БАҒДАРЛАМАЛАУ ТІЛІ

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

Қазақстан Республикасы
Жоғары оқу орындарының қауымдастығы

Brian W. Kernighan
Dennis M. Ritchie

С бағдарламалау тілі

Алматы, 2020

УДК 004
ББК 32. 973.202
С 11

Қазақстан Республикасы
Жоғары оқу орындарының қауымдастығы

Қазақ тіліне аударған:
Тюлепбердинова Гульнур Алпысқызы

С 11 **С бағдарламалау тілі.** Brian W. Kernighan,Dennis M. Ritchie. Алматы, «Фортуна Полиграф» баспасы ЖШС, 2020. – 392 бет.

ISBN 978-601-329-172-7

Біздің кітабымыздың мақсаты – оқырманға С бағдарламалау тілін үйренуге көмектесу. Басылым жаңадан бастаушыларға мүмкіндігінше тезірек программалауды үйреніп кетуге мүмкіндік береді. Оның негізін зерттеу, жазу және ережелер мен мысалдарды пысықтау тарауларда қамтылған. Біздің мысалдардың барлығы дерлік ажыратылған фрагменттер емес, нақты бағдарламалар.

Біздің тәжірибеміз С тілі – бағдарламалау үшін қолайлы, мәнерлі және икемді тіл екенін көрсетті. С-мен көбірек жұмыс істеңіз, сонда оны үйренуге және қолдануға ыңғайлы болады. Бұл кітап оны жақсы меңгеруге көмектеседі деп үміттенеміз.

УДК 004
ББК 32. 973.202

ISBN 978-601-329-172-7

Алғы сөз

Есептеуіш әлем 1978 жылы С тілінде бағдарламалау жарияланған сәттен бастап революцияға ұшырады. Үлкен компьютерлер әлдеқайда көп болғанымен, жеке компьютерлер он жыл бұрынғы мэйнфреймдермен бәсекелесу мүмкіндіктеріне ие болды.

Осы уақыт ішінде С да өзгерістерге ұшырады, бірақ ол UNIX операциялық жүйесінің тілі ретінде кең таралды.

С тілінің өсіп келе жатқан танымалдығы, жылдар ішіндегі тілдегі өзгерістер және құрастыруға қатыспайтын топтар бойынша компиляторлар – осы кітаптың бірінші басылымына қарағанда, тілді дәлірек және заманауи анықтау қажеттілігін көрсету мақсатында біріктірілген. 1983 жылы Американдық ұлттық стандарттар институты (ANSI) комитет құрды, оның мақсаты – бір мәнді және машинадан тәуелсіз анықтаманы шығару болды. Зерттеу нәтижесінде, С тіліндегі бағдарламалау жүйесінің үлгісі ойлап табылды.

Бұл үлгі туралы кітаптың бірінші шығарылымында көрсетілген, бірақ толық сипатталмаған және конструкцияларды ресімделмеген болатын. Ол функцияның жаңа формасын ұсынады және қарама-қарсы тексеруге мүмкіндік беретін декларация болып табылады, оның ерекшеліктері – енгізу мен шығару, жадты басқаруға арналған функциялардың кең жиынтығы, жолды манипуляциялау және ұқсас тапсырмаларға ие жаңа үлгі болуында. Ол бастапқы анықтамада айтылмаған мүмкіндіктерлің нақты әрекетін жасайды. Сонымен қатар, тілдің қай аспектісі болып қалатынын нақты белгілейді.

С бағдарламалау тілінің осы екінші басылымында ANSI анықтаған С үлгісі сипатталады. Біз тек қана заманауи түрде дамыған жазу түрлерін қолдануды қарастырдық. Ең айқын өзгеріс – бұл функция декларациясының және анықтаудың жаңа формасының қолданылуы. Қазіргі заманауи компиляторлар стандарттың көптеген мүмкіндіктерін қолдайды.

Біз бірінші басылымның көлемін қысқартуға тырыстық. Негізінде, С үлкен тіл емес және ол туралы мәліметтер үлкен кітапты қажет етпейді. Біз бағдарламалау орталығы болып табылатын маңызды сипаттамалардың экспозициясын жақсарттық. Оған қоса, біз бастап-

қы мысалдарды нақтылай отырып, бірнеше тарауларға жаңаларын қостық. Мысалы, күрделі декларацияларды өңдеу тәсілдерері және декларацияны сөзге немесе керісінше өзгертетін бағдарламалармен толықтырылған. Бұрынғыдай барлық мысалдар машинамен оқылатын мәтіннен тікелей тексерістен өткізілген.

Анықтамалық нұсқаулық А қосымшасы стандартты емес, бірақ біз оның кішірек кеңістіктегі маңыздылығын жеткізуге тырысамыз. Бұл бағдарламашыларды оңай түсінуге арналған, бірақ құрастырушы жазушылар үшін анықтама ретінде - бұл рөл стандарттың өзіне тиесілі. В қосымшасы стандартты кітапхана мүмкіндіктерінің қысқаша сипаттамасы болып табылады. Бұл да анықтама үшін арналған орындаушы емес, бағдарламашылар атынан. С қосымшасы – енгізілген өзгерістердің қысқаша түйіндемесі, яғни, түпнұсқасы.

Бірінші басылымдағы алғысөзде біз «С бағдарламалау тілімен неғұрлым көп жұмыс істеген сайын, ол соғұрлым ыңғайлы бола түседі» деп атап өткебіз. Бұл қасиет онымен он жылдан кейін де сақталады. Біз бұл кітап сізге С тілін түсінуге және практикада қолдануда пайдасын тигізеді деп сенеміз.

Біз кітаптың екінші басылымын шығаруға көмектескен достарымыз алдында қарыздармыз. Джон Бентли, Дуг Гуин, Дуг Макилрой, Питер Нельсон және Роб Пайк кітаптың әр бетте нақты ескертулер жасады. Денис Алиссон, Джей Кэмпбелл, Г. Р. Эмлинге ризамыз. Карен Фортганг, Аллен Голуб, Эндрю Хьюм, Дэйву Кристалл, Джон Линдерман, Дэйву Просер, Гену Спаффорд кітапты мұқият оқып шыққаны үшін және Крис Ван Уикке алғысымызды айтамыз. Биллден пайдалы кеңестер алдық және Питер Вайнбергер. Дейв Просер стандарт бөлшектеріне қатысты көптеген сұрақтарға жауап берді.

ANSI. Біз жергілікті тексеру үшін C++ Bjern Straustrup транслятырын кеңінен пайдаландық. Дев Кристалл бізге ANSI-С-компиляторды соңғы тексеру үшін берді. Рич Дрешлер кітапты жинауға өз үлесін қосты.

Барлығына алғысымыз шексіз.
Брайан В. Керниган, Деннис М. Ритчи

Мазмұны

Бірінші басылымның алғы сөзі

С тілі – қазіргі заманғы өрнектерді жазудың ықшам тәсілі бар әмбебап бағдарламалау тілі, ол – деректер құрылымын басқару механизмдері мен операторлардың бай жиынтығы. С тілі «өте жоғары деңгейдегі» немесе «үлкен» тілмен де өрнектелмейді, ол белгілі бір салаға қолдануға арналмаған. Дегенмен, көптеген міндеттерді шешу үшін кең мүмкіндіктер мен әмбебаптықтың арқасында олнеғұрлым қуатты тілдерге қарағанда ыңғайлы және тиімді.

Бастапқыда С тілі Денис Ритчи UNIX операциялық жүйесін жазу құралы ретінде құрылды, PDP-11 машиналары осы операциялық жүйе аясында жүзеге асырылды. Операциялық жүйе, Компилятор және Unix жүйесінің барлық қолданбалы бағдарламалары (соның ішінде осы кітаптың мәтінін дайындау үшін қолданылған) С-ға жазылған. Фирмалық С-компиляторлар бар және IBM/370, Honeywell 6000 және Interdata 8/32 басқа типті бірнеше машиналарда. С белгілі бір аппаратураға немесе жүйеге байланбаған, бірақ оған қолдау көрсетілетін басқа машиналарға қандай да бір өзгерістер енгізетін бағдарламаларды жазу оңай.

Біздің кітабымыздың мақсаты – оқырманға С бағдарламалау тілін үйренуге көмектесу.. Басылым жаңадан бастаушыларға мүмкіндігінше тезірек программалауды үйреніп кетуге мүмкіндік береді. Оның негізін зерттеу, жазу және ережелер мен мысалдарды пысықтау тарауларда қамтылған. Біздің мысалдардың барлығы дерлік ажыратылған фрагменттер емес, нақты бағдарламалар. Олардың барлығы машинада келтірілгендей кітапта да бейнеленген. Тілді тиімді пайдалануды көрсетуден басқа, жақсы жазу стилінің пайдалы алгоритмдері мен принциптерін көрсетуге тырыстық және олардың ақылға қонымды болатындай жобаладық.

Бұл кітап бағдарламалаудың кіріспе курсы емес. Оқырман алдын-ала «айнымалы», «тағайындау», «цикл», «функция» сияқты негізгі ұғымдармен таныс деп болжанып жазылған кітап. Дегенмен, бағдарламалау тілін жаңадан үйренушілерге де кітап өз септігін

тигізе алады, бірақ білімді мамандармен сұхбаттасу әдеқайда пайдалырақ болады.

Біздің тәжірибеміз С тілі — бағдарламалау үшін қолайлы, мәнерлі және икемді тіл екенін көрсетті. Оны үйрену оңай және С-мен көбірек жұмыс істеңіз, соғұрлым ол ыңғайлы болады. Біз бұл кітап оны жақсы меңгеруге көмектеседі деп үміттенеміз.

Көптеген достар мен әріптестердің сыны мен ұсыныстары бізге кітап жазуға көмектесті. Атап айтқанда, Майк Бианки, Джим Блу, Стью Фельдман, Дуг Макилрой, Рум, Боб Разин және Ларри Рослер. Біз Ахо, Стива Бьерна, Дана Дворак, Чака Хейли, Марион Харрис, Рика Холта, Стив Джонсон, Джон Маше, Баба Митц, Ральф Мухи, Питер Нельсон, Эллиот Пинсон, Билл Пейджер, Джерри Спивак, Кен Томпсон және Питер Вайнбергер, сондай-ақ Майк Леска мен Джо Осаннаға мәліметтерді жинауға ат салысқаны және баспаға дайындауға көмек көрсеткені үшін алғысымызды айтамыз.

Брайан В. Керниган,
Деннис М. Ритчи

1-тарау. Кіріспе

С тілімен жылдам танысудан бастайық. Біздің мақсатымыз – нақты бағдарламаларда кішкентай бөлшектерге, ресми ережелер мен ерекшеліктерге қарамай, тілдің маңызды элементтерін көрсету. Сондықтан біз толықтыққа және нақты дәлдікке ұмтылмаймыз (алайда, мысалдардың дұрыстығына назар аударамыз). Біз сізді пайдалы бағдарламаларды өзіңіз жаза алатын сәтке тезірек жеткізгіміз келеді. Мұны жылдамырақ жүзеге асыру үшін біз айнымалылар және тұрақтылар, арифметика, басқару есептеу тізбектілігі, функциялар және қарапайым енгізу-шығару тәсілдеріне назарымызды аударуымыз керек. Осы тарауда біз үлкен бағдарламаларды жазу кезінде қолданылатын маңызды тіл құралдары мен көрсеткіштер құрылымдары, операторлардың бай жиынтығының үлкен бөлігі, кейбір басқару нұсқаулары мен стандартты кітапхана туралы тақырыптарды қозғамаймыз.

Мұндай тәсілдің өз кемшіліктері де бар. Олардың ең маңыздысы - жеке тілдің тән қасиеті бір жерде толық сипатталмайды және мұндай қысқа оқыту кейбір ережелерді дұрыс қабылдамауға әкелуі мүмкін. Шектеулі уақытқа байланысты мысалдарда материалды беру сипаты тілдің барлық күші пайдаланылмайды, сондықтан олар қысқа болады. Біз бұл әсерлерді мүмкіндігінше жоюға тырыстық, дегенмен, ескерте кеткеніміз жөн деп шештік. Тағы бір кемшілігі келесі тарауларда қандай да бір сәттерді қайталау керек. Қайталаудың арқасында пайда болған кері әсерлер жойылады деп сенеміз.

Кез келген жағдайда тәжірибелі бағдарламашы осы тараудың материалын өз бағдарламалық қажеттіліктеріне жарата алады. Жаңадан бастағандарға оны оқуды өзі жазған бағдарламалармен толықтыруды ұсынамыз. Оқырмандарымыздың осы екі түрлері де осы тарауды негізге ала отырып, келесі 2-тарауда бағдарламалау тілдерінің элементтерімен танысады.

1.1. Бастайық

Жаңа бағдарламалау тілін үйренудің жалғыз жолы – бұл бағдарлама жазу. Зерттеу кезінде кез келген бірінші тіл әдетте келесі бағдарламаны жазуды ұсынады:

``сәлем,әлем'' сөзін экранға шығарамыз.

Міне, бірінші кедергі және оны жеңу үшін бағдарламаның мәтінін сәтті жасау керек. Дәлірек айтқанда,оны құрастыру, жүктеу, іске қосу және нәтиже қайда жіберілетінін түсіну қажет. Сіз оны меңгерген бойда қалғандары салыстырмалы түрде оңай болады.

"сәлем, әлем" С тілдік бағдарламасы төменде көрсетілгендей жазылады:

```
#include <stdio.h>
main()
{
printf("сәлем,әлем\n");
}
```

Бұл бағдарламаны қалай іске қосу сіз қолданатын жүйеге байланысты. Мысалы, UNIX амалдық жүйесінде атауды «.c» таңбаларымен аяқталатын файлда бастау керек, мысалы, hello.c файлында, содан кейін команданы қолдану арқылы құрастырылады.

```
cc hello.c
```

Егер сіз бәрін дұрыс жасасаңыз — таңбаны жіберіп алмаңыз және орфографиялық қателерді жібермесеңіз, онда компиляция "үнсіз" өтеді және сіз орындауға дайын a.out деп аталатын файлды аласыз. Егер сіз қазір бұл файлды командамен орындау үшін іске қоссаңыз,

```
a.out
```

бағдарлама экранға шығарады

```
hello, world
```

Басқа жүйелерде бағдарламаны іске қосу ережелері басқаша болуы мүмкін; олар туралы білу үшін, мамандармен сөйлесіңіз.

Енді бағдарламаның өзіне қатысты кейбір сәттерді түсіндіреміз. С тілдік бағдарламасы қандай мөлшерде болмасын функциялар мен айнымалылардан тұрады. Функциялар есептеулерді сипаттайтын, орындалуға тиіс және айнымалылар мен нұсқауларды қамтитын мәндерді сақтайды. С-дағы функциялар Фортранның кіші бағдарламалары мен функцияларына немесе Паскаль процедуралары мен оның функцияларына ұқсас. Бұл бағдарлама – main деп аталатын функция. Әдетте сіздер кез келген атауларды ойлап таба аласыздар, бірақ “main” - ерекше атау: кез келген бағдарлама өз есептеулерін бірінші нұсқаулығын main функциясынан бастайды.

Әдетте main өз жұмысын орындау үшін басқа функцияларды пайдаланады; олардың бірі бағдарламашының өзімен жазылады, ал басқалары қолында бар кітапханалардан дайын болады. Бірінші бағдарлама жолы:

```
#include <stdio.h>
```

компиляторға стандартты енгізу-шығару кітапханасы туралы ақпаратты қосу керек екенін хабарлайды. Бұл жол көптеген стандартты енгізу-шығару кітапханасы туралы ақпаратты қамтиды және бағдарламалардың бастапқы файлдарының басында кездеседі. Стандартты кітапхана 7-тараудың В қосымшасында сипатталған.

Функциялардың арасындағы деректерді беру тәсілдерінің бірі — бұл функция екіншісіне жүгінген кезде функциялар оған дәлелдер деп аталатын мәндердің тізімін беруі. Бұл тізім жақшаға алынады және функцияның атауынан кейін орналастырылады. Біздің мысалда main күтпейтін функция ретінде анықталған бос тізіммен () белгіленген.

```
#include <stdio.h> Стандартты кітапхана туралы
ақпаратты қосу
```

main() Ешқандай дәлелдері жоқ main деп аталатын функцияны анықтау

```
{main нұсқаулары фигуралы жақшаларға жатады.

printf("сәлем, әлем\n"); Main функциясы кітапхана функциясын тудырады
printf басып шығару үшін
таңбалар; \n — жаңа жолдың символы.

}
```

С тіліндегі алғашқы бағдарлама

Функцияның нұсқаулары фигуралық жақшаларға тұрады {}. main функциясы тек бір нұсқауды қамтиды.

```
printf ("сәлем, әлем\n");
```

Функция аты бойынша шақырылады, содан кейін жақшада дәлелдердің тізімі көрсетіледі. Осылайша, жоғарыда көрсетілген жол-printf функциясының "сәлем, әлем\n"аргументі бар қоңырауы, printf функциясы — бұл символдар тізбегін басып шығаратын қос тырнақшаға қамалған кітапхана функциясы, "сәлем, әлем\n" сияқты қос тырнақшадағы символдар тізбегі жол таңбалар немесе жол константасы деп аталады. Әзірге printf үшін аргументтер және басқа мүмкіндіктер ретінде біз тек таңбалар жолдарын қолданамыз.

С-де таңбалар жолының ішіндегі комбинация \n жаңа жолдың таңбасын көрсетеді және басып шығарғанда келесі жолдың сол жақ шетіне өтеді. Егер сіз \n жойсаңыз (эксперимент ретінде), онда басып шығаруды аяқтағаннан кейін машина жаңа жолға өтпейді. Мәтін аргументінің жаңа жол белгісі printf анық болуы керек. Егер сіз орындауға тырыссаңыз, мысалы,

```
printf(«сәлем, әлем
«);
```

компилятор қате туралы хабар береді.

Жаңа жолдың белгісі ешқашан автоматты түрде орнатылмайды, сондықтан бір жолды қадамдар бойынша printf-ке бірнеше рет жүгіну арқылы басып шығаруға болады. Біздің бірінші бағдарламаны төмендегідей жазуға болады:

```
#include<stdio.h>
main()
{
printf(“hello, “);
printf(“world”);
printf(“\n”);
}
```

Оны орындау нәтижесінде, сол жол басып шығарылады.

Назар аударсақ, \n тек бір таңбаны білдіреді. Басталатын таңбалардың ерекше комбинациялары және эскейп тізбектері деп аталатын кері қисық сызықтары үшін кеңінен қолданылады. С бағдарламалау тілінде \t, \b, \”, \\ символдары бар, олар сәйкесінше табуляцияны, бір символға кері қайтару, екі есе жақша және көлбеу сызықты білдіреді. Осы таңбалардың толық тізімі 2.3. параграфта берілген.

1.1 жаттығу Жүйеңізде “сәлем, әлем” басып шығаратын бағдарламаны орындаңыз. Бағдарламаның кейбір бөліктерін жою арқылы эксперимент жасап көріңіз және сіз қандай қателер туралы хабарларды алатыныңызды қараңыз.

1.2 жаттығу Егер printf аргументінің жол тұрақтылығына кірістіру \c болса, онда не пайда болатындығын анықтаңыз. (жоғарыда көрсетілген тізімге кірмейтін таңба)

1.2. Айнымалы және арифметикалық өрнектер

Төменде берілген бағдарлама °C = (5/9) (T-32) формуласы бойынша есептеулерді орындайды және кестені басып шығарады. Фаренгейт бойынша температуралардың Цельсий бойынша температураға сәйкестігі:

```
1    -17
20   -6
40   4
60   15
80   26
100  37
120  48
140  60
160  71
180  82
200  93
220  104
240  115
260  126
280  137
300  148
```

Алдыңғы сияқты, бұл бағдарлама main бір ғана жалғыз функциядан тұрады. Ол “сәлем, әлем” басып шығаратын бағдарламадан ұзағырақ болғанымен, шын мәнінде қиын емес. Онда біз бірнеше пікір, хабарландырулар, айнымалы, арифметикалық өрнектер, циклдар және форматтық қорытынды сынды жаңа мүмкіндіктерді қарастыра аламыз.

```
#include <stdio.h>

/* print Фаренгейт бойынша температура кестесін басып шығару
және Цельсий үшін fahr = 0, 20, ..., 300 */
main()
{
    int fahr, celsius;
    int lower, upper, step;
```

```
    lower = 0;        /* температура кестесінің төменгі шегі */
    upper = 300;      /* жоғарғы шегі */
    step = 20;        /* қадам */

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
fahr = fahr + step;
    }
}
```

Екінші жол:

/* print Фаренгейт бойынша температура кестесін басып шығару және Цельсий үшін fahr = 0, 20, ..., 300 */

бағдарламаның не істейтінін қысқаша түсіндіреді. Барлық /* және */ таңбалары арасында орналастырылған компилятор еленбейді және оныбағдарламаны түсіну үшін еркін пайдалануға болады. Пікірді кез келген бос табуляция немесе жаңа жолдың символы тұрған жерге қоюға болады.

С-да кез келген айнымалы ол қолданылғаннан бұрын жариялануы керек; әдетте, барлық айнымалылар функцияның басында бірінші орындалатын нұсқаулық алдында жарияланады. Хабарландыруда айнымалылардың қасиеттері сипатталады. Ол айнымалылардың түрі мен тізімінен тұрады, мысалы:

```
int fahr, celsius;
  int lower, upper, step;
```

Int түрі аталған айнымалылардың мәндері float түріне қарағанда бүтін екенін білдіреді және өзгермелі нүктелі мәндерді көрсетеді, яғни бөлшек бөлігі болуы мүмкін сандарды. Ауқымы екі түрдің мәндері қолданылатын машинаға байланысты. Int типті сандар 16-биттік (-32768-ден 32767-ге дейінгі диапазонда), сондай-ақ 32-биттік. Float түрді сандар әдетте кемінде 6 ондық сөздері бар 32 биттік сөздермен ұсынылады (шамамен 10-38-ден 1038-ге дейінгі диапазонда жатыр).

С-да int және float-дан басқа бірнеше базалық түрлері бар. Атап өтсек:

```
char — символ-дара байт;
short — қысқа бүтін сан ;
long  —  ұзын бүтін сан;
double — қос дәлдікпен өзгермелі нүкте.
```

Аталған типтегі нысандардың көлемі машинаға байланысты. Негізгі түрлерден: массивтер, құрылымдар мен бірлестіктер, базалық типтердің объектілеріне көрсеткіштер және осы мәндерді қайтаратын функциялар түрлерін жасауға болады Бұл туралы біз кейінірек тоқтала кетеміз.

Температураны түрлендіру бағдарламасындағы есептеулер беру нұсқауларынан басталады:

```
lower = 0;
upper = 300;
step = 20;
```

Олар оларда көрсетілген айнымалыларды бастапқы мәндерге орнатады. Кез келген нұсқаулық нүктелі үтірмен аяқталады.

Кестенің барлық жолдары бірдей жолмен есептеледі, сондықтан біз әрбір жол үшін бұл есептеу қайталанатын циклді пайдаланамыз. Қажетті әрекеттер while циклін орындайды:

```
while (fahr <= upper) {
 ...
}
```

Ол келесідей жұмыс істейді. Шарт жақшада тексеріледі. Егер ол шынайы болса (Fahr мәні аз немесе upper мәніне тең), онда цикл денесі орындалады (фигуралы жақшаларға жасалған үш нұсқаулық). Содан кейін шарт қайтадан тексеріледі, егер ол шын болса, цикл денесі қайтадан орындалады. Шарт жалған болған да (fahr upper-ден асқанда), цикл аяқталады және есептеулер келесі

цикл нұсқаулығымен жалғасады. Цикл үшін ешқандай нұсқаулар жоқ болғандықтан, бағдарлама жұмысын аяқтайды.

While циклінің денесі бір немесе бірнеше жақшаға алынған нұсқаулар болуы мүмкін. Мысалы:

```
while (i < j)
        i = 2 * i;
```

Сол және басқа да жағдайда while басқаруындағы нұсқауларды, біз бағдарламада төрт бос орын көрсетілген бір табуляция позициясына тең жылжумен жазамыз. Осының арқасында цикл ішінде орналасқан нұсқаулар анық көрінеді. Негізінде, С-компилятор сыртқы дизайнға назар аудармайды, бірақ дұрыс жерлерде шегіністер мен бос орын болуы адамға ақпаратты оңай қабылдауға ықпал жасай алады. Өрнектің логикалық құрылымы жақсы көрінуі үшін біз әр жолда тек бір нұсқауды және екі жағынан да бос орындарды қалдырып жазуды ұсынамыз. Жақшалардың орналасуы маңызды емес, бірақ бұл тұрғыда әртүрлі көзқарастар бар. Біз оларды қолданудың бірнеше жалпы стильдерінің біреуіне ғана тоқталамыз. Сіз өзіңізге ұнаған біреуін ғана нақты таңдаңыз және ең бастысы оны қатаң ұстаныңыз.

Есептеулердің көп бөлігі цикл денесінде орындалады. Фаренгейт бойынша температура Цельсий шкаласы бойынша температураға ауыстырылады және нұсқаулық арқылы айнымалы celsius беріледі.

```
celsius = 5 * (fahr-32) / 9;
```

5/9-ға қарапайым көбейтудің орнына, 5-ке көбейту және 9-ға бөлу себебі - С тілінде, басқа да көптеген тілдердегідей, бүтін бөлу қиылады, яғни, кез келген бөлшек бөлігі алынып тасталады. 5 және 9 бүтін сандар болғандықтан, 5/9 нөлге дейін кесіледі, сондықтан Цельсий бойынша барлық температура нөлдік деп хабарланады.

Бұл мысал бізге printf функциясы қалай жұмыс істейтіні туралы біраз мағлұмат бере берді. Printf функциясы туралы 7-тарауда егжей-тегжейлі сипатталатын болады. Оның бірінші аргументі - әрбір % таңбасы келесі аргументтердің біріне сәйкес келеді (екінші,

үшінші), ал % символынан кейін орналасқан ақпарат әрқайсысы шығатын аргументтер түрін көрсетеді. Мысалы, %d символы аргументті бүтін ондық түрінде және нұсқаулық түрінде Fahr-ді басып шығарады, табуляцияны орындайды (\t) және тұтас celsius-ді басып шығарады. Төмендегі мысалға назар аударыңыз:

```
printf("%d\t%d\n", fahr, celsius);
```

Бірінші Аргументтің әрбір спецификаторына printf функциясында (конструкция %-дан басталады) екінші дәлел, үшінші дәлел және т.б. сәйкес келеді. Алайда олар саны мен түрі бойынша келісілуі керек, әйтпесе қажет нәтижені ала алмауыңыз мүмкін.

Айтпақшы, printf С тілінің бір бөлігі емес. Жалпы, тілде ешқандай арнайы енгізу-шығаруды анықтайтын құрылымдар жоқ. Printf функциясы стандартты, С-бағдарламалар үшін қол жетімді пайдалы функция ғана. Printf қызметінің қасиеттері ANSI стандартымен және оның қасиеттер стандарт талаптарын қанағаттандыратын барлық С-жүйелерде бірдей болуы тиіс.

Сіздің толық назарыңызды С-ға шоғырландыру үшін біз 7-тарауға дейін енгізу-шығару туралы көп айта қоймаймыз. Атап айтқанда, біз форматты енгізу туралы әңгімені кейінге қалдырамыз. Сандарды енгізу қажет болса, 7.4 параграфында scanf функциясы туралы толық оқуға кеңес береміз.

Бұл функция printf-дан деректерді енгізетіндігімен ерекшеленеді.

Температураны түрлендіру бағдарламасына байланысты тағы екі мәселе бар. Олардың бірі (қарапайымы) – бұл нәтиже бірнеше жинақы емес көрінеді, себебі сандар колонкалардың оң жағында туралanбаған. Бұл мәселені шешу оңай, әрбір формат спецификаторларына %d-ны қосқан жағдайда, бағдарлама сандарды оң жақ шетіне қысып, басып шығарады. Мысалы, егер төменде көрсетілгендей жазсақ,

```
printf("%3d %6d\n", fahr, celsius);
```

әрбір жолда бірінші санды үш позицияның біреуіне, ал екіншісін алты позицияның бірінің жолына басып шығарамыз. Нәтижесінде экранға төмендегідей сандар шығады:

```
0       -17
20      -6
40       4
60      15
80      26
100      37
```

Екінші, неғұрлым маңызды мәселе, біз бүтін арифметиканы пайдаланамыз, сондықтан, Цельсий шкаласы бойынша температураны дәл есептеу емес. Мысалы, 0°F шын мәнінде (дәлдікпен онға дейін) -17.8 °C тең, -17 емес. Температураның дәл мәндерін алу үшін біз бүтін арифметиканы емес, өзгермелі нүктелі арифметиканы қолдануымыз керек. Бұл кейбір бағдарламада өзгерістерді талап етеді.

```
#include <stdio.h>

/* print Fahrenheit-Celsius table
for fahr = 0, 20, ..., 300; floating-point version */
main()
{
float fahr, celsius;
float lower, upper, step;

lower = 0;      /* lower limit of temperatuire scale */
upper = 300;    /* upper limit */
step = 20;      /* step size */

fahr = lower;
while (fahr <= upper) {
celsius = (5.0/9.0) * (fahr-32.0);
  printf("%3.0f %6.1f\n", fahr, celsius);
fahr = fahr + step;
}
}
```

Бағдарламада айтарлықтай өзгерістер орын алған жоқ. Ол алдыңғыдан fahr және celsius-ті float ретінде жариялағанымен ғана ерекшеленеді. Алдыңғы нұсқада 5/9 жазуға болмайтын болған, өйткені бөлшек бөлікті лақтыру нәтижесінде нәтижені

нөл деп береді. Константадағы ондық нүкте соңғы өзгермелі нүктелі сан ретінде қарастырылатынын көрсетеді және 5.0/9.0, осылайша, екі мәнді өзгермелі нүктемен бөлуден бөлек, ол бөлшек бөлігін тастауды көздейді. Арифметикалық операция бүтін болған жағдайда, операндалар бүтін арифметика ережелері бойынша орындалады.

Егер бір операнд қалқымалы нүкте, ал екіншісі – тұтас болса, операция орындалғанға дейін, соңғысы өзгермелі нүктелі сан өзгереді. Егер біз fahr-32 деп жазсақ, онда 32 автоматты түрде өзгермелі нүктелі санға өзгереді. Дегенмен, өзгермелі нүктемен тұрақты жазу кезінде біз әрқашан ондық нүктені пайдаланамыз, тіпті константалар шын мәнінде тұтас мәнге ие болған жағдайларда да. Бұл оқу бағдарламасының табиғатына назар аудару үшін жасалады.

Қандай жағдайларда тұтас өзгермелі нүктелі сандарға аударылатынын анықтайтын ережелер толығырақ, 2-тарауда қаралады.

Тағайындау және тексеру табиғи түрде жұмыс істейді, яғни операцияны орындау алдында int мәні float келтіріледі. Төмендегі жазбаға назар аударыңыз:

```
fahr = lower;
 while (fahr <= upper)
```

%3.0f ерекшелігі - Printf-те ені ондық нүктесіз және бөлшек бөліксіз үш позициядан аспайтын өрісте қалқымалы нүктелі санды басып шығаруды анықтауында (бұл жағдайда fahr саны).

%6.1f-тің ерекшелігі – басқа санның (celsius) ондық нүктеден кейін бір цифрмен алты позицияның өрісінде басып шығарып сипаттауында.

Экранға төмендегі сандар шығады:

```
0       -17.8
20     -6.7
40     4.4
```

Ені мен дәлдігін көрсетпеуге болады: %6f сан алты позициядан артық емес екенін білдірсе, ал ал %.2F-сан ондық нүктеден кейін екі сан бар, бірақ ені шектелмейтіндігін білдіреді; %f жай ғана сандарды қалқымалы нүктемен басып шығаруды көрсетеді.

%d	ондық бүтін басып шығару
%6d	алты позицияның өрісінде ондық бүтін басып шығару
%f	қалқымалы нүктелі санды басып шығару
%6f	алты позицияның өрісінде қалқымалы нүктесі бар санды басып шығару
%.2f	ондық нүктеден кейін екі цифрмен өзгермелі нүктелі санды басып шығару
%6.2f	алты позицияның өрісінде ондық нүктеден кейін өзгермелі нүкте және екі цифрмен санды басып шығару

Сонымен қатар, printf келесі сипаттамаларға рұқсат береді: сегіздік сан үшін %o символы; он алтылық сан үшін %x символы, таңба үшін %c символы; таңба жолы үшін %s және % символдары, ал өзі үшін %% символы қолданылады.

1-3 жаттығу. Тақырыпты кестеден жоғары басып шығару үшін температураны түрлендіру бағдарламасын өзгертіңіз.

1-4 жаттығу. Тиісті Цельсий және Фаренгейт кестесін басып шығару бағдарламасын жазыңыз.

1.3. For Нұсқаулығы

Бір бағдарламаны жазудың көптеген жолдары бар. Температураны түрлендіру бағдарламасын құрып көрейік.

```
#include <stdio.h>
/* Фаренгейт және Цельсий бойынша температура кестесін басып шығару */
main()
{
 int fahr;
      for (fahr = 0; fahr <= 300; fahr = fahr + 20)
  printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

Бұл бағдарлама бірдей нәтижені басып шығарады, бірақ ол, әрине, басқаша көрінеді. Басты айырмашылық – көптеген айнымалылардың болмауы. Тек қана int ретінде жарияланған fahr айнымалысы ғана қалды. Төменгі және жоғарғы шекаралар мен for қадамы – жаңа нұсқаулығында тұрақты түрінде болады, ал Цельсий температурасын есептейтін өрнек енді үшінші жеке тағайындау нұсқаулығында емес, printf функциясының аргументінде болады.

Соңғы өзгеріс жалпы ережені қолданудың мысалы болып табылады: мүмкін болатын кез келген контексте қандай да бір түрдегі айнымалы мәннің пайдалануында, сол күрделі өрнекті қолдануға болады. Осылайша, printf функциясының үшінші аргументінің орнында %6.1f спецификаторы бойынша өзгермелі нүкте мәні болуы керек, демек, бұл түрдегі кез келген өрнек болуы мүмкін.

Нұсқаулық for while циклінің жалпылауы болып табылатын циклді сипаттайды. Егер сіз оны бұрын жазылған while-мен салыстырсаңыз, онда ол қалай жұмыс істейтіні анық болады. Жақшаның ішінде нүкте үтірмен бөлінген үш өрнек бар. Бірінші өрнек-инициализация:

```
fahr = 0
```

циклға кірер алдында бір рет орындалады. Екінші-циклді жалғастыру шартын тексеру:

```
fahr <= 300
```

Шарт есептеледі, егер ол ақиқат болса, цикл денесі орындалады (біздің жағдайда бұл printf-ке бір үндеу). Содан кейін қадамның өсуі жүзеге асырылады:

```
fahr = fahr + 20
```

Осыдан кейін шарт қайтадан есептеледі. Цикл шарт жалған болған кезде аяқталады. While жағдайындағыдай for-цикл денесі бір нұсқаулықтан немесе фигуралы жақшаларға салынған бірнеше нұсқаулықтан тұруы мүмкін. Бұл үш өрнектің орнында (инициализация, шарттар және қадамның өсуі) еркін өрнектер тұра алады.

While мен FG арасындағы таңдау бағдарламаның айқындылығы тұрғысынан анықталады. Цикл for қадамды инициализациялау үшін ыңғайлы және бір-бірімен логикалық байланысы болған жағдайларда немесе бұл цикл while циклінен ықшам болған жағдайда, бір ғана басқару бөліктері бір жерде шоғырланған нұсқаулармен көрсетіледі.

1-5 жаттығу. Кестені кері ретпен, яғни 300 градустан 0-ге дейін басу үшін температураны түрлендіру бағдарламасын өзгертіңіз.

1.4. Атаулы константалар

Температураны түрлендіру бағдарламасын қарастырмас бұрын, тағы бір дайын нәрсе шығады. Бағдарлама бойынша 300, 20 сияқты " жұмбақ сандар " шашыраған кез дұрыс емес. Бағдарламаны оқып отырған адам олардың нені білдіретіндігіне күмәнданбайды да. Сонымен қатар, оларды басқа жүйелі тәсілмен ауыстыру қиындық туғызады. Осындай мәселелерді шешуге мүмкіндіктердің бірі – оларға мағыналы атаулар беру. #Define жолы таңбалы атауды немесе константа атауын анықтайды.

#define *мәтінге қойылған атау*

Осы кезден бастап кез келген атау пайда болған кезде (егер ол мәтінге енбесе, тырнақша және басқа атаудың анықтамасына кірмейді) оның орнына сәйкес келетін алмастыру мәтіні жазылады. Атаудың ауыспалы формасы бірдей: әріптер мен сандар тізбегі әріптен басталады. Ауыстыру мәтіні кез келген таңбалардың кезектілігі бола алады, тек сандарды ғана табуға болмайды.

```
#include <stdio.h>

#define LOWER  0       /* lower limit of table */
#define UPPER  300     /* upper limit */
#define STEP   20      /* step size */

/* print Fahrenheit-Celsius table */
main()
```

```
{
        int fahr;
    for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
  }
```

LOWER, UPPER және STEP шамалары айнымалы емес, сондықтан олар үшін хабарландырулар жоқ. Жалпы қабылданған келісім бойынша аталған константалардың аттары айнымалылардан ерекшелену үшін бас әріптермен теріледі. #define соңында нүктелі үтір қойылмайтындығына назар аударыңыз.

1.5. Таңбаларды енгізу-шығару

Енді біз мәтіндерді өңдеу бағдарламаларының ошағын қарастырамыз. Сіз өзіңіздің қолданысыңыздағы бағдарламалар мұнда талқыланатын бағдарламалардың прототиптерінің кеңейтілген нұсқалары болып табылатынын байқайсыз. Стандартты кітапхана өте қарапайым кіріс-шығыс моделін қолдайды. Мәтіннің енгізу-шығаруы қайдан немесе қайда бағыталғанына қарамастан,символдар ағынымен жұмыс істейді. Мәтіндік ағын – бұл әр жолдан бөлінген таңбалар тізбегі, олардың әрқайсысы нөл немесе одан да көп және жаңа жолдың символымен аяқталады. Кез келген енгізу ағынының осы модельге жауап беруін қадағалау міндеті кітапханаға жүктеледі. Бағдарламашы кітапхананы пайдалана отырып, бағдарламаның сыртында жолдың қандай түрінде берілетініне алаңдамауы керек.

Стандартты кітапхана бір таңбаны оқу және жазу үшін бірнеше мүмкіндіктерді қамтиды. Қарапайымдарының бірі – getchar және putchar. Бір getchar үшін келесі енгізу белгісі оқылады және бұл таңба нәтиже ретінде беріледі. Осылайша, орындалғаннан кейін,

```
c = getchar();
```

c айнымалысы келесі енгізу белгісін қамтиды. Әдетте таңбалар пернетақтадан келеді. Файлдардан енгізу 7-тарауда қаралады.

Putchar-ға жүгіну бір таңбаны басып шығаруға әкеледі. Сонымен,

```
putchar(c);
```

бүтін айнымалы с мазмұнын таңба түрінде (әдетте экранда) басып шығарады. Putchar шақырулары және printf еркін түрде ауыса алады. Нәтиже осы функцияларды шақырумен бірдей ретпен жасалады.

1.5.1. Файлды көшіру

Getchar және putchar функциялары бар болса, Енгізу-шығару туралы ештеңе білмей, таңқаларлық көптеген пайдалы бағдарламаларды жазуға болады. Қарапайым мысал – шығыс ағынынан кіріс ағынына символын бір-бірден көшіретін бағдарлама:

```
таңбаны оқу
        while (таңба файл соңындағы белгі емес)
жаңа ғана оқылған таңбаның нәтижесі
                таңбаны оқу
Converting this into C gives:

    #include <stdio.h>

 /* шығуды көшіру; 1-ші нұсқа */
main()
{
int c;

c = getchar();
while (c != EOF) {
putchar(c);
c = getchar();
}
}
```

Қатынас операторы != «теңемес» дегенді білдіреді.

Пернетақтадан енгізілетін немесе экранда пайда болатын әр-бір таңба комбинациямен кодталады. Char түрі арнайы символды сақтауға арналған, дегенмен, бұл үшін кез келген түрі де жарамды.

Біз түсіндіруді талап ететін бір маңызды себеп үшін int түрін пайдаланамыз және оны жасаймыз.

Әдеттегі оқылатын деректерден енгізудің соңын қалай ажыратуға болады? Бұл мәселенің шешімі – getchar функциясы кіріс ағыны таусылған кезде нәтиже ретінде ешқандай нақты символмен шатастыруға болмайтын мәнді беруі керек. Бұл мән EOF (файлдың соңы – файлдың end аббревиатурасы). Getchar функциясы беретін барлық ықтимал нәтижелерді ұсыну үшін осы түрден айнымалыны «жеткілікті» нәтижені жариялау керек. Бізге char түрі сәйкес келмейді, себебі с char түрінің кез келген мәнінен басқа, сақтау және EOF болуы үшін жеткілікті, яғни, «сыйымды» болуы керек. Сондықтан біз char емес, int пайдаланамыз.

EOF – бұл <stdio.h> анықталған бүтін сан. Бұл тұрақты мәннің қандай мәні бар екендігі маңызды емес, өйткені char түрінің мүмкін болатын кез келген мәнінен өзгеше болады. Атаулы тұрақты мәнді бірыңғай атаумен қолдану бағдарламаның белгілі бір сандық мәнге тәуелді болмауын қамтамасыз етеді, бұл басқа С жүйелерінде әр түрлі болуы мүмкін.

Көшіру бағдарламасын қысқартып жазуға болады. С-да кез келген өрнек, мысалы:

```
c = getchar()
```

бергеннен кейін сол жақ бөлігінің мәніне тең мәнімен өрнек ретінде түсіндіріледі. Бұл өрнек күрделі өрнектің ішінде болуы мүмкін дегенді білдіреді. Егер айнымалыны с өрнегін while циклінің шарттарын тексергенде қойса, көшіру бағдарламасын келесі түрде жазуға болады:

```
#include <stdio.h>

/* шығуды көшіру; 2-ші нұсқа */
main ()
{
int c;
```

```
while ((c = getchar()) != EOF)
putchar (c);
}
```

While циклі getchar-дан алынған мәнді с-ға жіберу арқылы бірден оның «файлдың соңы» болып табылмайтындығын тексереді. Егер олай болмаса, while циклінің денесі орындалады және таңба теріледі. Енгізу аяқталғаннан кейін while циклінің жұмысы да, ain де аяқталады.

Бұл нұсқада енгізу «орталықтандырылған» - бағдарламада getchar-ға бір ғана жүгіну бар. Нәтижесінде ол оқу кезінде ықшам және оңай қабылданады. Сіз жиі тексерумен бірге берілетін жазбаның осындай түріне тап болуыңыз керек (оған шамадан тыс қолдану бағдарламаны шатастыруы мүмкін, сондықтан біз аталған нысанды ақылға қонымды пайдалануға тырысамыз).

Шарт ішіндегі жақшалар, өрнектің айналасына қойылуы қажет. != -- ның басымдылығы = қарағанда жоғары. Жақшалар жоқ кезде != өрнегі = операциясына дейін орындалады. Осылайша,

```
  c = getchar() != EOF
```

жазбасының баламасы төмендегідей болады:

```
c = (getchar() != EOF)
```

Ал, бұл бізге қажет нәрсе емес. Айнымалы с файл соңындағы белгіні кездестіру немесе кездестірмеуіне байланысты 0 немесе 1 беріледі. (Бұл туралы толығырақ 2-тараудан қараңыз).

1-6 жаттығу. getchar()!= EOF өрнегінің мәні 0 немесе 1 екенін дәлелдеңіз.

1-7 жаттығу. EOF мәнін басып шығаратын бағдарламаны жазыңыз.

1.5.2. Символдарды есептеу

Келесі бағдарлама символдарды санаумен айналысады. Ол көшіру бағдарламасымен ұқсас қасиеттерге ие.

```
#include <stdio.h>

/* енгізілетін таңбаларды есептеу; 1-ші нұсқа */
main ()
{
        long nc;

        nc = 0;
        while (getchar() != EOF)

 ++nc;
printf ("%ld\n", nc);
}
```

++nc; нұсқаулығы++ жаңа операторын ұсынады, ол бірлікке көбейтуді білдіреді. Оның орнына nc=nc+1 жазуға болатын еді, бірақ ++nc әлдеқайда қысқарақ әрі тиімдірек. Бірлікке азайтуды білдіретін – операторы бар. ++ және -- операторлары префиксті (++nc), сондай-ақ постфиксті (nc++) бола алады. 2-тарауда көрсетілгендей, бұл екі форманың өрнектерінде әртүрлі мағыналар бар, бірақ ++nc және nc++ nc бірлігіне қосылады. Бұл жағдайда біз префикс жазбасына тоқтадық.

Таңбаларды есептеу бағдарламасы long түріндегі айнымалы соманы жинайды. Ұзын тұтас түрі кемінде 32 бит бар. Кейбір машиналарда int және long типтері бірдей болса да, int 16 биттік ең көп 32767 битті алатын машиналар бар, ал бұл салыстырмалы түрде аз сан және int есептегіші толып кетуі мүмкін. Printf-дегі %ld тиісті дәлел long түрі бар екенін көрсетеді.

Double түрін (яғни екі дәлдікпен float) қолдансаңыз, мәндердің одан да үлкен ауқымын қамтуға болады. Циклді жазудың басқа әдісін көрсету үшін while орнына for нұсқауларын да қолдануға болады.

```
#include <stdio.h>

/* енгізілетін таңбаларды есептеу; 2-нұсқа */
main ()
 {
      double nc;

      for (nc = 0; getchar() != EOF; ++nc)
           ;
      printf (“%.0f\n”, nc);
}
```

Printf %f спецификаторы float үшін де, double үшін де қолданылады; % 0f спецификаторы ондық нүктесіз және бөлшек бөліксіз басып шығаруды білдіреді (біздің жағдайда соңғысы жоқ).

Көрсетілген for-циклдің денесі бос, өйткені тексерулерден және есептегіштің өсімінен басқа ештеңе істеудің қажеті жоқ. Бірақ С грамматикасының ережелері for-циклінің денесінің болуын талап етеді. Бұл талапты орындау нұсқаулық деп аталатын оқшауланған үтірмен қамтамасыз етеді. Біз көрнекілік үшін бөлек жолда үтір мен нүкте қойдық.

Сонымен қатар, егер де бір таңбаны енгізбесе, онда getchar-ға бірінші рет жүгінген кезде while немесе for-да шарт орындалмайды және бағдарлама дұрыс нәтиже болатын нөл береді. Бұл өте маңызды. While және For циклдарының тартымды қасиеттерінің бірі – бұл шарт цикл денесі орындалғанға дейін тексерілетіндігі. Егер ештеңе істеудің қажеті болмаса, онда циклдің денесі ешқашан орындалмаса да, ештеңе істелмейді. Бағдарлама енгізілген символдардың нөлдік саны кезінде өзін сенімді ұстауы керек. While және For циклдер құрылғы өзі шекаралық жағдайда бағдарламаның дұрыс мінез-құлқына қосымша сенімділік береді.

1.5.3. Жолдарды есептеу

Келесі бағдарлама жолдарды есептейді. Жоғарыда айтылғандай, стандартты кітапхана кіріс мәтіндік ағынының әрқайсысы жаңа

жолдың символымен аяқталатын жолдар тізбегінен тұратын кіріс-шығыс моделін қамтамасыз етеді. Демек, жолдарды санау жаңа жолдың символдар санын санауға келеді.

```
#include <stdio.h>

/* кіріс ағыны жолдарын есептеу */
main ()
{
        int c, nl;
        nl = 0;
        while ((c = getchar()) != EOF)
 if (c == '\n')
                        ++nl;
printf ("%d\n", nl);
 }
```

Циклдің денесі енді if нұсқаулығын жасайды, оның бақылауында nl есептегішінің бірлікке өсуі болады. If нұсқаулығы жақшадағы шартты тексереді және егер ол шын болса, келесісін нұсқаулық бойынша орындайды (немесе фигуралық жақшаға жасалған нұсқаулықтар тобы). Біз тағы да бағдарламаның мәтінінде не басқарылатынын көрсету үшін шегіністерді жасаймыз.

С тіліндегі теңдіктің қос белгісі оператордың "тең" екендігін білдіреді. Белгінің екі еселенуі = теңдікті тексеру операторында С-да қолданылатын бірліктен ажырату үшін жасалған. Ескерту: С бағдарламалауды бастағандар кейде = жазады, ал ол === білдіреді. 2-тарауда көретініміздей, бұл жағдайда, нәтиже әдетте, компилятор ешқандай ескертуші хабарлама бермейтін нысанда жарамды өрнек болады.

Бір тырнақшаға алынған таңба – бұл таңбаның кодына тең бүтін сан (осы машинада қабылданған кодтауда). Бұл символдық константа деп аталады. Кішкентай мәндерді жазудың басқа жолы бар. Мысалы, 'A' символдық константа; ASCII таңбалар жиынтығында оның мәні A символының ішкі көрінісіне тең. Әрине, "A" константа рөлінде 65-ке қарағанда жақсырақ, өйткені бірінші жазбаның мағынасы

айқын және ол таңбаларды кодтаудың нақты әдісіне байланысты емес.

Жол константыларында қолданылатын Эскейп-бірізділік символдық константыларда да рұқсат етіледі. Осылайша,' \n ' AS-CII-де 10-ға тең жаңа жол таңбасының кодын білдіреді. '\n' бір таңбаны білдіреді (оның коды бүтін мән ретінде қарастырылады), ал "\ n" — бір таңбаны кездейсоқ көрсеткен жол константасы. Символдық және жол константалары арасындағы айырмашылық 2-тарауда егжей-тегжейлі талданады.

1.8-жаттығу. Бос орындарды, табуляцияларды және жаңа жолдарды санау үшін бағдарламаны жазыңыз.

1.9-жаттығу. Шығыс ағынына енгізу таңбаларын көшіретін және қатарынан тұрған бос орындарды бір бос орынға ауыстыратын бағдарламаны жазыңыз.

1.10-жаттығу. Енгізілетін таңбаларды Шығыс ағынына көшіретін бағдарламаны табуляция белгісін \t-ға, забой символын \b-ға және әрбір кері көлбеу белгіні \\ - ға ауыстыра отырып жазыңыз. Бұл табуляция және забой таңбаларын көрінетін етеді.

1.5.4. Сөздерді санау

Біздің пайдалы бағдарламалар сериясындағы төртінші жолдар, сөздер мен таңбаларды есептейді, сөз деп мұнда жаңа жолдың бос орындары, табуляциялары және символдары жоқ символдардың кез келген жолын айтып тұрмыз. Бұл бағдарлама UNIX жүйесінің wc бағдарламасының жеңілдетілген нұсқасы болып табылады.

```
#include <stdio.h>

#define IN 1 /* внутри слова */
#define OUT 0 /* вне слова */

/* жолдар, сөздер мен таңбаларды есептеу */
main ()
{
```

```
int c, nl, nw, nc, state;
state = OUT;
nl = nw = nc = 0;
while ((c = getchar()) != EOF) {
++nc;
 if (c == '\n' )
++nl;


if (c == " " || c == '\n' || c == '\t')
state = OUT;
else if (state == OUT) {
state = IN;
++nw;
}
printf ("%d %d %d\n", nl, nw, nc);
}
```

Әр кезде, сөздің бірінші таңбасын кездестіргенде, бағдарлама сөз санағышының мәнін 1-ге өзгертеді. Айнымалы state ағымдағы жағдайды белгілейді – біз іште немесе сөзден сыртында екенімізді. Алдымен оған "сөзден тыс" күйіне сәйкес келетін OUT мәні беріледі. Біз бағдарламаны түсінікті ету үшін 1 және 0 мәндерін емес, IN және OUT деп аталатын тұрақтарды пайдалануды жөн көреміз. Мұндай шағын бағдарламада бұл әдістің нәтижесі аз, бірақ үлкен бағдарламада оның айқындығын арттыру басынан бастап осындай стильде бағдарламаны жазуға жұмсалған қосымша күш-жігерді талап етеді. Сіз сиқырлы сандар тек атаулы константтар түрінде кездесетін бағдарламаларға үлкен өзгерістер енгізу оңайырақ екенін анықтайсыз.

```
nl = nw = nc = 0;
```

жолы барлық үш айнымалыны нөлге орнатады. Мұндай жазба қандай да бір ерекше құрылым емес және рұқсат етіледі, себебі тағайындау өз мәні бар өрнек болып табылады және белгілеу операциялары оңнан солға орындалады. Көрсетілген жол мынаған тең:

```
nl = (nw = (nc = 0));
```

Оператор || НЕМЕСЕ деген мағына береді, сондықтан:

```
if (c == ‘ ‘ || c == ‘\n’ || c == ‘\t’)
```

жолы «егер с бос орын болса, немесе с жаңа жол болса, немесе Стабуляция болса» деп оқылады. (көрінетін эскейп-реттілік \t табуляция белгісін білдіреді.) Сондай-ақ бар && операторы бар, ол ЖӘНЕ деген мағына береді. Оның басымдылығы || операторына қарағанда жоғары. && және || операторлармен байланысқан өрнектер солдан оңға қарай есептеледі; бұл ретте шарттың шынайылығы немесе жалған болуы анықталғаннан кейін есептеулер бірден үзіледі деп кепілдік беріледі. Егер с бос орын болса, онда жаңа жолдың немесе табуляцияның белгісіне мән бермесе де болады. Жеке жағдайда есептеудің бұл әдісі маңызды емес, бірақ ол көп ұзамай қарастыратын күрделі жағдайларда маңызды.

Мысалда else сөзі бар, ол if-да көрсетілген шарт шынайы емес болған жағдайда орындалатын баламалы әрекеттерді көрсетеді. Жалпы түрде шартты нұсқаулық осылай жазылады:

```
if (өрнек)
    нұсқаулық1
else
    нұсқаулық2
```

If-else конструкциясында екі нұсқаудың біреуі ғана орындалады. Егер өрнек шын болса, онда 1, Егер жоқ болса, онда 2-нұсқаулық орындалады. Осы екі нұсқаулықтың әрқайсысы бір немесе бірнеше нұсқаудан тұрады. Біздің бағдарламада else-ден кейінгі екі нұсқаулықты басқаратын if нұсқаулығы бар.

1.11-жаттығу. Сөздерді санау бағдарламасын қалай тексеруге болады? Қателік жіберілген болса, оны енгізудің ықтималдылығы қандай?

1.12-жаттығу. Әр жолда бір сөзді орналастыра отырып, өз жазбаңыздың мазмұнын басып шығаратын бағдарламаны жазыңыз.

1.6. Массивтер

Ал енді әр санды жеке-жеке санайтын бағдарламаны жазамыз, бөлгіш таңбалар (бос орындар, табуляция, жаңа-жолдар) және барлық басқа таңбалар. Бұл бірнеше жасанды бағдарлама, бірақ ол бір мысалда С тілінің бірнеше мүмкіндіктерін көрсетуге мүмкіндік береді.

Енгізілетін символдардың он екі санаты бар. Барлық он санауыштарды он жеке айнымалы түрінде емес, массивте сақтау ыңғайлы. Бұл бағдарламаның бір нұсқасы:

```
#include <stdio.h>

/* сандарды, таңбаларды-бөлгіштерді және басқа да символдарды
есептеу */
main()
{
        int c, i, nwhite, nother;
        int ndigit[10];

        nwhite = nother = 0;
        for (i = 0; i < 10; ++i)
            ndigit[i]= 0;

        while ((c = getchar()) != EOF)
              if (c >= '0' && c <= '9' )
                  ++ndigit[c - '0' ];
              else if (c == ' ' || c == '\n' || c == '\t')
                  ++nwhite;
     else            +
    +nother;

    printf ("цифры =");
     for (i=0; i < 10; ++i)
    printf(" %d", ndigit[i]);
    printf (", символы-разделители = %d, прочие = %d\n",
     nwhite, nother);
    }
```

Бұл бағдарламаның қорытындысы мынадай:

```
сандар = 9 3 0 0 0 0 0 0 0 1, бос орын = 123, басқа = 345
```

int ndigit[10];
ndigit int типті 10 мәнінен массивін жариялайды.

Индекс тұтас айнымалылармен (мысалы, 1) және тұтас константалармен пайда болатын кез келген бүтін өрнек болуы мүмкін.

Берілген бағдарлама сандарды кодтаудың белгілі бір сипаттарына сүйенеді. Мысалы, тексеру:

if (c >= ‘0’ && c <= ‘9’

с-дағы таңбаның сан екенін анықтайды. Егер бұл осылай болса, онда:

c - ‘0’

санның сандық мәні бар. Айтылғандар тек ‘0’, ‘1’, ...’9’ мәндерінің қатарында әрбір келесі мән алдыңғысынан 1-ге көп жағдайда ғана орындала алады. Бұл ереже барлық таңбалар жиынтығында сақталады.

Анықтама бойынша, char түрінің мәндері жай ғана шағын бүтін болып табылады, сондықтан арифметикалық өрнектердегі char түрінің айнымалылары int түрінің мәндеріне ұқсас. Бұл өте ыңғайлы; мысалы, С - ‘0’ 0-ден 9-ға дейінгі ықтимал мәндері бар бүтін өрнек бар, олар с айнымалысында сақталатын ‘0’-ден ‘9’-ға дейінгі символдарға сәйкес келеді.

Келесі фрагмент санның бөлгіш немесе басқа нәрсе символы болып табылатындығын анықтайды.

```
if (c >= ‘0’ && c <= ‘9’)
    ++ndigit[c - ‘0’];
else if (c ==’ ‘ || c == ‘\n’ || e == ‘\t’)
    ++nwhite;
else
    ++nother;
```

Көрініс құрылымы

```
if (условие1)
    инструкция1
else if (условие2)
    инструкция2
```

```
else
    инструкция n
```

бағдарламада бар бірнеше баламалы жолдардың бірін таңдау үшін жиі қолданылады. Шарттар реті бойынша жоғарыдан төменге қарай олардың біреуі қанағаттандырылғанға дейін есептеледі; бұл жағдайда оған сәйкес нұсқаулық орындалады және барлық конструкцияның жұмысы аяқталады. (Нұсқаулардың кез келгені фигуралы жақшадағы нұсқаулардың тобы болуы мүмкін.) Егер шарттардың ешқайсысы қанағаттандырылмаса, else үшін бірден орналасқан соңғы нұсқаулық орындалады. Егер else және одан кейінгі нұсқау болмаса (сөздерді есептеу бағдарламасында болғандай), онда ешқандай әрекет мүлдем жасалмайды. Бірінші if және соңғы else арасында көріністің комбинациясы болуы мүмкін.

```
else if (условие)
    инструкция
```

Олар бірнеше болған кезде, бағдарламаны біз мұнда көрсеткендей пішімдеу ақылға қонымдырақ. Егер әрбір келесі if алдыңғы else-ге қатысты оңға жылжытса, онда ұзын тексеру каскадында мәтін беттің оң жақ шетіне тым жақын басылады.

3-тарауда сөз болатын switch нұсқаулығы С тілінде көп жолды тармақтауды бейнелеудің басқа әдісін қамтамасыз етеді. Ауысу шарты берілген жиынтыққа кіретін тұрақтылардың біреуімен бүтін түрдегі кейбір өрнектің мәнінің сәйкес келген кезде қолданылады. Switch көмегімен іске асырылған біздің бағдарламаның нұсқасы 3.4 параграфында келтірілген.

1.13-жаттығу. Енгізілетін сөздердің гистограммасының ұзындығын басып шығаратын бағдарламаны жазыңыз. Гистограмманы көлденең жолақтармен салу оңай. Тік жолақтарды салу-қиынырақ тапсырма.

1.14-жаттығу. Енгізілетін символдардың кездесу жиілігінің гистограммасын басып шығаратын бағдарламаны жазыңыз.

1.7. Функциялар

С-дағы функциялар Фортрандағы кіші бағдарламалар мен функциялар немесе Паскалдағы процедуралар мен функциялар сияқты рөл атқарады. Функция кейбір есептеуді бөлек ресімдеудің ыңғайлы жолын қамтамасыз етеді және оны қалай жүзеге асырылғанына қарамастан әрі қарай пайдаланады. Функциялар жазылғаннан кейін, олар қалай жасалғанын ұмытып, олар не істей алатынын білу жеткілікті. Функцияны С-да пайдалану механизмі ыңғайлы, жеңіл және тиімді. Сіз жиі тек бір рет туындаған қысқа функцияларды кездестіруіңіз мүмкін; олар бір айқын бағдарлама алуды мақсат етіп, функция түрінде безендірілген.

Әлі күнге дейін біз main, getchar және putchar сияқты дайын функцияларды пайдаландық. С-да Фортрандағы ** сияқты дәрежеге салу операторы жоқ. Сондықтан функцияны анықтау механизмін power(m, n) функциясының мысалында бейнелейміз, ол бүтін m-ді тұтас оң дәрежеге көтереді. Іс жүзінде қолдану үшін бұл функция ең оңтайлының бірі емес, өйткені тек кіші бүтін дәрежелермен жұмыс істейді, алайда, ол иллюстрация бола алады. (Стандартты кітапханада xy есептейтін pow(x, u) функциясы бар.)

Сонымен, бізде power функциясы және оның қызметтерін пайдаланатын негізгі функциясы бар, сондықтан барлық бағдарлама келесідей:

```
#include <stdio.h>
int power(int m, int n);

/* power функциясының тесті */
main()
{
 int i;
for (i = 0; i < 10; ++i)
printf("%d %d %d\n", i, power(2,i), power(-3, i));
return 0;
}

/* возводит base в n-ю степень; n >= 0 */
```

```
int power(int base, int n)
{
int i, p;
P = 1;
for (i = 1; i <= n; ++i)
p = p * base;
return p;
}
```

Кез келген функцияның анықтамасы мынадай:

```
түрі – нәтиже аты – функция (параметрлер тізімі, егер ол бар болса)
{
    хабарландыру
    нәтижесі
}
```

Функциялардың анықтамалары кез келген тәртіпте бір немесе бірнеше бастапқы файлдарда орналасуы мүмкін, бірақ кез келген функция толығымен бір бірімен орналасуы керек. Егер бағдарламаның бастапқы мәтіні бірнеше файлға бөлінген болса, оны құрастыру және жүктеу үшін бір файлды пайдаланғаннан гөрі бірнешесі дұрысырақ; бірақ бұл тілге емес, операциялық жүйеге қатысты. Әзірге біз екі функция бір файлда тұр деп ойлаймыз, сондықтан сіз С бағдарламаларын іске қосуға қатысты алған білімдеріңіз жеткілікті болады.

Келесі жолда main функциясынан power екі рет қосылады:

```
printf(“%d %d %d\n”, i, power(2,i), power(-3,1));
```

Әрбір power функциясын шақыруда екі аргументтер беріледі, және әр уақытта басты main бағдарламасы жауапты бүтін сан алады, ол содан кейін тиісті пішімге келтіріледі және басып шығарылады. Power (2,1) өрнегінің ішінде 2 немесе 1 сияқты бүтін мәнді білдіреді.)

Power анықтаудың бірінші жолында:

```
int power(int base, int n)
```

параметрлер түрлері, функцияның аты және нәтиже түрі көрсетіледі. Параметр атаулары power ішінде жергілікті, бұл кез келген басқа функция үшін жасырылған, сондықтан басқа кіші бағдарламалар өз мақсаттары үшін бірдей атауларды еркін пайдалана алады. Соңғы мәлімдеме I және P: I-де power және I-де main-де айнымалылар үшін де ортақ ештеңе жоқ.

Бұдан әрі параметр ретінде біз параметрлер тізімінен айнымалыны дөңгелек жақшаға қойылған және функцияны анықтағанда берілген, ал аргумент-функцияны қолданғанда қолданылатын мән деп атаймыз. Кейде сол мағынада біз ресми дәлел мен нақты дәлел терминдерін қолданамыз.

Power функциясы есептейтін мән return нұсқаулығымен main-ге қайтарылады. Return сөзінің артында кез келген өрнек болуы мүмкін:

```
return өрнек;
```

Функция қандай да бір мәнді міндетті түрде қайтармайды. Есептеу процесінде біз соңғы жабылатын фигуралық жақшамен мәтінде көрсетілген функцияның соңына шығамыз. Шақыру функциясы оған қайтарылатын нәтижені елемейтін жағдай да болуы мүмкін.

Сіз main соңында return нұсқаулығына назар аударған боларсыз. Main кез – келген басқа функция сияқты болғандықтан, ол іс жүзінде бағдарлама іске қосылған ортада оны шақырған адамға нәтиже бере алады. Әдетте нөлдік мән қайтарылады, бұл орындалудың қалыпты аяқталғанын білдіреді. Нөлдік емес мән ерекше немесе қате аяқталғаны туралы сигнал береді. Осы уақытқа дейін қарапайымдылық үшін біз return-ді main-ге түсірдік, бірақ осы сәттен бастап return-ге бағдарламалар өзінің аяқталуының жай-күйі туралы операциялық жүйеге хабарлауға тиіс екенін еске салу ретінде қоямыз.

```
int power(int m, int n);
```

main алдында тұрған power функциясы екі nt аргументін күтеді және int түрінің нәтижесін қайтарады. Бұл функция прототипі деп аталатын хабарлама анықтаумен және барлық power қоңыраулармен келісілген болуы керек. Егер функцияны анықтау немесе шақыру прототипіне сәйкес келмесе, бұл қате болып табылады.

Параметрлер атаулары келісуді талап етпейді. Шын мәнінде прототипте олар еркін немесе мүлдем болмауы мүмкін, яғни прототипті былай да жазуға болады:

```
int power(int, int);
```

Дегенмен, сәтті таңдалған аттар бағдарламаны түсіндіреді және біз оны жиі қолдана аламыз.

Тарихи анықтама. ANSI-C-дің тілдің ерте нұсқаларынан ең үлкен айырмашылықтары - жарнама және функцияларды анықтау тәсілдерінен тұрады. С-ның бірінші нұсқасында power функциясын келесі түрде орнату қажет болды:

```
/* power: base-ті n-ші дәрежеге көтереді; n >= 0 */
/* (С тілінің ескі стиліндегі нұсқасы) */
power(base, n)
int base, n;
(
    int i, p;

   P = 1;
   for (i = 1; i <= n; ++i)
       p = p * base;
   return p;
}
```

Мұнда параметрлер аттары дөңгелек жақшаларда көрсетілген, ал олардың түрлері бірінші ашылатын фигуралық жақшаның алдында берілген. Параметр түрі туралы нұсқау болмаған жағдайда, оның int түрі бар деп саналады. (Функцияның денесі өзгерген жоқ.)

С бірінші нұсқасына сәйкес бағдарламаның басында power сипаттамасы келесідей болуы керек:

```
int power();
```

Параметрлер тізімін орнату мүмкін болмады, сондықтан компилятор power-қа жүгінудің дұрыстығын тексеру де мүмкін болмады. Хабарландыру болмаған кезде power функциясы int түрінің мәнін қайтарады деп болжанғандықтан, бұл жағдайда хабарландыру толығымен түсірілуі мүмкін еді.

Функциялардың прототипі үшін жаңа синтаксис компиляторды аргументтер саны мен олардың түрлерінде қателерді табуды жеңілдетеді. Ескі хабарландыру синтаксисі және функцияны анықтау әлі де ANSI стандартымен рұқсат етіледі, бірақ егер компилятор жаңа синтаксисті кем дегенде өтпелі кезеңде қолдаса, біз тек оны пайдалануды ұсынамыз.

1.15-жаттығу. Температураны түрлендіру бағдарламасын бөлек функцияға түрлендіру арқылы қайта жазыңыз.

1.8. Аргументтер. Мән бойынша шақыру

Дәлелдер берудің бұл тәсілі сілтеме бойынша шақырудан біршама ерекшеленеді, Fortran-да және Паскальдағы параметр үшін var-спецификация, бұл жоспардың жергілікті көшірмелеріне емес, дәлелдерге өздері қол жеткізуге мүмкіндік береді.

Негізгі айырмашылық мынада: С-да шақырылатын функция шақырылатын функцияның айнымалысын тікелей өзгерте алмайды: ол тек оның жеке, уақытша көшірмесін өзгерте алады.

Алайда, маңызы жағынан шақыруды тілдің кемшіліктеріне емес, оның қасиеттеріне жатқызу керек. Осы қасиеттің арқасында, әдетте, бөтен айнымалылардың аз санын қамтитын ықшам бағдарламаны жазуға болады, себебі параметрлерді тиісті түрде инициалданған жергілікті айнымалылар деп қарастыруға болады. Мысал ретінде, осы сипат қолданылған power функциясының тағы бір нұсқасын ұсынамыз.

```
/* power: base-ті n-ші дәрежеге көтереді: n >= 0; 2 нұсқа */
int power(int base, int n)
```

```
{
    int p;
    for (p = 1; n > 0; --n)
    p = p * base;
    return p;
}
```

N параметрі мұнда уақытша айнымалы рөлінде шығады, онда for циклінде азаю тәртібімен оның мәні нөлге жеткенге дейін қадамдар санының есебі жүргізіледі. Бұл ретте цикл есептегіші үшін қосымша і айнымалы қажеттілігі жоғалады. Power ішіндегі n-мен не істесек де, бұл көшірмесін power функциясы шақырғанда берілген дәлелге әсер етпейді.

Егер қаласаңыз, функция шақырушы бағдарламада айнымалыны өзгерте алатындай етіп жасауға болады. Бұл үшін соңғысы өзгеріске жататын мекен-жайды (айнымалыға көрсеткіш) беруі тиіс, ал шақырылатын функцияда көрсеткіш ретінде тиісті параметрді жариялау және ол арқылы осы айнымалыға жанама қатынауды ұйымдастыру керек. Көрсеткіштерге қатысты толық мағлұматтарды біз 5-тарауда қарастырамыз.

Аргумент ретінде массив беру механизмі өзгеше. Аргумент массивтің аты болған кезде, функциялар осы массивтің басталу мекен-жайы болып табылатын мән беріледі; массивтің ешқандай элементтері көшірілмейді. Алынған мәнге қатысты индекстеу арқылы функция массивтің кез келген элементіне қол жеткізе алады. Бұл туралы әңгіме келесі параграфта айтылатын болады.

1.9. Символдық массивтер

С тіліндегі массивтің ең көп таралған түрі - символдық массивтер. Символдық массивтерді және олармен жұмыс істеу функцияларын пайдалануды көрсету үшін мәтіндік жолдар жиынтығын оқитын және олардың ең ұзынын басып шығаратын бағдарламаны жазамыз. Оның схемасы өте қарапайым:

```
while (тағы да жол бар ма?)
```

```
 if (бұл жол алдыңғы жолдың ең ұзынынан да ұзынырақ)
        оны еске сақтау
ұзындығын еске сақтау
ең ұзын жолды теру
```

Схемадан бағдарлама табиғи түрде бөліктерге бөлінетінін көруге болады. Олардың бірі жаңа жолды алады, екіншісі оны тексереді, үшіншісі есте сақтайды, ал қалғандары есептеу процесін басқарады.

Процесс бөлікке анық ыдырағаннан кейін, оны С-ға аудару жақсы болар еді. Сондықтан, алдымен басқа жолды алу үшін getline функциясын жазамыз. Біз бұл функцияны басқа қолданбалар үшін пайдалы етуге тырысамыз. Кем дегенде getline мүмкіндігі туралы сигнал беруі керек. Файлдың соңында немесе егер ол жолдың ұзындығын берсе одан да жақсы, файл таусылған жағдайда нөлге тең. Нөл файлдың соңының белгісі үшін жарамды, өйткені нөлдік ұзындық жолы жоқ; тіпті жаңа жолдың бір ғана белгісі бар жол ұзындығы 1-ге тең болады.

Біз алдыңғысынан да ұзын жолды тапқан кезде, оны есте сақтау қажет болады. Мұнда жаңа жолды сенімді орынға көшіре алатын екінші функция, copy қажет болады.

Ақырында, getline және copy функцияларын басқаратын басты бағдарлама қажет. Біздің бағдарламамыз қалай көрінеді:

```
#include <stdio.h>
#define MAXLINE 1000 /* енгізілетін жолдың ең үлкен мөлшері */

int getline(char linef], int MAXLINE);
void copy(char to[], char fromf]);

/* ең ұзын жолды басып шығару */
main()
{
int len; /* ағымдағы жолдың ұзындығы */
int max; /* қаралған жолдардың максималды ұзындығы */
char line[MAXLINE]; /* ағымдағы жол */
char longest[MAXLINE]; /* ең ұзын жол */
```

```
max = 0;
while ((len = getline(line, MAXLINE)) > 0)
if (len > max) {
max = len;
copy(longest, line);
}
if (max > 0) /* қандай-да бір жол болды ма? */
printf("%s", longest);
 return 0;
}

/* getline: S-да жолды оқиды, ұзындығын қайтарады */
int getline(char s[], int lim)
{
int c, i;

for (i = 0; i < lim-1 && (c = getchar()) != EOF && c != '\n'; ++i)
s[i] = c;
if (c == 'n'; {
s[i] = c;
++i;
}
s[i] = '\0';
return i;
}

/* copy: 'from' ды 'to' ға көшіреді; to жеткілікті үлкен */
void copy(char to[], char from[])
{
 int i;

  i = 0;
 while ((to[i] = from[i]) != '\0')
++i;
}
```

Бағдарламаның басында сипатталған getline және copy функциялары main сияқты бірдей файлда.

main және getline функциялары бірнеше дәлелдер мен қайтарылатын мән арқылы өзара әрекеттеседі. getline аргументтері жол арқылы анықталады.

```
int getline(char s[], int lim);
```

Көріп отырғанымыздай, оның бірінші аргумент s массиві, ал екіншісі, lim int түрі бар. Анықтаудағы массивтің мөлшерін белгілеу жадты резервтеуді мақсат етеді. getline-дің өзінде s массивінің ұзындығын орнатудың қажеті жоқ, себебі оның өлшемі main-де көрсетілген. Қоңырау шалу бағдарламасын қайтару үшін getline қуат функциясы сияқты return пайдаланады. Жоғарыда келтірілген жол getline-ге int мәнін қайтаратыны туралы айтады, бірақ егер индикатор болмаса, int сөзі getline-дан бұрын алынып тасталуы мүмкін.

Кейбір функциялар нәтиже мәнін қайтарады, басқалары (сору сияқты) ешқандай мән берместен, кейбір әрекеттерді жасау үшін қажет. Сору-дағы нәтиже түрінің орнында void тұр. Бұл – функция ешқандай мәнді қайтармайды деген айқын нұсқау.

Getline функциясы ол жасаған массивтің соңында таңбалар жолының ұшын белгілеу үшін ‘\0’ (null-нөлдік байтпен кодталған таңба) таңбасын орналастырады. Бірақ нөлдік тоқтату туралы келісім тұрақты жол тәрізді жағдайда да сақталады.

“hello\n”

Бұл жағдайда осы жолдың таңбаларынан массив «соңында» жасалады.

h	e	l	l	0	\n	\0

Printf пішіміндегі %s спецификациясы жоғарыда көрсетілген таңбалардың аргументі жол екенін көрсетеді. сору функциясы өз жұмысында оқылатын дәлел басқа таңбалармен бірге көшіретін ‘\0’ символымен аяқталады. (Барлық айтылғандар ‘\0’ әдеттегі мәтін ішінде жоқ деп болжанылады.)

Осындай шағын бағдарламамен жұмыс істеу кезінде кейбір конструктивті қиындықтарға тап болуымыз мүмкін екенін атап өткен жөн. Мысалы, жол рұқсат етілген мөлшерден асатын болса, main не істеу керек? Getline функциясы сенімді жұмыс істейді: егер массив толы болса, ол жаңа жолдың таңбасын білмесе де, қайта

жіберуді тоқтатады. Getline-ден жолдың ұзындығын алып, MAX-LINE-ге сәйкес келетінін көріп, main негізгі бағдарламасы осы ерекше жағдайды "ұстап" және оны жеңе алады. Қысқартушылық үшін біз бұл істің сипаттамасын осында қалдырамыз.

Getline пайдаланушылары енгізілетін жолдар қанша ұзақ болатынын алдын-ала біле алмайды, сондықтан getline толып кетуін тексереді. Бірақ пайдаланушылар көшірілетін жолдардың өлшемдері белгілі (немесе олар біле алады), сондықтан мұнда қосымша бақылау қажет емес.

1.17-жаттығу. 80-нен астам таңбадан тұратын барлық енгізілетін жолдарды басып шығару бағдарламасын жазыңыз.

1.18-жаттығу. Әрбір енгізілетін жолда қатарынан тұрған бос орындар мен табуляция таңбаларын бір бос орынға ауыстыратын және бос жолдарды алып тастайтын бағдарламаны жазыңыз.

1.19-жаттығу. S жолында символдарды кері тәртіпте орналастыратын reverse(s) функциясын жазыңыз. Оны әрбір енгізілетін жол кері тәртіпте болатын бағдарламаны жазу кезінде қолданыңыз.

1.10. Сыртқы айнымалылар және көріну аймағы

Айнымалы line, longest және басқалар тек main функцияларына ғана тиесілі. Олар main ішінде жарияланғандықтан, оларға тікелей басқа мүмкіндіктер болмайды. Басқа функциялардың айнымалысына қатысты да дәл солай. Мысалы, і getline-да соору-да і-ге ешқандай қатысы жоқ. Функцияның әрбір жергілікті айнымалысы тек осы функцияға жүгіну кезінде пайда болады және одан шыққаннан кейін жоғалады. Сондықтан мұндай айнымалылар басқа тілдердің терминологиясын автоматты деп атайды. (4-тарауда static жад класы талқыланады, ол жергілікті айнымалы өз мәндерін шақырулар арасында сақтауға мүмкіндік береді.)

Автоматты айнымалылар кіру және шығу функциясымен бір уақытта құрылып, жойылғандықтан, олар өз мәндерін сақтаймайды

да, әр функцияға жаңа өтініш сайын жаңадан орнатылуы тиіс. Егер бұл болмаса, олардың құрамында "қоқыс" болады.

Автоматты айнымалыға балама ретінде кез келген функциядан олардың аттары бойынша жүгінуге рұқсат етілетін сыртқы айнымалыларды анықтауға болады. (Бұл механизм Фортрандағы COMMON саласына және Паскалдағы ең сыртқы блоктағы айнымалыларды анықтауға ұқсас.) Сыртқы айнымалылар барлық жерде қол жетімді болғандықтан, оларды деректер бойынша функциялар арасындағы байланыс үшін аргументтердің орнына пайдалануға болады. Сонымен қатар, сыртқы айнымалылар тұрақты болғандықтан, функцияны орындау кезеңінде пайда болып, жоғалмағандықтан, олардың мәндері оларды орнатқан функциялардан қайтарылғаннан кейін де сақталады.

Сыртқы айнымалы кез келген функцияның мәтінінен тыс бір рет анықталуы керек; бұл жағдайда оған жады бөлінеді. Ол оны қолданғысы келетін барлық функцияларда жариялануы керек. Хабарландыру айнымалы түрі туралы ақпаратты қамтиды. Хабарландыру қажетті ақпарат контекстен алынған кезде, extern нұсқаулығы түрінде анық немесе анық емес болуы мүмкін. Айтылғанды нақтылау үшін, біз line, longest және max пайдаланып ең ұзын жолды басып шығару бағдарламасын сыртқы айнымалы ретінде қайта жазамыз. Бұл барлық үш функцияның шақыруларында, хабарландыруларында және денелерінде өзгерістерді талап етеді.

```
#include <stdio.h>

#define MAXLINE 1000 /* енгізілетін жолдың ең үлкен мөлшері */

int max; /* қаралған жолдардың максималды ұзындығы */
char line[MAXLINE]; /* ағымдағы жол */
char longest[MAXLINE]; /* ең ұзын жол */

int getline(void);
void copy(void);

/* ең ұзын жолды басып шығару; мамандандырылған нұсқасы */
main ()
```

```
{
int len;
extern int max;
 extern char longest[];

max = 0;
while ((len = getline()) > 0)
 if (len > max) {
max = len;
 copy();
}
 if (max > 0) /* кем дегенде бір жол болды */
printf("%s", longest);
return 0;
}

/* getline: мамандандырылған нұсқасы */
int getline(void)
{
 int c, i;
extern char line[];

 for (i=0; i < MAXLINE-1 && (c=getchar()) != EOF && c != '\n'; ++i)
 line[i] = c;
if(c == '\n') {
line[i]= c;
++i;
}
line[i] = '\0';
return i;
}

/* copy: мамандандырылған нұсқасы */
void copy (void)
{
 int i;
extern char line[], longest[];

i = 0;
while ((longest[i] = line[i]) != '\0')
++i;
}
```

Main, getline және copy үшін сыртқы айнымалылар біздің

мысалымыздың басында анықталады, онда оларға түрі мен жады беріледі. Сыртқы айнымалыларды анықтау жергілікті айнымалыларды анықтаудан айырмашылығы жоқ, бірақ олар функциялардан тыс орналасқандықтан, бұл айнымалылар сыртқы болып саналады. Сыртқы айнымалыны пайдалану үшін, ол ең алдымен тиісті айнымалының атын хабарлау керек. Бұл, мысалы, сыртқы айнымалы хабарландырудан айырмашылығы бар, ол тек extern кілт сөзінен басталады.

Кейбір жағдайларда extern хабарламасын түсіруге болады. Егер сыртқы айнымалыны анықтау бастапқы файлда ол пайдаланылатын функциядан жоғары болса, онда extern хабарландыруында қажеті жоқ. Осылайша, main, getline және copy extern хабарландырулары артық. Әдетте, сыртқы айнымалыларды анықтау бастапқы файлдың басында орналасады және олар үшін барлық extern хабарландырулар түсіріледі.

Егер бағдарлама бірнеше бастапқы файлдарда орналасқан болса және сыртқы айнымалы 1 файлында анықталған болса, және 2 файлында және 3 файлында пайдаланылатын болса, онда 2 файлында және 3 файлында extern хабарландырулары міндетті, себебі барлық үш файлдарда функциялар бір сыртқы айнымалыға айналады. Іс жүзінде, әдетте, сыртқы айнымалылар мен функциялардың барлық хабарландыруларын тақырып (header-файл) деп аталатын жеке файлға жинау және оны әрбір бастапқы файлдың басына #include арқылы орналастыру ыңғайлы. Header-файл атауларында ортақ келісім бойынша h жұрнақ қолданылады. Бұл файлдарда, атап айтқанда <stdio.h>, сондай-ақ стандартты кітапхана функциялары сипатталады. Тақырып файлдары туралы толығырақ 4-тарауда, ал стандартты кітапханаға қатысты 7-тарауда және В қосымшасында айтылады.

Getline және copy мамандандырылған нұсқаларында дәлелдер жоқ болғандықтан, олардың прототиптерін getline() және copy () түрінде қою қисынды болып көрінеді. Бірақ ескі С-бағдарламалармен үйлесімділік тұрғысынан стандарт аргументтердің сәйкестігінің барлық тексерулерін өшіру үшін сигнал ретінде бос тізімді

қарастырады. Сондықтан бақылауды сақтау және аргументтердің болмауын анық көрсету керек болғанда, void сөзін қолдану керек. Біз 4-тарауда осы мәселеге ораламыз.

Бұл параграфта сыртқы айнымалыға қатысты біз анықтама мен хабарландыру ұғымдарын өте мұқият қолданамыз. "Анықтама" айнымалы жад құрылатын және оған берілетін жерде орналасады;" хабарландыру " айнымалы табиғаты бекітілген жерде орналастырылады, бірақ оған ешқандай жад берілмейді.

Барлық айнымалыларды сыртқы етіп жасау үрдісін атап өту керек. Өйткені, бір қарағанда, бұл байланыстарды жеңілдетуге әкеледі, аргументтер тізімі қысқа болады, ал айнымалылар қажет болған жерде қол жетімді болады; алайда олар қол жетімді және қажет емес ол жерде. Сондықтан сыртқы айнымалыларға шамадан тыс назар үлкен қауіп төндіреді, ол деректер бойынша анық емес бағдарламаларды құруға әкеледі, себебі айнымалылар күтпеген және тіпті жұмбақ жолмен өзгеруі мүмкін. Сонымен қатар, мұндай бағдарламаны өзгерту қиын. Ең ұзын жолды іздеу бағдарламасының екінші нұсқасы біріншіге қарағанда нашар, осы себептерге байланысты олар жұмыс істейтін нақты айнымалылардың аттарына сәйкес келетін екі пайдалы функциялардың ортақтығы бұзады.

Сонымен, біз С ядросы деп атауға болатын нәрсені қарастырдық. Сипатталған "кірпіштер" айтарлықтай мөлшерде пайдалы бағдарламаларды жасау үшін жеткілікті және оқуды үзіп, оған біраз уақыт арнасаңыз керемет болар еді. Келесі жаттығуларда біз сізге жоғарыда қарастырылғаннан гөрі бірнеше күрделі бағдарламаларды жасауды ұсынамыз.

1.20-жаттығу. Енгізілетін мәтінде табуляция таңбаларын ауыстыратын detab бағдарламасын қажетті бос орындар санымен жазыңыз (келесі табуляцияның аяғына дейін). Табуляцияның аяғы бір-бірінен белгіленген қашықтықта орналасқан, айталық, N позиция арқылы. N — айнымалы мән түрінде немесе белгілі тұрақты түрінде қалай орнатуға болады?

1.21-жаттығу. Бос орындардан тұратын жолдарды ауыстыратын entab бағдарламасын табуляциялардың және бос орындардың ең аз санымен басып шығарылған мәтіннің түрі өзгермейтіндей етіп жазыңыз. Detab сияқты табуляция аяғын пайдаланыңыз. Кезекті "тоқтауға" шығу үшін бір бос орын жарамды, қайсысы жақсы-бос орын немесе табуляция?

1.22-жаттығу. Мәтін жолдары n-ші позициядан оң шықпайтындай етіп кіріс ағынының таңбаларын басып шығаратын бағдарламаны жазыңыз. Бұл ұзындығы n-дан асатын әрбір жол келесі жолдарға көшіре отырып басылуы тиіс дегенді білдіреді. Тасымалдау орнын n-ші позицияның сол жағында орналасқан бөлгіш-символдан басқа соңғы символдан кейін "іздеу" керек. Сіздің бағдарламаңыз өте ұзын жолдар жағдайында, сондай-ақ n позицияға дейін бос орын немесе табуляция символының бірде-біреуі кездеспейтініне көз жеткізіңіз.

1.23-жаттығу. Кез келген С-бағдарламадан барлық түсініктемелерді алып тастайтын бағдарламаны жазыңыз. Таңбалар мен жол тұрақтарын дұрыс өңдеуді ұмытпаңыз. С-дағы пікірлер бір-біріне салынбауы мүмкін.

1.24-жаттығу. С-бағдарламаларды тексеретін бағдарламаны қарапайым синтаксистік қателерге барлық түрдегі жақшалардың теңгерілмегендігі сияқты жазыңыз. Тырнақшалар (бір және екі), эскейп-тізбектер (\... және ескертулер. (Бұл жалпы жағдайға жазған кезде, жазылуы қиын бағдарлама.)

2-Тарау. Типтер, операторлар және өрнектер

Айнымалы және тұрақты бағдарлама бар негізгі деректер объектілері болып табылады. Айнымалылар олардың типтері және бастапқы мәндері белгіленетін хабарландыруларда атап көрсетіледі. Операциялар осы айнымалылармен жасалатын әрекеттерді анықтайды. Өрнектер жаңа мәндерді алу үшін айнымалылар мен константаларды біріктіреді, нысан түрі осы нысан қабылдай алатын мәндерді және олардың үстінен орындалатын әрекеттерді анықтайды. Аталған "кірпіштер" осы тарауда талқылау тақырыбы болады.

ANSI стандарты негізгі түрлері мен өрнектердегі шағын өзгерістер мен қосымшалардың айтарлықтай санын бекітті. Кез келген бүтін түрінің белгісі болуы мүмкін, signed және белгісі жоқ, unsigned. Белгісі жоқ константаларды және он алтылық символдық константаларды қарастырайық. Өзгермелі нүктемен операциялар енді бірдей дәлдікпен рұқсат етіледі. Жоғары дәлдікті қамтамасыз ететін ұзын double түрі енгізілді. Жол тұрақтылары компиляция кезінде ("желімделеді") бұрылады. Тілдің бөлігі мәндер ауқымын орнату түрі үшін формальды тізімдеу (enum) болды. Оларды кез келген өзгертулерден қорғау үшін нысандарды const деп белгілеуге рұқсат етіледі. Жаңа түрлердің енгізілуіне байланысты бір арифметикалық түрден екіншісіне автоматты түрлендіру ережелері кеңейтілді.

2.1. Айнымалылардың атаулары

Біз бұл туралы 1-тарауда ештеңе айтпағанымызбен, бірақ айнымалы және атаулы константалардың аттары туралы кейбір шектеулер бар. Аттар әріптер мен цифрлардан құрастырылады; бірінші символ әріп болуы тиіс. "_" символы әріп болып саналады. Астын сызу белгісі әріппен саналады; кейде айнымалылардың ұзын атауларын қабылдауды жақсарту үшін қолдануға ыңғайлы. Айнымалылардың атауларын астын сызудан бастамаңыз, өйткені көптеген кітапханалық бағдарламалардың айнымалылары дәл осы белгіден басталады. Үлкен

және кіші әріптер әртүрлі, сондықтан x және X – бұл екі түрлі атау. Әдетте С бағдарламаларында кіші әріптермен айнымалыларды, ал үлкен – атаулы константаларды тереді.

Ішкі атаулар үшін алғашқы 31 символ маңызды. Функциялардың және сыртқы айнымалылардың атаулары үшін маңызды символдардың саны 31-ден аз болуы мүмкін, өйткені бұл атаулар ассемблерлер мен жүктеушілер мен тіл бақыланбайды. Сыртқы атаулардың бірегейлігі қандай регистрде бей-жай терілген 6 таңба шегінде ғана кепілдік беріледі. If, else, int, float және т.б. кілт сөздер сақталған және оларды айнымалылардың атаулары ретінде қолдануға болмайды. Олардың барлығы төменгі регистрде (яғни шағын әріптермен) теріледі.

Олардың мақсатына сәйкес айнымалы мағыналы атауларды беру ақылға қонымды және оларды бір-бірімен шатастыру қиынға соғады. Біз жергілікті айнымалылар үшін қысқа атауларды қолдануды жөн көреміз, әсіресе цикл есептегіштер үшін, ал,сыртқы айнымалылар үшін ұзақ атауларды қолдануға болады.

2.2. Деректердің түрлері мен өлшемдері

С-да тек бірнеше база типтері бар:
char — рұқсат етілген символ жиынтығынан бір таңбасы бар жеке байт;
int — тұтас, әдетте машинадағы тұтастың табиғи көрінісін бейнелейді;
float — дара дәлдіктің өзгермелі нүктесі бар сан;
double — қос дәлдігі өзгермелі нүктелі сан.

Сондай-ақ, көрсетілген базалық үлгілермен бірге пайдалануға болатын бірнеше квалификаторлар бар. Мысалы, short (қысқа) және long (ұзын) квалификаторлар:

```
shortintsh;
longintcounter;
```

Мұндай хабарландыруларда int сөзін тастап кетуге де болады, әдетте солай болады да.

Егер ортақ мағынада қайшылықтар болмаса, shortint және longint

ұзындығы әртүрлі болуы керек, ал int осы машинада табиғи өлшемге сәйкес келуі қажет. Көбінесе short квалификаторы сипатталған бүтіндігін ұсыну үшін ұзын-32 бит, ал int — немесе 16 немесе 32 бит үлгісінің мәніне 16 бит беріледі. Компиляторды әзірлеушілер өз компьютерінің сипаттамаларына сәйкес және мынадай шектеулерді сақтай отырып, тиісті өлшемдерді өздері таңдауға құқылы: short және int түрлерінің мәндері кемінде 16 бит ұсынылады; long түрі-кемінде 32 бит; short өлшемі int өлшемінен артық емес, ал ол өз кезегінде ұзын өлшемнен артық емес.

Signed (белгісі бар) немесе unsigned (белгісі жоқ) квалификаторларын char түріне және кез келген бүтін түрге қолдануға болады. Unsigned мәндері әрқашан оң немесе нөлге тең және 2n модулі бойынша арифметика заңдарына бағынады, мұнда n – түр көрінісіндегі биттер саны. Мысалы, егер char мәніне 8 бит берілсе, онда unsignedchar 0 – ден 255-ке дейін, ал signedchar-128-ден 127-ге дейін (екілік қосымша коды бар машинада). Тек char түрінің мәндері таңбалы немесе белгісіз болып табыла ма, іске асыруға байланысты, бірақ кез келген жағдайда басып шығарылатын таңбалардың кодтары оң болады.

Longdouble түрі жоғары дәлдіктегі өзгермелі нүктелі арифметикаға арналған. Бүтін болған жағдайындай, өзгермелі нүктесі бар объектілердің өлшемдері іске асырылуына байланысты; float, double және longdouble бір өлшеммен ұсынылуы мүмкін немесе екі немесе үш түрлі өлшеммен де болуы мүмкін.

Машина мен компилятордың басқа сипаттамаларымен бірге барлық өлшемдерге арналған атаулы константалар стандартты тақырып файлдарында <limits бар.h> және <float.h> (В қосымшасын қараңыз).

2.1-жаттығу. Стандартты тақырып файлдарынан тиісті мәндерді басып шығару арқылы және тікелей есептеу арқылы signed және unsigned ретінде сипатталған char, short, int және long мәндерінің ауқымын беретін бағдарламаны жазыңыз. Әртүрлі типтегі

қалқымалы нүктелі сандар ауқымын анықтаңыз. Бұл ауқымдарды есептеу қиынырақ.

2.3. Тұрақтылар

Бүтін тұрақты, мысалы, 1234-те int түрі бар.Long типті тұрақты l немесе L әрпімен аяқталады, мысалы 123456789L; int ретінде елестету мүмкін емес тым үлкен тұтас ұзын ретінде ұсынылады. Unsigned long constantes u немесе U әрпімен аяқталады және ul немесе UL аяқталуы unsigned long тұрақты түрі туралы айтады.

Өзгермелі нүктелі константалардың ондық нүктесі (123.4) немесе экспоненциалды бөлігі (1e-2) немесе тағы басқалар бар. Егер олардың аяғы болмаса, олар double түріне жатады деп саналады. f немесе F аяқталуы float түрін, ал l немесе L — long double түрін көрсетеді.

Бүтін мән ондықтан басқа сегіз немесе он алтылық көрініске ие болуы мүмкін. Егер тұрақты нөлден басталса, онда ол сегіздік түрде, егер 0x немесе 0X болса, онда — он алтылық түрінде ұсынылған. Сегіздік және он алтылық константалардың жазбалары L (long түрін көрсету үшін) және U (егер константа таңбасыз екенін көрсету қажет болса) әрпімен аяқталуы мүмкін. Мысалы, тұрақты 0XFUL-нің 15 мәні және unsigned long түрі бар.

Символдық константа бір тырнақшамен жабылған таңба түрінде жазылған, мысалы ‘x’. Символ константасының мәні осы машинадағы символдар жиынтығынан символдың сандық коды болып табылады. Мысалы, ASCII кодтауында ‘0’ символдық тұрақты 48 мәні бар, ол 0 сандық мәніне ешқандай қатысы жоқ. Кодтау әдісіне байланысты қандай да бір мәнді емес (мысалы , 48) жазғанда, біз бағдарламаны кодтың жеке мәнінен тәуелсіз жасаймыз. Сонымен қатар ол оңай оқылады. Символдық константалар басқа таңбалармен салыстыру үшін жиі қолданылатын болса да, кез келген басқа бүтін сияқты сандармен операцияларға қатыса алады.

Символдық және жолдық константалардағы кейбір таңбалар эскейп дәйектеу көмегімен жазылады, мысалы \n (жаңа жолдың символы); мұндай тізбектер екі таңбамен бейнеленеді, бірақ біреуін білдіреді. Сонымен қатар, еркін сегіздік кодты түрінде орнатуға болады

‘\ooo’
онда ooo — бір, екі немесе үш сегіздік сандар (0…7) немесе

‘\xhh’

мұнда hh — бір, екі немесе одан көп он алтылық сан (0…9, a…f, A…F). Осылайша біз былай жаза аламыз:

```
#define VTAB ‘\013’ /* ASCII тік табуляция */
#define BELL ‘\007’ /* ASCII-ге қоңырау шалу */
```

немесе он алтылық түрінде:

```
#define VTAB ‘\xb’ /* ASCII тік табуляция */
#define BELL ‘\x7’ /* ASCII-ге қоңырау шалу */
```

```
\a сигнал-қоңырау \\ кері көлбеу сызық
\b бір қадамға қайтару (забой)\? сұрақ белгісі
\f аударма-беттер \’ жалғыз тырнақша
\n жаңа-жол \” қос тырнақша
\r қайтару-каретка лар \ooo сегіздік код
\t көлденең-табуляция \xhh он алтылық код
\v тік-табуляция
```

Символдық тұрақты ‘\0 ‘ - null символы деп аталатын нөлдік мәндегі символ. Тек 0 орнына өрнектің таңбалық сипатын белгілеу үшін ‘\0’ жазбасын жиі пайдаланады, бірақ сол және басқа жағдайда да жазба нөлді білдіреді.

Тұрақты өрнектер – бұл тек тұрақтармен жұмыс істейтін өрнектер. Мұндай өрнектер орындау кезінде емес, компиляция кезінде есептеледі, сондықтан оларды кез келген жерде қолдануға болады, мысалы,

```
#define MAXLINE 1000
char line[MAXLINE+1];
```

немесе

```
#define LEAP 1 /* in leap years - аспалы жылдарда */
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

Жол тұрақтысы немесе жол литералы – бұл екі тырнақшаға салынған нөл немесе одан да көп таңбалар, мысалы,

```
«Мен жол тұрақтысымын»
```

Немесе

```
«» /* бос жол */
```

Тырнақшалар жолға кірмейді, тек оның шектегіші болып табылады. Символдік константалар сияқты, жолдарға эскейп-бірізділікті қосуға болады;\», мысалы, қос тырнақшаны білдіреді. Жол тұрақтыларын құрастыру кезінде («желімдеу») реттеуге болады; мысалы, екі жолды жазу.

```
«Сәлем,» « әлем!»
```

келесі бір жол жазбасына тең:

```
«Сәлем, әлем!»
```

Көрсетілген сипат ұзын жолдарды бөліктерге бөлуге және осы бөліктерді жеке жолақтарға орналастыруға мүмкіндік береді.

Шын мәнінде, жол тұрақтысы – символдар массиві. Жолдың ішкі көрінісінде соңында «\0 « нөлдік белгісі бар, сондықтан жол үшін жад қос тырнақшалар арасында орналасқан таңбалар санынан бір байтқа көп талап етіледі. Бұл жолдың ұзындығына шектеу жоқ, бірақ оның ұзындығын анықтау үшін бүкіл жолды көру қажет. Strlen(s) функциясы s жолының ұзындығын аяқтаушы ‘\0’таңбасын ескермей есептейді. Төменде біздің нұсқа бойынша жазылған функция:

```
/* strlen: жолдың ұзындығын қайтарады s */
int strlen(char s[])
{
int i;
 i = 0;
while (s[i] != '\0')
++i;
  return i;
}
```

strlen функциясы және жолдарға қолданылатын кейбір басқалар стандартты <string.h> тақырып файлында сипатталған <string.h>.

Символдық константа және жол бір символды қамтығанымен, олар бірдей емес екендігі есіңізде болсын. 'x' символы «x» символы сияқты емес. 'X' жазбасы стандартты символдық жиынтықтан x әріптерінің кодына тең бүтін мәнді білдіреді, ал «x» жазбасы – бір символ (X әрпі) және '\0' символдар массиві.

С-да санақ константасының тағы бір түрі бар. Тізімдеу – бұл тұтас тұрақты тізім, мысалы,

```
enum boolean { NO, YES };
```

enum5 – дегі бірінші атау 0, келесі-1 және т.б. (егер константалардың мәндері үшін айқын ерекшеліктер болмаса). Егер барлық мәндер арнайы болмаса, онда олар келесі екі мысалдағы сияқты соңғы ерекше мәннен бастап прогресті жалғастырады:

```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t',
               NEWLINE = '\n', VTAB = '\v', RETURN = '\r' };
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
              JUL, AUG, SEP, OCT, NOV, DEC };
/* FEB- да 2, MAR-да 3 және т.б */
```

Әр түрлі тізімдердегі атаулар бір-бірінен ерекшеленуі керек. Бір тізімнің ішіндегі мәндер сәйкес келуі мүмкін.

enum құралы тұрақтыларға аттарды берудің ыңғайлы жолын қамтамасыз етеді, ал #define мәнінен айырмашылығы тұрақты бұл әдіс автоматты түрде жасалуы мүмкін. Enum типті айнымалыларды

жариялауға рұқсат етілсе де, компилятор осы айнымалы мәндердің олардың түріне кіретінін бақылауға міндетті емес. Бірақ мұндай тексерудің өзі жиі enum-ді #define-ге қарағанда жақсы етеді. Сонымен қатар, жөндеуші символ түрінде enum типті айнымалы мәндерді басып шығаруға мүмкіндік алады.

2.4. Жариялау

Барлық айнымалылар пайдаланылғаннан бұрын жариялануы керек, ал кейбіреулері анық емес – контекстен алынуы мүмкін. Жариялау түрін сипаттайды және осы түрдегі бір немесе бірнеше айнымалылардың тізімін қамтиды, мысалы:

```
int lower, upper, step;
char c, line [1000];
```

Айнымалыларды хабарландырулар бойынша еркін түрде таратуға болады, сондықтан жоғарыда көрсетілген тізімдерді келесі түрде жазуға болады:

```
int lower;
int upper;
int step;
char c;
char line[1000];
```

Жазбаның соңғы нысаны көп орын алады, дегенмен, оның да жақсы тұстары бар, себебі ол әрбір хабарламаға пікір қосуға мүмкіндік береді. Сонымен қатар, ол келесі модификациялар үшін ыңғайлы.

Оның хабарландыруында айнымалы инициалдалуы мүмкін:

```
char esc = '\\';
int i = 0;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

Автоматты емес айнымалыны инициализациялау тек бір рет жүзеге асырылады, – бағдарлама орындала бастамас бұрын, бастапқы мән тұрақты өрнек болуы керек. Нақты инициалданған автоматты айнымалы функцияға немесе блокқа кіргенде әр жолы бастапқы мәнді алады, оның бастапқы мәні кез келген өрнек болуы мүмкін. Сыртқы және статикалық айнымалылар нөлдік мәндерді алады. Автоматты айнымалы, анық инициалдалмаған, белгісіз мәндерді («қоқыс») қамтиды.

Хабарландыруда кез келген айнымалыға const квалификаторы оның мәні бұдан әрі өзгермейтіндігін көрсету үшін қолданылуы мүмкін.

```
const double e = 2.71828182845905;
const char msg[] = «ескерту: «;
```

Массивке қатысты const квалификаторы оның элементтерінің ешқайсысы өзгермейтіндігін көрсетеді. Const нұсқауын сондай-ақ функция бұл массивті өзгертпейтінін жариялау үшін аргумент-массивке қолдануға болады:

```
int strlen(const char[] );
```

Const квалификаторымен белгіленген айнымалыны өзгертуге әрекет ету әрекеті компилятордың жүзеге асырылуына байланысты.

2.5. Арифметикалық операторлар

Бинарлық (яғни екі операндпен) арифметикалық операторлар +, -, *, /, сондай-ақ % модулі бойынша бөлу операторы болып табылады. Бүтін бөлікті бөлу бөлшек бөлігін қандай болса да лақтырып тастаумен бірге жүреді. Өрнек х-ны у-ге бөлуден қалған, демек, х-ға бөлінсе, нөл береді. Мысалы, жыл 4-ке бөлінген болса, бірақ 100-ге бөлінбейді. Сонымен қатар, егер ол 400-ге бөлінген болса, бір жыл. Демек,

```
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)
```

```
    printf(«%d аспалы жыл\n», year);
else
    printf(«%d аспалы емес жыл\n», year);
```

% операторы float және double типті операндаларға қолданылмайды. Орындау кезінде бөлшек бөлік қай жаққа (санның ұлғаюы немесе азаюы жағына) кесіледі / және теріс операндалармен % операция нәтижесінің белгісі қандай болатындығы машинаға байланысты.

Екі жақты операторлар + және - операторлардың басымдығынан төмен *, / және % қарағанда төмен, ол өз кезегінде унарлы операторлардың басымдығынан төмен + и -. Бір басым деңгейдің арифметикалық операциялары солдан оңға қарай орындалады.

Осы тараудың соңында (2.12-параграф) 2.1-кесте келтіріледі, онда барлық операторлардың басымдықтары және оларды орындау кезектілігі берілген.

2.6. Қатынас операторлары және логикалық операторлар

Қатынастар операторлары:

```
>>= <<=
```

Олардың барлығы бірдей басымдыққа ие. Олардың артында теңдікке салыстыру операторларының басымдығы бар:

```
== !=
```

Қатынас операторлары арифметикалыққа қарағанда төмен басымдыққа ие, сондықтан I < lim-1 сияқты өрнек i < (lim-1) сияқты орындалады, яғни біздің ойлағанымыздай.

Логикалық операторлар & және / / қызықтырақ. && мен || операторлары арасындағы өрнектер солдан оңға қарай есептеледі. Есептеу нәтиженің шынайылығы немесе жалған екендігі белгілі болған кезде тоқтатылады. Көптеген С-бағдарламалар осы сипатқа сүйенеді, мысалы, getline функциясындағы цикл, біз ол туралы 1-тарауда талқылағанбыз:

```
for (i = 0; i < lim-1 && (c = getchar()) != EOF && c != '\n'; ++i)
    s[i] = c;
```

Келесі символды оқып алмас бұрын, s массивінде орын баржоғын тексеру керек, басқаша айтқанда, алдымен I < lim-1 шартының сақталуын тексеру керек. Егер бұл шарт орындалмаса, біз келесі таңбаны оқып, есептеуді жалғастырмауымыз керек. Сонымен қатар, getchar-ға жүгінгенге дейін EOF-мен салыстыру дұрыс емес болар еді; сондықтан getchar шақыруы және тағайындау көрсетілген тексеру алдында орындалуы керек.

&& операторлардың басымдылығы || операторға қарағанда жоғары, алайда олардың басымдылығы қатынас және теңдік операторларының басымдығынан төмен. Бұл түрдің көрінісі мынадай:

```
i < lim-1 && (c = getchar()) != '\n' && c != EOF
```

қосымша жақшаларды қажет етпейді. Бірақ, != басымдылығы жоғары болғандықтан,

```
(c = getchar()) ! = '\n'
```

'\n ' салыстыру үшін жақшалар міндетті түрде болуы қажет.

Анықтама бойынша қатынас немесе логикалық өрнектің сандық нәтижесі 1, егер ол шын болса, ал егер ол жалған болса 0.

Унарылы оператор ! нөл емес операндты 0-ге, ал нөлді 1-ге түрлендіреді. Әдетте оператор ! түр конструкцияларында қолданылады.

```
if (!valid)
```

оның баламасы:

```
if (valid == 0)
```

Жазу формаларының қайсысы жақсы екенін айту қиын. Көрініс дизайны !valid жақсы оқылады («егер дұрыс болмаса»), бірақ күрделі өрнектерде оны түсіну оңай емес.

2.2-жаттығу. && и || операторларды пайдаланбай, жоғарыда келтірілген for-циклға баламалы циклды жазыңыз.

2.7. Типтерді түрлендіру

Егер оператордың операндалары әртүрлі түрлерге тиесілі болса, онда олар кейбір жалпы түрге келтіріледі. Келтіру ереженің аз санына сәйкес орындалады. Әдетте, қандай да бір ақпаратты жоғалтпай, операндтарды мәндер диапазоны аз операндтарға үлкен диапазонды айналдыратын түрлендірулер ғана автоматты түрде жасалады, мысалы, F+I сияқты өрнектегі өзгермелі нүктелі санға бүтін санды түрлендіруге болады. Ақпарат жоғалуы мүмкін өрнектер (айталық, ұзын бүтін айнымалылардың қысқа түрлерін тағайындағанда немесе өзгермелі нүктелі бүтін айнымалы мәндерді тағайындағанда) ескерту тудыруы мүмкін, бірақ олар рұқсат етіледі.

Char түрінің мәндері – бұл жай ғана шағын бүтін және оларды арифметикалық өрнектерде еркін пайдалануға болады, бұл таңбалармен манипуляцияларды жеңілдетеді. Мысал ретінде, сандар тізбегін оның сандық эквивалентіне түрлендіретін atoi функциясын қарапайым іске асыруды келтіреміз.

```
/* atoi: s-ны бүтінге түрлендіру */
int atoi(char s[])
{
 int i, n;
n = 0;
for (i = 0; s[i] >= ‘0’ && s[i] <= ‘9’; ++i)
n = 10 * n + (s[i] - ‘0’);
return n;
}

1-тарауда айтылғандай, өрнек
s[i] - ‘0’
```

s[i] сақталған символдың сандық мәнін береді, өйткені ‘0’, ‘1’ және т.б. мәндері үздіксіз өсіп отыратын тізбекті құрайды.

Char-ны int-ге келтірудің тағы бір мысалы, ASCII жиынтығынан бір таңба, егер ол бас әріппен болса, кіші әріптерге айналады. Егер таңба бас әріппен болмаса, lower оны өзгертпейді.

```
/* lower: тек ASCII үшін кіші әріптен түрлендіру */
int lower(int c)
{
    if (c >= ‘A’ && c <= ‘Z’ )
        return c + ‘a’ - ‘A’;
    else
        return c;
}
```

ASCII жағдайында бұл бағдарлама дұрыс жұмыс істейді, себебі жоғарғы және төменгі регистрлердің аттас әріптері арасында бірдей қашықтық (егер оларды сандық мәндер ретінде қарастырса). Сонымен қатар, латын әліпбиі-тығыз, яғни A және Z әріптерінің арасында тек әріптер орналасқан. EBCDIC жиынтығы үшін соңғы шарт орындалмайды, сондықтан біздің бағдарлама бұл жағдайда әріптерді ғана түрлендірмейді.

Стандартты тақырып файлы <ctype. B қосымшасында сипатталған h> таңбаларды символ жиынтығына қарамастан тексеруге және түрлендіруге мүмкіндік беретін функциялар тобын анықтайды. Мысалы, tolower(c) функциясы C әрпін төменгі регистр кодында қайтарады, егер ол жоғарғы регистр кодында болса, сондықтан tolower-жоғарыда қарастырылған lower функциясын әмбебап ауыстыру. Тексеру:

```
c >= ‘0’ && c <= ‘9’
```

оны мынаған ауыстыруға да болады:

```
isdigit(c)
```

Содан кейін біз <ctype.h>. функцияларын пайдаланамыз.

Таңбаларды бүтін санға түрлендіруге қатысты бір тәсіл бар: тіл char типті айнымалылардың таңбалы немесе белгісіз екендігін

анықтамайды. Char int-ге түрлендіру кезінде теріс бүтін болуы мүмкін бе? Әр түрлі архитектурасы бар машиналарда жауаптар әр түрлі болуы мүмкін. Кейбір машиналарда жалғыз үлкен бит бар char типті мәні теріс тұтастыққа айналады («белгіні тарату» арқылы). Басқа char int-ға түрлендіру сол жақтағы нөлдерді қосу арқылы жүзеге асырылады және осылайша алынған мән әрдайым оң болады.

Басып шығарылатын таңбалардың стандартты жиынтығының кез келген белгісі ешқашан теріс сан болмайтынына кепілдік беріледі, сондықтан өрнектерде мұндай таңбалар әрдайым оң операндалар болып табылады. Бір машиналарда char айнымалы типті еріксіз сегізбитті код теріс сан, ал екіншісінде оң болуы мүмкін. Аналогты емес деректер сақталатын char типті айнымалылардың үйлесімдігі үшін сигналды немесе unsigned деп анықталуы керек.

I > j өрнектеріндегі қарым-қатынас сияқты && және || операторлармен өзгеретін логикалық өрнектерді анықтайды-егер ол шын болса, 0, егер жалған болса, 1 мәні бар болуы шарт. Осылайша,

d = c >= ‘0’ && c <= ‘9’

егер c саны болса, d-ді 1 мәніне орнатады және 0 әйтпесе. Алайда, isdigit сияқты функциялар шындық ретінде кез келген нөл емес мән бере алады. If, while, for және т.б. ішіндегі тексеру орындарында «ақиқат» жай ғана «нөл емес» дегенді білдіреді.

Белгісіз арифметикалық түрлендірулер, әдетте, табиғи түрде жүзеге асырылады. Жалпы жағдайда, операция орындалмас бұрын, екі операндасы бар (бинарлы оператор) + немесе * тәрізді оператордың әртүрлі операндалары болса, «төменгі» түрі «жоғарыға» дейін көтеріледі. Нәтиже жоғары түрге ие болады. А қосымшасының 6 параграфында түрлендіру ережелері дәл тұжырымдалған. Егер көріністе операндтар болмаса, бейресми ережелердің келесі жиынтығымен қанағаттандыруға болады:

▪ Егер операндтардың біреуі long double түріне тиесілі болса, екіншісі де long double-ге келеді.

▪ Олай болмаған жағдайда, операндтардың кез келгені double түріне тиесілі болса, екіншісі де double-ге келтіріледі.

▪ Егер операндтардың кез келгені float түріне жататын болса, онда екіншісі float-ке келтіріледі.

▪ Олай болмаған жағдайда char және short типтерінің операндтары int-ға келтіріледі.

▪ Ақыр соңында, егер операндтардың бірі long болса, екіншісі де long болады.

Айта кетейік, float типті операндалар double түріне автоматты түрде келтірілмейді; бұл тіл нұсқасы бастапқы нұсқадан ерекшеленеді. Жалпы айтқанда, кітапханада жинақталған математикалық функциялар <math.h>, қос дәлдікпен есептеулерге негізделген. Негізінен float үлкен массивтерде жадты үнемдеу үшін пайдаланылады және жиі емес — уақыт пен жад шығыны тұрғысынан екі есе дәлдікпен арифметика тым қымбат болатын машиналарда шотты жеделдету үшін.

Түрлендіру ережелері unsigned типті операндтардың пайда болуымен күрделенеді. Мәселе мынада, таңбалы және белгісіз мәндерді салыстыру әртүрлі машиналарда өзгеше болуы мүмкін бүтін түрлердің өлшеміне байланысты. Мысалы, int түрінің мәні 16 бит алады, ал long – 32 бит түрінің мәні. Содан кейін – 1L < 1U, себебі 1U unsigned int түріне тиесілі және signed long түріне дейін көтеріледі. Бірақ, -1L > 1UL, себебі-1L unsigned long түріне дейін көтеріледі және үлкен оң сан ретінде қабылданады.

Өзгерістер беру кезінде де орын алады: иеленудің оң бөлігінің мәні сол жақ бөліктің түріне келтіріледі, ол нәтиженің түрі болып табылады.

Char түрі белгіні тарату арқылы немесе жоғарыда сипатталған басқа тәсілмен int-ға айналады.

Long int түрі қысқа int-ға немесе Char типті мәндерге үлкен разрядтарды лақтыру арқылы түрлендіріледі. Мәселен,

```
int i;
char c;
i = c;
c = i;
```

с мәні өзгермейді. Бұл char-ды int-ға немесе жоқ-қа ауыстырған кезде белгі таратылғанына қарамастан әділ. Алайда, егер тағайындау кезектілігін өзгертсе, ақпаратты жоғалтуы мүмкін.

Егер x float, а i — int түріне жататын болса, онда x = I және I = X түрлендірулерді шақырады, ал float int-ға аудармасы бөлшек бөлікті лақтырумен бірге жүреді. Егер double float-қа аударылса, онда мән дөңгелектенеді немесе қиылады; бұл іске асырылуға байланысты.

Функцияны шақырудағы дәлел өрнек болғандықтан, оның функциясын беру кезінде түрдің түрленуі де мүмкін. Прототипі болмаған кезде char және short түріндегі аргументтер int, а float — double-ға аударылады. Сондықтан біз char немесе float түріндегі дәлелдерді пайдаланғанда да int немесе double түріндегі дәлелдерді жарияладық.

Ақыр соңында, кез келген өрнек үшін нақты («күшпен») келтірудің деп аталатын унарлы операторды қолдана отырып, оның түрін түрлендіруді көрсетуге болады. Көрініс құрылымы:

```
(аты-типі) өрнек
```

жоғарыда аталған ережелер бойынша жақшада көрсетілген түрге өрнек келтіреді. Келтіру операциясының мағынасын елестетуге болады: өрнек көрсетілген түрдің кейбір айнымалысына қалай тағайындалады және бұл айнымалы бүкіл конструкцияның орнына қолданылады. Мысалы, sqrt кітапхана функциясы double түріндегі аргументке есептелген және егер ол басқа нәрсе болса (sqrt <math сипатталған.h>). Сондықтан, егер n бүтін түрі болса, біз былай жаза аламыз:

```
sqrt((double) n)
```

N мәні функция берілгенге дейін, double-ге аударылады. Айта кету керек, келтіру операциясы көрсетілген түрдегі n мәнін шығарады, бірақ айнымалы n өзіне әсер етпейді. Тастау операторының басымдығы кез келген нотариаттық операторға қарағанда жоғары, оны осы тараудың соңындағы кестеден көре аласыз.

Аргументтер функция прототипінде сипатталған жағдайда, функцияны шақырғанда, қажетті түрлендіру автоматты түрде орындалады. Осылайша, sqrt функциясының прототипі болған кезде:

```
double sqrt(double);

sqrt-ге тағайындалғанға дейін

root2 = sqrt(2);
```

бүтін 2-ге келтіру операциясын анық көрсетпей, double 2.0 мәніне автоматты түрде аударылады.

Жасанды сандарды генератордың ауыспалы нұсқасына және «тұқым» бастамашы функциясына келтіру операциясын көрсетеміз. Генератор және функция стандартты кітапханаға кіреді.

```
unsigned long int next = 1;

/* rand: жалған кездейсоқ бүтін санды қайтарады 0...32767 */
int rand(void)
{
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}

/* srand: «семя» – ны rand үшін орнатады() */
void srand(unsigned int seed)
{
     next = seed;
}
```

2.3-жаттығу. 0x немесе 0X-тен басталатын он алтылық сандар тізбегіне тиісті бүтін санға түрлендіретін htol (s) функциясын жазыңыз. Он алтылық сандар - 0 ... 9, a ... f, A ... F таңбалары.

2.8. Инкремент және декемент операторлары

С-да айнымалыларды көбейту және азайтуға арналған екі ерекше оператор бар. Инкремент ++ операторы өзінің операндығына 1 қосады, ал декемент операторы 1 шегереді. Біз бірнеше рет айнымалылардың мәнін кеңейту үшін ++ пайдаландық, мысалы,

```
if (c == '\n')
    ++nl;
```

++ және -- операторларының ерекшелігі, оларды префиксті (айнымалыдан бұрын орналастыра отырып) және постфиксті (айнымалыдан кейін орналастыра отырып) операторлар ретінде пайдалануға болады. Екі жағдайда да n мәні 1 — ге артады, бірақ ++n өрнегі n мәнін қолданғанға дейін арттырады, ал n++ одан кейін. n-де 5 бар делік, сонда:

```
x = n++;
```

x мәні 5, ал

```
x = ++n;
```

x мәнін 6 мәніне орнатады. Сол және басқа жағдайда n 6-ға тең болады. Инкремент және декемент операторлары тек айнымалыларға қолдануға болады. (I+j)++ сияқты өрнектер жарамсыз.

Егер айнымалының мәнін ұлғайту немесе азайту қажет болса (бірақ оның мәнін алмаңыз), мысалы мынадай болса,

```
if (c == '\n')
    nl++;
```

ол жағдайда префиксті немесе постфиксті операторлардың қайсысын таңдағанымыз маңызды емес. Бірақ белгілі бір түрдегі оператор талап етілетін жағдайлар бар. Мысалы, s жолынан c сәйкес келетін барлық таңбаларды жоятын squeeze(s, c) функциясын қарастырайық:

```
/* squeeze: s-дан барлығын өшіреді*/
void squeeze(char s[], int c)
{
      int i, j;
      for (i = j = 0; s[i] != '\0'; i++)
          if (s[i] != c)
              s[j++] = s[i];
      s[i] = '\0';
}
```

С-дан өзгеше символ болған сайын, ол ағымдағы j-ші позицияға көшіріліп, содан кейін ғана j айнымалысы 1-ге артады, осылайша келесі символды қабылдауға дайындалады. Бұл келесі әрекеттермен сәйкес келеді:

```
if (s[i] != c) {
s[j] = s[i];
j++; }
```

Тағы бір мысал 1-тараудан бізге белгілі getline функциясы. Онда келтірілген жазба:

```
if (c == '\n' ) {
    s[i] = c;
    ++i;
}
```

Ықшамдап та жаза аласыз:

```
if (c == '\n' )
    s[i++] = c;
```

Үшінші мысал ретінде стандартты strcat(s, t) функциясын қарастырайық , ол мына t жолын s жолының соңына орналастырады. Біз strcat-ты ешқандай нәтиже бермейтіндей етіп жаздық. Шын мәнінде, кітапханалық strcat көрсеткішті нәтиже жолына қайтарады.

```
/* strcat: t жолын s жолының соңына орналастырады */
void strcat (char s[], char t[])
{
```

```
int i, j;
i = j = 0;
while (s[i] != '\0') /* s-ның аяғын табамыз */
 i++;
while ((s[i++] = t[j++]) != '\0') /* t-көшіреміз */
;
}
```

Келесі символды t-ден s-ге көшіру кезінде постфиксті оператор ++ I-ге де, j-ге де қолданылады.

2.4-жаттығу. s2 жолында кездесетін барлық таңбаларды s1-ден алып тастайтын squeeze(s1, s2) функциясының нұсқасын жазыңыз.

2.5-жаттығу. Any(s1, s2) функциясын жазыңыз, ол s2 немесе -1 символдарының кез келгенімен сәйкес келген бірінші символ тұрған s1-дегі позицияны қайтарады (егер s1-ден бір символ s2-ден бір символ сәйкес келмесе). (Strpbrk стандартты кітапханалық функциясы бірдей, бірақ таңбаның позициясының нөмірін емес, таңбаның көрсеткішін береді).

2.9. Бинарлық операторлар

С-да биттерді манипуляциялау үшін алты оператор бар. Оларды тек тұтас операндаларға, яғни char, short, int және long типтеріне, таңбалы және белгісіз операндаларға қолдануға болады.

```
& – бинарлық ЖӘНЕ.
| – бинарлық НЕМЕСЕ.
^ – бинарлық НЕМЕСЕ алып тасталынған.
<< –  солға жылжыту.
>> – оңға жылжыту.
~ – бинарлық жоққа шығару (унарлық).
```

Оператор & (бинарлық ЖӘНЕ) жиі кейбір разряд тобын нөлге келтіру үшін қолданылады. Мысалы:

```
n = n & 0177;
```

кіші жетіден басқа барлық разрядтарды n-де нөлдейді.

Оператор | (бинарлық НЕМЕСЕ) разрядтарды орнату үшін қолданылады;

```
x = x | SET_ON;
```

SET_ON бірліктеріне сәйкес келетін х разрядтарында бірліктерді орнатады.

Оператор ^ (бинарлық жоққа шығару) әрбір разрядта, егер операндтардың тиісті разрядтары әр түрлі мағынаға ие болса, және олар сәйкес болғанда 0-ді белгілейді.

& и | қисынды операторларды солдан оңға қарай есептеу кезінде ақиқатқа мән беретін && және || логикалық операторлардан ажырата білу керек. Мысалы, егер х 1 болса, у 2 болса, онда х & у нөлді, ал х && у нөлді береді.

<< және > операторлары сол жақ теріс емес болуы тиіс. Операндты оң жақ операндпен берілген биттік позициялар санына солға немесе оңға жылжытады. Осылайша, х << 2 босаған биттерді нөлдермен толтыра отырып, х мәнін 2 позицияға солға жылжытады, бұл х-ға 4-ке көбейту эквивалентті. Сырғымасыз шаманың оңға жылжуы әрқашан босаған разрядтарды нөлдермен толтырумен жүреді. Бір машиналарда белгілік шаманың оңға жылжуы («арифметикалық ығысу») белгінің таралуымен, екіншісінде – босаған разрядтарды нөлдермен толтырумен («логикалық ығысу») жүргізіледі.

Унарлы оператор ~ тұтас «айналдырады», яғни әрбір бір битті нөлдік және керісінше айналдырады. Мысалы:

```
x = x & ~077
```

х соңғы 6 разрядты нөлдейді. X & ~077 жазбасы сөздің ұзындығына байланысты емес, сондықтан ол х & 0177700 қарағанда жақсы, өйткені соңғы х 16 бит алады. Машинаға тәуелді емес жазу

нысаны ~077 шот кезінде қосымша шығындарды талап етпейді, өйткені ~077 — компиляция кезінде есептелетін тұрақты өрнек.

Кейбір бинарлы операциялары суреттеу үшін getbits (x, p, n функциясын қарастырамыз, ол оң шетіне басу арқылы p позициясынан бастап x қиып алынған N биттер өрісін қалыптастырады. 0 бит-шеткі оң бит, ал n және p – мағыналы оң сандар. Мысалы, getbits (x, 4, 3) нәтиже ретінде 4, 3 және 2 бит x мәнін қайтарады, оларды оң шетіне қысады. Міне, бұл функция:

```
/* getbits: p позициясынан бастап, n битін алады */
unsigned getbits(unsigned x, int p, int n)
{
    return (x >> (p+1-n)) & ~(~0 << n);
}
```

Өрнек x >> (p+1-n) бізге қажетті өрісті оң жаққа жылжытады. Тұрақты ~0 бір бірліктен тұрады және оның солға қарай n битке (~0 < n) жылжуы осы тұрақты оң жақ шеті n нөлдік разрядтарды алады. Тағы бір бинарлы инверсия операциясы ~ оң жақта n бірлік алуға мүмкіндік береді.

2.6-жаттығу. setbits (x, p, n, y) функциясын жазыңыз, онда n биттер p-ші позициядан бастап y оң разрядтарының n-ге ауыстырылуы керек (қалған биттер өзгермейді).

2.7-жаттығу. p позициясынан бастап инверттелген n биттермен x мәнін қайтаратын invert (x, p, n) функциясын жазыңыз (қалған биттер өзгермейді).

2.8-жаттығу. x-ті n разрядқа оңға жылжытатын rightrot (x, n) функциясын жазыңыз.

2.10. Меншікті өрнектер мен операторлар

Өрнек:

```
i = i + 2
```

сол жақта тұрған айнымалысы оң жақта да қайталанады. Оның қысқартылған түрін былай жазуға болады:

```
i += 2
```

+= операторы меншікті өрнек деп аталады.

Көптеген бинарлық операторлар (ұқсас + және сол және оң операндалары бар) op-операторлардың бірі болатын, меншікті op= операторына сәйкес келеді.

```
+ - * / % << >> & ^ |
```

Егер өрнек $_1$ және өрнек$_2$ өрнектер болса, онда:

өрнек $_1$ op= өрнек$_2$

эквалиентті

өрнек$_1$ = (өрнек$_1$) op (өрнек $_2$)

Бір ғана айырмашылық бар - өрнек$_1$ тек бір рет есептеледі. Өрнек$_2$ айналасында жақшаларға назар аударыңыз:

```
x *= y + 1
```

эквалиентті

```
x = x * (y + 1)
```

мынаған емес:

```
x = x * y + 1
```

Мысал ретінде, біз bitcount функциясын, бүтін түрдегі аргументте бірлік биттердің санын есептейміз.

```
/* bitcount: x-та бірліктерді санау */
```

```
int bitcount (unsigned x)
{
     int b;

    for (b = 0; x != 0; x >>= 1)
         if (x & 01)
              b++;
    return b;
}
```

Бұл бағдарлама жұмыс істейтін машинаға қарамастан, x аргументін unsigned деп жариялап, еркін жылжу кезінде босайтын биттер таңбалы битпен емес, нөлдермен толтырылатынына кепілдік береді.

Қысқа мерзімділіктен басқа, тағайындау операторлары адамның ойлауына сәйкес келетін артықшылықтарға ие. Біз «2-ге і-ні қосу» немесе «і-ге 2-ге ұлғайту» деп айтамыз, «і-ге 2-ні қосып, і нәтижесін қайтару» емес, сондықтан i += 2 өрнегі i = i + 2 өрнегіне қарағанда жақсырақ. Сонымен қатар, кейбір күрделі өрнектерде:

```
yyval[yypv[p3+p4] + yypv[p1]] += 2
```

меншіктеу операторының арқасында += жазу түсіну үшін оңай болады, өйткені мұндай жазба кезінде оқырманға екі ұзын өрнектерді мұқият салыстыру қажет емес, олар сәйкес келетіндігін немесе келмейтіндігіын анықтамайды. Мұндай меншіктеу операторлары компиляторға тиімді кодты құруға көмектесетінін есте ұстаған жөн.

Біз меншіктеу мәні бар екенін және өрнектің ішінде қолданылуы мүмкін екенін көрдік; мысал:

```
while ((c = getchar()) != EOF)
        ...
```

Өрнектерде басқа меншіктеу операторлары да кездеседі (+ = , - = және т.б.).

Меншіктеу аяқталғаннан кейін оның сол жақ операндының

түрі мен мәні кез келген меншіктеу өрнегінің түрі мен мәні болып табылады.

2.9-жаттығу. Қосымша код қолданылған сандарға қатысты, x &= (x-1) өрнегі x-дағы ең оң 1-ді жояды. Bitcount функциясының жылдам нұсқасын жазу кезінде осы бақылауды пайдаланыңыз.

2.11. Шартты өрнектер

Нұсқаулық

```
if (a > b)
    z = a;
else
    z = b;
```

z -ға екі мәнді a және b көбірек жібереді. Шартты өрнек тернарлық (яғни үш операндасы бар) оператордың көмегімен жазылған ?:'', бұл және оған ұқсас конструкцияларды жазудың басқа жолын ұсынады. Өрнекте:

$$өрнек_1 ? өрнек_2 : өрнек_3$$

бірінші $өрнек_1$ өрнегі есептеледі. Егер оның мәні нөл (ақиқат) болмаса, онда $өрнек_2$ өрнегі есептеледі және бұл өрнектің мәні барлық шартты өрнектің мәні болады. Олай болмаған жағдайда, $өрнек_3$ өрнегі есептеледі және оның мәні шартты өрнектің мәні болады. Айта кету керек, $өрнек_2$ және $өрнек_3$ өрнектерінен біреуі ғана есептеледі. Осылайша, z-ға a және b-дан көп орнату үшін жазуға болады

```
z = (a > b) ? a : b;      /* z = max(a, b) */
```

Шартты өрнек шын мәнінде өрнек екенін және оны өрнек рұқсат етілетін кез келген жерде қолдануға болатынын атап өткен жөн. Егер $өрнек_2$ және $өрнек_3$ әртүрлі түрлерге тиесілі болса, онда нәтиже түрі бұрын осы тарауда айтылған қайта құру ережелерімен анықталады. Мысалы, егер f float түрі болса, a n – int түрі болса,

онда өрнектің түрі:

```
(n > 0) ? f : n
```

n мәнінің оң немесе оң емес екеніне қарамастан, float болады.

Жақшаға шартты түрде бірінші сөйлемді жасау міндетті емес, себебі ?: басымдылығы өте төмен (төмен басымдыққа тең меншіктеу ие), бірақ біз әрқашан осыны істеуді ұсынамыз, себебі, жақшаларының арқасында өрнектегі шарт жақсы қабылданады.

Шартты өрнек жиі бағдарламаны қысқартуға мүмкіндік береді. Мысал ретінде әрбір жолда 10-нан жиым элементтерінің n-ді басып шығаруды қамтамасыз ететін цикл келтіреміз; циклдің әрбір жолы, соңғысын қоса алғанда, жаңа жолдың символымен аяқталады:

```
for (i = 0; i < n; i++)
    printf("%6d%c", a[i], (i%10==9 || i==n-1) ? '\n' : ' ');
```

Жаңа жолдың белгісі әрбір оныншы элементтен кейін және n-ші элементтен кейін жіберіледі. Барлық басқа элементтердің артында бос орын бар. Бұл бағдарлама өте күрделі көрінеді, бірақ ол if-else-ді қолданатын эквивалентті бағдарламаға қарағанда ықшамырақ. Міне, тағы бір жақсы мысал:

```
printf("Сізде %d элементі бар%s.\n", n, (n%10==1 && n%100 ! = 11) ?
      " " : ((n%100 < 10 || n%100 > 20) && n%10 >= 2 && n%10 <= 4) ?
      "а" : "ов");
```

2.10-жаттығу. Бас әріптерді кішіге ауыстыратын lower функциясын шартты өрнек (if-else конструкциясы емес) арқылы жазыңыз.

2.12. Есептеудің басымдығы мен кезектілігі

2.1-кестеде барлық операторлардың есептеуінің басымдықтары мен кезектілігі көрсетілген, соның ішінде біз әлі де қарастырмағандар да бар. Бір жолда аталған операторлар бірдей басымдыққа ие; жолдар

басымдықтардың кемуі бойынша реттелген; мысалы, *, / және % басымдылығы бірдей және бинарлы + мен - басымдылығынан жоғары. “Оператор” () функцияны шақыруға жатады. Операторлар -> и . (нүкте) қол жеткізуді қамтамасыз етеді элементтері құрылымдар; олар туралы 6-тарауда айтылатын б, сол жерде sizeof операторы туралы да қарастыратын боламыз. Операторлар * (көрсеткіш бойынша жанама өтініш) және & (объектінің мекенжайын алу) 5-тарауда талқыланады. “Үтір” операторы 3-тарауда қаралатын болады.

Операторлардың басымдылығы мен ассоциативтілігі

Операторлар	**Орындалады**
() [] -> .	солдан оңға қарай
! ~ ++ -- + - * & (тип) sizeof	оңнан солға қарай
* / %	солдан оңға қарай
+ -	солдан оңға қарай
<< >>	солдан оңға қарай
< < = > > =	солдан оңға қарай
== ! =	солдан оңға қарай
&	солдан оңға қарай
^	солдан оңға қарай
|	солдан оңға қарай
&	оңнан солға қарай
||	оңнан солға қарай
?:	солдан оңға қарай
= += -= *= /= %= &= ^= |= <<= >>=	
,	

Ескерту. Бірыңғай операторлар +, -, * және & бірдей екілік операторларға қарағанда жоғары басымдыққа ие.

&, ^ және | меншіктік операторларының басымдылығы == және ! қарағанда төмен екенін ескеріңіз !

```
if ((x & MASK) == 0) ...
```

Дұрыс нәтиже алу үшін жақшаларды пайдалану керек.

С көптеген тілдер сияқты операторлардың операндтарын есептеу кезектілігін белгілемейді (& &, ||, ?: және , қоспағанда). Мысалы, нұсқауда көрініс:

```
x = f() + g();
```

f бұрын g немесе керісінше есептелуі мүмкін. Осыдан, егер функциялардың бірі басқа функция тәуелді айнымалының мәнін өзгертетін болса, онда x нәтижеге орналастырылатын есептеулер кезектілігіне байланысты болуы мүмкін. Есептеулердің қажетті дәйектілігін қамтамасыз ету үшін аралық нәтижелерді уақытша айнымалыларда есте сақтауға болады.

Функция аргументтерін есептеу кезектілігі де анықталмады, сондықтан әр түрлі компиляторларда:

```
printf("%d %d\n", ++n, power(2, n)); /* НЕВЕРНО */
```

сәйкес келмейтін нәтижелер бере алады. Функцияны шақыру нәтижесі компилятор n — дейін немесе power-ге жүгінгеннен кейін ұлғайту командаларын жасаған кезде байланысты болады. Өзіңізді ықтимал жанама әсерден қорғау үшін төменде көрсетілген өрнекті жазу жеткілікті:

```
++n; printf("%d %d\n", n, power(2, n));
```

Функцияларға жүгінулер, берілген иемденулер, инкрементті және декрементті операторлар кейбір айнымалылардың мәндерін есептеу кезінде өзгеретіндіктен көрінетін "жанама әсер" береді. Жанама әсері бар кез келген өрнекте өрнектің нәтижесінің өрнекке кіретін айнымалы мәндерінің өзгеру кезектілігіне қарай көрінетін тәуелділігі жасырылуы мүмкін. Мұндай, мысалы, әдеттегі жағымсыз жағдайда

```
a[i] = i++;
```

сұрақ туындайды: a массиві ескі немесе өзгертілген I мәнімен индекстеледі ме? Компиляторлар осы жазбаны түсіндіруде көрінетін бағдарламаны керемет жасай алады. Стандарт саналы түрде осындай сұрақтардың көпшілігі компиляторлардың қалауы бойынша қалдырылады, себебі ең жақсы есептеу тәртібі машина архитектурасымен анықталады. Стандартты дәлелдерді есептеу кезінде барлық жанама әсерлері функцияға кірер алдында көрінеді. Шынын айтқанда, printf мысалында бұл бізге көмектеспейді.

Қорытындылай келе, қандай тілді пайдалансаңыз да есептеу кезегіне байланысты бағдарламаларды жазу жақсы идеялардың бірі емес. Әрине, неден аулақ болу керек екендігін білу керек, бірақ егер сіз әртүрлі машиналарда жанама әсерлердің қалай пайда болатынын білмесеңіз, онда жеке іске асыру ерекшеліктерінде жеңіске жетуді күтуге болмайды.

3-Тарау. Басқару ағыны

Есептеулер орындалатын тәртіп басқару нұсқаулықтарымен анықталады. Біз алдыңғы мысалдарда осындай ең көп таралған басқару құрылымдарымен кездестік, ал, осында біз қаралып өткен тізімді аяқтаймыз.

3.1. Нұсқаулар мен блоктар

x = 0, немесе i++, немесе printf (...) өрнегінің егер соңында нүктені үтірмен қойса, нұсқаулық болады, мысалы:

```
x = 0;
i++;
printf (…);
```

С-да үтірлі нүкте Паскаль тілі сияқты бөлгіш емес, нұсқаулықтың қорытынды символы болып табылады.

Фигуралы жақшалар {және} хабарландырулар мен нұсқауларды құрама нұсқауға немесе блокқа біріктіру үшін қолданылады, синтаксис тұрғысынан бұл жаңа құрылым бір нұсқаулық ретінде қабылданады. Функцияның денесін құрайтын нұсқаулар тобын жиектейтін фигуралық жақшалар – бұл бір мысал; екінші мысал – бұл if, else, while немесе for кейін орналастырылған нұсқауларды біріктіретін жақшалар. (Айнымалылар кез келген блоктың ішінде жариялануы мүмкін, бұл туралы әңгіме 4-тарауда өтеді.) Оң жақ жабатын фигуралы жақшадан кейін блоктың соңында үтірмен нүкте қойылмайды.

3.2. If-else құрылымы

If-else нұсқаулығы шешім қабылдау үшін пайдаланылады. Формальды оның синтаксисі:

```
if (өрнек)
  1-нұсқаулық
```

```
else
  2-нұсқаулық
```

else-бөлігі болмауы да мүмкін. Алдымен өрнек есептеледі және егер ол шын болса (яғни нөлден жақсы), нұсқау орындалады. Егер өрнек жалған болса (яғни оның мәні нөлге тең) және else-бөлігі болса, онда 2-нұсқаулық орындалады.

If жай өрнектің сандық мәнін тексеретіндіктен, шартты кейде қысқартылған түрде жазуға болады. Осылай, жазу

```
if (өрнек)
```

```
if (өрнек ! = 0)
```

қарағанда қысқарақ.

Кейде мұндай қысқартулар табиғи және айқын, басқа жағдайларда, керісінше, бағдарламаны түсінуді қиындатады.

Бір-біріне салынған if-конструкциялардың бірінде else-бөлігінің болмауы жазбаның бір мағыналы түсіндірілуіне әкелуі мүмкін. Бұл белгісіздік else өзінің else жоқ ең жақын if-мен байланыстыруға рұқсат береді. Мысалы

```
if (n > 0)
    if (a > Ь)
        z = a;
    else
        z = b;
```

else ішкі if-ге қатысты, біз оны шегіністер арқылы көрсеттік. Егер бізге басқа түсіндіру қажет болса, фигуралық жақшаларды дұрыс орналастыру керек:

```
if (n > 0) {
    if (a > b)
        z = a;
}
else
     z = b;
```

Төменде әркелкілік әсіресе қауіпті болған жағдайдың мысалын көре аласыз:

```
if (n >= 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0) {
            printf ( “ . . . “ ) ;
            return i;
        }
else /* НЕВЕРНО */
    printf(“қате-терiс n\n”);
```

Шегіністер арқылы біз бізге не қажет екенін анық көрсеттік, бірақ компилятор бұл ақпаратты қабылдамайды және else ішкі if-ге жатқызады. Мұндай қателерді іздеу өте қиын. Мұнда келесі кеңес орынды: if-ті фигуралы жақшалармен жабыңыз.

Айтпақшы, z = a в кейін үтірлі нүктеге назар аударыңыз.

```
if (a > b)
    z = a;
else
    z = b;
```

Мұнда ол міндетті, өйткені грамматика ережелері бойынша if үшін нұсқаулықты ұстану керек, ал z = a сияқты өрнек нұсқаулық бойынша әрқашан нүктелі үтірмен аяқталады.

3.3. Else-if құрылымы

```
if (өрнек)
    нұсқаулық
else if (өрнек)
    нұсқаулық
else if (өрнек)
    нұсқаулық
else if (өрнек)
    нұсқаулық
else
    нұсқаулық
```

Құрылымы жиі кездеседі, ол туралы бөлек тақырып қозғауға да болады. If нұсқауларының келтірілген тізбегі – көп сатылы шешім қабылдауды сипаттаудың ең жалпы тәсілі. Өрнектер рет-ретімен есептеледі; “ақиқат” мәнімен өрнек кездескен соң, оған сәйкес нұсқаулық орындалады; бұл жерде тексерулердің реттілігі аяқталады. Мұнда сөз астында нұсқаулық бір нұсқауды немесе нұсқаудың бір тобы фигуралық жақшадағы нұсқауды білдіреді.

Соңғы else-барлық алдыңғы шарттар орындалмаса, бөлігі іске қосылады. Кейде соңғы бөлігінде ешқандай әрекет жасау қажет емес, бұл жағдайда фрагмент

```
else
    нұсқаулық
```

қате (“мүмкін емес”) жағдайды бекіту үшін төмендетуге немесе пайдалануға болады.

V массивіндегі x мәнінің екілік іздеу функциясын қарастырайық. V элементтері өсу ретімен реттелген деп болжанады. Функция x-та v (0-ден n-1-ге дейінгі сан), егер x бар болса, және -1 , егер жоқ болса.

Егер x осы мәннен аз болса, онда іздеу аймағы V массивінің “жоғарғы” жартысы болады, керісінше болса – “төменгі”. Қалай болғанда да, келесі қадам-таңдалған жартысы алынған орта элементпен салыстыру. Ауқымның “бұрылу” процесі іздеу ауқымы бос болғанша немесе мән табылғанға дейін жалғасады. Бинарлы іздеу функциясын жазамыз:

```
/* binsearch: найти x в v[0] <= v[1] <= … <= v[n-1] */
int binsearch(int x, int v[], int n)
{
    int low, high, mid;

    low = 0;
    high = n - 1 ;
     while (low <= high) {
```

```
            mid = (low + high) / 2;
            if (x < v[mid])
                high = mid - 1;
            else if (x > v[mid])
                low = mid + 1 ;
            else /* ұқсастық бар */
                return mid;
        }
        return -1; /* ұқсастық жоқ */
    }
```

Әрбір іздеу қадамында орындалатын негізгі әрекет-v[mid] элементімен x (аз, көп немесе тең) мәнін салыстыру; бұл салыстыру else-if құрылымдарын табиғи түрде тапсыру.

3.1-жаттығу. Біздің бинарлық іздеу бағдарламасында цикл ішінде екі тексеру жүзеге асырылады, бірақ біреуі ғана болуы мүмкін (циклден тыс тексерулер саны ұлғайған кезде). Цикл ішінде бір тексеруді қарастыра отырып, бағдарламаны жазыңыз. Орындау уақытының айырмашылығын бағалаңыз.

3.4. Switch қосқышы

Switch нұсқаулығы көптеген жолдардың бірін таңдау үшін қолданылады. Ол өрнектің мәні бірнеше бүтін константалардың мәндерінің бірімен сәйкес келетінін тексереді және осы мәнге сәйкес бағдарлама тармағын орындайды

```
switch (өрнек) {
case тұрақты өрнек:
инструкции case тұрақты өрнек:
инструкции default: нұсқаулық
}
```

Case әрбір тармағы бір немесе бірнеше бүтін константалармен немесе тұрақты өрнектермен белгіленген. Есептеулер тұрақты өрнектің мәнімен сәйкес келетін case тармағынан басталады. Барлық case тармақтарының константалары бір-бірінен ерекшеленуі керек. Егер тұрақтардың ешқайсысы дұрыс емес екені анықталса, онда default сөзімен белгіленген тармақ орындалады, егер ондай болса,

әйтпесе ештеңе жасалмайды. Case және default бұтақтарын кез келген тәртіпте орналастыруға болады.

1-тарауда біз әр санның, символдар бөлгіштердің (Бос орындар, табуляция және жаңа жолдар) және барлық басқа символдардың мәтініне кіруді санайтын бағдарламаны жаздық. Онда біз if…else if…else. тізбегін қолдандық. Енді switch қосқышымен осы бағдарламаның нұсқасын ұсынамыз:

```
#include <stdio.h>

main() /* сандарды, таңбаларды-бөлгіштерді және басқа да символдарды
есептеу */
{
        int c, i, nwhite, nother, ndigit[10];

        nwhite = nother = 0;
        for (i = 0; i < 10; i++)
            ndigit[i] = 0;
        while ((c = getcharO) != EOF) {
            switch (c) {
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                ndigit[c - '0' ]++;
                break;
            case ' ':
            case '\n':
            case '\t':
                nwhite++;
                break;
            default:
                nother++;
                break;
            }
}
printf ("цифр =");
for (i= 0; i < 10; i++)
    printf (" %d", ndigit[i]);
printf (", бөлгіш таңбалар = %d, басқалар = %d\n", nwhite, nother);    return
0;
}
```

Break нұсқаулығы switch қосқышынан дереу шығуды тудырады. Case тармағын таңдау белгіге өту ретінде жүзеге асырылғандықтан, case тармағын орындағаннан кейін, егер ештеңе істемесе, бағдарлама келесі тармаққа төмен түседі. Break және return нұсқаулары — қосқыштың ең көп таралған шығу құралдары. Break нұсқаулығы while, for және do-while циклдерінен мәжбүрлеп шығу үшін де қолданылады (біз бұл туралы кейінірек тағы айтатын боламыз).

Case тармақтарын " өтпелі " орындау аралас сезімдерді тудырады. Бір жағынан, бұл жақсы, өйткені бірнеше case тармақтарын біз мысалда сандармен бірге келдік. Бірақ екінші жағынан, бұл әрбір бұтақтың соңында келесіге ауысудан аулақ болу үшін break қоюға тура келеді. Тармақтарда дәйекті өту – сенімсіз нәрсе, бұл қателерге толы, әсіресе бағдарламаны өзгерту. Бір есептеу үшін бірнеше таңбалы жағдайды қоспағанда, өтпелі жолды мүмкіндігінше сирек қолдануға тырысыңыз, бірақ егер сіз оны қолдансаңыз, міндетті түрде түсініктеме бергеніңіз жөн.

Сіз үшін бір кеңес: тіпті соңғы тармақтың соңында (біздің мысалда default кейін) break нұсқаулығын орналастырыңыз, бірақ логика тұрғысынан оған ешқандай қажеттілік жоқ. Алайда, бұл кішкентай сақтық сізді сізге бір күні тағы бір case тармағын қосу кезінде қажет болғанда сақтайды.

3.2-жаттығу. escape(s, t) функциясын жазыңыз, ол мәтінді t-дан s-ге көшіргенде жаңа Жол және табуляция сияқты таңбаларды "символдардың көрінетін тізбектеріне" (\n және \t сияқты) түрлендіреді. Switch нұсқаулығын пайдаланыңыз. Тізбектердің эскейпін нақты таңбаларға кері түрлендіретін функцияны жазыңыз.

3.5. While және for циклдері

Біз while және for циклдарымен осыған дейінгі тақырыптарда кездестік. Циклде

```
while (өрнек)
    нұсқаулық
```

өрнек есептеледі. Егер оның мәні нөлден жақсы болса, онда нұсқаулық орындалады және өрнекті есептеу қайталанады. Бұл цикл өрнек нөлге тең болғанша жалғасады, содан кейін есептеулер нұсқаулықтан кейін бірден орналасқан нүктеден жалғасады.

Нұсқаулық for

```
for (өрнек1; өрнек2; өрнек3)
      нұсқаулық
```

мына құрылымға эквивалентті

```
өрнек1;
while (өрнек2) {
          нұсқаулық
          өрнек3;
}
```

3 бөлімде талқыланатын жалғастырушы нұсқаулықтағы мінез-құлықтағы айырмашылықтарды қоспағанда.

Грамматика тұрғысынан for циклінің үш компоненті еркін өрнектер болып табылады, бірақ өрнек$_1$ және өрнек$_2$ – функцияларды тағайындау немесе шақыру дегенді, ал өрнек$_2$ – қарым-қатынас дегенді білдіру. Осы үш өрнектің кез келгені болмауы мүмкін, бірақ нүктені үтірмен түсіруге болмайды. Өрнек$_1$ немесе өрнек3 болмаған жағдайда олар цикл конструкциясында жоқ деп саналады; өрнек$_2$ болмаған жағдайда оның мәні әрдайым шынайы деп болжанады. Мысалы,

```
for (;;) {
     ...
}
```

“шексіз” цикл да бар, оның орындалуы басқа жолмен, мысалы break немесе return нұсқауларымен үзіледі.

Қандай цикл таңдауға болады: while немесе for – бұл өз қалауыңызда. Осылайша,

```
while ((c = getchar()) == ' ' || c == '\n' || c == '\t')
      ; /* бөлгіш таңбаларды айналып өту */
```

инициализация да, параметрді қайта санау да жоқ, сондықтан мұнда while қолайлы.

Кейбір айнымалы мәнінің қарапайым инициализациясы және қадамдық өсуі бар жерде, бұл циклда оның ұйымдастыру бөлігі жазбаның басында шоғырланған. Мысалы, массивтің алғашқы n элементтерін өңдеу циклінің басталуы келесі түрге ие:

```
for (i = 0; i < n; i++)
    …
```

Бұл Фортран мен Паскалдағы for-циклдарға ұқсас. Алайда, ұқсастықтар дәл емес, өйткені С-да индекс және оның шекті мәні циклдің ішінде өзгеруі мүмкін және циклден шыққаннан кейін і индексінің мәні әрқашан анықталған. Циклдің үш компоненті ерікті өрнектермен болуы мүмкін болғандықтан, for-циклдарды ұйымдастыру тек арифметикалық прогрессия жағдайымен шектелмейді. Дегенмен, инициализация мен инкрементацияға қатысы жоқ есептеу циклінің тақырыбына қосу нашар стиль деп саналады. Тақырып тек циклды басқару операциялары үшін қалдыру керек.

Көп әсерлі мысал ретінде жолды оның сандық эквивалентіне түрлендіретін atoi бағдарламасының басқа нұсқасын келтіреміз. Бұл 2-тарауда қарастырылған белгілермен салыстырғанда, ол сол таңбаларды елемейтін бөлгіштерге (егер олар болса) және сандар алдында тұра алатын + және - белгілеріне дұрыс жауап береді. (4-тарауда өзгермелі нүктелі сандар үшін ұқсас түрлендіруді жүзеге асыратын atof нұсқасы қарастырылады).

Бағдарламаның құрылымы енгізілетін ақпараттың түрін көрсетеді:

```
егер бар болса, бөлгіш таңбаларды елемеу
егер бар болса, белгі алыңыз
бүкіл бөлігін алып, оны түрлендіру
```

Әрбір қадамда жұмыстың белгілі бір бөлігі орындалады және оның нәтижесі нақты бекітіледі, содан кейін келесі қадамда пайдаланылады. Деректерді өңдеу санның бір бөлігі бола алмайтын бірінші таңбада аяқталады.

```
#include <ctype.h>

/* atoi: s бүтін санға түрлендіру; 2-нұсқасы*/
int atoi(char s[])
{
    int i, n, sign;

    /* бөлгіш таңбаларды елемеу */
    for (i = 0; isspace(s[i]); i++)
        ;
    sign = (s[i] == '-') ? -1: 1;
    if (s[i] == '+' || s[i] == '-') /* белгі рұқсаттамасы */
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10 * n + (s[i] - '0' );
    return sign * n;
}
```

Стандартты кітапханада жолды ұзын бүтін (long int) — strtol функциясы (В қосымшасының 5-параграфын қараңыз).

Циклді басқаруды орталықтандыру беретін артықшылықтар бірнеше циклдар бір-біріне салынған кезде одан да айқын болады. Бұл алгоритмнің негізгі идеясы ерте кезеңдерде көрші элементтер емес, бір-бірінен алыс қалған элементтермен салыстырылады. Бұл жаппай тәртіпсіздікті тез жоюға әкеледі, соның арқасында кейінгі кезеңде аз жұмыс қалады. Салыстырылатын элементтер арасындағы аралық біртіндеп бірлікке дейін азаяды және осы сәтте сұрыптау көрші элементтердің әдеттегі орнына азаяды. Shellsort бағдарламасының келесі түрі бар:

```
/* shellsort: сортируются v[0] ... v[n-1] өсу тәртібімен */
void shellsort (int v[], int n)
{
      int gap, i, j, temp;
      for (gap = n/2; gap > 0; gap /= 2)
          for (i = gap; i < n; i++)
```

```
        for (j = i - gap; j >= 0 && v[j] > v[j + gap]; j -= gap) {
            temp = v[j];
            v[j] = v[j + gap];
            v[j + gap] = temp;
        }
}
```

Мұнда бір-біріне салынған үш цикл қолданылған. Сыртқы салыстырмалы элементтер арасындағы gap аралығын басқарады, оны N/2-ден нөлге дейін екіге бөлу арқылы қысқартады. Орташа цикл элементтерді таңдайды. Ішкі бір-бірінен gap қашықтықта тұрған элементтердің әрбір жұпты салыстырады және элементтерді реттелмеген жұптарда ауыстырады. Gap міндетті түрде бірлікке байланысты болғандықтан, барлық элементтер реттеледі. For циклінің әмбебаптығы сыртқы циклді басқаларға ұқсас етіп жасауға мүмкіндік береді, бірақ ол арифметикалық прогресс емес.

Соңғы С операторы-бұл “, “ (үтір), ол көбінесе for нұсқауларында қолданылады. Үтірмен бөлінген өрнектердің жұбы солдан оңға қарай есептеледі. Нәтиженің түрі мен мәні оң жақ өрнектің түрі мен мәні болып табылады, бұл әр үш компоненттің for нұсқауларында бірнеше өрнектермен қатар екі индексті жүргізуге мүмкіндік береді. Мұны сол s жолында нәтиже қалдырып, s жолын “бұрады” reverse(s) функциясының мысалында көрсетеміз:

```
#include <string.h>

/* reverse: s жолын бұрады (s нәтижесі) */
void reverse(char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

Үтір, бөлісетін функциялар дәлелдері, хабарландырулардағы айнымалылар және т.б. операторларға кедергі болмайды және солдан оңға қарай есептеулерді қамтамасыз етпейді.

Оператор ретінде үтірлерді қалыпты пайдалану керек. Олар ең алдымен бір-бірімен тығыз байланысты конструкцияларда (reverse бағдарламасының for-циклында сияқты), сондай-ақ макростарда орынды көп сатылы есептеулер бір сөйлеммен көрсетілуі тиіс. Үтір-reverse бағдарламасында оператор ретінде бұл алмасуды бір жеке операция ретінде ойлай отырып, жолдың элементтердің тексерілетін жұптарында таңбаларды айырбастау кезінде де пайдалануға болады:

```
for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    c = s[i], s[i] = s[j], s[j] = c;
```

3.3-жаттығу. A-Z сияқты қысқартылған жазбаны ауыстыратын expand (s1, s2) функциясын s1 жолында abc. . . xyz толық жазбасына балама жазыңыз. . . S1-де әріптер (бас және кіші) мен сандар жіберіледі. a-b-c, a-z0-9 және-a-b. сияқты жағдайларды шеше білу керек.

3.6. do-while циклі

1-тарауда айтылғандай, while және for циклдарында циклдің аяқталу шарттарын тексеру жоғарыда орындалады. С-да циклдің тағы бір түрі бар, онда бұл тексеру while және for қарағанда төменгі жағында цикл денесінің әрбір өткеннен кейін жасалады, яғни дене кем дегенде бір рет орындалғаннан кейін. dowhile циклында келесі синтаксис бар:

```
do
    нұсқаулық
while (өрнек);
```

Алдымен нұсқаулық орындалады, содан кейін өрнек есептеледі. Егер ол шынайы болса, онда нұсқаулық қайтадан жасалады және

т.б. өрнек жалған болған кезде цикл жұмысын аяқтайды. Do-while циклі Паскалдағы repeat-until цикліне тең, бұл бірінші жағдайда циклді жалғастыру шарты, ал екіншісінде — оның аяқталу шарты көрсетіледі.

Тәжірибе көрсеткендей, do-while циклы while және for-дан әлдеқайда аз қолданылады . Дегенмен, уақыт өте келе оған қажеттілік, мысалы, санды символдар жолына түрлендіретін itoa (atoi-ге кері қатынасы) функциясында пайда болады. Бұл түрлендіруді орындау күткендегіден әлдеқайда қиын болды, себебі қарапайым Алгоритмдер сандарды керісінше жасайды. Біз алдымен сандардың кері дәйектілігі қалыптасатын нұсқаға тоқтадық, содан кейін ол кері қайтарылады.

```
/* itoa: n-ді s жолына түрлендіру */
void itoa (int n, char s[])
{
      int i, sign;
      if ((sign = n) < 0) /* белгіні сақтаймыз */
          n = -n; /* n оң жасаймыз */
      i = 0;
      do { /* сандарды кері ретпен генерациялаймыз */
           s[i++] = n % 10 + '0'; /* келесі сан */
      } while ((n /= 10) > 0); /* оны алып тастау */
       if (sign < 0)
            s[i++] = '-';
       s[i] = '\0';
       reverse(s);
}
```

Мұнда do-while дизайны қажет немесе кем дегенде ыңғайлы, өйткені n нөлге тең болса да, s-ке кем дегенде бір таңба жіберіледі. Циклдің денесінде біз бір нұсқаулықты фигуралы жақшалармен бөлеміз (олар артық болса да), оқылмаған оқырман while циклінің басында while сөзін қате қабылдамайды.

3.4-жаттығу. Сандарды ұсыну үшін қосымша код қолданылатын болса, біздің itoa нұсқасы модуль бойынша ең үлкен теріс санмен орындай алмайды, оның мәні -(2n1) тең, мұнда n – сөз өлшемі.

Осыған қарағанда, бұл туындаған. Бағдарламаны орындалатын машинаға қарамастан, көрсетілген санның дұрыс мәнін беретін етіп өзгертіңіз.

3.5-жаттығу. itob(n, s, b) функциясын жазыңыз, ол b негізі бойынша санды білдіретін s жолына бүтін n аударады.

3.6-жаттығу. itoa нұсқасын өрістің ең төменгі енін көрсететін қосымша үшінші дәлелмен жазыңыз. Қажет болған жағдайда түрлендірілген Сан сол жақта бос орындармен толықтырылуы тиіс.

3.7. Break және continue нұсқаулары

Кейде циклдің басында немесе соңында жүзеге асырылатын тексеру нәтижесі бойынша емес, басқа тәсілмен циклдан шығу ыңғайлы болады. Бұл мүмкіндік for, while және do-while циклдері үшін, сондай-ақ switch қосқышы үшін break нұсқаулығын ұсынады. Бұл нұсқаулық оның циклдарының немесе қосқыштарының ішіндегі ең жылдам шығуды тудырады.

Келесі функция, trim, жолдан соңғы бос орындарды, табуляцияларды, жаңа жолдың таңбаларын алып тастайды; break онда циклден шығу үшін сол жақтан анықталған бірінші таңбадан бөлек қолданылады.

```
    /* trim: соңғы бос орындарды, табуляцияларды және жаңа жолдарды
жояды */
    int trim(char s[])
    {
        int n;
        for (n = strlen(s)-1; n >= 0; n--)
            if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
                break;
        s[n+1] = '\0';
        return n;
    }
```

strlen функциясымен жолдың ұзындығын алуға болады. for циклы оны кері ретпен қарап, соңынан бастап, бос орын, табуляция және жаңа жолынан басқа символ кездестіргенге дейін қарайды. Цикл

мұндай таңба анықталғаннан кейін немесе n теріс болған кезде үзіледі (яғни барлық жол қаралады). Функция бос немесе тек бөлгіш таңбалардан тұратын жағдайда да дұрыс әрекет ететініне көз жеткізіңіз.

continue нұсқаулығы break сияқты, бірақ әлдеқайда аз қолданылады. Ол ең жақын көлемді циклды (for, while немесе do-while) келесі Итерация қадамын бастауға мәжбүр етеді. While және do-while үшін бұл жағдайды тексеруге, ал for – қадамның өсуіне дереу өтуді білдіреді. Continue нұсқаулығын тек циклдарға ғана қолдануға болады, бірақ switch емес. Циклда орналасқан қосқыш қосқышының ішінде, ол осы циклдің келесі итерациясына ауысады.

Мұнда a массивінің теріс емес элементтерін ғана өңдейтін бағдарлама фрагменті (теріс).

```
for ( i = 0 ; i < n; i++) {
    if (a[i] < 0) /* теріс элементтерді өткізу */
        continue;    /* оң элементтерді өңдеу */
}
```

Continue нұсқаулығына циклдың қалған бөлігі күрделі болған кезде жиі жүгінеді, ал оның шарттарын керісінше ауыстыру және тағы бір деңгейді енгізу тіркеме деңгейлерінің тым көп болуына әкеледі.

3.8. Goto және белгілер нұсқаулары

С-да көптеген goto нұсқаулары мен оларға көшу үшін белгілер бар. Қатаң айтқанда, бұл нұсқаулықта ешқандай қажеттілік жоқ және іс жүзінде онсыз әрдайым оңай. Әлі күнге дейін біз кітапта goto-ны пайдаланбадық.

Алайда, goto пайдалы болуы мүмкін жағдайлар бар. Кейбір терең ішкі құрылымда өңдеуді үзіп, екі немесе одан да көп Ішкі циклдардан бірден шығу қажет болған кездегі ең әдеттегі жағдай. Мұнда break нұсқаулығы көмектеспейді, себебі ол тек ішкі циклден шығу-

ды қамтамасыз етеді. Мысал ретінде келесі құрылымды қарастырайық:

```
for (…)
      for   (…) {
            …
            if (disaster) /* егер апат болса */
                goto error; /* қатеге кету */
      }
      …

error: /* қатені өңдеу */
тәртіпсіздікті жою
```

Егер қате жағдайды өңдеу бағдарламасы тривиальды болмаса және қате бірнеше жерде орын алуы мүмкін болса, бағдарламаның мұндай ұйымы ыңғайлы.

Белгі қос нүктеден кейін әдеттегі айнымалы атқа ие. Белгіге goto көмегімен осы функцияның кез келген жерінен өтуге болады, яғни белгі функцияның бүкіл бойында көрінеді.

Тағы бір мысал ретінде осы мәселені қарастырайық: a және b массивтерінде сәйкес келетін элементтер бар-жоғын анықтау. Оны іске асырудың мүмкін нұсқаларының бірі мынадай:

```
for (i = 0 ; i < n; i++)
     for (j = 0; j < m; j++)
          if (a[i] == b[j])
              goto found;
/* ұқсас элементтер жоқ */
…
found:
/*ұқсастық табылды: a[i] == b[j] */
 …
```

Сәйкес элементтерді табу бағдарламасын goto-сыз жазуға болады, бірақ бұл үшін қосымша тексерулер мен басқа айнымалыны төлеңіз:

```
found = 0;
for (i = 0; i < n && !found; i++)
     for (j = 0; j < m && !found; j++)
          if (a[i] == b[j])
              found = 1;
      if (found)
          /* ұқсастық табылды: a[i-1] == b[j-1] */
          …
      else
          /* ұқсас элементтер жоқ */
          …
```

Кейбір сирек жағдайлардан басқа, goto қолданылатын бағдарламалар, әдетте, goto-сыз сол міндеттерді шешетін бағдарламаларға қарағанда түсіну және сүйемелдеу үшін қиын. Біз осы мәселедегі догматика болмаса да, goto-ға осы нұсқаулықты мүлдем қолдансаңыз, өте сирек жүгіну керек деп ойлаймыз.

4-тарау. Функциялар мен бағдарламаның құрылымы

Функциялар үлкен есептеу тапсырмаларын ұсақ есептерге бөледі және бағдарлама жасауды "нөлден" әр жолы бастамай, басқа әзірлеушілер жасаған нәрселерді пайдалануға мүмкіндік береді. Тиісті түрде таңдалған функцияларда бағдарламаның басқа бөліктері үшін елеусіз олардың жұмыс істеу бөлшектері жасырынып тұрады, бұл бағдарламаны тұтастай анық етеді және оған өзгерістер енгізуді жеңілдетеді.

Тіл функциялардың тиімді және қарапайым болуы үшін жобаланған. Әдетте Си бағдарламалары аз ғана үлкен емес, шағын функциялардан тұрады. Бағдарламаны бір немесе бірнеше бастапқы файлдарда орналастыруға болады. Бұл файлдарды бөлек құрастыруға және бұрын құрастырылған кітапхана мүмкіндіктерімен бірге жүктеуге болады. Мұнда жүктеу процесі қарастырылмайды, себебі ол әртүрлі жүйелерде ерекшеленеді.

Функцияны жариялау және анықтау – бұл ANSI стандарты тілге ең маңызды өзгерістер енгізілген аймақ. 1-тарауда көргеніміздей, функция сипаттамасында дәлелдердің түрлерін орнатуға рұқсат етілген. Функцияны анықтау синтаксисі де өзгертілді, сондықтан қазір хабарландыру және функцияларды анықтау бір-біріне сәйкес келеді. Бұл компиляторға бұрынғыдан әлдеқайда көп қателер табуға мүмкіндік береді. Сонымен қатар, егер дәлелдердің түрлері тиісті түрде жарияланса, онда қажетті дәлелдерді түрлендіру автоматты түрде орындалады.

Стандарт атаулардың көріну аймағын анықтайтын ережелерге айқындық енгізеді; атап айтқанда, ол әрбір сыртқы объект үшін бір ғана анықтама болуын талап етеді. Онда инициализация құралдары жалпыланған: енді автоматты массивтер мен құрылымдарды инициализациялауға болады.

Сондай-ақ, С прцессоры жақсартылған. Ол шартты компиляция директиваларының кең жиынтығын қамтиды, макронгументтер-

ден тырнақшаларда жолдарды генерациялау мүмкіндігін береді, сонымен қатар, макро кеңейту процесін басқарудың жетілдірілген механизмін қамтиды.

4.1. Функциялар туралы негізгі мәліметтер

Бастайық, бағдарламаны құрастырамыз, онда таңбалар жолы түрінде берілген кейбір «үлгі» бар енгізілетін мәтін жолдарын басып шығару. (Бұл бағдарлама UNIX жүйесінің grep функциясының жеке жағдайын білдіреді.) Мысал қарастырайық: мәтін жолдарында «ould» үлгісін іздеу нәтижесінде.

Ah Love! could you and I with Fate conspire
To grasp this sorry Scheme of Things entire,
Would not we shatter it to bits — and then
Re-mould it nearer to the Heart's Desire!
біз одан аламыз

Ah Love! could you and I with Fate conspire
Would not we shatter it to bits — and then
Re-mould it nearer to the Heart's Desire!

Үлгіні іздеу жұмыстары үш кезеңге бөлінеді:

```
while (тағы жол бар)
    if (жол үлгісі бар)
        оны теріп шығару
```

Іздеу процесінің барлық үш құрамдас бөлігі main функциясына қойылса да, берілген құрылымды сақтау және оның әрбір бөлігін жеке функция ретінде іске асыру жақсы. Бір үлкен бөлікке қарағанда, үш шағын бөлікпен айналысу оңай, өйткені іске асырудың маңызды емес ерекшеліктері функцияларда жасырылса, олардың бір-біріне жағымсыз әсер ету ықтималдығы аз. Бұдан басқа, функциялар түрінде ресімделген тиісті бөліктер басқа бағдарламаларда да пайдалы болуы мүмкін.

“While (тағы да жол бар)” конструкциясы getline-да іске асырылған (1-тарауды қараңыз), ал “оны басып шығару” сөзін дайын printf функциясымен жазуға болады. Осылайша, біз берілген үлгінің жолға кіретінін анықтайды.

Бұл мәселені шешу үшін біз strindex(s, t) функциясын жазамыз, ол t жолы басталатын s жолында орын (индекс) көрсетеді, немесе -1, егер s-да t болмаса. Егер одан әрі бізге үлгі бойынша күрделі анықтау қажет болса, біз strindex-ті басқа функцияға ауыстырамыз, бұл ретте бағдарламаның қалған бөлігін өзгеріссіз қалдырамыз. (strindexr кітапхана функциясы strstr функциясына ұқсас және тек индексті емес, көрсеткішін қайтарады.)

Мұндай жобалаудан кейін, оның “егжей-тегжейлі” айқын көрінеді. Біз жалпы бағдарлама туралы түсінік бар және оның бөліктері қалай өзара әрекеттесетінін білеміз. Біздің бағдарламада іздеу үлгісі оның әмбебаптығын төмендетеді. 5-тарауда біз символдық массивтерді инициализациялау мәселесіне қайта ораламыз және бағдарламаны іске қосу кезінде орнатылған параметрді қалай жасау керектігін көрсетеміз. Мұнда getline функциясының бірнеше өзгертілген нұсқасы бар және оны 1-тарауда қарастырылған нұсқамен салыстыру қажет.

```
#include <stdio.h>
#define MAXLINE 1000 /* енгізілетін жолдың ең үлкен мөлшері */

int getline(char line[], int max);
int strindex(char source[], char searchfor[]);

char pattern[] = “ould”; /* іздеу үлгісі */

/* үлгі бар барлық жолдарды табу */
main()
{
      char line[MAXLINE];
      int found = 0;
      while (getlinedine, MAXLINE) > 0)
            if (strindex(line, pattern) >= 0) {
                printf (“%s”, line);
                found++;
```

```
            }
        return found;
}

/* getline: s-да жолды оқиды, ұзындығын қайтарады */
int getline(char s[], int lim)
{
    int c, i;
    i = 0;
    while (--lim > 0 && (c = getchar()) != EOF && c != '\n' )
        s[i++] = c;
    if (c == '\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}

/* strindex: s-дағы t орнын есептейді немесе береді -1, егер s-да t жоқ болса */
int strindex (char s[], char t[])
{
    int i, j, k;
    for (i = 0; s[i] != '\0'; i++) {
        for (j = i, k = 0; t[k] != '\0' && s[j] == t[k]; j++, k++)
            ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}
```

Кез келген функцияның анықтамасы мынадай:

```
түрі – нәтиже, аты – функция (аргументтерді жариялау)
{
    жариялау және нұсқаулық
}
```

Анықтаманың жекелеген бөліктері, мысалы, «ең төменгі» функцияны анықтауда болмауы мүмкін.

```
dummy() {}
```

ол ештеңе есептемейді және ештеңе қайтармайды. Бағдарламаны әзірлеу процесінде мұндай ештеңе бөлмейтін функция «орын сақтаушысы»ретінде пайдалы. Егер нәтиже түрі төмен болса, онда функция int түрінің мәнін қайтарады.

Кез келген бағдарлама – бұл жай ғана айнымалылар мен функциялардың анықтамаларының жиынтығы. Функциялар арасындағы байланыстар дәлелдер, қайтарылатын мәндер және сыртқы айнымалылар арқылы жүзеге асырылады. Бастапқы файлда функциялар кез келген ретпен орналасуы мүмкін; бастапқы бағдарламаны файлдардың кез келген санына бөлуге болады, бірақ функциялардың ешқайсысы кесілген болмауы тиіс.

Return нұсқаулығы нәтижені шақырылатын функциядан шақырушыға қайтару механизмін іске асырады. Return сөзінің артында кез келген өрнек болуы мүмкін:

```
return өрнек;
```

Қажет болса, өрнек қайтарылатын функция түріне беріледі. Өрнек жиі жақшада жасалады, бірақ олар міндетті емес.

Шақыру функциясы қайтарылатын мәнді елемеуге құқылы. Сонымен қатар, return өрнегі болмауы мүмкін, содан кейін ешқандай мән шақыру функциясына қайтарылмайды. Басқару шешуші мәнсіз шақыру функциясына, сондай-ақ есептеулер «соңына» (яғни функцияның соңғы жабатын фигуралы жақшасына) жеткен жағдайда қайтарылады. Бір функцияда бір return өрнектері бар, ал басқалары – жоқ жағдай тыйым салынған (бірақ ескерту керек). Нәтиже return-ге «ұмытып» функциясы барлық жағдайларда, ол міндетті түрде «қоқыс»береді.

Main функциясы үлгі бойынша іздеу бағдарламасында табылған жолдар санын нәтиже ретінде қайтарады. Бұл сан осы бағдарлама туындаған ортада қол жетімді.

Бірнеше бастапқы файлдарда орналасқан С-бағдарламаларды құрастыру және жүктеу механизмдері әртүрлі болуы мүмкін. UNIX жүйесінде, мысалы, бұл жұмыстарды 1-тарауда көрсетілген cc командасы орындайды. Соңғы мысалымыздың үш функциясы үш түрлі файлда орналасқан: main.c, getline.c және strindex.c. сонда команда

```
cc main.c getline.c strindex.c
```

компиляция нәтижесін main объект модульдерінің файлдарына орналастыру арқылы көрсетілген файлдарды құрастырады. Егер қате болса, мысалы, main файлында.c, содан кейін оны қайта құрастыруға болады және нәтиже келесі пәрменді орындау арқылы бұрын алынған нысан файлдарымен жүктеуге болады:

```
cc main.c getline.o strindex.o
```

cc командасы стандартты файл кеңейтімдерін пайдаланады.»c» және».o» бастапқы файлдарды объектіден ажырату үшін.

4.1-жаттығу. strindex(s, t) функциясын жазыңыз, ол оң жақ t-да егер кіру анықталмаған болса. s-ге немесе -1-ге кіру орнын береді.

4.2. Мақсатсыз мәндерді қайтаратын функциялар

Алдыңғы мысалдарда функция нәтижелік мәндерді (void) қайтармаған, немесе int типті мәндерді қайтарған жоқ. Функцияның нәтижесі басқа түрге ие болған кезде қалай болуы керек? Көптеген есептеу функциялары, мысалы, sqrt, sin және cos double түріндегі мәндерді қайтарады; басқа арнайы функциялар басқа түрлердің мәндерін бере алады. Функцияның мақсатсыз мәнін қалай қайтаратынын көрсету үшін, s жолын екі есе дәлдіктің өзгермелі нүктесімен тиісті санға ауыстыратын atof(s) функциясын жазамыз. Atof функциясы atoi функциясын кеңейтеді, оның екі нұсқасы 2 және 3 тарауларында қаралды. Оның ондық нүктесімен, сондай-ақ біреуі болмауы мүмкін бүтін және бөлшек бөліктерімен белгісі бар (ол болуы мүмкін) ісі болады. Біздің нұсқасы енгізілген сандарды түрлен-

дірудің жоғары сапалы бағдарламасы емес; мұндай бағдарлама айтарлықтай көп жад талап етеді. Atof функциясы стандартты бағдарлама кітапханасына кіреді; оның сипаттамасы <stdlib.h> тақырып файлында бар.

Ең алдымен, қайтарылатын мәннің түрін atof өзі жариялау керек, себебі бұл int түрі емес. Түрдің сілтемесі функция атауынан бұрын болады.

```
#include <ctype.h>

/* atof: s жолын double-ге түрлендіру */
double atof (char s[])
{
      double val, power;
      int i, sign;
      for (i = 0; isspace (s[i]); i++)
          ; /* сол жақ бөлгіш таңбаларды елемеу */
      sign = (s[i] == '-') ? -1 : 1;
      if (s[i] == '+' || s[i] == '-')
          i++;
      for (val = 0.0; isdigit (s[i]); i++)
           val = 10.0 * val + (s[i] - '0');
      if (s[i] =='.')
          i++;
      for (power = 1.0; isdigit(s[i]); i++) {
           val = 10.0 * val + (s[i] - '0');
           power *= 10.0;
      }
      return sign * val / power;
}
```

Сонымен қатар, шақырушы бағдарлама atof мақсатсыз мәнді қайтарады деп білгені маңызды. Мұны қамтамасыз етудің бір жолы – шақырушы бағдарламада atof-ті нақты сипаттау. Мұндай сипаттама төменде қарапайым калькулятор бағдарламасында (чек кітапшасының балансын тексеру үшін жеткілікті) көрсетіледі, ол әрбір енгізілетін жолды сан ретінде қабылдайды, оны ағымдағы сомаға қосады және оның жаңа мәнін басып шығарады.

```
#include <stdio.h>
```

```
#define MAXLINE 100

/* қарапайым калькулятор */
main()
{
      double sum, atof (char[]);
      char line[MAXLINE];
      int getline (char line[], int max);

      sum = 0;
      while (getline(line, MAXLINE) > 0)
            printf («\t%g\n», sum += atof(line));
      return 0;
}
```

Жариялауда:

```
double sum, atof (char[]);
```

sum-екі түрдегі айнымалы, ал atof – функция char[] түріндегі бір аргументті қабылдайды және екі түрдегі нәтижені қайтарады делінген.

Atof функциясын хабарлау және анықтау бір-біріне сәйкес келуі керек. Егер бір бастапқы файлда atof функциясы және оған main-ге жүгінудің әртүрлі түрлері болса, онда бұл сәйкессіздік компилятормен қате ретінде бекітіледі. Бірақ егер atof функциясы бөлек құрастырылған болса (бұл мүмкін), онда түрлердің сәйкессіздігі анықталмайды және atof double түрінің мәнін қайтарады, ол main функциясы int ретінде қабылдайды, бұл мағынасыз нәтижеге әкеледі.

Бұл соңғы мәлімдеме, бәлкім, сізді таңқалдырады, себебі бұрын хабарландырулар мен анықтамаларға сәйкес келу қажеттілігі туралы айтылды. Сәйкессіздіктің себебі функцияның прототипі мүлдем жоқ болғандығынан болуы мүмкін немесе функция бірінші пайда болған кезде, мысалы,

```
sum += atof(line)
```

Егер сөйлемде бұрын жарияланбаған атау кездессе, одан кейін ашу жақшасы бар болса, онда мұндай атау контекст бойынша int түрінің нәтижесін қайтаратын функцияның атауы болып саналады; бұл ретте оның дәлелдеріне қатысты ештеңе болмайды. Егер хабарлауда функциялар дәлелдер көрсетілмесе,

```
double atof();
```

бұл жағдайда atof дәлелдері туралы ештеңе белгісіз және оның параметрлеріне сәйкес келетін барлық тексерулер өшіріледі деп саналады. Бос тізімнің мұндай арнайы интерпретациясы жаңа компиляторларға ескі Си-бағдарламаларды таратуға мүмкіндік береді. Бірақ жаңа бағдарламаларда оны пайдалану жақсы идея емес. Егер функцияда дәлелдер болса, оларды сипаттаңыз, егер олар болмаса, void сөзін пайдаланыңыз.

Тиісті түрде сипатталған atof функциясы бар, біз таңбалар жолын бүтін мәнге түрлендіретін atoi функциясын жаза аламыз:

```
/* atoi: s жолын atof арқылы int-ге түрлендіру */
int atoi (char s[])
{
    double atof (char s[]);
    return (int) atof (s);
}
```

Жариялау түріне және return нұсқауларына назар аударыңыз. Өрнектің мәні

```
return өрнек;
```

нәтиже ретінде қайтарылмас бұрын, функция түріне келтіріледі. Демек, atoi функциясы int мәнін қайтарады, double типті atof есептеу нәтижесі return нұсқауында автоматты түрде int түріне айналады. Түрлендіру кезінде ақпаратты жоғалту мүмкін және кейбір компиляторлар бұл туралы ескертеді. Келтіру операторы түрді түрлендіру қажеттілігін анық көрсетеді және кез келген ескерту хабарын басады.

4.2-жаттығу. Atof функциясын 123.456e-6 сияқты түр сандарымен жеңе алатындай етіп толықтырыңыз, мантиссадан кейін e (немесе E) келесі тәртіппен тұруы мүмкін (мүмкін, белгімен).

4.3. Сыртқы айнымалылар

С-дағы бағдарлама әдетте көптеген сыртқы нысандармен жұмыс істейді:ауыспалы және функциялар. Сын есім “сыртқы” (external) функциялардың ішінде анықталған дәлелдер мен айнымалыларға қатысты сын есімнің “ішкі” қарама-қарсы. Сыртқы айнымалылар функциялардан тыс анықталады және көптеген мүмкіндіктер үшін ықтимал қол жетімді. Функциялардың өздері әрқашан сыртқы объектілер болып табылады, өйткені С-да басқа функциялардың ішінде функцияларды анықтауға тыйым салынады. Әдепкі бойынша әр түрлі файлдарда қолданылатын бірдей сыртқы атаулар бірдей сыртқы нысанға (функцияларға) жатады. (Стандартта бұл сыртқы байланыстарды редакциялау деп аталады (external linkage7).) Осы мағынада сыртқы айнымалылар фортрандағы COMMON аймағына және Паскалдағы ең сыртқы блоктың айнымалысына ұқсас. Кейінірек біз сыртқы функциялар мен айнымалылардың тек бір бастапқы файлдың ішінде көрінетінін көрсетеміз.

Сыртқы айнымалылар барлық жерде қол жетімді болғандықтан, оларды аргументтер мен қайтарылатын мәндер арқылы байланыстарға балама ретінде функциялар арасындағы байланыстырушы деректер ретінде пайдалануға болады. Кез келген функция үшін сыртқы айнымалы аты дұрыс жарияланса, оның атымен қол жетімді.

Егер айнымалы функциялар саны үлкен болса, соңғы арасындағы байланыстар сыртқы айнымалылар арқылы ұзын аргументтер тізімдеріне қарағанда ыңғайлы және тиімді болуы мүмкін. Бірақ 1-тарауда айтылғандай, бұл мәлімдемеге сыни қарау керек, өйткені мұндай тәжірибе бағдарламаның құрылымын нашарлатады және деректер бойынша функциялар арасындағы байланыстың тым көп болуына әкеледі.

Сыртқы айнымалы пайдалы, себебі олар үлкен іс-қимыл аймағы мен өмір сүру уақыты бар. Автоматты айнымалылар функцияның ішінде ғана бар, олар функцияға кіру кезінде пайда болады және одан шыққан кезде жоғалады. Сыртқы айнымалылар, керісінше, үнемі бар, сондықтан олардың мәндері функцияларға жүгінулер арасында да сақталады. Сонымен, егер екі функцияны бірдей деректерді пайдалануға тура келсе және олардың бірде-біреуі екіншісін тудырмаса, онда бұл жалпы деректерді функцияға және кері аргументтер арқылы беруге емес, сыртқы айнымалылар түрінде ресімдеуге ыңғайлы болады.

Келтірілген пайымдауларға байланысты мысалдарды қарастырайық. Өзімізге +, -, * және/операторларын түсінетін калькулятор бағдарламасын жазу міндетін қоямыз. Бұл калькулятор сөйлемдердің инфиксті жазбасы емес, поляк тіліне бағытталса, жазуға оңай болады. (Кері поляк жазба кейбір қалта калькуляторларында және Forth және Postscript сияқты тілдерде қолданылады.

Кері поляк жазбасында әрбір оператор өз операндыларынан кейін жүреді. Инфикс жазбасындағы өрнек, айталық

```
(1 - 2) * (4 + 5)
```

Жақшалар қажет емес, есептеулерде екіұштылық болмайды, өйткені әрбір оператор үшін қанша операндтар қажет екені белгілі.

Біздің бағдарламаны іске асыру өте оңай. Әрбір операнд стекке жіберіледі; егер оператор кездескен болса, стектен операндтардың тиісті саны алынады (екілік операторлар жағдайында екі) және операция орындалады, содан кейін нәтиже стекке жіберіледі. Біздің мысалда 1 және 2 сандар стекке жіберіледі, содан кейін олардың айырмашылығы -1. Одан әрі стекке 4 және 5 сандар жіберіледі, содан кейін олардың сомасымен ауыстырылады (9). 1 және 9 сандар шыныда олардың шығаруымен ауыстырылады (яғни -9). Жаңа жолдың таңбасын кездестіргенде, бағдарлама стек мәнін шығарып, оны басып шығарады.

Осылайша, бағдарлама өзінің әрбір қадамында кезекті қарсы алатын оператор немесе операндты өңдейтін циклден тұрады:

```
while (келесі элемент соңы емес)
    if (сан)
        оны стекке жіберу
    else if (оператор)
        стек операндадан алу
        операцияны орындау
        нәтиже стекке жіберілсін
    else if (новая-строка)
        стектің жоғарғы жағынан сан алып басып шығару
    else
        қате
```

“Стекке жіберу” және “стектен алу” операциялары өздігінен тривиальды, бірақ оларға қателерді табу және бейтараптандыру тетіктерін қосу кезінде жеткілікті ұзақ болады. Сондықтан олар жақсы барлық бағдарлама бойынша тиісті кодты қайталаудан гөрі жеке функциялар түрінде рәсімдеңіз. Оған қоса, әрине, келесі операторды немесе операндты алу үшін жеке функцияға ие болу керек.

Біз әлі қарастырмаған басты мәселе – бұл стекті қайда орналастыру және оған тікелей қол жеткізуді қандай функциялар туралы мәселе. Бұл жағдайда, ол үшін, мысалы, “stack” (“stack”) және “stack” (“stack”) функцияларына аргументтер ретінде стектің өзі мен ағымдағы позициясын беруге болады. Бірақ main функциялары стек қатысты айнымалы іс жоқ, — ол стек сандар орналастыру және оларды одан алу үшін ғана операциялар қажет. Сондықтан біз стек және онымен байланысты ақпаратты push және pop функциялары үшін қол жетімді, бірақ main үшін қол жетімді емес сыртқы айнымалыларда сақтауды шештік.

Эскизден бағдарламаға өту өте оңай. Егер бағдарлама енді бір бастапқы файлда орналасқан мәтін ретінде ұсынылса, ол келесі түрге ие болады:

```
#include /* кез келген мөлшерде болуы мүмкін */
#define /* кез келген мөлшерде болуы мүмкін */
```

```
Main үшін функцияны жариялау

main () {...}

push және pop үшін сыртқы айнымалылар

void push (double f) {...} double pop (void) {...}

int getop(char s[] ) {...}
```

getop функциясы тудыратын кіші бағдарламалар

Кейінірек біз осы бағдарламаның мәтінін екі немесе одан да көп файлдарға бөлуге болатынын талқылаймыз.

Main функциясы – оператордың немесе операндтың түріне байланысты сол немесе басқа тармаққа басқаруды беретін үлкен switch қосқышы бар цикл. Мұнда 3.4 параграфында қарастырылған switch қосқышын қолданудың типтік жағдайы ұсынылған.

```
#include <stdio.h>
#include <stdlib.h> /* для atof() */

#define MAXOP 100 /* макс. размер операнда или оператора */ #define
NUMBER ‘0’ /* признак числа */

int getop (char []);
void push (double);
double pop (void);

/* калькулятор с обратной польской записью */
main ()
{
        int type;
        double op2;
        char s[MAXOP];
        while ((type = getop (s)) != EOF) {
                switch (type) {
                case NUMBER:
                        push (atof (s));
                        break;
```

```
                case '+':
                    push (pop() + pop());
                    break;
                case '*':
                    push (pop() * pop());
                    break;
                case '-':
                    op2 = pop();
                    push (pop() - op2);
                    break;
                case '/' :
                    op2 = pop();
                    if (op2 != 0.0)
                        push (pop() / op2);
                    else
                        printf("ошибка: деление на нуль\n");
                        break;
                    case '\n' :
                        printf("\t%.8g\n", pop());
                        break;
                    default:
                        printf("ошибка: неизвестная операция %s\n", s);
                        break;
                }
        }
    return 0;
    }
```

+ Және * операторлары коммутативті болғандықтан, операндаларды стектан алу тәртібі маңызды емес, алайда, - мен / операторлары жағдайында сол мен оң жақтағы операндалар басқаша болуы керек. Сонымен

```
push(pop() - pop()); /* Д9РЫС ЕМЕС*/
```

pop-ға жүгіну кезектілігі анықталмаған. Дұрыс кезектілікке кепілдік беру үшін, стектің бірінші мәні уақытша айнымалыны тағайындауы керек.

```
#define MAXVAL 100 /* максималды стек тереңдігі */

int sp = 0; /* стекте келесі бос орын */
```

```
double val[ MAXVAL ]; /* стек */

/* push: f мәнін стекке қою */
void push(double f)
{
      if (sp < MAXVAL)
          val[sp++] = f;
      else
           printf( “қателік: стек толы, %g не сыймайды\n”, f);
}

/* pop: стектің шыңынан алып, нәтиже ретінде беріңіз */
double pop(void)
{
      if (sp > 0)
          return val[--sp];
      else {
          printf( “қате: стек бос\n”);
          return 0.0;
       }
}
```

Айнымалы функциядан тыс анықталса, сыртқы болып саналады. Осылайша, push және pop үшін қол жетімді болуы керек стек және стек индексі осы функциялардан тыс анықталады. Бірақ main стек немесе стек позициясын пайдаланбайды, сондықтан олардың көрінісі main-ден жасырын болуы мүмкін.

Getop іске асырумен айналысамыз-келесі оператор немесе операндты алатын функциялар. Біз өте қарапайым тапсырманы шешуіміз керек. Дәлірек: Бос орындар мен табуляцияларды өткізіп жіберу талап етіледі; егер келесі таңба — сан емес және ондық нүкте емес болса, оны беру керек; әйтпесе, ондық нүктелі сандар жолын жинақтау керек, егер ол болса, және нәтиже ретінде NUMBER санын беру керек.

```
#include <ctype.h>

int getch(void);
void ungetch(int);
```

```
/* getop: келесі оператор немесе операнд алады */
int getop(char s[])
{
    int i, c;

   while ((s[0] = c = getch()) == ‘ ‘ || c == ‘\t’ )
         ;
    s[1] = ‘\0’;
    if (!isdigit(c) && c != ‘.’)
        return c; /* не число */
    i = 0;
    if (isdigit(c)) /* біз бүтін бөлігін жинаймыз */
        while (isdigit(s[++i] = c = getch()))
              ;
    if (c == ‘.’) /* бөлшек бөлігін жинаймыз */
        while (isdigit(s[++i] = c = getch()))
              ;
    s[i] = ‘\0’;
    if (c != EOF)
        ungetch(c);
    return NUMBER;
}
```

Getsh және ungetch функциялары қалай жұмыс істейді? Көптеген жағдайларда бағдарлама артық нәрсені оқығанға дейін қажет нәрсенің бәрін оқып шыққанын “елестете” алмайды. Сонымен, санның жинақталуы саннан басқа символ кездескенге дейін жасалады. Бірақ бұл бағдарлама бір таңбаны қажет болғаннан көп оқығанын және соңғы таңбаны санға қосуға болмайтынын білдіреді.

Бұл мәселені қажетсіз сипатты қайтаруға болатын «кері қайтару» операциясы арқылы шешуге болады.. Содан кейін бағдарлама қажет болғаннан көбірек бір таңбаны санаған сайын, бұл әрекет оны кіріске қайтарады, ал қалған бағдарлама бұл таңба мүлдем оқылмаған сияқты әрекет етеді. Бақытымызға орай, символдың кері жіберуінің сипатталған механизмі бір-бірімен келісілген функциялардың жұптары арқылы оңай модельделеді, оның ішінде getch келесі енгізу таңбасын жеткізеді, ал ungetch таңбаны кері кіріс ағынына жібереді, сондықтан getch келесі қадамда оны қайтадан аламыз.

Олар қалай бірге жұмыс істейтінін білу қиын емес. Бұл функция екі функцияға да қол жетімді таңбалар массиві болып табылатын кейбір буфердегі кері жіберілетін таңбаны есте сақтайды; getch буферден оқиды немесе буфер бос болса, getchar-ға жүгінеді. Буфердегі ағымдағы символдың жағдайын көрсететін индексті қарастыру керек.

Getch және ungetch функциялары буфер мен индексті бірге пайдаланады, өйткені соңғы мәндер қоңыраулар арасында сақталуы тиіс. Сондықтан буфер мен индекс осы бағдарламаларға қатысты сыртқы болуы керек және getch, ungetch және олар үшін ортақ айнымалыларды келесі түрде жаза аламыз:

```
#define BUFSIZE 100

char buf[BUFSIZE]; /* ungetch арналған буфер */
int bufp = 0; /* келесі, буфердегі бос орын */

int getch(void) /* символды алу (мүмкін қайтарылған) */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) /* таңбаны енгізуге қайтару */
{
     if (bufp >= BUFSIZE)
         printf (“ungetch: слишком много символов\n”);
     else
         buf[bufp++] = c;
}
```

Стандартты кітапхана бір таңбаны қайтаруды қамтамасыз ететін ungetc функциясын қамтиды (7-тарауды қараңыз). Біз жалпы тәсілді көрсету үшін, қайтарылатын таңбаларды есте сақтау үшін массивті пайдаланды.

4.4-жаттығу. Стектің жоғарғы элементін (оны стекте сақтай отырып) басып шығаруға, стектің екі жоғарғы элементін орнына өзгертуге болатын командаларды қосыңыз. Стекті Тазалау пәрменін енгізіңіз.

4.5-жаттығу. Sin, exp және pow кітапханалық функцияларды бағдарламада пайдалану мүмкіндігін қарастырыңыз. Кітапхана <math.h> қараңыз. В қосымшасында (4-параграф).

4.6-жаттығу. Айнымалылармен жұмыс істеу үшін командаларды енгізіңіз (әр қайсысы латын әліпбиінің бір әрпімен ұсынылған аты бар 26 айнымалыға дейін қамтамасыз ету оңай). Басып шығарылған мәндердің ең соңғысын сақтауға арналған айнымалыны қосыңыз.

4.7-жаттығу. S жолын кіріс ағынына қайтаратын ungets(s) бағдарламасын жазыңыз. Ungets buf және bufr айнымалылары туралы "білу" керек пе, әлде ол тек ungetch функциясын пайдалану жеткілікті ме?

4.8-жаттығу. Кері қайтарылатын таңбалардың саны 1-ден аспайды делік. Осы фактіні ескере отырып, getch және ungetch функциясын өзгертіңіз.

4.9-жаттығу. Біздің функцияларымызда EOF қайтару мүмкіндігі жоқ. EOF-ті қайтару үшін не істеу керек деп ойлаңыз және бағдарламаны түзетіңіз.

4.10-жаттығу. Калькулятор бағдарламасының негізіне бүкіл жолды оқитын getline функциясын қолдануға болады; бұл ретте getch және ungetch қажеттілігі жойылады. Бұл тәсілді іске асыратын бағдарламаны жазыңыз.

4.4. Көріну аймағы

С бағдарламасы тұратын функциялар мен сыртқы айнымалылар әр кезде бәрін бірге құрастырудың қажеті жоқ. Бастапқы мәтінді бірнеше файлда сақтауға болады. Бұрын құрастырылған бағдарламаларды кітапханалардан жүктеуге болады. Осыған байланысты келесі сұрақтар туындайды:

- Компиляция кезінде пайдаланылатын айнымалыларды дұрыс жариялау үшін хабарландыруды қалай жазуға болады?
- Жүктеу кезінде бағдарламаның барлық бөліктері қажетті түрде байланысты болуы үшін хабарландыруларды қандай тәртіппен орналастыру керек?

▪ Бір ғана көшірмесі болуы үшін хабарландыруды қалай ұйымдастыруға болады?

▪ Сыртқы айнымалыларды қалай инициализациялау керек?

Бірнеше файлға калькулятор бағдарламасын бөлшектей бастайық. Әрине, бұл бағдарлама оны файлдарға бөлу үшін тым аз, алайда біздің бағдарламаны бұзу үлкен бағдарламаларда туындайтын мәселелерді көрсетуге мүмкіндік береді.

Атаудың көріну аймағы – бұл атауды пайдалануға болатын бағдарламаның бір бөлігі. Функцияның басында жарияланған Автоматты айнымалылар үшін, көріну аймағы олар жарияланған функция болып табылады. Әр түрлі функциялардың жергілікті айнымалылары, алайда бірдей атаулары бар, бір-бірімен байланысы жоқ. Сол мәлімдеме жергілікті айнымалы функцияның параметрлеріне қатысты да әділ.

Сыртқы айнымалының немесе функцияның әрекет ету аймағы ол жарияланған бағдарлама нүктесінен компиляцияға жататын файлдың соңына дейін созылады. Мысалы, егер main, sp, val, push және pop көрсетілген тәртіпте бір файлда анықталса және т.с.с.

```
main() {…}

int sp = 0; double val[MAXVAL];

void push(double f){…}

double pop(void) {…}
```

sp және val айнымалылары тек олардың аттары бойынша push және pop-дан жіберуге болады; бұл үшін қосымша жарнамалар қажет емес. Айта кетейік, main-де бұл атаулар ит пен pop сияқты көрінбейді.

Алайда, егер сыртқы айнымалыға ол анықталғанға дейін немесе ол басқа файлда анықталғанға дейін сілтеме жасау қажет болса, онда оның хабарландыруы extern сөзімен белгіленуі тиіс.

Оны анықтаудан сыртқы айнымалыны хабарландыруды ажырату маңызды. Хабарландыру айнымалының қасиеттерін (ең алдымен оның түрін) жариялайды, ал анықтама, сонымен қатар, оған жад бөлуге әкеледі. Егер жолдар:

```
int sp;
double val[MAXVAL];
```

барлық функциялардан тыс орналасқан, онда олар sp және val сыртқы айнымалыларын анықтайды, яғни олар үшін жадты бөледі, сонымен қатар бастапқы файлдың қалған бөлігі үшін хабарландыру қызметін атқарады. Ал мына:

```
extern int sp;
extern double val[];
```

файлдың қалған бөлігі үшін sp — int типті айнымалы, a val — double типті массив (оның көлемі басқа жерде анықталған); сонымен қатар айнымалы да, массив де құрылмайды және жад оларға берілмейтінін жариялайды.

Бастапқы бағдарлама тұратын файлдардың барлық жиынтығында әрбір сыртқы айнымалы үшін жалғыз анықтама болуы керек; сыртқы айнымалыға қол жеткізу үшін басқа файлдар extern хабарландыруы болуы керек. (Алайда, extern хабарландыруын анықтама бар файлға да қоюға болады.) Массивтердің анықтамаларында олардың мөлшерін көрсету қажет, extern хабарландыруларында міндетті емес.

Сыртқы айнымалыны тек анықтамада инициализациялауға болады.

Біздің бағдарламаны осы жолмен ұйымдастыру екіталай болғанымен, бірақ біз бір файлда push және pop анықтаймыз, ал val және sp-басқа жерде оларды инициализациялайды. Бұл ретте байланыс орнату үшін осындай анықтамалар мен хабарландырулар қажет болады:

```
1 файлында:
extern int sp;
extern double val[];
void push(double f ) {…}
double pop(void) {…}
2 файлында:
int sp = 0;
double val[MAXVAL];
```

Extern хабарландырулары 1 файлының басында және функциялардың анықтамаларынан тыс болғандықтан, олардың әрекеті барлық функцияларға таралады, ал бір хабарландыру жиынтығы барлық файл үшін жеткілікті. Сол extern-хабарландыруларды ұйымдастыру бағдарлама бір файлдан тұратын, бірақ sp және val анықтамалары оларды пайдаланғаннан кейін орналасқан жағдайда да қажет.

4.5. Тақырып файлдары

Енді калькулятордың компоненттерінің айтарлықтай үлкен өлшемдері бар екенін және бұл жағдайда оларды бірнеше файлдарға қалай бөлуге болатынын ойлаймыз. Main бағдарламасын біз main деп аталатын файлға орналастырамыз.c; push, pop және олардың айнымалылары екінші файлда, stack.c; а getop-үшінші, getop.соңында, getch және ungetch төртінші getch файлында орналастырамыз.c; біз оларды басқа функциялардан бөліп алдық, өйткені нақты бағдарламада олар алдын ала құрастырылған кітапханадан алынады.

Оқырманға ескертетін тағы бір сәт бар-анықтамалар мен жарнамалар бірнеше файлдармен бірге қолданылады. Біз бұл хабарландырулар мен анықтамаларды орталықтандырғымыз келеді, олар үшін бір ғана көшірме бар. Содан кейін бағдарлама оны дамыту процесінде оңай түзетіледі және дұрыс жағдайда сақталады. Ол үшін жалпы ақпарат calc тақырып файлында орналасқан.h, ол қажет болған жағдайда басқа файлдарға қосылатын болады. (#Include жолы 4.11 параграфында сипатталады.) Нәтижесінде файлдық құрылымы төменде көрсетілген бағдарламаны аламыз:

calc.h:

```
#define NUMBER ‘0’
void push(double);
double pop(void);
int getop(char[]);
int getch(void);
void ungetch(int);
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include “calc.h”
#define MAXOP 100
main() {
…
}
```

getop.c:

```
#include <stdio.h>
#include <stype.h>
#include “calc.h”
getop (){
…
}
```

stack.c:

```
#include <stdio.h>
#include “calc.h”
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
…
}
double pop(void) {
…
}
```

getch.c:

```
#include <stdio.h>
#define BUFSIZE 100
char buf [BUFSIZE];
int bufp = 0;
int getch(void) {
…
}
```

```
void ungetch(int) {
…
}
```

Әрбір файл оған жұмыс істеу үшін қажетті ақпаратқа ғана ие болу үшін ұмтылу арасындағы ымыраға келу сөзсіз және іс жүзінде тақырып файлдары көп болған жағдайда айналысу өте қиын. Кейбір орташа өлшемнен аспайтын бағдарламалар үшін әрқайсысы екі түрлі файлда пайдаланылатын барлық нысандар бірге жиналған бір тақырып файлы болуы мүмкін; сондықтан біз мұнда келіп тұрдық. Үлкен бағдарламалар үшін тақырып файлдары көп күрделі ұйым қажет.

4.6. Статикалық айнымалылар

Stack файлындағы sp және val айнымалылары.c, сондай-ақ buf және bufp getch.c осы файлдардың жеке функцияларын пайдаланады және оларға басқа біреуге қол жеткізуді ашудың қажеті жоқ. Сыртқы айнымалыға немесе функцияға қолданылған static көрсету файлдың соңында тиісті объектінің көріну аймағын шектейді. Бұл атауларды жасыру тәсілі. Мысалы, buf және bufp айнымалылары сыртқы болуы керек, өйткені олар getch және ungetch мүмкіндіктерін бірлесіп пайдаланады, бірақ олар getch және ungetch функцияларын “пайдаланушыларға” көрінбейді.

Статикалық жады әдеттегі хабарландырудың алдында орналасқан static сөзімен ерекшеленеді. Егер біз қарастыратын екі функция және екі айнымалы бір файлда құрастырылған болса, төменде көрсетілген мысал ретінде:

```
static char buf[BUFSIZE]; /* ungetch арналған буфер */
static int bufp = 0; /* келесі, buf-дағы бос орын */
int getch(void) { ... }
void ungetch(int c) { ... }
```

онда ешқандай басқа бағдарлама buf-ге де, bufp-ге де қол жеткізе алмайды және бұл атауларды басқа файлдарда мүлдем басқа мақсаттар үшін еркін пайдалануға болады. Сол сияқты, тек push және

pop жұмыс істейтін sp және val айнымалыларын жариялаудың алдында static нұсқауын орналастыра отырып, біз оларды басқа функциялардан жасыра аламыз.

Нұсқау static көбінесе үшін пайдаланылады айнымалы, бірақ бірдей табыспен оны қолдануға болады және функциялары. Әдетте функциялардың атаулары жаһандық және бағдарламаның кез келген жерінен көрінеді. Егер функция static сөзімен белгіленген болса, онда оның аты анықталған файлдан тыс көрінбейді.

static декларацияны ішкі айнымалылар үшін де қолдануға болады. Автоматты айнымалылар сияқты ішкі статикалық айнымалы функциялар жергілікті болып табылады, бірақ автоматты айнымалылардан айырмашылығы, олар тек функцияның ұзақтығында пайда болмайды, бірақ үнемі тұрады. Бұл дегеніміз ішкі статикалық айнымалылар функция ішіндегі мәліметтердің тұрақты сақталуын қамтамасыз етеді.

4.11-жаттығу. Getop функциясын ungetch функциясына қажеттілік болмайтындай етіп өзгертіңіз. Кеңес: ішкі статикалық айнымалыны пайдаланыңыз.

4.7. Регистрлік айнымалы

Register хабарлауы компиляторға бұл айнымалы қарқынды қолданылатынын хабарлайды. Идея – register-де жарияланған айнымалы машиналар тіркеліміне орналастыру, осының арқасында бағдарлама қысқа және жылдам болуы мүмкін. Алайда компилятор бұл нұсқауды елемеуге құқылы.

Register жариялауы келесідей:

```
register int x;
register char c;
```

register хабарландыруы тек Автоматты айнымалыға және функция-

ның формальды параметрлеріне ғана қолданылуы мүмкін. Соңғысы үшін бұл төмендегідей:

```
f( register unsigned m, register long n)
{
    register int i;
    …
}
```

Іс жүзінде аппаратураның мүмкіндіктерімен байланысты регистрлік айнымалыларға шектеулер бар. Регистрде әрбір функцияның аздаған айнымалы саны ғана орналасуы мүмкін және тек белгілі бір түрлер. Регистердің артық хабарландырулары еш әсер етпейді, өйткені регистрлер жетпеген немесе регистрге орналастыру үшін ауыспалыларға қатысты еленбейді. Сонымен қатар, регистр айнымалысына қатысты, ол үшін шын мәнінде тіркелім бөлінгеніне немесе бөлінбеуіне қарамастан, мекенжай ұғымы анықталмаған (5-тарауды қараңыз). Регистрлік айнымалылардың саны мен типтеріне нақты шектеулер машинаға байланысты.

4.8. Блоктық құрылым

С-дағы функцияларды басқа функциялардың ішінде анықтауға болмайтындықтан, ол Паскальда және оған ұқсас тілдерде рұқсат етілген мағынада бағдарламаның блоктық құрылымына жол беретін тіл болып табылмайды. Бірақ функциялардың ішіндегі айнымалыларды блоктық-құрылымдық мәнде анықтауға болады. Айнымалының хабарландыруларын (инициализациямен бірге) функцияның басында ғана емес, құрамдас нұсқаулықты ашатын кез келген сол жақ фигуралық жақшадан кейін де орналастыруға рұқсат етіледі. Осындай тәсілмен сипатталған айнымалы көлемді блоктарда орналасқан сол атаумен айнымалыларды “көлеңкелейді” және тиісті оң жақ жақшаға дейін бар. Мысалы,

```
if (n > 0) {
    int i; /* жаңа i айнымалының сипаттамасы */
    for (i = 0; i < n; i++)
        …
}
```

і айнымалының көріну аймағы n>0 кезінде орындалатын if тармағы болып табылады; және бұл айнымалы осы блоктан тыс орналасқан кез келген і-ге ешқандай қатысы жоқ. Блокта жарияланған және инициалданған Автоматты айнымалы, блокты кіргенде әр жолы инициалданады. Static айнымалылары блокқа бірінші кіргенде тек бір рет инициалданады.

Автоматты айнымалы және формальды Параметрлер Сыртқы айнымалыларды және сол аттармен функцияларды "көлеңкелейді". Мысалы,

```
int x;
int y;

f(double x)
{
      double y;
      …
}
```

F функциясының ішінде екі түрі параметрі ретінде қарастырылады, ал F тыс int сыртқы айнымалы түрі болып табылады. Y айнымалысы туралы да айтуға болады.

Бағдарламалау стилі тұрғысынан әртүрлі айнымалылар үшін бірдей атауларды қолданбау керек, себебі шатасуы мен қателердің пайда болуы тым үлкен.

4.9. Инициализация

Біз бастамашылық туралы бірнеше рет айтқан болатынбыз, бірақ басқа да мәселелерді талқылау барысында ғана. Бұл параграфта біз әр түрлі сынып жадының бастамашылдығын анықтайтын барлық ережелерді жинақтаймыз.

Сыртқы және статикалық ауыспалылар үшін айқын инициализация болмаған жағдайда олардың нөлденуіне кепілдік беріледі; Автоматты және регистрлік ауыспаларда белгісіз бастапқы мәндері

(“қоқыс”) болады. Скалярлық айнымалыларды олардың анықтамаларында инициалдауға болады, содан кейін = белгісі және тиісті өрнек:

```
int x = 1;
char squote = '\'';
long day = 1000L * 60L * 60L * 24L; /* күн миллисекундта */
```

Сыртқы және статикалық айнымалылар үшін инициализациялық өрнектер тұрақты болуы тиіс, бұл ретте инициализация бағдарламаны орындау басталғанға дейін бір рет қана жүзеге асырылады. Автоматты және регистрлік айнымалыларды инициализациялау функцияға немесе блокқа кірген сайын орындалады. Мұндай айнымалылар үшін инициализациялық өрнек міндетті емес тұрақты. Бұл функциялардың шақыруларын қоса алғанда, бұрын белгілі бір мәндерді пайдаланатын кез келген өрнек болуы мүмкін. Мысалы, 3.3 параграфында сипатталған екілік іздеу бағдарламасында инициализацияны жазуға болады:

```
int binsearch(int x, int v[], int n)
{
    int low = 0;
    int high = n - 1;
    int mid;

ал бұлай емес:

int low, high, mid;

low = 0;
high = n - 1;
```

Шын мәнінде, автоматты айнымалыны инициализациялау – бұл меншіктеу нұсқаулығының қысқа жазбасы. Қандай жазба жақсырақ екендігі – әркімнің өз пікірі. Осы уақытқа дейін біз негізінен айқын меншіктеуді пайдаландық, өйткені хабарландыруларда инициализация аз көрінеді және айнымалыны пайдалану орнынан ары қарай сақтайды.

Массивті оны анықтау кезінде үтірмен бөлінген инициализаторлар тізімінің фигуралы жақшаның көмегімен инициалдауға болады. Мысалы, әр айдағы күн санының мәні бар days массивін инициализациялау үшін жазуға болады:

```
int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Егер массивтің көлемі көрсетілмесе, онда компилятор массивтің ұзындығын берілген инициализаторлардың саны бойынша есептейді; біздің жағдайда олардың саны 12-ге тең.

Егер инициализаторлар саны массивтің ұзындығын анықтағанда көрсетілген саннан аз болса, онда сыртқы, статикалық және автоматты айнымалылар үшін қалған элементтер нөлдік болады. Бастамашылардың тым көп санын қою қате деп саналады. Тілде инициализатордың қайталануын қоюға да, барлық алдыңғы мәндерді тапсырмай массивтің орташа элементтерін инициалдауға да мүмкіндік жоқ.

Символьді массивтерді инициализациялау – ерекше жағдай: фигуралы жақшалар мен үтіктелген конструкцияның орнына таңбалар жолын пайдалануға болады. Мысалы, мұндай жазба болуы мүмкін:

```
char pattern[] = “ould”;
```

жазбаның қысқа баламасы:

```
char pattern[] = {‘o’, ‘u’, ‘l’, ‘d’, ‘\0’};
```

Бұл жағдайда массивтің өлшемі беске тең (төрт әдеттегі символ және соңғы символ ‘\0’).

4.10. Рекурсия

С-да функцияларға рекурсивті қарауға жол беріледі, яғни функция өзіне тікелей немесе жанама түрде жүгіне алады. Сандарды таңбалар жолы түрінде басып шығаруды қарастырайық. Бұрын айтылғандай, сандар кері ретпен жасалады – кіші сандар

үлкендерден бұрын алынады және олар дұрыс ретпен басылуы керек.

Мәселені екі жолмен шешуге болады. Біріншісі-белгілі бір массивтегі сандарды есте сақтау, содан кейін оларды кері ретпен басып шығару; сондықтан бұл 3.6-параграфта қарастырылған itoa функциясында жасалды. Екінші әдіс-printd алдымен барлық ескі сандарды басып, содан кейін соңғы кіші санды басып шығаратын рекурсияны пайдалану. Бұл бағдарлама, оның алдыңғы нұсқасы сияқты, теріс сан модулі бойынша ең үлкен пайдалану дұрыс емес.

```
#include <stdio.h>

/* printd: n бүтін ондық сан ретінде басып шығарады */
void printd(int n)
{
    if (n < 0) {
        putchar('-');
        n = -n;
    }
    if (n / 10)
        printd(n / 10);
    putchar(n % 10 + '0');
}
```

Функцияның өзі рекурсивті түрде келгенде, әрбір келесі өтініш оның алдыңғы жиынтықтардан тәуелсіз Автоматты айнымалылардың жаңа толық жиынтығын алумен бірге жүреді. Мысалы, printd(123) бірінші шақыруда аргумент n = 123, екінші шақыруда printd 12, үшінші шақыруда — 1 мәнін алады. Үшінші деңгейдегі printd функциясы 1-ді басып, екінші деңгейге қайтарады, содан кейін 2-ні басып, бірінші деңгейге қайтарады. Мұнда ол 3-ті басып, жұмысын аяқтайды.

Берілген массив үшін қалған элементтерді екі ішкі жиынтыққа бөлетін бір элемент таңдалады – бұл аз және одан кем емес нәрсе. Сол тәсіл екі алынған ішкі жиынға да рекурсивті түрде қолданылады. Егер Ішкі жиында екі элементтен аз болса, онда сұрыптайтын ештеңе жоқ және рекурсия аяқталады.

Біздің жылдам сұрыптау нұсқасы, әрине, барлық мүмкіндіктердің арасында ең жылдам емес, бірақ ең оңай. Бөлгіш элемент ретінде біз ортадағы элементті қолданамыз.

```
/* qsort: сортирует v[left]...v[right] по возрастанию */
void qsort(int v[], int left, int right)
{
      int i, last;
      void swap(int v[], int i, int j);
      if (left >= right) /* егер ештеңе істемесе */
         return; /* екі элементтен аз массивінде */
      swap(v, left, (left + right)/2); /* делящий элемент */
      last = left; /* v[0] ауыстырылады */
      for(i = left+1; i <= right; i++) /* бөлімдерге бөлу */
           if (v[i] < v[left])
                swap(v, ++last, i);
      swap(v, left, last); /* бөлгіш элементті қайта қосамыз */
      qsort(v, left, last-1);
      qsort(v, last+1, right); }
```

Біздің бағдарламада ауыстыру операциясы жеке функция (swap) түрінде жасалған, себебі qsort үш рет кездеседі.

```
/* swap: орындарын ауыстыру v[i] и v[j] */
void swap(int v[], int i, int j)
{
      int temp;
      temp = v[i];
      v[i] = v[j];
      v[j] = temp;
}
```

Стандартты кітапхана кез келген түрдегі нысандарды сұрыптауға мүмкіндік беретін qsort функциясына ие.

Рекурсивті бағдарлама жадты үнемдеуді қамтамасыз етпейді, өйткені сізге бір жерде өңделетін немесе орындалатын құндылықтар жиынтығын сақтау қажет. Оның рекурсивті емес эквивалентімен салыстырғанда көбінесе қысқарақ және жазуы мен түсіну оңайырақ болып келеді. Бағдарламалардың бұл түрлері әсіресе ағаштар сияқты рекурсивті түрде анықталған мәліметтер құрылымын өңдеу

үшін пайдалы; осы тақырыпқа байланысты келтірілген мысалды 6.5-бөлімінен қараңыз.

4.12-жаттығу. Rrintd-де қолданған идеяларды қолданыңыз, itoa функциясының рекурсивті нұсқасын жазу; басқаша айтқанда, рекурсивті бағдарлама арқылы бүтін санды сандар жолына айналдырыңыз.

4.13-жаттығу. Reverse(s) функциясының рекурсивті нұсқасын жазыңыз, ол жолдың элементтерін сол жолға кері ретпен ауыстырады.

4.11. С тілінің препроцессоры

С тілінің кейбір мүмкіндіктері компиляцияның бірінші қадамында жұмыс істейтін препроцессормен қамтамасыз етіледі. Ең жиі екі мүмкіндік пайдаланылады: #includ, компиляция кезінде кейбір файлдардың кірістіру мазмұнын қамтиды және #define, бір мәтіндік тізбектерді келесіге ауыстырады. Бұл параграфта шартты компиляция және аргументтермен макро қалпына келтіру талқыланады.

4.11.1. Файлды қосу

#Include құралы #define және жарнамаларды оңай басқаруға мүмкіндік береді. Түрдің кез келген жолы

```
#include “имя-файла”
 немесе
#include <имя-файла>
```

Файл атауы бар файл мазмұнымен ауыстырылады. Егер файл аты қос тырнақшаға салынса, онда әдетте, файл бағдарламаның бастапқы файлдары арасында ізделеді; егер ондай болмаса немесе файл аты < және > бұрыштық жақшаларға салынса, онда іздеу іске асыру кезінде анықталған ережелер бойынша жүзеге асырылады. Қосылатын файлда #include жолдары болуы мүмкін.

Жиі бастапқы файлдар #define жалпы нұсқауларына сілтеме жасайтын #include жолдарынан және шығыс хабарландыруларынан

немесе <stdio.h> сияқты тақырып файлдарынан қажетті кітапхана функцияларының прототиптерінен басталады.. (Қатаң айтқанда, бұл қосылымдар міндетті түрде файлдар емес; тақырыптарға қол жеткізудің техникалық бөлшектері нақты іске асыруға байланысты.)

Құрал #include-үлкен бағдарламаны жарнамалау бірге жинаудың жақсы жолы. Ол барлық бастапқы файлдар бірдей анықтамалар мен ауыспалы хабарландыруларды қолдануға кепілдік береді, бұл әсіресе жағымсыз қателерге жол бермейді. Әрине, енгізілген файлға өзгерістер енгізгенде, оған байланысты барлық файлдар қайта жасалуы керек.

4.11.2. Макроподстановка

Анықтау макроподстановки түрі:

```
#define аты алмастырушы-мәтін
```

Макро-қалпына келтіру қарапайым ауыстыру үшін қолданылады: лексема аты бар барлық жерлерде, орнына алмастырғыш мәтін орналастырылады. #Define атаулары әдеттегі айнымалылардың аттары сияқты ережелер бойынша беріледі. Ауыстыратын мәтін еркін болуы мүмкін. Әдетте ауыстыратын мәтін #define сөзі орналасқан жолды аяқтайды, бірақ ұзын анықтамаларда оны келесі жолдарда жалғастыруға болады, әр созылатын жолдың соңында кері көлбеу сызық\. #Define-де анықталған атаудың көріну аймағы осы анықтамадан файлдың соңына дейін созылады. Макро-қалпына келтіру анықтау ерте #define-анықтамалар болуы мүмкін. Қалпына келтіру тырнақшаға жасалған мәтіндерден тыс орналасқан атауларға ғана жүзеге асырылады. Мысалы, егер yes #define арқылы анықталса, онда printf (“YES”) немесе YESMAN ішінде ешқандай қойылу болмайды.

Кез келген атауды еркін алмастырушы мәтінмен анықтауға болады. Мысалы,

```
#define forever for(;;) /* шексіз цикл */
```

шексіз цикл үшін жаңа forever сөзін анықтайды.

Макро қалпына келтіру дәлелдермен анықталуы мүмкін, соның салдарынан ауыстыратын мәтін берілген параметрлерге байланысты өзгереді. Мысалы, max келесі түрде анықтаймыз:

```
#define max(A, B) ((A) > (B) ? (A) : (B))
```

Max қызметтері әдеттегі функцияларға ұқсас болса да, олар тек мәтінді алмастырады. Әрбір формальды параметр (бұл жағдайда A және B) оған сәйкес аргументпен ауыстырылады. Осылайша, жол

```
x = max(p+q, r+s);
```

мына жолға ауыстырылады:

```
x = ((p+q) > (r+s) ? (p+q) : (r+s));
```

Аргументтер ауыстырудың кез келген түріне жол бергендіктен, көрсетілген max анықтамасы кез келген түрдегі деректер үшін қолайлы, сондықтан функциялардың көмегімен тапсырма болған жағдайда әртүрлі деректер үшін әртүрлі max жазу қажет емес.

Егер сіз max жұмысын мұқият талдасаңыз, онда кейбір суасты тастарды табыңыз. Өрнектер екі рет есептеледі және егер олар жанама әсер етсе (инкременттік операциялар немесе енгізу-шығару функцияларына байланысты), бұл жағымсыз салдарға әкелуі мүмкін. Мысалы,

```
max(i++, j++) /* О ДҰРЫС ЕМЕС */
```

I және j екі рет ұлғаяды. Сонымен қатар, қажетті есептеу тәртібін қамтамасыз ету үшін жақшалар қамқорлық қажет. Анықтау кезінде не болатынын ойлаңыз

```
#define square(x) x*x /* ДҰРЫС ЕМЕС */
```

square(z+1) шақыру.

Дегенмен, макроқұралдар өз қадір-қасиеттеріне ие. Оларды пайдаланудың практикалық мысалы - <stdio.h>-дан getchar және putchar жиі қолданылуы әрбір өңделетін таңбаға қоңырау шалу уақытын болдырмау үшін макростармен іске асырылған тармағын таңдаңыз. <Ctype.h> функциялары әдетте макростар арқылы жүзеге асырылады.

#define әрекетін #undef көмегімен жоюға болады:

```
#undef getchar
int getchar(void) {…}
```

Әдетте, бұл сол атаумен осы функцияның макроанықтауын ауыстыру үшін жасалады.

Егер тырнақшадағы жолдарда болса, формальды параметрлердің атаулары ауыстырылмайды. Алайда, егер ауыстыратын мәтінде формальды параметрдің алдында # белгісі тұрса, бұл параметр тырнақшаға жазылған аргументке ауыстырылады. Бұл түзету шығару макрос жасау үшін, мысалы, жолдардың конкатенациясы (желімдеу) біріктірілуі мүмкін:

```
#define dprint(expr) printf(#expr " = %g\n", expr)
```

Үндеу

```
dprint(x/y);
```

айналады

```
printf("x/y" " = %g\n", x/y);
```

ал екеуінің нәтижесі ретінде аламыз

```
printf("x/y = %g\n", x/y);
```

Нақты аргументтің ішінде әрбір белгі “ \-ге ауыстырылады”, ал әрқайсысы \ таңбасы \ \ - ге ауыстырылады, сондықтан қою нәтижесі дұрыс символдық тұрақтылыққа әкеледі.

Операторы макрасширлеулерде дәлелдерді анықтауға мүмкіндік береді. Егер алмастырушы мәтінде параметр # # - мен қатар тұрса, онда ол оған сәйкес аргументпен ауыстырылады, ал # операторы және оны қоршаған таңбалар-бөлгіштер шығарылады. Мысалы, paste макроанықтамасында екі аргументті анықтайды.

```
#define paste(front, back) front ## back
```

сондықтан paste (name, 1) name1 атауын жасайды.

Операторының қосымша қолдану ережелері анықталмаған; ## қатысты басқа мәліметтерді А қосымшасында табуға болады.

4.14-жаттығу. Swap(t, x, y) макрос түрінде анықтаңыз, ол x және y дәлелдері арасында көрсетілген t түрінің мәндерімен алмасуды жүзеге асырады. (блоктық құрылымды қолданыңыз.)

4.11.3. Шартты компиляция

Алдын алу барысын шартты нұсқаулардың көмегімен басқаруға болады. Олар компиляция кезінде есептелетін шарттың мәніне байланысты бағдарламаның қандай да бір мәтінін іріктеп қосу құралы болып табылады.

#If жолында көрсетілген тұрақты бүтін өрнек есептеледі. Бұл өрнек бірде-бір sizeof операторынан немесе түрге және бірде-бір enum константасынан болмауы керек. Егер ол нөлдік емес болса, онда барлық келесі жолдар қосылады #endif, немесе #elif, немесе #else. (Prepressor нұсқаулығы #elif else if сияқты.) Defined(аты) өрнегі #if 1, егер атау анықталған болса және керісінше жағдайда 0 болады.

Мысалы, hdr.h тақырып файлын қайта қосудан сақтандыру үшін оны келесідей ресімдеуге болады:

```
#if !defined(HDR)
#define HDR

/* мұнда мазмұн hdr.h */

#endif
```

hdr.h файлын бірінші рет қосқан кезде HDR аты анықталады, ал кейінгі қосуларда prepressor HDR аты қазірдің өзінде анықталғанын және бірден #endif-ке ауысатынын анықтайды. Бір файлды бірнеше рет қосудан аулақ болу керек болғанда бұл әдіс пайдалы болуы мүмкін. Егер оны жүйелі түрде қолдансаңыз, онда әрбір тақырып файлы пайдаланушының осы сабағынан босатып, оған байланысты тақырып файлдарын өзі қосады.

Міне, жүйе атын тексеру тізбегінің мысалы, қосу үшін қажетті файлды таңдауға мүмкіндік береді:

```
#if SYSTEM == SYSV
#define HDR “sysv.h”
#elif SYSTEM == BSD
#define HDR “bsd.h”
#elif SYSTEM == MSDOS
#define HDR “msdos.h”
#else
#define HDR “default.h”
#endif #include HDR
```

#Ifdef және #ifndef нұсқаулары оларға берілген атау анықталған немесе жоқ екенін тексеру үшін арнайы жасалған. Сондықтан, #if иллюстрациясы үшін жоғарыда келтірілген бірінші мысал осы түрде жазуға болады:

```
#ifndef HDR
#define HDR

/* мұнда мазмұн hdr.h */

#endif
```

5-тарау. Көрсеткіштер мен массивтер

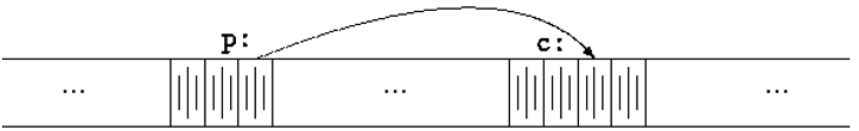
Көрсеткіш-айнымалы мекен-жайы бар айнымалы. Көрсеткіштер С-да кеңінен қолданылады, өйткені кейбір жағдайларда онсыз оңай болмайды, бірақ ішінара олармен бағдарламалар әдетте қысқа және тиімдірек. Көрсеткіштер мен массивтер бір-бірімен тығыз байланысты; осы тарауда біз бұл тәуелділікті қарастырамыз және оны қалай пайдалануға болатынын көрсетеміз.

Goto-мен қатар көрсеткіштер бір кездері аз жұмыс істейтін бағдарламаларды жазу үшін ең жақсы құрал болып жарияланды. Сондықтан ол, егер оларды ақылсыз пайдаланса. Өйткені, өте оңай көрсеткіш көрсететін нәрсе мүлдем жағымсыз. Белгілі бір пәнді сақтаған кезде көрсеткіштер арқылы айқындық пен қарапайымдылыққа қол жеткізуге болады. Біз сізді сендіруге тырысамыз.

ANSI стандартымен Енгізілген өзгерістер, негізінен, нұсқаулармен жұмыс істеу үшін нақты ережелерді тұжырымдауға байланысты. Стандарт бағдарламашылардың жинақталған оң тәжірибесін және компиляторларды әзірлеушілердің сәтті жаңалықтарын заңдастырды. Сонымен қатар, char * орнына жалпылама көрсеткіш түрі ретінде void * түрі ұсынылады (void-дегі көрсеткіш).

5.1. Көрсеткіштер мен мекенжайлар

Жадты ұйымдастырудың жеңілдетілген схемасын қарастырайық. Типтік машинаның жадысы жүйелі түрде нөмірленген немесе бағытталған ұяшықтардың массиві болып табылады, олар жеке немесе байланысқан бөліктермен жұмыс істеуге болады. Кез келген машинаға қатысты келесі тұжырымдар дұрыс: бір байт char түрінің мәнін сақтай алады, екі биттік ұяшықтар short типті тұтас ретінде, ал төрт биттік ұяшықтар – толық ұзын түрі ретінде қарастырылуы мүмкін. Көрсеткіш – бұл адрес сақталатын ұяшықтар тобы (әдетте, екі немесе төрт). Осылайша, егер c char түрі болса, және P-көрсеткіш c, онда жағдай келесідей:



Бірыңғай оператор & объектінің мекенжайын береді, сондықтан нұсқаулық

```
p = &c;
```

айнымалы p ұяшықтың мекен-жайын тағайындайды (p с көрсетеді деп айтады). Оператор & жадта орналасқан нысандарға ғана қолданылады: айнымалы және массив элементтеріне. Оның операндом өрнек де, тұрақты да, регистрлік айнымалы де бола алмайды.

Бірыңғай оператор * жанама қол жеткізу операторы бар. Көрсеткішке қолданылған, ол осы көрсеткіш көрсететін объектіні береді. x және y int түрі бар деп есептейік, int бойынша ір-көрсеткіш. Келесі бірнеше жолдар көрсеткіштер қалай жарияланатынын және операторлар & және *қалай пайдаланылатынын көрсету үшін арнайы ойластырылған.

```
int x = 1, y = 2, z[10];
int *ip; /* ip - int көрсеткіш */
ip = &x; /* енді ip x көрсетеді */
y = *ip; /* енді y 1-ге тең */
*ip =0; /* x енді 0-ге тең*/
ip = &z[0]; /* ip енді z [0] нұсқайды*/
```

Жариялау x, y және z бізге бұрыннан таныс. ip жариялауының индикациясы

```
int *ip;
```

біз оны мнемоникалық етіп жасауға тырыстық - ол былай дейді: «өрнек *ip int түрі бар». Айнымалыны жарнамалау синтаксисі бұл айнымалыны кездестіретін өрнектер синтаксисіне «бейімделеді». Көрсетілген қағидат функциялардың хабарландыруларында да қолданылады. Мысалы, жазу

```
double *dp, atof (char *);
```

*dp және atof(s) өрнектері double түрі бар, ал atof функциясының аргументі char көрсеткіші бар.

Сіз, бәлкім, көрсеткіш тек белгілі бір түрдегі нысандарды көрсетуге рұқсат етілетінін байқадыңыз. (Бір ерекшелік бар: «void-дегі көрсеткіш» кез келген түрдегі нысандарды көрсете алады, бірақ мұндай көрсеткішке жанама қол жеткізу операторын қолдануға болмайды. Біз 5.11 параграфында бұған ораламыз).

Егер ip х бүтін түрін көрсетсе ,онда * ip қолдануға болатын кез келген жерде пайдалануға болады; мысалы,

```
*ip = *ip+ 10;
```

*ip 10-ға арттырады.

Бірыңғай операторлар * және & қосу, сондықтан арифметикалық операторлар қарағанда жоғары басымдыққа ие

```
y = *ip + 1
```

ip көрсетеді және оған 1 қосады, ал нәтиже у айнымалысын береді.

```
*ip += 1
```

және

```
(*ip)++
```

Жақшаның соңғы жазбасында қажет, өйткені егер олар болмаса, көрсеткіштің мәні артады, ол нені көрсетпейді. Бұл бірыңғай операторлар * және ++ бірдей басымдыққа ие және орындау тәртібі – оңнан солға.

Ақыр соңында, көрсеткіштер өздері айнымалы болғандықтан, мәтінде олар жанама қатынау операторынсыз да кездеседі. Мысалы, егер iq int-да басқа көрсеткіш болса, онда

```
iq = ip
```

ip және iq бірдей нысанды көрсету үшін ip мазмұнын iq-ге көшіреді.

5.2. Көрсеткіштер мен функциялар дәлелдері

С-дағы функциялар параметр мәндерін олардың дәлелдері ретінде қабылдайтын болғандықтан, шақырылған функцияның өзгермелі өзгеруінің тікелей мүмкіндігі болмайды. Сұрыптау бағдарламасында бізге екі реттелмеген элементті алмастыратын swap функциясы қажет болды. Алайда, жазу жеткіліксіз

```
swap(a, b);
```

мұнда swap функциясы келесідей анықталады:

```
void swap(int x, int y) /* ДҰРЫС ЕМЕС */
{
      int temp;
      temp = x;
      x = y;
      y = temp;
}
```

Swap тек a және b айнымалыларының көшірмелерін алады, ол оған жүгінген бағдарламаның A және b айнымалыларына әсер ете алмайды.

Қажетті әсер алу үшін, шақыратын бағдарлама көрсеткіштерді өзгертілуі тиіс мәндерге беру керек:

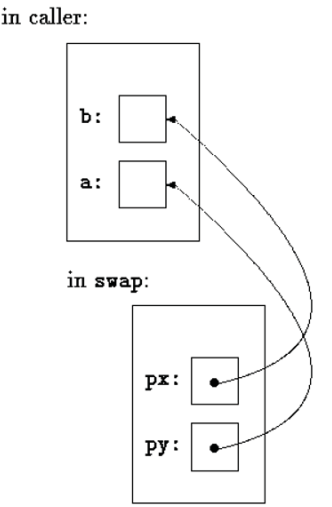
```
swap(&a, &b);
```

Оператор & айнымалы мекен-жайын алады, өйткені, &a көрсеткіш a бар. Сол swap функциясында параметрлер көрсеткіштер ретінде жариялануы тиіс, сонымен қатар параметрлер мәндеріне қол жеткізу жанама жүзеге асырылады.

```
void swap(int *px, int *py) /* қайта орнату *px және * py */
{
        int temp;
        temp = *px;
        *px = *py;
        *py = temp;
}
```

Графикалық түрде бұл келесідей:

шақырушы бағдарламада:



Аргументтер-көрсеткіштер оны тудыратын нысандарға кіруге мүмкіндік береді және бұл нысандарды өзгертуге мүмкіндік береді. Мысалы, getint функциясын қарастырайық, ол бір бүтін санның еркін пішімінде енгізуді және оны мәтіндік көріністен int түріне аударуды жүзеге асырады. Getint кіріс ағыны таусылған жағдайда, алынған санның мәнін қайтаруы немесе файлдың соңы туралы EOF мәнімен сигнал беруі керек. Бұл мәндер әртүрлі арналар арқылы қайтарылуы тиіс, өйткені аударма нәтижесінде алынған сан EOF-мен ешқашан сәйкес келмейді деп есептеуге болмайды.

Шешімдердің бірі – getint файл күйінің сипаттамасын нәтиже ретінде береді (таусылған немесе таусылмаған), ал санның мәні дәлел ретінде берілген көрсеткішке сәйкес орналастырылады. Ұқсас схема scanf бағдарламасында да әрекет етеді, оны біз 7.4-параграфта қарастырамыз.

Төменде көрсетілген цикл getint көмегімен алынған бүтін сандармен кейбір массивті толтырады.

```
int n, array[SIZE], getint (int *);
for (n = 0; n < SIZE && getint (&array[n]) != EOF; n++)
     ;
```

Getint-ке әрбір кезекті өтініштің нәтижесі array[n] - ге жіберіледі және n бірлікке артады. Getint функциялары array[n] элементінің мекен-жайы беріледі. Егер мұны істемесе, getint шақырушы бағдарламаға аударылған бүтін санды қайтарудың жолы болмайды.

```
#include <ctype.h>

int getch (void);

void ungetch (int);
/* getint: * pn енгізуден келесі бүтін оқиды */
int getint(int *pn)
{
     int c, sign;
     while (isspace(c = getch()))
          ; /* бөлгіш таңбаларды өткізу */
      if (!isdigit(c) && c != EOF && c ! = '+' && c != '-' ) {
          ungetch (c); /* сан емес */
          return 0;
       }
       sign = (c = = ' - ' ) ? -1 : 1;
       if (c == '+' || c == '-')
                    c = getch();
       for (*pn = 0; isdigit(c); c = getch())
            *pn = 10 * *pn + (c - '0' ) ;
       *pn *= sign;
        if (c != EOF)
```

```
            ungetch(c);
        return c;
}
```

getint-те барлық жерде *pn деп int типті әдеттегі айнымалыны біл-діреді. Ungetch функциясы getch-пен бірге (4.3-параграф) қосым-ша оқылған таңбаны кері жіберу мүмкіндігін қамтамасыз ету үшін бағдарламаға қосылған.

5.1-жаттығу. getint функциясы - немесе +, одан кейін сан бол-майды, ол нөлдің “дұрыс” көрінісі ретінде түсінеді. Бағдарламаны осындай жағдайларда оқылған белгіні қайта енгізуге қайтаратын-дай етіп түзетіңіз.

5.2-жаттығу. getfloat — getint аналогы функциясын өзгермелі нүктелі сандар үшін жазыңыз. getfloat функциясы беретін нәтиже мәні қандай болады?

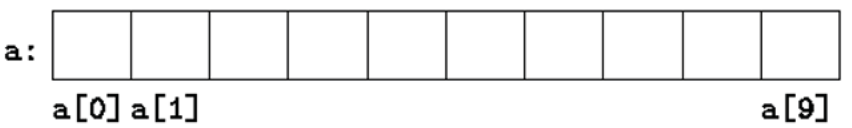
5.3. Көрсеткіштер мен массивтер

С-да көрсеткіштер мен массивтер арасындағы байланыс бар және бұл байланыс соншалықты тығыз, бұл құралдарды бірге қарастыру жақсы. Индекстеу операциясы жүзеге асыратын массив элементіне кез келген қол жеткізу көрсеткішпен орындалуы мүмкін. Нұсқағыштармен нұсқасы жалпы жылдам жұмыс істейді, бірақ оны түсіну қиындықтар туғызуы мүмкін.

Жариялау

```
int a[10];
```

10 өлшемді А массивін анықтайды, яғни 10 тізбекті объектіден тұратын блок а[0], ал[1],..., ал[9].



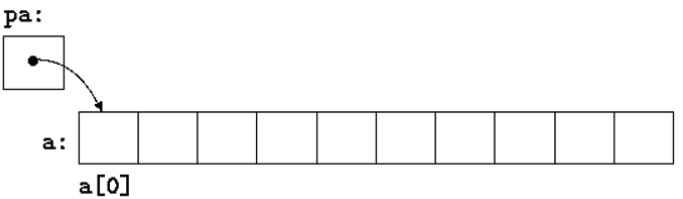
a[i] жазбасы бізді массивтің i элементіне жібереді. Егер pa int көрсеткіші болса, яғни

```
int *pa;
```

меншіктеу нәтижесінде

```
pa = &a[0];
```

pa a нөлдік элементін көрсетеді, басқаша айтқанда, pa a элементінің мекен-жайын қамтиды.



Енді меншіктеу өрнегі

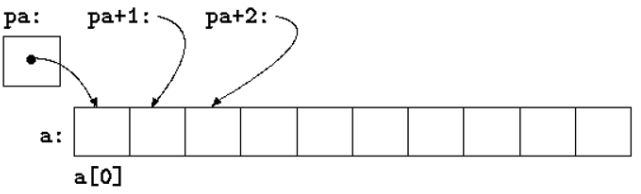
```
x = *pa;
```

a[0] мазмұнын x ішінде көшіретін болады.

Егер pa массивтің кейбір элементін көрсетсе, онда pa+1 анықтау бойынша келесі элементті көрсетеді: pa+i — pa-дан кейінгі i элементке, а pa-i — pa-дан кейінгі i элементке. Осылайша, егер pa a[0] көрсетсе, онда

```
*(pa+1)
```

мазмұн бар a [1], a+i-адрес a[i], a *(pa+i) - мазмұн a[i].



«Көрсеткішке 1 қосу «деген сөздердің мағынасы, көрсеткіштермен кез келген арифметиканың мағынасы сияқты, ра+1 келесі объектіге, ал ра+1 — ра-дан кейін 1-ге көрсету болып табылады.

Көрсеткіштермен индекстеу мен арифметика арасында өте тығыз байланыс бар. Айнымалының немесе массив түрінің өрнегінің мәнін анықтау бойынша массивтің нөлдік элементінің мекен-жайы бар. Меншіктеуден кейін

```
pa = &a[0];
```

ра және а бірдей мәнге ие. Массив атауы оның бастапқы элементінің орналасуының синонимі болғандықтан, ра=&а[0] меншіктеуін келесі түрде жазуға болады:

```
pa = a;
```

Одан да таңқаларлығы (бір қарағанда) а[і]-ды *(а+і) деп жазуға болады. а[і] есептей отырып, С оны бірден *(а+і) деп түрлендіреді; жазбаның көрсетілген екі нысаны эквивалентті. Сонымен, & және + і операторын қолдану нәтижесінде алынған жазбалар және олар эквивалентті болады, яғни екі жағдайда да бұл а-дан кейінгі 1-элементтің адресі болады. Екінші жағынан, егер ра-көрсеткіш болса, онда оны индекспен пайдалануға болады, яғни ра[і] жазбасы *(ра+і) жазбаға тең. Қысқаша айтқанда, массивтің элементін ығыстырылған көрсеткіш түрінде де, индексті массивтің атауы түрінде де бейнелеуге болады.

Массив атауы мен массив атының рөлін атқаратын көрсеткіш арасында бір айырмашылық бар. Көрсеткіш айнымалы, сондықтан сіз ра=а немесе ра++жазуға болады. Бірақ массив атауы айнымалы емес, және а=ра немесе а++ сияқты жазбалар рұқсат етілмейді.

Егер массив атауы функцияға берілсе, соңғысы аргумент ретінде оның бастапқы элементінің мекен-жайын алады. Шақырылатын функцияның ішінде бұл аргумент мекен-жайы бар жергілікті айнымалы болып табылады. Біз белгіленген фактіні пайдаланып, жол-

дың ұзындығын есептейтін strlen функциясының тағы бір нұсқасын жаза аламыз.

```
/* strlen: жолдың ұзындығын қайтарады */
int strlen(char *s)
{
    int n;
    for (n = 0; *s != '\0' ; s++)
        n++;
    return n;
}
```

S айнымалы көрсеткіш болғандықтан, оған ++ операциясы қолданылады; s ++ функциясы strlen қатынасқан функцияның таңбалар жолына әсер етпейді. Strlen функциясын жеке пайдалануға арналған көрсеткіштің бірнеше көшірмесін 1-ге көбейтеді. Бұл барлық қоңыраулар, мысалы:

```
strlen(«Сәлем, әлем»); /* жол тұрақтысы */
strlen(array); /* char array[100]; */
strlen(ptr); /* char *ptr; */
```

дұрыс екенін білдіреді.

Формальды параметрлер

```
char s[];
```

және

```
char *s;
```

функцияны анықтауда эквивалентті. Біз соңғы нұсқаға артықшылық береміз, себебі ол s көрсеткіші бар екенін анық көрсетеді. Егер функциялар аргумент ретінде беріледі аты массив, онда ол оны ыңғайлы – не ретінде қарастыра алады массив атауы, немесе Көрсеткіш ретінде және тиісінше, онымен келеді. Ол тіпті орынды және түсінікті көрінсе, жазбаның екі түрін де пайдалана алады.

Функциялар массивтің бір бөлігін беруге болады, бұл үшін дәлел массивтің басына көрсетілуі керек. Мысалы, егер а-массив болса, онда жазбаларда

```
f(&a[2])
```

немесе

```
f(a+2)
```

f функциялары а элементінен басталатын ішкі массив мекен-жайы беріледі. F функциясының ішінде параметрлерді сипаттау төмендегідей көрінуі мүмкін

```
f(int arr[]) {…}
```

немесе

```
f(int *arr) {…}
```

Демек, f үшін, бұл параметр бүкіл массив емес, массивтің бөлігін көрсетеді, маңызды емес.

Егер массивтің элементтері бар деген сенім болса, онда нөлдік элементке қатысты «кері» жағына индекстеу мүмкін; өрнектер p[-1], p[-2] және т.б. тіл синтаксисіне қайшы келмейді және тікелей p[0] алдында тұрған элементтерге айналады. Әрине, массивтің шекарасынан «шығуға» және сол арқылы жоқ нысандарға жүгінуге болмайды.

5.4. Адрестік арифметика

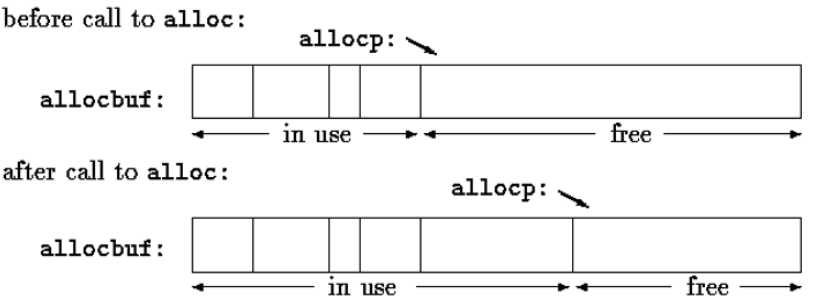
Егер p массивтің кейбір элементтеріне көрсеткіш болса, онда p++ p келесі элементке көрсететіндей арттырады, ал p += i оны бұрын көрсеткеннен кейін i элементке көрсететіндей арттырады. Бұл және оған ұқсас құрылымдар — адрестік арифметика деп аталатын көрсеткіштерден арифметиканың ең қарапайым мысалдары.

С-адрестік арифметикаға жақындауда және біркелкі. Бұл көрсеткіштердің, массивтердің және адрестік арифметиканың бір тіліндегі қосылысы – оның күшті жақтарының бірі. Екі бағдарламадан тұратын қарапайым жадты таратқыштың құрылысымен баяндалады. Бірінші, alloc(n), char типті тізбектелген ұяшықтардың n көрсеткішін қайтарады; alloc-қа қарайтын бағдарлама бұл ұяшықтар символдарды есте сақтау үшін пайдаланылуы мүмкін. Екінші, afree(p), оны қайта кәдеге жарату үшін жадты босатады. Алгоритмнің қарапайымдылығы afree-ге жүгінулер alloc-қа тиісті жүгінулерге қатысты кері тәртіпте жасалады деген болжаммен негізделген. Осылайша, alloc және afree жұмыс істейтін жады-бұл стек («соңғы кірді, бірінші кетті» принципі бар тізім). Стандартты кітапханада malloc және free функциялары бар, олар тек аталған шектеулерсіз бірдей болады; 8.7-параграфта біз олардың қалай көрінетінін көрсетеміз.

Alloc функциясын жүзеге асыру оңай, егер ол allocbuf деп атайтын char типті үлкен массивтің бөліктерін береді деп ойласаңыз. Бұл массив alloc және afree функцияларын жеке пайдалануға беріледі. Олар массивтің индекстерімен емес, көрсеткіштермен айналысатындықтан, басқа бағдарламалар оның атын білудің қажеті жоқ. Сонымен қатар, бұл массивті alloc және afree сияқты бастапқы файлда анықтауға болады, бұл файлдан тыс көрінбейді. Іс жүзінде, мұндай массив аты жоқ болуы мүмкін, өйткені ол операциялық жүйеден malloc көмегімен сұрауға және кейбір атаусыз жад блогына көрсеткіш алуға болады.

Әрине, allocbuf массивінің қанша элементтері бар екенін білу керек. Біз allocp көрсеткішін енгіземіз, ол бірінші бос элементті көрсетеді. Егер n үшін жад сұралса alloc alloc ағымдағы allocp мәнін (яғни бос блоктың басталу мекен-жайы) қайтарады, содан кейін allocp көрсеткіші келесі бос аймақты көрсету үшін оны n-ға көбейтеді. Егер кеңістік жоқ болса, онда alloc нөл береді. afree[p] функциясы allocp-ді allocbuf массивінен тыс шықпаса ғана p мәніне орнатады.

alloc шақырудан бұрын:
alloc шақыруынан кейін:




```
#define ALLOCSIZE 10000 /* қол жетімді кеңістіктің өлшемі */

static char allocbuf[ALLOCSIZE]; /* alloc үшін жады */
static char *allocp = allocbuf; /* бос орынға нұсқау */

char *alloc(int n) /* меңзерді n таңбаға қайтарады */
{
                        if (allocbuf + ALLOCSIZE - allocp >= n) {
                            allocp += n; /* кеңістік бар */
                            return allocp - n; /* ескі p */
                        } else /* пространства нет */
                            return 0;
}

void afree(char *p) /* p көрсететін жадты босатады */
{
      if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
          allocp = p;
}
```

Жалпы жағдайда, көрсеткіш, кез келген басқа айнымалы сияқты, инициализациялауға болады, бірақ ол үшін нөл немесе өрнек сияқты мағыналармен, тиісті түрдегі бұрын анықталған деректердің мекен-жайына әкеледі. Жариялау

```
static char *allocp = allocbuf;
```

allocp char көрсеткішін анықтайды және allocbuf массивінің мекен-жайын инициалдайды, өйткені бағдарлама басталар алдында

allocbuf массиві бос. Көрсетілген хабарландырудың осындай түрі болуы мүмкін:

```
static char *allocp = &allocbuf[0];
```

массив атауы және оның нөлдік элементінің мекен-жайы болғандықтан.

Тексеру

```
if (allocbuf + ALLOCSIZE - allocp >= n) { /* жарайды */
```

N таңбаларға сұранысты қанағаттандыру үшін жеткілікті кеңістіктің болуын бақылайды. Егер жад жеткілікті болса, allocp үшін жаңа мән allocbuf соңғы элементінің келесі позициясына қарағанда көрсетілмеуі керек. Осы талапты орындау кезінде alloc таңбалар блогының басына көрсеткіш береді (функцияның түрін жариялауға назар аударыңыз). Егер талап орындалмаса, alloc функциясы жадтың жетіспейтіні туралы сигнал беруі керек. С нөлдің деректер үшін ешқашан дұрыс мекен-жай болмайтынына кепілдік береді, сондықтан біздің жадымыз жетіспеген жағдайда оны апат белгісі ретінде пайдаланамыз.

Көрсеткіштер және тұтас бір-бірімен алмастырылатын объектілер болып табылмайды.Тұрақты нөл – бұл ережеден жалғыз ерекшелік: оны көрсеткішке тағайындауға болады және көрсеткіш нөлдік тұрақтымен салыстыруға болады. Нөл – бұл көрсеткіш үшін арнайы мән екенін көрсету үшін нөл санының орнына, әдетте, <stdio. h> файлында анықталған NULL – константты жазады. Осы сәттен бастап біз оны қолданамыз.

Тексеріс

```
if (allocbuf + ALLOCSIZE - allocp >= n) { /* жарайды */
```

және

```
if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
```

көрсеткіштер бар арифметиканың бірнеше маңызды қасиеттерін көрсетеді. Біріншіден, кейбір ережелерді сақтаған кезде көрсеткіштерді салыстыруға болады.

Егер p және q бір массивтің элементтерін көрсетсе, онда оларға қатынас операторлары == , != ,<, > = және т. б.қолдануға болады.

```
p < q
```

шын мәнінде, егер p q қарағанда массивтің ерте элементін көрсетсе. Бірақ бір массивтің элементтерін көрсетпейтін көрсеткіштер үшін арифметикалық операциялардың немесе салыстырулардың нәтижесі анықталмаған. (Бір ерекшелік бар: көрсеткіштер арифметикасында жоқ “массивінен кейінгі” элементтің мекенжайын, яғни массивке тағы бір элементті қоссаңыз, соңғы болатын сол “элементтің” мекенжайын пайдалануға болады).

Екіншіден, сіз байқаған шығарсыз, көрсеткіштер мен бүтіндерді қосуға және шегеруге болады. Құрылым

```
p + n
```

бұл p-ны көрсететін объект түріне сай келмейтін объектіден кейін n-ші орын алатын объектінің мекен-жайын білдіреді; n объект мөлшеріне сәйкес келетін коэффициентке автоматты түрде көбейтіледі. Егер, мысалы, int төрт байтты алса, онда көбейту коэффициенті төртке тең болады.

Сондай-ақ көрсеткіштерді шегеруге жол беріледі. Мысалы, егер p және q бір массивтің және p<q элементтерін көрсетсе, онда q-p+1 p-дан q-ға дейінгі элементтердің саны бар. Бұл факт strlen тағы бір нұсқасын жазу кезінде пайдалануға болады:

```
/* strlen: s жолдың ұзындығын қайтарады */
int strlen(char *s)
```

```
{
    char *p = s;
    while (*p != '\0' )
        p++;
    return p - s;
}
```

Өз хабарландыруында p s мәнімен инициаланады, яғни басында p жолдың бірінші символын көрсетеді. While циклінің әрбір қадамында келесі таңба тексеріледі; цикл '\0' кездескенше жалғасады. Келесі символға p көрсеткішінің әрбір жылжуы p++ нұсқауымен орындалады және ps айырмашылығы өткен символдардың санын береді, яғни жолдың ұзындығын. (Жолда таңбалар саны int айнымалы түрінде оны сақтау үшін тым үлкен болуы мүмкін. ptrdiff_t түрі, екі көрсеткіштің айырмашылығын (белгісі бар) сақтау үшін жеткілікті, <stddef.h> файлында анықталған. Алайда, өте абай болған жөн, біз қайтарылатын нәтиже үшін size_t түрін қолдануымыз керек, бұл жағдайда біздің бағдарлама стандартты кітапханалық нұсқаға сәйкес келеді. size_t түрі sizeof операторымен қайтарылатын толық түрі бар.)

Көрсеткіштермен Арифметика түрін ескереді: егер ол char-дан көп жадты алатын float мәндерімен айналысса және float-қа p-көрсеткіш болса, онда p++ p-ны келесі float мәніне жылжытады. Бұл char емес, float типті элементтермен айналысатын alloc-тың басқа нұсқасы alloc және afree барлық char-да float-те қарапайым алмастыруға болады. Көрсеткіштермен барлық операциялар көрсеткіштер көрсететін нысандардың өлшеміне сәйкес автоматты түрде түзетіледі.

Көрсеткішпен келесі операцияларды жүргізуге болады: көрсеткіш мәнін сол типтегі басқа көрсеткішке меншіктеу, көрсеткіш пен бүтіндікті қосу және азайту, бір массивтің элементтеріне көрсететін екі көрсеткішті шегеру және салыстыру, сондай-ақ көрсеткішке нөлді беру және көрсеткішті нөлге салыстыру. Көрсеткіштермен басқа операцияларды жүргізуге жол берілмейді. Екі көрсеткішті қосуға, оларды көбейтуге, бөлуге, жылжытуға, разрядтарды ерек-

шелеуге болмайды; көрсеткіш float немесе double типті мәнмен бүктеуге болмайды; бір типті көрсеткішке алдын ала келтіру операциясын орындамай, тіпті басқа типті көрсеткішті беруге болмайды (тек void* типті көрсеткіштер ғана емес).

5.5. Функцияның символдық көрсеткіштері

Төмендегі түрде жазылған жол тұрақтысында

“Мен жол”

таңбалар массиві бар. Ішкі көріністе бұл массив ‘\0’ нөлдік белгісімен аяқталады, ол арқылы бағдарлама жолдың ұшын таба алады. Бос емес жад ұяшықтарының саны қос тырнақшалар арасында орналастырылған таңбалар санынан біреуден көп.

Көбінесе жол тұрақтылары функциялардың аргументтері ретінде қолданылады, мысалы,

```
printf (“сәлем, әлем\n”);
```

Мұндай символдық жол бағдарламада пайда болған кезде, оған кіру символдық көрсеткіш арқылы жүзеге асырылады; printf таңбалар массивінің басына көрсеткіш алады. Дәлірек айтқанда, жол тұрақтылығына қол жеткізу оның бірінші элементіне көрсеткіш арқылы жүзеге асырылады.

Жол тұрақтылары функциялар аргументтері ретінде ғана емес қажет. Мысалы, pmessage айнымалысы ретінде жарияланса

```
char *pmessage
```

онда меншіктеу

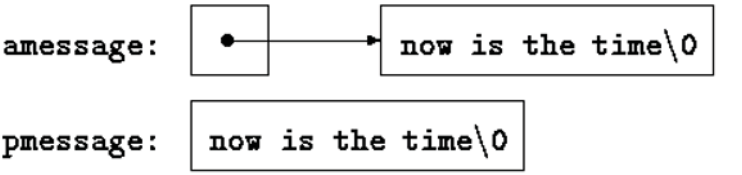
```
pmessage = “now is the time”;
```

оған көрсеткіш символдық массивке орналастырады, бұл ретте жолдың өзі көшірілмейді, тек көрсеткіш оған көшіріледі. С-да бір тұтас жолмен жұмыс істеуге арналған операциялар қарастырылмаған.

Келесі анықтамалар арасында маңызды айырмашылық бар:

```
char amessage[] = “now is the time”; /* массив */
char *pmessage = “now is the time”; /* нұсқау */
```

amessage – бұл көрсетілген таңбалар тізбегі және ‘\0’ орналастырылған көлемі бар массив. Жиым ішіндегі жеке таңбалар өзгеруі мүмкін, бірақ amessage әрқашан бір жад орнын көрсетеді. Керісінше, оған pmessage жол тұрақтысын көрсету үшін инициалданған көрсеткіш бар. Көрсеткіш мәнін өзгертуге болады, содан кейін соңғысы басқа нәрсені көрсетеді. Сонымен қатар, тұрақты мазмұнды өзгертуге тырыссаңыз, нәтиже белгісіз болады.



Біз стандартты кітапханадан алған екі пайдалы бағдарламаның бірнеше өзгертілген нұсқаларында көрсеткіштер мен массивтерге қатысты қосымша ойларды суреттейміз. Олардың біріншісі, strcpy (s, t) функциясы t жолын s жолына көшіреді. Мен s = t-ді тікелей жазғым келеді, бірақ мұндай оператор таңбаларды көшірмейді. Таңбаларды көшіру үшін цикл құру керек. Массивті қолдана отырып, strcpy-дің бірінші нұсқасы келесі формада болады:

```
/* strcpy: t-ны s-ға көшіреді; индекстелетін массив нұсқасы */
void strcpy(char *s, char *t)
{
      int i;
      i = 0;
     while ((s[i] = t[i]) != ‘\0’ )
            i++;
}
```

Салыстыру үшін strcpy көрсеткіштермен мысал келтіреміз:

```
/* strcpy: t-ны s-ға көшіреді: 1 нұсқасы (көрсеткіштермен)*/
void strcpy(char *s, char *t)
{
      while ((*s = *t) != '\0' ) {
            s++;
            t++;
       }
}
```

Тек аргументтер мәндерінің көшірмелері берілгендіктен, strcpy s және t параметрлерін жергілікті айнымалы ретінде еркін пайдалана алады. Олар көшірілетін t жолында '\0' кездескенге дейін әр массивтердің әрқайсысында келесі таңбаға әр жолы жылжитын көрсеткіштер бойынша тиісті түрде инициалданған.

Іс жүзінде strcpy осылай жазылмайды. Тәжірибелі бағдарламашы қысқа жазуды таңдайды:

```
/* strcpy: t-ны s-ға көшіреді; 2 нұсқасы (көрсеткіштермен)*/
void strcpy(char *s, char *t)
{
      while ((*s++ = *t++) != '\0')
            ;
}
```

Мұнда s және t өсуі циклдің басқару бөлігінде жүзеге асырылады. *t++ мәні-бұл мәні ұлғайғанға дейін t айнымалысын көрсететін символ; постфикстік оператор ++ ол көрсеткен таңба алынғанша t көрсеткішін өзгертпейді. s -ға қатысты да солай: алдымен символ s ескі мәнін көрсететін позицияда есте қалады және содан кейін ғана s айнымалы мәні артады. Жіберілетін таңба бір уақытта '\0' - мен салыстырылатын мән болып табылады. Нәтижесінде барлық таңбалар, соның ішінде '\0' соңғы таңбаны көшіреді.

'\0' - мен салыстыруды байқасаңыз, мұнда артық (С-да өрнектің нөл емес мәні оның шынайылығы ретінде түсіндіріледі), біз бағдарламаның мәтінінің тағы бір және соңғы қысқаруын жасай аламыз:

```
/* strcpy: t-ны s-ға көшіреді; 3 нұсқасы (көрсеткіштермен)*/
```

```
void strcpy(char *s, char *t)
{
      while ((*s++ = *t++) != '\0')
            ;
}
```

Бір қарағанда, біз алған нәрсе жұмбақ болып көрінсе де, осындай жазба әлдеқайда ыңғайлы және оны меңгеру керек, өйткені С-бағдарламаларда сіз онымен жиі кездесетін боласыз.

Стандартты кітапханадан strcpy функциясына қатысты <string.h>, онда ол өзінің нәтижесі ретінде жолдың жаңа көшірмесіне меңзерді қайтарады.

Мұнда қарастыратын екінші бағдарлама strcmp(s, t). Ол s және t жолдарының таңбаларын салыстырады және егер s жолы тиісінше лексикографиялық аз, тең немесе одан көп болса, теріс, нөлдік немесе оң мәнді қайтарады.

```
/* strcmp: < 0 мәнін береді s < t, 0 мәні s == t, > 0 мәні s > t */
int strcmp(char *s, char *t)
{
     int i;
     for (i = 0; s[i] == t[i]; i++)
          if (s[i] == '\0')
               return 0;
     return s[i] - t[i];
}
```

Көрсеткіштерді пайдалана отырып, сол бағдарлама:

```
/* strcmp: < 0 мәнін береді s < t, 0 мәні s == t, > 0 мәні s > t */
int strcmp(char *s, char *t)
{
     for ( ; *s == *t; s++, t++)
          if (*s == '\o')
               return 0;
     return *s - *t;
}
```

Өйткені операторлары ++ және -- префиксті немесе постфиксті болуы мүмкін болғандықтан, басқа да оларды біріктіру * операторы *кездеседі. Мысалы:

```
*--p
```

бұл көрсеткіш бойынша таңба алынбас бұрын p-ны азайтады. Мысалы, келесі екі өрнек:

```
*p++ = val; /* val стекке орналастыру */
val = *--p; /* стектен таңбаны алып, val орналастыру*/
```

стекке жіберу және стектен алу үшін стандартты болып табылады (4.3-параграфты қараңыз.).

Осы параграфта айтылған функцияларды, сондай-ақ жолдармен жұмыс істейтін басқа да стандартты функцияларды жариялау тақырып файлында <string.h> бар.

5.3-жаттығу. Көрсеткіштерді пайдалана отырып, 2-тарауда қарастырылатын strcat функциясын жазыңыз (strcat(s, t) функциясы t жолын s жолының соңына көшіреді).

5.4-жаттығу. Strend(s, t) функциясын жазыңыз, ол 1-ді береді, егер t жолы s жолының соңында орналасқан болса немесе керісінше жағдайла нөл.

5.5-жаттығу. Strncpy, strncat және strncmp кітапханалық функциялардың нұсқаларын жазыңыз, олардың саны n-дан аспайтын аргументтерінің бірінші Символдарымен жұмыс істейді. мысалы, strncpy (t, s, n) s-дегі t таңбаларының n-ден артық емес көшіреді.

5.6-жаттығу. Алдыңғы тараулар мен жаттығулардан тиісті бағдарламаларды алып, индекстеудің орнына көрсеткіштерді пайдаланып оларды қайта жазыңыз. Getline (1 және 4-тараулар), atoi, itoa бағдарламалары және олардың нұсқалары (2, 3 және 4-тараулар), reverse (3-тарау), сондай-ақ strindex және getop (4-тарау).

5.6 Көрсеткіштер массивтері; Көрсеткіштер көрсеткіштері

Кез келген басқа айнымалылар сияқты, көрсеткіштерді массивтерге топтастыруға болады. Оны көрсету үшін әліпбилік ретпен мәтіндік жолдарды сұрыптайтын бағдарламаны жазамыз; бұл UNIX жүйесінің sort бағдарламасының жеңілдетілген нұсқасы болады.

3-тарауда біз тұтас массивін реттейтін Шелл бойынша сұрыптау функциясын алып келдік, ал 4-тарауда оны жақсартып, шапшаңдықты арттырдық. Сол алгоритмдер мұнда да қолданылады, бірақ қазір олар әр түрлі ұзындығы мен салыстыруы мүмкін мәтіндік жолдарды өңдейді немесе оларды бір операцияда орындау мүмкін емес. Бізге еркін ұзындықтағы мәтіндік жолдармен ыңғайлы және тиімді жұмыс істеуге мүмкіндік беретін кейбір деректер көрінісін таңдау қажет.

Бұл үшін жолдар басындағы көрсеткіштер массивін қолданамыз. Жадтағы жолдар бір-біріне жақын орналасқандықтан, әрбір жеке жолға оның бірінші символына көрсеткіш арқылы қол жеткізу оңай. Көрсеткіштердің өздері массив түрінде ұйымдастыруға болады. Екі жолды салыстыру мүмкіндіктердің бірі-strcmp функцияларына көрсеткіштерді беру. Жолдың орындарын өзгерту үшін олардың көрсеткіштері (жолдардың өздері емес) жиымындағы орындарды ауыстыру жеткілікті болады.

Мұнда бірден екі мәселе түсіріледі: біреуі — жадыны басқарудың күрделілігіне байланысты, екіншісі – жолдардың өздерін ауыстырған кезде үлкен үстеме шығындармен.

Сұрыптау процесі үш кезеңге бөлінеді:
енгізуден барлық жолдарды оқу

енгізілген жолдарды сұрыптау
оларды ретімен басып шығару

Әдеттегідей, біз мәселенің табиғи бөлінуіне сәйкес келетін функцияларды таңдаймыз және осы функцияларды басқаратын негізгі бағдарламаны жазамыз. Сұрыптау кезеңін іске асыруды біраз уақытқа кейінге қалдырамыз және мәліметтер құрылымы мен енгізу-шығаруға назар аударамыз.

Енгізу бағдарламасы барлық жолдардың таңбаларын оқу және есте сақтау, сондай-ақ жолдардағы көрсеткіштер массивін салу керек. Сонымен қатар, енгізілген жолдардың санын есептеу керек — бұл ақпарат сұрыптау және басып шығару үшін қажет. Кіріс функциясы тек жолдардың соңғы санымен ғана жұмыс істей алады, егер олар тым көп енгізілсе, ол белгілі бір мәнді береді, ол ешқашан жолдар санына сәйкес келмейді, мысалы -1.

Шығару бағдарламасы тек жолдарды басып шығарады, алапта олардың көрсеткіштері орналасқан тәртіппен айналысады.

```
#include <stdio.h>
#include <string. h>

#define MAXLINES 5000 /* ең көп жолдар саны */

char *lineptr[MAXLINES]; /* жолдар көрсеткіштері */

int readlines(char *lineptr[], int nlines);
void wntelines(char *lineptr[], int nlines);
void qsort(char *lineptr[], int left, int right);

/* сортировка строк */
main()
{
      int nlines; /* оқылған жолдар саны */
      if ((nlines = readlinesdineptr, MAXLINES)) >= 0) {
         qsort(lineptr, 0, nlines-1);
         writelines(lineptr, nlines);
         return 0;
      } else {
```

```
            printf(«қате: тым көп жолдар \n»);
            return 1;
        }
    }

#define MAXLEN 1000 /* ең көп жолдар саны */

int getline(char *, int);
char *alloc(int);

/* readlines: жолдарды оқу */
int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];
    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
          if (nlines >= maxlines I ! (p = alloc(len)) == NULL)
              return -1;
          else {
              line[len-1] = '\0'; /* \n символын аламыз */
              strcpy(p, line);
              lineptr[nlines++] = p;
           }
     return nlines;

/* writelines: жолдарды басып шығару */
void writelines(char *lineptr[], int nlines)
{
      int i;
      for (i = 0; i < nlines; i++)
         printf(«%s\n», lineptr[i]);
}

getline функциясы 1.9 тақырыптан алынған

Бұл жердегі жаңалық — lineptr жариялауы:

char *lineptr[MAXLINES]
```

онда lineptr MAXLINES элементтердің массиві бар, олардың әрқайсысы char көрсеткіш болып табылады. Басқаша айтқанда, lineptr [i] – таңбаға көрсеткіш, ал *lineptr[i] – ол көрсететін таңба (мәтін жолының бірінші символы).

Lineptr-аты массивтің болғандықтан, оны көрсеткіш ретінде түсіндіруге болады, яғни, біз мұны алдыңғы мысалдарда жасадық және writelines келесі түрде қайта жазуға болады:

```
/* writelines: жолдарды басып шығару */
void writelines(char *lineptr[], int nlines)
{
    while (nlines-- > 0)
        printf( «%s\n», *lineptr++);
}
```

Алдымен *lineptr бірінші жолды көрсетеді; әрбір көрсеткіш өсуі *lineptr келесі жолды көрсетеді және бұл nlines нөлге дейін жасалады.

Енді енгізу мен шығару туралы түсінген кезде, сіз сұрыптауға кірісе аласыз. 4-тарауында сипатталған жылдам сұрыптауды бірнеше өзгерту керек: жариялауды өзгертулер енгізу керек, ал салыстыру операциясын strcmp-ге жіберу арқылы ауыстыру керек. Алгоритм бірдей қалды және бұл бізге оның дұрыстығына белгілі бір сенімділік береді.

```
/* qsort: сортирует v[left]...v[right] по возрастанию */
void qsort(char *v[], int left, int right)
{
      int i, last;
      void swap(char *v[], int i, int j);

       if (left >= right) /* массивте ештеңе жасалмайды */
           return; /* екі элементтен кем */
       swap(v, left, (left+ right)/2);
       last = left;
       for (i = left+1; i <= right; i++)
            if (strcmp(v[i], v[left]) < 0)
                swap(v, ++last, i);
       swap(v, left, last);
       qsort(v, left, last-1);
       qsort(v, last+1, right);
}
```

Шағын түзетулер қайта орналастыру бағдарламасында да талап етіледі.

```
/* swap: v[i] мен v[j] орындарын ауыстыру */
void swap(char *v[], int i, int j)
{
    char *temp;

    temp = v[i];
    v[i] = v[j];
    v[j] = temp;
}
```

V массивтің әрбір элементі (яғни lineptr) символдағы көрсеткіш болғандықтан, temp v — сонда temp және v элементтері арасында қайта жіберуді жүзеге асыруға болады бірдей түрі болуы керек.

5.7-жаттығу. Readlines жаңа нұсқасын жазыңыз, ол alloc бағдарламасы арқылы жадты сұрамай, main-да анықталған массивте жолдарды есте сақтайды. Бұл бағдарлама қаншалықты жылдам?

5.7. Көп өлшемді массивтер

С-да тікбұрышты көп өлшемді массивтерді қоюға мүмкіндік бар, алайда, іс жүзінде көрсеткіштер массивімен салыстырғанда олар айтарлықтай сирек қолданылады. Бұл параграфта біз олардың кейбір қасиеттерін көрсетеміз.

«Күн-ай» күнін «жыл күніне» және кері аудару міндетін қарастырайық. Мысалы, 1 наурыз – бұл 60-күн немесе 61-күн. Осы түрлендірулер үшін екі функцияны анықтаймыз: day_of_year функциясы ай мен күнді жыл күні, a month_day – жыл күні Ай мен күні түрлендіреді. Соңғы функция екі мәнді есептейтіндіктен, ай мен күн дәлелдері көрсеткіштер болады. Сонымен шақыру

```
month_day( 1988, 60, &m, &d)
```

айнымалы m мәні 2, a d — 29 (29 ақпан).

Біздің функцияларымызға бірдей ақпарат қажет, атап айтқанда әр айдың күндері бар кесте. Бұл кестелер әртүрлі болғандықтан, екі

өлшемді массивте екі бөлек жолды иелену оңайырақ, бұл есептеу кезінде ақпанмен ерекше жағдайды бақылауға қарағанда. Түрлендіруді орындайтын массив пен функциялардың келесі түрі бар:

```
static char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31} };

/* day_of_year: жыл күнін ай және күн бойынша анықтайды */
int day_of_year(int year, int month, int day)
{
     int i, leap;
     leap = year%4 == 0 && year%100 != 0 i I year%400 == 0;
     for (i = 1; i < month; i++)
         day += daytab[leap][i];     return day;
}

/* month_day: жыл күні бойынша ай мен күнді анықтайды */
void month_day(int year, int yearday, int *pmonth, int *pday)
{
      int i, leap;
      leap = year%4 == 0 && year%100 != 0 || year%400 == 0;
      for (i = 1; yearday > daytab[leap][i]; i++)
          yearday -= daytab[leap][i];
      *pmonth = i;
      *pday = yearday;
}
```

Логикалық өрнектің арифметикалық мәні (мысалы, leap есептелген өрнектер) нөлге (өтірік) немесе бірлікке (шындық) тең екенін еске саламыз, сондықтан біз оны daytab массивінде индекс ретінде пайдалана аламыз.

Daytab массиві day_of _year және month_day функцияларына қатысты сыртқы болуы керек, себебі ол басқа да қажет. Біз оны Char түрін белгісіз шағын бүтін үшін char түрін қолдануға заңдылығын көрсету үшін жасадық.

Daytab массиві – бұл екі өлшемді массив, біз әлі де жоқ. Қатаң айтқанда, С — да екі өлшемді массив бір өлшемді массив ретінде қарастырылады, оның әрбір элементі – массив. Сондықтан индекстеу осылай бейнеленген:

```
daytab[i][j] /* [жол] [баған] */
```

былай емес:

```
daytab[i,j] /* ДҰРЫС ЕМЕС */
```

С-дағы екі өлшемді массивтің ерекшелігі тек жазба түрінде, қалған жағдайда оны басқа тілдегідей түсіндіруге болады. Элементтер жолдармен есте қалады, демек, оларды жадта орналасқан ретпен ауыстырған кезде, ең оң индекс жиі өзгереді.

Жиым бастапқы мәндердің тізімімен фигуралы жақшаларға салынған; екі өлшемді массивтің әрбір жолы тиісті кіші бағанмен инициаланады. Нөлдік баған 1-ден 12-ге дейінгі айдың табиғи нөмірлерімен сәйкес келетін индекстер үшін ғана daytab басына қосылды. Мұнда бірнеше жад ұяшықтарын үнемдеу мағынасы жоқ, ал индексті түзету қажет емес бағдарлама анық көрінеді.

Бұл жағдайда жолдар саны маңызды емес, себебі функциялар жол массивіне берілетін болады, олардың әрқайсысы int түріндегі 13 мәннен тұратын массиві бар, біздің жеке жағдайда int түріндегі 13 мәннен тұратын массив болып табылатын объектілерге көрсеткіш бар. Осылайша, егер daytab массиві кейбір f функциясына берілсе, онда бұл функцияны келесідей анықтауға болады:

```
f(int daytab[2][13]) {…}
```

Оның орнына былай жазуға болады:

```
f(int daytab[][13]) {…}
```

мұнда жолдар саны маңызды емес болғандықтан, немесе

```
f(int (*daytab)[13]) {…}
```

Соңғы жазба параметр 13 int түрі мәндерінің массивіне көрсеткіш бар екенін жариялайды. Мұнда жақшалар қажет, өйткені квадрат жақшалар [] *қарағанда жоғары басымдыққа ие. Жақшалары жоқ жариялау

```
int *daytab[13]
```

char-дағы 13 сілтемелерінің массивін анықтайды. Жалпы жағдайда тек бірінші Өлшем (бірінші индекске сәйкес келетін) сұралмауға болады, барлық басқа спецификация қажет.

5.12 параграфында біз күрделі хабарландыруларды қарауды жалғастыр

5.8-жаттығу. Day_of_year және month_day функцияларында енгізілетін күндердің дұрыстығын тексеру жоқ. Бұл кемшіліктерді жойыңыз.

5.8. Көрсеткіштер массивтерін инициализациялау

month_name(n) функциясын жазамыз, ол көрсеткішті N-айдың атауы бар таңбалар жолына қайтарады. Бұл функция статикалық массивті пайдалануды көрсету үшін өте ыңғайлы. Month_name функциясы өзінің жеке иелігінде жол жиымын бар, оның біреуі меңзерді қайтарады. Төменде осы атау массиві қалай басталатынын көрсетеміз.

Бастапқы мән тапсырмасының синтаксисі алдыңғы инициализацияның синтаксисіне ұқсас:

```
/* month_name: n-ші айдың атын қайтарады */
char *month_name(int n)
{
    static char *name[] = {
        “Дұрыс емес ай”,
        “Қаңтар”, “Ақпан”, “Наурыз”,
        “Сәуір”, “Мамыр”, “Маусым”,
        “Шілде”, “Тамыз”, “Қыркүйек”,
        “ Қазан”, “ Қараша”, “Желтоқсан”
    };

    return (a < 1 || n > 12) ? name[0] : name[n];
}
```

Таңбаларға арналған көрсеткіштер массиві name хабарландыруы сұрыптау бағдарламасындағы lineptr хабарландыруы сияқты

бірдей. Инициализатор-әр жол тізімі, олардың әрқайсысы массивте белгілі бір орынға сәйкес келеді. І-жолдың символдары бір жерде орналасқан және оларға көрсеткіш name[i] - де есте қалады. Name массивінің мөлшері көрсетілмегендіктен, компилятор оны көрсетілген бастапқы шамалар саны бойынша есептейді.

5.9. Көп өлшемді массивтерге қарсы көрсеткіштер

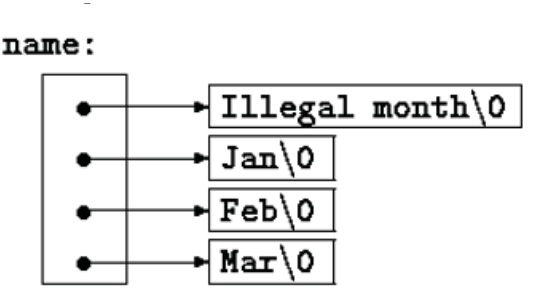
С бағдарламалауды бастаушылар кейде екі өлшемді массив пен көрсетілген мысалдағы ат сияқты көрсеткіштер массивінің айырмашылығы неде екенін түсінбейді. Келесі екі анықтама үшін:

```
int a[10][20];
int *b[10];
```

a[3][4] және b[3][4] жазбалары int типті кейбір мағынаға синтаксистік дұрыс болады. Бірақ тек қана а екі өлшемді массив болып табылады: int типті екі жүз элемент үшін жад бөлінеді, ал а элементінің ығысуын есептеу [жол][баған] массивтің басынан 20 х жол + баған формуласы бойынша оның тікбұрышты табиғатын ескеретін жүргізіледі. В үшін тек 10 көрсеткіш анықталды, ал инициализациясыз. Инициализация анық статикалық немесе бағдарламада болуы керек. Мысалы, әрбір В элементі жиырма элементті массивті көрсетеді, нәтижесінде бір жерде int типті 200 мәндер және көрсеткіштер үшін тағы 10 ұяшықтар орналасатын кеңістік бөлінеді. Көрсеткіштер массивінің маңызды артықшылығы - бұл массивтің жолдары әртүрлі ұзындықтарға ие болуы мүмкін. Осылайша, в массивінің әрбір элементі міндетті түрде жиырма элементті векторды көрсетпейді; біреуі екі элементті, екіншісі – елу элементті көрсете алады, ал кейбіреулері ешнәрсе көрсетпеуі мүмкін.

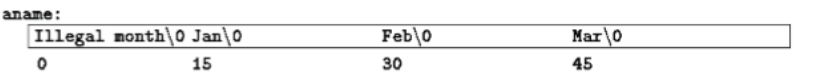
Мұнда біздің пайымдауымыз бүтін мәндерге қатысты болды, бірақ көрсеткіш массивтері көбінесе month_name функциясында болғандай, ұзындығы бойынша ерекшеленетін символдар жолдарымен жұмыс істеу үшін қолданылады. Көрсеткіштер массивінің анықтамасын және оған сәйкес суретті салыстырыңыз:

```
char *name[] = {“Дұрыс емес ай”, “Қаңтар”, “Ақпан”, “Наурыз”};
```



екі өлшемді массив үшін хабарландыру және суретпен:

```
char aname[][15] = {“Дұрыс емес. ай”, “Қаңтар”, “Ақпан”, “Наурыз”};
```



5.9-жаттығу. Day_of_year және month_day бағдарламаларын индекстердің орнына көрсеткіштер арқылы қайта жазыңыз.

5.10. Командалық жол аргументтері

С қолдауын қамтамасыз ететін амалдық ортада пәрмен жолы арқылы іске қосылатын бағдарлама дәлелдерін немесе параметрлерін беру мүмкіндігі бар. Қоңырау кезінде main екі дәлел алады. Бірінші, әдетте argc деп аталатын (argument count қысқарту) командалық жолда берілген дәлелдердің саны. Екінші, argv (argument vector-ден) дәлелдерді қамтитын символдық жолдар массивіндегі көрсеткіш болып табылады. Бұл жолдармен жұмыс істеу үшін әдетте бірнеше деңгейдің көрсеткіштері қолданылады.

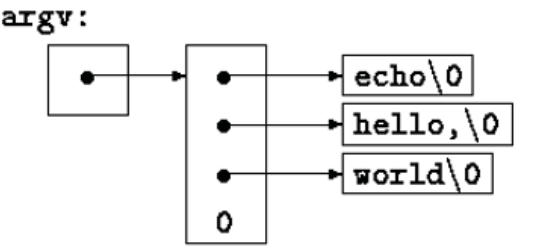
Қарапайым мысал-echo (“эхо”) бағдарламасы, ол бір жолда өз командалық жолының дәлелдерін басып, оларды бір-бірінен бос орындармен бөліп шығарады. Осылайша, команда

```
echo Сәлем, әлем!
```

Басып шығарады

```
Сәлем, әлем!
```

Argv келісімі бойынша [0] шақырылатын бағдарламаның аты бар, сондықтан мәні ешқашан 1-ден кем болмайды. Егер argc 1-ге тең болса, командалық жолда бағдарлама атауынан кейін ешқандай дәлел жоқ. Біздің мысалда argc 3 тең және тиісінше argv, argv [1] және argv [2] “echo”, “Сәлем,” және “ әлем!”. Бірінші қосымша дәлел-argv [1], соңғысы-argv[argc1]. Сонымен қатар, стандарт argv[argc] әрдайым бос көрсеткіш болуын талап етеді.



Echo бағдарламасының бірінші нұсқасы argv-ді символдық көрсеткіштер массиві ретінде түсіндіреді.

```
#include <stdio.h>

/* эхо пәрмен жолының аргументтері: нұсқасы 1 */
main(int argc, char *argv[])
{
        int i;
        for (i = 1; i < argc; i++)
             printf(“%s%s”, argv[i], (i < argc-1) ? “ “ : “”);
        printf(“\n”);
        return 0;
}
```

Argv-көрсеткіштер массивіне көрсеткіш болғандықтан, біз онымен индекстелетін массив ретінде емес, көрсеткіш ретінде жұмыс істей аламыз. Келесі бағдарлама argv өсіміне негізделген, ол әр

жеке сәтте оның мәні char-дағы кезекті көрсеткішін көрсетеді; argc таусылған кезде көрсеткіштерді қайта бөлу аяқталады.

```
#include <stdio.h>

/* эхо пәрмен жолының аргументтері: нұсқасы 1 2 */
main(int argc, char *argv[]) {
      while (--argc > 0)
            printf("%s%s", *++argv[i], (i < argc-1) ? " " : "");
      printf("\n");
      return 0;
}
```

Argv аргументі-аргументтер жолдарының массивінің басына көрсеткіш. h++argv ++ префиксті операторын пайдалану әкеледі бірінші болып argv[0] басып шығаруға емес, argv[1] басып шығарылуына әкеледі. Көрсеткіштің әрбір кезекті өсуі бізге *argv көрсететін келесі дәлел береді. Сонымен қатар, argc мәні 1-ге азаяды және ол нөлге жеткенде, барлық дәлелдер басылады.

Printf нұсқауын жазуға болады:

```
printf((argc > 1) ? "%s " : "%s", *++argv);
```

Көріп отырғанымыздай, printf пішімі де өрнек болуы мүмкін.

Екінші мысал ретінде 4.1-параграфта қарастырылған үлгіні іздеу бағдарламасын алайық және оны біршама жетілдіреміз. Есіңізде болса, іздеу үлгісі біз бағдарламаға терең "монтаждадық" және бұл жақсы шешім емес. Біздің бағдарламамыз unіха-дан grep-ге ұқсас, яғни іздеу үлгісі командалық жолда бірінші аргументпен қойатындай етіп құрайық.

```
#include <stdio.h> #include <string.h>

#define MAXLINE 1000

int getline(char *line, int max);

/* find: 1 аргументі берілген үлгі бар жолдарды басып шығару */
```

```
main(int argc, char *argv[])
{
        char line[MAXLINE];
        int found = 0;
        if (argc != 2)
            printf("find үлгісін пайдаланыңыз\n ");
        else
            while (getline(line, MAXLINE) > 0)
                if (strstr(line, argv[1]) != NULL) {
                    printf ("%s", line);
                    found++;
                }
             return found;
}
```

Стандартты strstr(s, t) функциясы S немесе NULL жолында бірінші кездескен t жолына көрсеткіш қайтарады. Функция тақырып файлында жарияланған <string.h>.

Бұл модель басқа құрылымдарды көрсеткіштермен көрсету үшін одан әрі дами алады. Біз тағы екі қосымша аргументті енгіземіз делік. Олардың бірі үлгі кездесетін жолдан басқа барлық жолдарды басып шығаруды ұйғарады; екіншісі — әрбір шығарылатын жолдың алдында оның реттік нөмірін басып шығаруды ұйғарады.

UNIX жүйесіндегі С-бағдарламалар үшін жалпы келісім бойынша минус белгісі дәлел алдында міндетті емес белгі немесе параметр енгізеді. Мысалы, егер-х тапсырманы қарама-қарсы өзгертетін "басқа" сөзінің белгісі болса, а-n жолдарды нөмірлеу қажеттілігін көрсетеді, онда команда

```
find -x -n үлгі
```

көрсетілген үлгі табылмаған барлық жолдарды басып шығарады және одан басқа әрбір жолдың алдында оның нөмірін көрсетеді.

Міндетті емес дәлелдерді кез келген тәртіппен орналастыруға рұқсат етіледі, бұл ретте бағдарламаның қалған бөлігі ұсынылған дәлелдердің санына байланысты емес. Сонымен қатар, пайдала-

нушы, мысалы, қосымша дәлелдерді біріктіре алатын болса, ыңғайлы болар еді:

ind -nx үлгі

Ал, енді бағдарламамызды жазайық.

#include <stdio.h> #include <string.h>

#define MAXLINE 1000

int getline(char *line, int max);

```
/* find: 1 аргументтен жолдарды үлгілермен басып шығару */
main(int angc, char *argv[])
{
      char line[MAXLINE];
      long lineno = 0;
      int c, except = 0, number = 0, found = 0;

      while (—argc > 0 && (*++argv)[0] == '-')
            while (c = *++argv[0])
              switch (c) {
                case 'x':
                      except = 1;
                      break;
                case 'n' :
                      number = 1;
                      break;
                default:
                      printf( "find: неверный параметр %c\n", c);
                      argc = 0;
                      found = -1;
                      break;
                 }
        if (argc != 1)
            printf ("Используйте: find -x -n образец\n");
        else
            while (getline(line, MAXLINE) > 0) {
                  lineno++;
                  if ((strstr(line, *argv) != NULL) != except) {
                       if (number)
                            printf("%ld:", lineno);
                       printf("%s", line);
```

```
                found++;
                }
            }
        return found;
}
```

Келесі аргументті алу алдында argc 1 азаяды, ал argv келесі аргументке "жылжиды". Цикл аяқталғаннан кейін argc қателері болмаған кезде әлі өңделмеген аргументтер саны бар, ал argv олардың бірін көрсетеді. Осылайша, argc 1, ал *argv үлгіге тең болуы керек. * ++argv-бұл дәлел-жол көрсеткіші, а (*++argv) [0] - оның басқа жолмен сілтеме жасай алатын алғашқы символы:

```
**++argv
```

Индекстеу операторы [] * және ++ қарағанда жоғары басымдыққа ие болғандықтан, дөңгелек жақшалар міндетті, онсыз өрнек * ++(argv[0]) сияқты түсіндіріледі. Біз дәл осындай өрнек нақты Аргументтің таңбаларын қарайтын ішкі циклде қолданылады. Ішкі циклда *++argv[0] өрнек argv[0] көрсеткішін ұзартады.

Көрсеткіштер үшін күрделі өрнектерге қажеттілік жиі емес. Бірақ егер бұл орын алса, көрсеткіш есептеу процесін екі немесе үш қадамға бөле отырып, сіз бұл өрнектің қабылдануын жеңілдетесіз.

5.10-жаттығу. Әрбір оператор мен операндтың жеке дәлелмен ұсынылған командалық жолмен берілген өрнектің поляк кері жазбасын түсіндіретін expr бағдарламасын жазыңыз. Мысалы,

```
expr 2 3 4 + *
```

2x(3 + 4) өрнегі сияқты есептеледі.

5.11-жаттығу. Entab және detab бағдарламаларын жетілдіру (1.20 және 1.21- жаттығуларын қараңыз), сондықтан дәлелдер арқылы табуляцияның "тоқтары" тізімін қоюға болады.

5.12-жаттығу. Entab және detab мүмкіндіктерін түрді қолданғанда кеңейтіңіз

```
entab -m +n
```

табуляция” табуляция m-ші позициядан басталып, әр n позициядан кейін орындалды. Әдепкі бағдарлама мінез-құлқының пайдаланушыға ыңғайлы нұсқасын жасаңыз (дәлел жоқ кезде).

5.13-жаттығу. Соңғы енгізілген жолдардың n басып шығаратын tail бағдарламасын жазыңыз. Әдепкі бойынша n мәні 10 тең, бірақ қаласаңыз, n аргументі арқылы орнатуға болады. Түрдің айналымы

```
tail –n
```

соңғы жолдардың n басып шығарады. Бағдарламаны жадты ең жақсы түрде қолданатындай етіп жазыңыз; жолдарды есте сақтауды 5.6-параграфта сипатталған сұрыптау бағдарламасындағы сияқты, белгіленген жол өлшемі бар екі өлшемді массив негізінде емес ұйымдастырыңыз.

5.11. Функциялар көрсеткіштері

С-де функцияның өзі айнымалы емес, бірақ функцияның көрсеткішін анықтауға және онымен әдеттегі айнымалы сияқты жұмыс істеуге болады: тағайындау, массивте орналастыру, функцияның параметрі ретінде беру, функцияның нәтижесі ретінде қайтару және т.б. бұл мүмкіндіктерді көрсету үшін осы тарауда кездескен сұрыптау бағдарламасын қолданамыз. Оны міндетті емес аргументті тапсырғанда-n енгізілетін жолдар лексикографиялық тәртіпте емес, олардың сандық мәні бойынша ретке келтіретіндей етіп өзгертеміз.

Сұрыптау, әдетте, үш бөлікке бөлінеді: объектілер жұбының реттілігін анықтайтын салыстыру; объектілер жұбы орнын ауыстыратын орын ауыстыру және барлық объектілер ретке келтірілгенге дейін салыстыру мен орнын ауыстыруды жүзеге асыратын сұрыптау алгоритмі. Сұрыптау алгоритмі салыстыру және ауыстырып қою операцияларына байланысты емес, сондықтан оны салыстыру және ауыстырып қою функциясының әр түрлі параметрлері ретінде бере отырып, сұрыптаудың әр түрлі өлшемдеріне теңшеуге болады.

Екі жолды лексикографиялық салыстыру strcmp функциясымен орындалады (біз бұл функцияны бұрын қаралған сұрыптау бағдарламасында пайдаландық); сондай-ақ бізге екі жолды сандық мән ретінде салыстыратын және салыстыру нәтижесін strcmp беретін түрде қайтаратын numcmp бағдарламасы қажет болады. Бұл функциялар main алдында жарияланады, ал көрсеткіш олардың біріне qsort функциялары беріледі. Басты мақсатқа назар аудару үшін біз дәлелдер ұсынған кезде ықтимал қателерді талдаудан бас тарта отырып, өзімізге тапсырманы жеңілдетдік.

```
#include <stdio. h> #include <string. h>

#define MAXLINES 5000 /* ең көп жолдар саны */

char *lineptr[MAXLINES]; /* мәтін жолының көрсеткіштері */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);
void qsort(void *lineptr[], int left,
      int right,     int (*comp)(void *, void *));
int numcmp(char *, char *);

/* жолдарды сұрыптау */
main(int argc, char *argv[])
{
       int nlines; /* оқылған жолдар саны */
       int numeric = 0; /* 1, егер сан бойынша сұрыпталған болса */
       if (argc > 1 && strcmp(argv[1], “-n”) == 0)
            numeric = 1;
       if ((nlines = readlinesdineptr, MAXLINES)) >= 0) {
            qsort((void **) lineptr, 0, nlines-1,
                  (int (*)(void*, void*))(numeric ? numcmp : strcmp));
            writelines(lineptr, nlines);
            return 0;
       } else {
            printf(“Тым көп жолдар енгізілді \n”);
            return 1;
       }
}
```

Qsort, strcmp және numcmp функцияларына қоңырау шалу кезінде олардың аттары осы функциялардың мекен-жайы ретінде қарас-

тырылады, сондықтан массив атауынан бұрын қажет болмағандықтан, & операторы олардың алдында қажет емес.

Біз qsort-ті тек сипаттамалық жолдарды ғана емес, кез-келген мәліметтерді өңдей алатын етіп жаздық. Прототиптен көріп отырғаныңыздай, qsort функциясы оның көрсеткіштері, екі бүтін сан және екі аргументтері бар функцияны оның дәлелдері ретінде күтеді. Жалпы типтегі void * көрсеткішінің дәлелі келтірілген. Кез-келген меңзерді * түрін жоққа шығаруға болады және керісінше ақпаратты жоғалтпай, сондықтан qsort-қа аргументтерді бірінші болып босқа * айналдыру арқылы жүгінуге болады. Салыстыру функциясының ішінде оның дәлелдері қажет типке шығарылады. Шын мәнінде, бұл түрлендірулер дәлелдерді ұсынуға әсер етпейді, олар тек компилятор үшін типтегі консистенцияны қамтамасыз етеді.

```
/* qsort: сортирует v[left]...v[ right] өсу бойынша */
void qsort(void *v[], int left, int right,
      int (*comp)(void *, void *))
{
      int i, last;
      void swap(void *v[], int, int);

       if (left >= right) /* ештеңе жасалмайды, егер */
           return; /* массивте екі элементтен аз */
        swap(v, left, (left + right )/2);
        last = left;
        for (i = left+1; i <= right; i++)
             if ((*comp)(v[i], v[left]) < 0)
                 swap(v, ++last, i);
        swap(v, left, last);
        qsort(v, left, last-1, comp);
        qsort(v, last+1, right, comp);
}
```

Жариялауларға мұқият назар салыңыз. Qsort функциясының төртінші параметрі:

```
int (*comp)(void *, void *)
```

comp – бұл екі аргументі бар функцияға көрсеткіш және int түрінің нәтижесін береді.

Жолда comp пайдалану

```
if ((*comp)(v[i], v[left]) < 0)
```

“comp-бұл функцияға көрсеткіш” хабарламасымен келісіледі, сондықтан * comp – бұл функция және

```
(*comp)(v[i], v[left])
```

— оған жүгіну. Жариялаудың дұрыс түсіндірілуін қамтамасыз ету үшін мұнда жақшалар қажет; оларсыз хабарландыру

```
int *comp(void *, void *) /* ДҰРЫС ЕМЕС */'
```

comp — бұл көрсеткіш int-ға қайтаратын функция, бұл қажет емес

.

Біз екі жолды салыстыратын strcmp функциясын қарастырдық. Төменде numcmp функциясы, ол екі жолды салыстырады, оларды Сан ретінде қарастырады; олар алдын ала atof функциясымен сандық мәндерге аударылады.

```
#include <stdlib.h>

/* numcmp: s1 мен s2 сан ретінде салыстырады */
int numcmp(char *s1, char *s2)
{
    double v1, v2;
    v1 = atof(s1);
    v2 = atof(s2);
    if (v1 < v2)
        return -1;
    else if (v1 > v2)
        return 1;
    else
        return 0;
}
```

Екі көрсеткішті ауыстыратын swap функциясы, біз void* - ге ауысатыннан басқа, осы тарауда бұрын әкелген нәрсеге ұқсас.

Екі көрсеткішті ауыстыратын swap функциясы, біз осыған дейін осы тарауда мысалдарда қолданған функцияларға ұқсас, сілтегіш мәлімдемелері void * ауыстырылғанын санамағанда.

```
void swap(void *v[], int i, int j)
{
        void *temp;
        temp = v[i];
        v[i] = v[j];
        v[j] = temp;
}
```

Сұрыптау бағдарламасын көптеген басқа мүмкіндіктермен толықтыруға болады; олардың кейбіреулері жаттығулар ретінде ұсынылады.

5.14-жаттығу. Сұрыптау бағдарламасын -r параметріне жауап беретіндей етіп өзгертіңіз, бұл нысандарды кері тәртіпте, яғни кему ретімен сұрыптау керектігін көрсетеді. -r -n-мен бірге жұмыс істейтініне көз жеткізіңіз.

5.15-жаттығу. Бағдарламаға міндетті емес -f параметрді енгізіңіз, оның тапсырмасы төменгі және жоғарғы регистр таңбаларын (мысалы, а және А тең болған жағдайда) ажыратпайды.

5.16-жаттығу. Бағдарламада міндетті емес параметр -d, ол салыстыру кезінде тек әріптерді, сандарды және бос орындарды ескереді. Бағдарламаны осы параметр -f параметрімен бірге жұмыс істей алатындай етіп ұйымдастырыңыз.

5.17-жаттығу. Бағдарламада өрістермен жұмыс істеу мүмкіндігін іске асырыңыз: жол ішіндегі өрістер бойынша сұрыптау мүмкіндігі. Әрбір өріс үшін параметрлер жиынтығын қамтамасыз етіңіз. Бұл кітаптың пәндік көрсеткіші параметрлерімен реттелген: -df терминдер үшін және- n беттер нөмірлері үшін.

5.12. Күрделі жариялаулар

Кейде С жарнамалардың синтаксисі үшін сынға түседі, әсіресе функциялардың көрсеткіштері бар жағдайда. Бұл синтаксис біздің объектілерді жарнамалау және оларды пайдалану сияқты әрекет нәтижесінде пайда болды. Қарапайым жағдайларда бұл синтаксис жақсы, бірақ күрделі жағдайларда ол қиындық тудырады, өйткені хабарландырулар жақшамен жабылған және солдан оңға қарай оқу мүмкін емес. Мәселе келесі екі жариялаудың айырмашылығын көрсетеді:

```
int *f(); /* f: int-ға ук-тікөрсеткіштерді қайтаратын функция */
int (*pf)(); /* pf: int функцияға қайтаратын көрсеткіш */
```

Префиксті * оператордың басымдығы () басымдықтан төмен, сондықтан екінші жағдайда жақша қажет.

Іс жүзінде күрделі хабарландырулар сирек кездесетін болса да, оларды қалай түсіну керек, ал қажет болса, оларды қалай құрастыру керек екенін білу маңызды.

Жақсы әдісті көрсетеміз: хабарландырулар typedef көмегімен шағын қадамдармен қозғала отырып, синтездеуге болады; бұл әдіс 6.7-параграфта қарастырылған. Осы параграфта дұрыс С-жариялауды оларға сәйкес ауызша сипаттамаларға және кері түрлендіруді жүзеге асыратын екі бағдарлама мысалында біз жариялауды құрастырудың басқа тәсілін көрсетеміз. Ауызша сипаттама солдан оңға қарай оқылады.

Бірінші бағдарлама, dcl-күрделі. Ол келесі мысалдарда көрсетілгендей, С- жариялаулау сипаттамаларға түрлендіреді:

```
char **argv
   argv: көрсеткіш, көрсеткішке, char-ға
int (*daytab)[13]
   daytab: int массив[13] көрсеткіші

int *(daytab)[13]
   daytab: көрсеткіштегі массив[13] int
```

```
void *comp()
   comp: функц. қай-тару. көрсеткіш, void
void (*comp)()
   comp: көрсеткіш, функц. қай-тару.
void char (*(*x())[])()
   x: функц. қай-тару. көрсеткіш, массивке [] көрсеткіштен, функцияларға.
қай-тару. char
char (*(*x[3])())[5]
```

x: массив [3] көрсеткіштен функц. қай-тару. көрсеткіш Char массивінде[5]

Dcl функциясы өз жұмысында хабарлаушы грамматиканы пайдаланады. Бұл грамматика А қосымшасының 8.5-параграфында қатаң жазылған, ал жеңілдетілген түрде былай жазылады:

хабарлаушы:	қосымша * жеке хабарландырушы
жеке хабарландырушы:	аты (хабарлаушы)
	жеке хабарландырушы ()
	шын мәнінде хабарлаушы [міндетті емес мөлшері]

Қарапайым тілмен айтқанда, хабарландырушы өзі-хабарландырушы бар, оның алдында * тұра алады (яғни, бір немесе бірнеше жұлдызша), онда-хабарландырушының аты бар, немесе жақшадағы хабарландырушы, немесе келесі жақшамен жеке жариялаушы немесе жеке-хабарландырушы жұп шаршы жақшамен және оның ішінде өлшем орналасуы мүмкін.

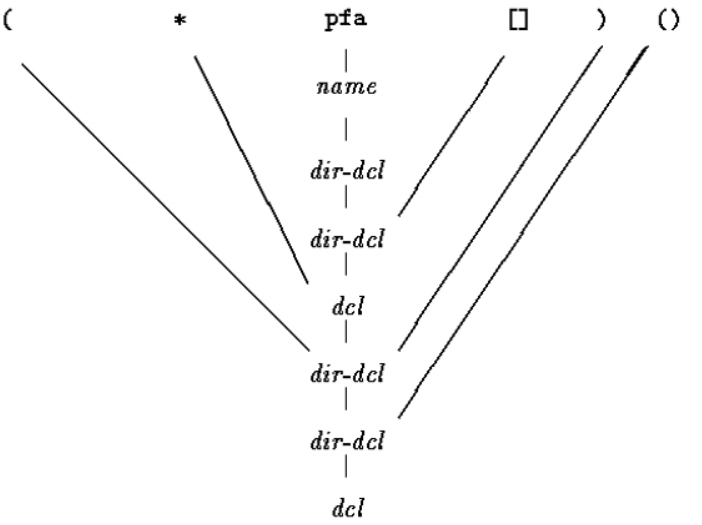
Бұл грамматиканы жарнамаларды грамматикалық талдау үшін пайдалануға болады. Мысалы, мұндай хабарландырушыны қарастырайық:

```
(*pfa[])()
```

pfa атауы аты ретінде жіктеледі, демек, нақты хабарлаушы ретінде. Содан кейін pfa [] нақты хабарлаушы ретінде танылады, ал *pfa []-хабарлаушы ретінде, демек, (*pfa[]) нақты хабарлаушы бар. Бұдан әрі, (*pfa []) () жеке хабарландырушы және осылайша хабарлаушы бар. Бұл грамматикалық талдау келесі бетте келтірілген талдау

ағашымен өрнектеуге болады (онда-хабарландырушы неғұрлым қысқа, атап айтқанда,жеке хабарлаушылар көрсетілген.).

Хабарландырушыны өңдеу бағдарламасының өзегі dcl және dirdcl функцияларының жұбы болып табылады. Грамматика рекурсивті түрде анықталғандықтан, бұл функциялар жарнаманың жекелеген бөліктерін тану бойынша бір-біріне рекурсивті түрде жүгінеді. Талқыланатын бағдарламада грамматикалық талдау үшін қолданылған әдіс рекурсивті түсіру деп аталады.




```
/* dcl: хабарландырушыны талдау */
void dcl(void)
{
      int ns;

      for (ns = 0; gettoken() == '*'; ) /* жұлдызшаларды есептеу */
            ns++;
      dirdcl();
      while (ns-- > 0)
            strcat(out, " көрсеткіш");
}

/* dirdcl: хабарландырушының өзін талдау */
```

```
void dlrdcl(void) {
        int type;

        if (tokentype == '(') { /* ( dcl ) */        dcl();        if (tokentype != ')')
            printf( "қате кетті )\n");
} else if (tokentype == NAME) /* ауыспалы атау */
        strcpy(name, token);
else
        printf("қате: name немесе (dcl)\n" болуы керек);
while ((type = gettoken()) == PARENS || type == BRACKETS)
        if (type == PARENS)
            strcat(out, " функц. қайт.");
        else {
            strcat(out, " массив");
            strcat(out, token);
            strcat(out, " шығады");
        }
}
```

Берілген бағдарламалар тек көрнекі мақсаттарға арналған және олар толығымен сенімді емес. Dcl-ге келетін болсақ, оның мүмкіндіктері айтарлықтай шектеулі. Ол char және int сияқты қарапайым типтермен ғана жұмыс істей алады және функциялардағы аргумент түрлерімен және const сияқты анықтағыштармен жұмыс істей алмайды. Қосымша кеңістіктер оған қауіпті. Қате жағдайдан шығу үшін ол ешқандай шара қолданбайды, сондықтан оған дұрыс емес сипаттамалар да қарсы. Осы кемшіліктерді жоюға біз жаттығуға қалдырамыз.

Төменде жаһандық айнымалылар және бағдарламаның негізгі мәні келтірілген.

```
#include <stdio.h>
#include <string.h>
 #include <ctype.h>

#define MAXTOKEN 100

enum {NAME, PARENS, BRACKETS };

void dcl(void); void dirdcl(void);
int gettoken(void);
```

```
int tokentype; /* соңғы лексеманың түрі */
char token[MAXTOKEN]; /* соңғы лексеманың мәтіні */
char name[MAXTOKEN]; /* аты */
char datatype[MAXTOKEN]; /* түрі = char, int және т.б. */
char out[1000]; /* берілетін мәтін */

main() /* хабарландыруды ауызша сипаттауға түрлендіру */
{
     while (gettoken() i= EOF) {  /* жолда 1-лексема */
          strcpy(datatype, token); /* бұл деректер түрі */
          out[0] = '\0';
          dcl(); /* жолдың қалған бөлігін талдау
          if (tokentype != '\n' )
               printf ( " синтаксистік қате\n " ) ;
          printf("%s; %s %s\n", name, out, datatype);
      }
      return 0;
}
```

Gettoken функциясы бос орындар мен табуляцияларды өткізіп, содан кейін енгізуден келесі лексеманы алады; "лексема" (token) — бұл атау немесе жұп дөңгелек жақшалар, немесе жұп квадрат жақшалар (онда орналасқан санмен болуы мүмкін) немесе кез келген басқа бір бірлік символ.

```
int gettoken(void) /* келесі лексиканы қайтарады */
{
     int c, getch(void);
     void ungetch(int);

     char *p = token;
     while ((c = getchQ) == ' ' ! ! c == ' \ t ' )
          ;
      if (c == '(') {
           if ((c = getch()) == ')') {
              strcpy(token, "()");
              return tokentype = PARENS;
           } else {
              ungetch(c);
              return tokentype = '(';
           }
       } else if (c == '[') {
           for (*p++ = c; (*p++ = getch()) != ']'; )
               ;
```

```
            *p = '\0' ;
            return tokentype = BRACKETS;
        } else if (isalpha(c)) {
            for (*p++ = c; isalnum(c = getch()); )
                *p++ = c;
            *p = '\0';
            ungetch(c);
            return tokentype = NAME;
         } else
            return tokentype = c;
}
```

Getch және ungetch функциялары 4-тарауында қаралды.

Кері түрлендіру оңай іске асырылады, әсіресе, егер артық жақшалар пайда болатынына мән бермесе. Undcl бағдарламасы "x" сияқты фразаны айналдырады, біз көрсететін "char-ды қайтаратын функцияларға көрсеткіштер массивіне қайтаратын функция бар"

```
x () * [] * () char
```

және жариялау

```
char (*(*x())[])()
```

Мұндай қысқартылған кіріс синтаксисі gettoken функциясын қайта пайдалануға мүмкіндік береді. Undcl функциясы dcl сияқты сыртқы айнымалыларды пайдаланады.

```
/* undcl : сөздік сипаттаманы жарнамаға түрлендіреді */
main ()
{
      int type;
      char temp[MAXTOKEN];

      while (gettoken() != EOF) {
            strcpy(out, token);
            while ((type = gettoken()) != '\n')
                  if (type == PARENS || type == BRACKETS)
                        strcat(qut, token);
                  else if (type == '*') {
                        sprintf(temp, "(*%s)", out);
```

```
                strcpy(out, temp);
            } else if (type == NAME) {
                sprintf(temp, “%s %s”, token, out);
                strcpy(out, temp);
            } else
                printf (“қате элемент %s \n”, token фразасында);
        printf(“%s\n”, out);
    }
    return 0;
}
```

5.18-жаттығу. DCL кіріс ақпаратында қателерді өңдейтіндей етіп өзгертіңіз.

5.19-жаттығу. Undcl-ді қосымша жақшаларды жасамайтындай етіп өзгертіңіз.

5.20-жаттығу. DCL мүмкіндіктерін кеңейтіңіз, DCL жарнамаларды функция дәлелдері, const сияқты біліктіліктер және т. б.

6-тарау. Құрылымдар

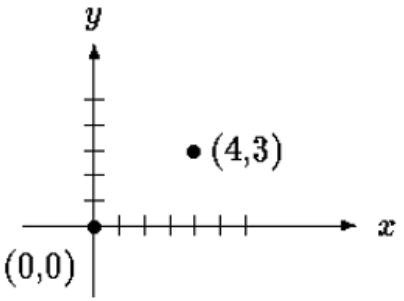
Құрылым – бір атаумен топтастырылған бір немесе бірнеше айнымалы (әр түрлі типтан болуы мүмкін). Кейбір тілдерде, атап айтқанда Паскальда, құрылымдар жазбалар деп аталады. Құрылым күрделі деректерді ұйымдастыруға көмектеседі (әсіресе үлкен бағдарламаларда), өйткені өзара байланысқан айнымалылар тобын көптеген жеке элементтер ретінде емес, біртұтас ретінде түсіндіруге мүмкіндік береді.

Құрылымның дәстүрлі мысалы-төлем ведомосінің жолы. Бұл сипаттамалардың кейбіреулері өздері құрылым бола алады: мысалы, толық аты бірнеше компоненттен тұрады (тегі, аты және әкесінің аты); мекен-жайы, тіпті жалақы. Тағы бір мысал (С-ға тән) – графика аймағынан: нүкте координаттар жұбы бар, тіктөртбұрыш нүктелер жұбы бар және т. б.

Құрылымдарға қатысты ANSI стандартымен енгізілген негізгі өзгерістер – олар үшін тағайындау операцияларын енгізу. Құрылым көшірілуі мүмкін, оларға тағайындау операциялары орындалуы мүмкін, оларды аргументтер ретінде функцияларға беруге болады, ал функциялар оларды нәтиже ретінде қайтаруы мүмкін. Көптеген компиляторларда бұл мүмкіндіктер бұрыннан іске асырылған, бірақ қазір олар стандартта дәл айтылған. Автоматты құрылымдар мен массивтер үшін де бастамашылық жасауға болады.

6.1. Құрылымдар туралы негізгі мәліметтер

Бірнеше графикалық құрылымдарды құрастырамыз. Негізгі объект ретінде x және y координаттары бар нүкте шығады.



Көрсетілген екі компонентті келесідей жарияланған құрылымға орналастыруға болады:

```
struct point {
    int x;
    int y;
};
```

Құрылымның хабарландыруы struct кілт сөзінен басталады және фигуралы жақшаларға салынған хабарландырулар тізімін қамтиды. Struct сөзінің артында құрылым тегімен аталатын атау болуы мүмкін (біздің жағдайда point). Тег осы түрдің құрылымына атау береді және одан әрі фигуралы жақшаларға салынған хабарландырудың сол бөлігінің қысқаша белгісі бола алады.

Құрылымда аталған айнымалылар 10 элементтері (members) деп аталады. Кез келген коллизиясыз элементтер мен тегтердің аттары қарапайым айнымалылардың (яғни элементтер емес) аттарымен сәйкес келуі мүмкін, себебі олар әрқашан контекст бойынша ерекшеленеді. Сонымен қатар, элементтердің бірдей атаулары әртүрлі құрылымдарда болуы мүмкін, бірақ егер жақсы бағдарламалау стилін ұстансаңыз, бірдей атаулар мағынасы бойынша жақын объектілерге ғана беріледі.

Құрылымды жариялау түрін анықтайды. Элементтердің тізімін жабатын оң жақ жақ жақшадан кейін айнымалыларды кез келген базалық түрдің атауынан кейін көрсетуге болады. Осылайша, өрнек

```
struct {…} x, y, z;
```

синтаксис тұрғысынан өрнекке ұқсас

```
int x, y, z;
```

ал екіншісі x, y және z-ті көрсетілген түрдің айнымалыларымен жариялайды; және екіншісі тиісті өлшемдегі жадты бөлуге әкеледі.

Айнымалылардың тізімі жоқ құрылымды жариялау жадты сақтамайды; ол жай ғана үлгіні немесе құрылым үлгісін сипаттайды. Алайда, құрылымда тег болса, онда бұл тег одан әрі құрылымдық нысандарды анықтау кезінде пайдалануға болады. Мысалы, жоғарыда көрсетілген point құрылымы сипаттамасымен жол

```
struct point pt;
```

struct point типті pt құрылымдық айнымалысын анықтайды. Құрылымдық айнымалыны анықтағанда, оның элементтерінің инициализаторлар тізімін тұрақты өрнектер түрінде қалыптастыра отырып, инициалдауға болады:

```
struct point maxpt = { 320, 200 };
```

Автоматты құрылымдарды тиісті түрдің құрылымын қайтаратын функцияға тағайындау немесе айналдыру арқылы инициализациялауға болады.

Құрылымның жеке элементіне қол жеткізу түрдің конструкциясы арқылы жүзеге асырылады:

```
аты-құрылымы.элемент
```

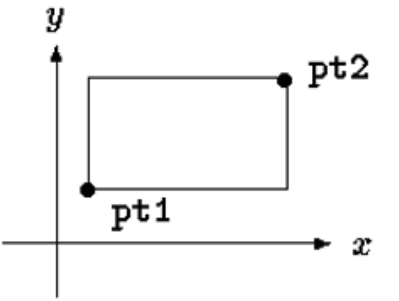
Құрылым элементіне қол жеткізу операторы. (нүкте) құрылым атауын және элемент атауын қосады. pt нүктесінің координаттары сияқты басып шығару үшін келесі printf қызметі жарамды:

```
printf("%d,%d", pt.x, pt.y);
```

Тағы бір мысал: координаталар басынан (0,0) pt қашықтықты есептеу үшін жазуға болады

```
double dist, sqrt(double);
dist = sqrt((double)pt.x * pt.x + (double)pt.y * pt.y);
```

Құрылымдар бір-біріне салынуы мүмкін. Тіктөртбұрыштың ықтимал көріністерінің бірі – оның диагональдарының бірінің бұрыштарында жұп нүкте:




```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

Rect құрылымы екі point құрылымын қамтиды. Егер біз screen деп жарияласақ

```
struct rect screen;
```

онда

```
screen.pt1.x
```

screen арқылы pt1 нүктесінің координатына жүгінеді.

6.2. Құрылымы мен функциялар

Құрылымдарға қатысты жалғыз мүмкін операциялар – бұл оларды көшіру, меншіктеу, адресті & арқылы алу және оның элементтеріне қол жеткізуді жүзеге асыру. Көшіру және меншіктеу аргументтер функцияларына беруді және олардың мәндерін қайтаруды қамтиды. Құрылымдарды салыстыруға болмайды. Құрылымды оның элементтерінің тұрақты мәндерінің тізімімен инициализациялауға болады. Автоматты құрылымды тағайындау арқылы да инициализациялауға болады.

Құрылымдармен жақсы танысу үшін нүктелер мен тіктөртбұрыштарды басқаратын бірнеше функцияларды жазамыз. Сұрақ туындайды: аталған нысандарды функцияларға қалай беруге болады? Кем дегенде үш тәсіл бар: компоненттерді бөлек беру, барлық құрылымды толығымен беру және көрсеткішті құрылымға беру. Әрбір тәсілдің өз артықшылықтары мен кемшіліктері бар.

makepoint бірінші функциясы екі мәнді алады және point құрылымын қайтарады.

```
/* makepoint: x және y компоненттері бойынша нүктені қалыптастырады */
struct point makepoint(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

Айта кетсек: дәлелдің аты мен құрылым элементінің аты арасында ешқандай қақтығыс пайда болмайды; сонымен қатар, ұқсастық ол белгілеген объектілердің туыстығын көрсетеді.

Енді makepoint көмегімен кез келген құрылымның динамикалық инициализациясын орындауға немесе қандай да бір функция үшін құрылымдық дәлелдерді қалыптастыруға болады:

```
struct rect screen; struct point middle;
struct point makepoint(int, int);

screen.pt1 = makepoint(0, 0);
screen.pt2 = makepoint(XMAX, YMAX);
middle = makepoint((screen.pt1.x + screen. pt2.x)/2,
      (screen.pt1.y + screen.pt2.y)/2);
```

Келесі қадам нүктелерге әртүрлі операцияларды жүзеге асыратын бірқатар функцияларды айқындаудан тұрады. Мысал ретінде келесі функцияны қарастырайық:

```
/* addpoint: екі нүктені қосу */
struct point addpoint(struct point p1, struct point p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

Мұнда екі аргументтер мен қайтарылатын мән – құрылымдар. Біз құрамдас бөліктерді тікелей p1-ге көбейтеміз және бұл үшін уақытша айнымалыны пайдаланбаймыз, бұл үшін құрылымдық параметрлер кез келген басқа сияқты мән бойынша беріледі.

Басқа мысал ретінде ptinrect функциясын келтіреміз, ол біз оған қатысты келісімді қабылдайтын тіктөртбұрыштың ішіндегі нүкте бар-жоғын тексереді, оның сол және төменгі жағы, бірақ жоғарғы және оң жағы жоқ.

```
/* ptinrect: 1 қайтарады, егер p г –да болса және 0, керісінше жағдайда*/
int ptinrect(struct point p, struct rect r) {
    return p.x >= r.ptl.x && p.x < r.pt2.x
        && p.y >= r.ptl.y && p.y < r.pt2.y;
}
```

Бұл жерде тіктөртбұрыш стандартты түрде ұсынылған, яғни pt1 нүктесінің координаттары pt2 нүктесінің тиісті координаттарынан аз. Келесі функция тіктөртбұрышты канондық түрде алуға кепілдік береді.

```
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

/* canonrect: тіктөртбұрыштың координаттарын канонизациялау */
struct rect canonrect(struct rect r) {
    struct rect temp;
    temp.pt1.x = min(r.pt1.x, r.pt2.x);
    temp.ptl.y = min(r.pt1.y, r.pt2.y);
    temp.pt2.x = max(r.pt1.x, r.pt2.x);
    temp.pt2.y = max(r.pt1.y, r.pt2.y);
    return temp;
}
```

Егер функциялар үлкен құрылым берілсе, оны толығымен көшіргеннен гөрі, көрсеткішті оған беру тиімдірек. Құрылымдағы көрсеткіштер әдеттегі айнымалы көрсеткіштерден еш айырмашылығы жоқ.

```
struct point *pp;
```

pp – бұл struct point типті құрылымға арналған көрсеткіш. Егер pp point құрылымын көрсетсе, онда *pp-бұл құрылымның өзі, ал (*pp).x және (*pp).y-оның элементтері. pp көрсеткішін пайдалана отырып, біз жаза аламыз

```
struct point origin, *pp;
pp = &origin;
printf (“origin: (%d,%d)\n”, (*pp).x, (*pp).y);
```

Жақшалар (*pp).x қажет, себебі оператордың басымдығы . қарағанда жоғары басымдық *. Өрнек *pp.x * (pp.x) дұрыс емес, өйткені pp.x көрсеткіш болып табылмайды.

Құрылымдағы көрсеткіштер өте жиі пайдаланылады, сондықтан оның элементтеріне қол жеткізу үшін жазбаның тағы бір, қысқа нысаны ойлап тапты. Егер p – құрылымға көрсеткіш болса, онда

```
p - > элемент-құрылымдар
```

оның жеке элементі бар. (Оператор - > белгіден тұрады, одан кейін бірден > белгісі болады.) Сондықтан printf түрінде қайта жазуға болады

```
printf(“origin: (%d,%d)\n”, pp->x, pp->y);
```

Операторлар . және - > солдан оңға қарай орындалады. Осылайша, хабарландырулар болған жағдайда,

```
struct rect r, *rp = &r;
```

келесі төрт өрнек мынаған тең болады:

```
r.pt1.x rp->pt1.x (r.pt1).x (rp->pt1).x
```

Құрылым элементтеріне қол жеткізу операторлары . және - > шақыру операторларымен бірге функциялар () және массив индекстеу [] басымдықтар иерархиясында ең жоғары орын алады және кез келген басқа операторлардан бұрын орындалады. Мысалы, егер хабарландыру орнатылса,

```
struct {
int len;
char *str;
} *p;
```

онда

```
++p->len
```

Len құрылымы элементінің мәнін көбейтеді, p көрсеткіші емес, өйткені бұл өрнекте жақшалар бар: ++(p->Len). Операцияларды орындау тәртібін өзгерту үшін айқын жақшалар қажет. Осылайша, жылы (++p) - >len, len мәнін алуға бұрын, бағдарлама прастит көрсеткіш p. В (p++)>len көрсеткіш p len мәні алынады кейін артады (соңғы жағдайда жақша міндетті емес).

Сол ережелер бойынша *p - >str str көрсететін нысанның мазмұнын білдіреді; *p - >str++ ол көрсеткен нысанның мәнін алғаннан кейін str көрсеткіші өседі (*s++ сияқты); (*p->str)++ str көрсететін нысанның мәнін арттырады; *p++ - >str str нені көрсеткетінін алғаннан кейін p артады.

6.3. Құрылым массивтері

Әр кілт сөздің Си-бағдарлама мәтініне кіру санын анықтайтын бағдарламаны қарастырайық. Біз кілт сөздерді жолдар массиві және кілт сөздер санауыштар түрінде бүтін массив түрінде сақтай білуіміз керек. Мүмкін нұсқалардың бірі-екі параллель массиві бар:

```
char *keyword[NKEYS];
int keycount[NKEYS];
```

Алайда, олар параллель екендігі туралы факт, бізге басқа сақтау ұйымын – құрылымдардың массиві арқылы көрсетеді. Әрбір кілт сөзді жұп сипаттамалары сипаттауға болады

```
char *word;
int count;
```

Мұндай жұптар массивті құрайды.

```
struct key {
    char *word;
    int count;
} keytab[NKEYS];
```

key түрінің құрылымын жариялайды және әрбір элемент осы түрдің құрылымы болып табылатын және бір жерде жады бөлінетін keytab массивін анықтайды. Мұны басқаша жазуға болады:

```
struct key {
    char *word;
    int count;
};
struct key keytab[NKEYS];
```

Keytab атаулардың тұрақты жиынтығын қамтиды, өйткені ол сыртқы массив жасау және анықтау кезінде бір рет инициалдау оңай. Құрылымдарды инициализациялау бұрын көрсетілген инициализацияға ұқсас – анықтаудан кейін фигуралық жақшада жасалған инициализаторлар тізімі:

```
struct key {
    char *word;
    int count;
} keytab[] = {
    “auto”, 0,
    “break”, 0,
    “case”, 0,
    “char”, 0,
    “const”, 0,
    “continue”, 0,
    “default”, 0,
     ...
    “unsigned”, 0,
    “void”, 0,
    “volatile”, 0,
    “while”, 0
};
```

Инициализатор конфигурация құрылымымен сәйкес келу үшін жұбымен қойылады. Қатаң айтқанда, әрбір жеке құрылым үшін бірнеше инициализаторлар, мысалы,

```
{ “auto”, 0 },
{ “break”, 0 },
{ “case”, 0 },
```

Дегенмен, инициализаторлар – қарапайым тұрақтылар немесе символдардың жолдары және олардың барлығы бар болса, ішкі жақшалар қажет емес. Keytab массив элементтерінің саны бастамашылардың саны бойынша есептеледі, себебі олар толық ұсынылған, ал төртбұрышты жақшаның ішінде” [] “ ештеңе көрсетілмеген.

Кілт сөздерді есептеу бағдарламасы keytab анықтамасынан басталады. Main бағдарламасы getword функциясына бірнеше рет хабарласып, оның әр шақыруында келесі сөз алады. Әрбір сөз keitab-да іздейді. Бұл үшін 3 тарауында жазылған екілік іздеу функциясы қолданылады. Кілт сөздер тізімі әліпбилік ретпен реттелуі керек.

```
#include <stdio.h> #include <ctype.h> #include <string.h>

#define MAXWORD 100

int getword(char *, int);
int binsearch(char *, struct key *, int);

/* С-дағы кілт сөздерді санау */
main()
{
      int n;
      char word[MAXWORD];

      while(getword(word, MAXWORD) != EOF)
            if (isalpha(word[0]))
                if ((n = binsearch(word, keytab, NKEYS)) >= 0)
                    keytab[n].count++;
      for (n = 0; n < NKEYS; n++)
            if (keytab[n].count > 0)
                 printf(«%4d %s\n», keytab[n].count, keytab[n].word);
return 0;
}

/* binsearch: найти слово в tab[0] ... tab[n - T] */
int binsearch(char *word, struct key tab[], int n)
{
     int cond;
     int low, high, mid;

     low = 0;
     high = n - 1 ;
     while (low <= high) {
           mid = (low + high)/2;
           if ((cond = strcmp(word, tab[mid].word)) < 0)
               high = mid - 1;
          else if (cond > 0)
               low = mid + 1;
```

```
            else
                return mid;
        }
        return -1;
    }
```

Кейінірек біз getword функциясын қарастырамыз, ал қазір біз оның әрбір шақыруында бірінші дәлелмен берілген массивте есте қалатын тағы бір сөз екенін білуіміз жеткілікті.

NKEYS-keytab-дағы кілт сөздердің саны. Біз осындай сөздердің санын қолмен есептей алар болсақ, оны машинамен жасау әлдеқайда жеңіл және қауіпсіз, әсіресе кілт сөздер тізімі өзгеруі мүмкін. Мүмкін шешімдердің бірі-инициализаторлар тізімінің соңына бос көрсеткішті (null) орналастыру және содан кейін соңғы элемент кездеспейінше keytab элементтерін циклде іріктеу.

Бірақ оңай шешім болуы мүмкін. Массивтің өлшемі компиляция кезінде толық анықталғандықтан және массивтің элементтерінің саны оның жеке элементінің өлшеміне тең болғандықтан, массивтің элементтерінің санын формула бойынша есептеуге болады.

```
keytab мөлшері / struct key мөлшері
```

С-да sizeof унарлы операторы бар, ол компиляция кезінде жұмыс істейді. Оны кез келген нысанның өлшемін есептеу үшін қолдануға болады. Өрнек

```
sizeof нысан
```

және

```
sizeof (аты-түрі)
```

көрсетілген нысанның немесе байттағы түрдің өлшеміне тең бүтін мәндерді береді. (Қатаң айтқанда, sizeof size_t түрі <stddef тақырып файлында анықталған белгісіз бүтін береді.h>.) Объектіге келетін болсақ, ол айнымалы, массив немесе құрылым болуы мүмкін.

Түрдің атауы негізгі типтің атауы болуы мүмкін (int, double ...) немесе алынған типтің атауы, мысалы, құрылым немесе сілтегіш.

Біздің жағдайда, кілт сөздердің санын есептеу үшін, массивтің мөлшерін бір элементтің өлшеміне бөлу керек. Көрсетілген есептеу nkeys мәнін орнату үшін #define нұсқаулығында қолданылады:

```
#define NKEYS (sizeof keytab / sizeof(struct key))
```

Дәл сол нәтижені басқа жолмен алуға болады — массивтің мөлшерін оның белгілі бір элементінің өлшеміне бөлуге болады:

```
#define NKEYS (sizeof keytab / sizeof keytab[0])
```

Мұндай жазбалардың артықшылығы-олардың түрін өзгерту кезінде түзетудің қажеті жоқ.

Prepressor түрлердің аттарына назар аудармағандықтан, sizeof операторы #if-да қолдануға болмайды. Бірақ #define-де препроцессор өрнегі есептелмейді, сондықтан біз ұсынған жазбаға жол беріледі.

Енді getword функциясы туралы сөйлесейік. Біз жазды getword бірнеше неғұрлым жалпы түрде, ол үшін талап етіледі біздің бағдарлама, бірақ ол осы болды, айтарлықтай қиын. Getword функциясы кіріс ағынынан келесі «сөз» алады. Сөз деп – әріптен басталатын әріптер-сандар тізбегі немесе бөлгіш-символдан өзгеше жеке символ.

```
/* getword: енгізуден келесі сөзді немесе таңбаны қабылдайды */
int getword (char *word, int lim)
{
    int c, getch(void);
    void ungetch(int);
    char *w = word;
    while (isspace(c = getch()))
        ;
    if (c != EOF)
        *w++ = c;
```

```
    if (!isalpha(c)) {
        *w = '\0';
        return c;
    }
    for ( ; --lim > 0; w++)
        if (!isalnum(*w = getch())) {
            ungetch(*w);
            break;
        }
    *w = '\0';
    return word[0];
}
```

Getword функциясы біз 4-тарауда айтып өткен getch және ungetch-ге жүгінеді. Әріптер жиынтығы аяқталғаннан кейін getword қосымша таңбаны алды. Ungetch-ге жүгіну оны кіріс ағынына кері қайтарады. Getword сондай-ақ isspace-таңбаларды – бөлгіштерді өткізу үшін, isalpha – әріптерді сәйкестендіру үшін және isalnum-әріптерді-сандарды тану үшін пайдаланылады. Олардың барлығы стандартты тақырып файлында сипатталған <ctype.h>.

6.1-жаттығу. Біздің getword нұсқасы астын сызу белгісін, жол тұрақтарын, түсініктемелерді және прпроцессордың басқару жолдарын дұрыс өңдемейді. Бағдарламаның жетілдірілген нұсқасын жазыңыз.

6.4. Құрылымға көрсеткіштер

Құрылым мен құрылым массивіндегі сілтемелерге қатысты кейбір сәттерді суреттеу үшін, индекстер орнына көрсеткіштер массив элементтерін алу үшін пайдалана отырып, түйінді сөздерді есептеу бағдарламасын қайта есептейміз.

Keytab массивінің сыртқы хабарландыруы өзгеріссіз қалады, ал main және binsearch өзгертілуі керек.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

```
#define MAXWORD 100

int getword(char *, int);
struct key *binsearch(char *, struct key *, int);

/* С кілт сөздерін санау: көрсеткіштері бар нұсқа */
main()
{
       char word [MAXWORD];
       struct key *p;
       while (getword(word, MAXWORD) != EOF)
              if (isalpha(word[0]))
                   if ((p = binsearch(word, keytab, NKEYS)) != NULL)
                        p->count++;
       for (p = keytab; p < keytab + NKEYS; p++)
             if (p->count > 0)
                  printf("%4d %s\n", p->count, p->word);
       return 0;
}

/* binsearch: word сөзін tab[0]...tab[n-1] табу */
struct key *binsearch(char *word, struct key *tab, int n)
{
      int cond;
      struct key *low = &tab[0];
      struct key *high = &tab[n];
     struct key *mid;

   while (low < high) {
         mid = low + (high - low) / 2;
         if ((cond = strcmp(word, mid->word)) < 0)
            high = mid;
         else if (cond > 0) ,
            low = mid + 1 ;
         else
            return mid;
    }
    return NULL;
 }
```

Бұл бағдарламаның кейбіреулері егжей-тегжейі түсіндіруді талап етеді. Біріншіден, binsearch функциясының сипаттамасы ол бүтін емес, struct key көрсеткішін қайтарады фактісін көрсетуі тиіс; бұл

функция прототипінде де, binsearch функциясында да жарияланған. Егер binsearch сөз тапса, онда ол оған көрсеткіш береді, әйтпесе ол қайтарады NULL. Екіншіден, keytab элементтеріне біздің бағдарламада қол жеткізу көрсеткіштер арқылы жүзеге асырылады. Бұл binsearch-да елеулі өзгерістер талап етті. Low және high үшін бастамашылары енді массивтің соңынан кейін бірден басына және орнына көрсеткіштер қызмет етеді. Формула арқылы орта элементтің орнын есептеу

```
mid = (low + high) / 2 /* ДҰРЫС ЕМЕС */
```

жарамайды, себебі, көрсеткіштерді қосуға болмайды. Дегенмен, оларға шегеру операциясын қолдануға болады және high-low элементтердің саны бар болғандықтан, меншіктеу

```
mid = low + (high-low) / 2
```

mid-ті low и high ортасында жатқан элементке айналдырады.

Бағдарламаның жаңа нұсқасына көшкен кезде ең маңыздысы — қате көрсеткіштер пайда болмайтындай және массивтен тыс шығуға әрекет жасамайтындай етіп жасау. Мәселе мынада &tab[1] және &tab[n] массивтің шекарасынан тыс. Бірінші мекен-жай дұрыс емес, екінші мекен-жайға кіруге болмайды. Тіл ережелері бойынша, алайда, массивтің соңында (яғни &tab[n]) келесі жад ұяшығының мекен-жайы көрсеткіштер арифметикасында дұрыс қабылданатынына кепілдік беріледі.

Main басты бағдарламасында біз былай жаздық:

```
for (p = keytab; p < keytab + NKEYS; p++)
```

Егер p — бұл құрылымға көрсеткіш болса, онда p-мен операцияларды орындау кезінде құрылымның мөлшері ескеріледі. Сондықтан p++ массивтің келесі құрылымдық элементіне шығу үшін, ал жағдайды тексеру циклді уақытында тоқтатады.

Алайда, құрылымның өлшемі оның элементтерінің өлшемдерінің сомасына тең деп ойлауға болмайды. Әр түрлі ұзындықтағы объектілерді теңестіру нәтижесінде құрылымда атаусыз "тесіктер" пайда болуы мүмкін. Мысалы, егер char типті айнымалы бір байт, ал int — төрт байт алса, онда құрылым үшін

```
struct {
    char c;
    int i;
};
```

бес емес, сегіз байт қажет болуы мүмкін. sizeof операторы дұрыс мәнді қайтарады.

Соңында, бағдарлама форматына қатысты бірнеше сөз. Егер функция күрделі түрдің мәнін қайтарса, мысалы, біздің жағдайда ол көрсеткішті құрылымға қайтарады:

```
struct key *binsearch(char *word, struct key *tab, int n)
```

онда "binsearch" функция атауы оңай емес. Мұндай жағдайларда кейде мына түрде де жазады:

```
struct key * binsearch(char *word, struct key *tab, int n)
```

Қандай формада артықшылық беру-әркімнің өз қалауында. Сізге ұнайтын нәрсені таңдап, оны ұстаныңыз.

6.5. Өзіне сілтемелері бар құрылымдар

Біз жалпы тапсырманы шешкіміз келеді делік, кіру ағынының кез келген сөздеріне қарсы алу жиілігін есептейтін бағдарлама жазу. Сөздер тізімі алдын ала белгісіз болғандықтан, біз оны алдын ала ұйымдастыруға және екілік іздеуді қолдана алмаймыз. Ол бұрын кездескен немесе жоқ анықтау үшін әрбір алынған сөзді сызықтық іздеуді пайдалану ақылға сыйымсыз болар еді – бұл жағдайда бағдарлама тым баяу жұмыс істейтін еді. (Дәлірек бағалау: бұл бағдарламаның жұмыс уақыты сөз санының квадратына

пропорционалды.) Еркін сөздер тізімін тиімді шешу үшін деректерді қалай ұйымдастыруға болады?

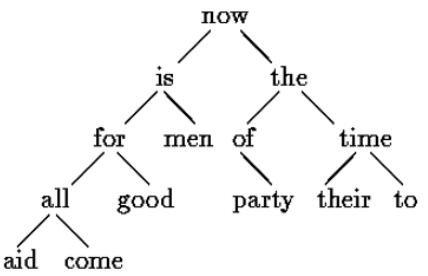
Тәсілдердің бірі - әрбір жаңа сөзді бар тәртіптілікті бұзбайтындай орынға қою арқылы алынған сөздердің тәртіптілігін үнемі сақтау. Бұл сызықты массивте сөздерді қозғалту керек емес, — кем дегенде, бұл процедура тым ұзақ болғандықтан. Оның орнына екілік ағаш деп аталатын деректер құрылымын қолданамыз.

Ағашта әрбір жеке сөзге "түйін" бар: сөздер мәтінінің көрсеткіші;

- кездесу санының санауышы;
- сол жақ торапқа көрсеткіш;
- оң жақ торапқа көрсеткіш;

Әр түйінде бір немесе екі ұл болуы мүмкін немесе түйінде мүлде ұл болмайды.

Ағаштағы түйіндер кез келген түйінге қатысты сол жақ ішкі тармақта тек лексикографиялық жағынан берілген түйін сөзінен кіші сөздерді, ал оң жақта одан үлкенірек сөздер болады. Міне, фраза үшін салынған ағаштың көрінісі - «now is the time for all good men to come to the aid of their party» («барлық жақсы адамдарға өз партияларына көмектесетін уақыт келді»), процестің соңында әр жаңа сөзге жаңа түйін қосылады:



Жаңадан келген сөздің ағашқа орналастырылғанын анықтау үшін, тамырдан бастаңыз, бұл сөзді түбір түйініндегі сөзбен салыс-

тырыңыз. Егер олар сәйкес келсе, онда сұрақтың жауабы оң болады. Егер жаңа сөз ағаштан шыққан сөзден аз болса, онда іздеу сол жақ тармақта, егер көп болса, оң жақта жалғасады.

Егер таңдалған бағытта бағыныңқы тармақ болмаса, онда бұл сөз ағашта жоқ, ал ішкі тармақтың жоқтығын білдіретін бос орын дәл сізге жаңа сөзбен түйінді «ілу» керек. Сипатталған процесс негізінен рекурсивті болып табылады, өйткені кез келген түйінде іздеу нәтижені филиал түйіндерінің бірінде іздейді. Тиісінше, түйін қосу және ағашты басып шығару үшін мұнда рекурсивті функцияларды қолдану тиімді.

Төрт компоненттен тұратын құрылым түрінде ыңғайлы түрде ұсынылған түйіннің сипаттамасына оралайық:

```
struct tnode { /* ағаш түйін */
    char *word; /* мәтіндік көрсеткіш */
    int count; /* кездесулер саны */
    struct tnode *left; /* сол жақ торап*/
    struct tnode *right; /* оң жақ торап*/
};
```

Берілген түйіннің рекурсивті анықтамасы қауіпті болып көрінуі мүмкін, бірақ ол дұрыс. Құрылым өзіне кіре алмайды, бірақ

```
struct tnode *left;
```

left-ді tnode-те емес, tnode-те көрсеткіш ретінде жариялайды.

Кейде өзара байланысқан құрылымдарға қажеттілік туындайды: бір-біріне сілтеме жасайтын екі құрылым. Бұл тапсырманы жеңуге мүмкіндік беретін әдіс келесі фрагментпен көрсетіледі:

```
struct t {
    …
    struct s *p; /* p s-ға нұсқайды*/
    };
struct s {
    …
    struct t *q; /* q t-ға нұсқайды */
};
```

Барлық бағдарлама таңқаларлық аз екендігі рас, ол біз жазған getword сияқты көмекші бағдарламаларды пайдаланады. Басты бағдарлама getword арқылы сөздерді оқиды және оларды addtree арқылы ағашқа енгізеді.

```
#include <stdio.h>
#include <ctype. h>
#include <string.h>

 #define MAXWORD 100

struct tnode *addtree(struct tnode *, char *);
void treeprint(struct tnode *);
int getword(char *, int);

/* сөздердің кездесу жиілігін есептеу */
main()
{
      struct tnode *root;
      char word[MAXWORD];

      root = NULL;
      while (getword (word, MAXWORD) != EOF)
            if (Isalpha(word[0]))
                root = addtree(root, word);
      treeprint(root);
      return 0;
}
```

Addtree функциясы рекурсивті. Бірінші сөз main функциясы ағаштың жоғарғы деңгейіне (ағаш түбірі) қояды. Әрбір жаңадан келіп түскен сөз түйін сөзімен салыстырылады және addtree-ге рекурсивті үндеу арқылы солға немесе оңға түседі. Біраз уақыттан кейін бұл сөз ағаштағы сөздердің кез келгенімен (бұл жағдайда есептеуішке 1 қосылады) сәйкес келеді, немесе бағдарлама жаңа түйін жасау және оны ағашқа қосу үшін сигнал болып табылатын бос позицияны кездестіреді. Жаңа торапты жасау addtree оған ата-ананың торабына салынған көрсеткішті қайтарады.

```
struct tnode *talloc(void);
char *strdup(char *);
```

```
/* addtree: p сөзіне немесе астына w сөзі бар түйін қосады */
struct tnode *addtree(struct tnode *p, char *w)
{
    int cond;
    if (p == NULL) { /* сөз алғаш рет кездеседі */
        p = talloc(); /* жаңа түйін жасалады */
        p->word = strdup(w);
        p->count = 1;
        p->left = p->right = NULL;
    } else if ((cond = strcmp(w, p->word)) == 0)
        p->count++; /* бұл сөз кездесті */
    else if (cond < 0) /* сол жақ түбірден кішкентай */
        p->left = addtree(p->left, w);
    else /* оң жақ түбірден үлкен */
        p->right = addtree(p->right, w);
    return p;
}
```

Жаңа түйін үшін жад talloc бағдарламасы арқылы сұралады, ол бір ағаш түйінін сақтау үшін жеткілікті бос орынға көрсеткіш қайтарады, ал жаңа сөзді бөлек жад орнына көшіру strdup көмегімен жүзеге асырылады. (Біз бұл бағдарламаларды кейінірек қарастырамыз.) Ағашқа жаңа торап ілінген кезде есептегіштің инициализациясы және тораптарға сілтегіштердің нөлденуі орын алады. Біз strdup және talloc мәндерін алған кезде орындалуға тиіс қателерді бақылауды төмендеттік.

Treeprint функциясы ағашты лексикографиялық ретпен басып шығарады; әрбір торап үшін ол сол жақ (осы торап сөзінен аз сөздер), содан кейін сөздің өзі және соңында оң жақ (осы торап сөзінен көп сөздер) басып шығарады.

```
/* treeprint: p ағашын реттеп, басып шығару */
void treeprint(struct tnode *p)
{
    if (p != NULL) {
        treeprint(p->left);
        printf("%4d %s\n", p->count, p->word);
        treeprint(p->right);
    }
}
```

Егер сіз рекурсияның қалай жұмыс істейтінін егжей-тегжейлі білмесеңіз, жоғарыда келтірілген ағашта treeprint әрекеттерін "жоғалтыңыз".

Практикалық ескерту: егер ағаш "теңгерілмеген" болса (сөздер кездейсоқ ретпен түспеген кезде не болады), онда бағдарламаның жұмыс уақыты өте арта алады. Сөздер қазірдің өзінде реттелген болса, бұл жағдайда есептеу шығындары сызықтық іздеу сияқты болады. Бұл кемшіліктен зардап шекпейтін екілік ағаштың жалпыламасы бар, бірақ біз оларды сипаттамаймыз.

Осы мысал талқылауды аяқтамас бұрын, тақырыптан қысқаша ауытқып, жадты сұрау механизмі туралы әңгімелестік. Әлбетте, бұл жады әр түрлі объектілерге арналған болса да, жадты бөлетін бір ғана функцияға ие болғым келеді. Бірақ егер бірдей функция жадты қамтамасыз етсе, айталық, char көрсеткіштері үшін де, struct node көрсеткіштері үшін де, екі сұрақ туындайды. Бірінші: белгілі бір түрдегі объектілер теңестірілуі тиіс машиналардың көпшілігінің талаптарын қалай жеңуге болады (мысалы, int жиі жұп мекен-жайлардан бастап орналасуы тиіс)? Екінші: әртүрлі түрдегі көрсеткіштерді қайтаруға мәжбүр болатын жад таратқышты қалай жариялау керек?

Жалпы айтқанда, теңестіру талаптарын кейбір артық жады есебінен оңай орындауға болады. Алайда, бұл үшін қайтарылатын көрсеткіш туралауға байланысты кез келген шектеулерді қанағаттандыратындай болуы тиіс. 5-тарауда сипатталған alloc функциясы кез келген нақты туралауға кепілдік бермейді, сондықтан біз мұны жасайтын стандартты malloc кітапханалық функциясын пайдаланамыз. 8-тарауда біз оны жүзеге асырудың бір жолын көрсетеміз.

Malloc сияқты функциялардың типтерін декларациялау туралы мәселе кез келген тілде қатаң түрдегі тексерумен бірге сүріну болып табылады. С-та сұрақ табиғи жолмен шешіледі: malloc сілтегішті босқа қайтаратын функция ретінде жарияланды. Алынған

көрсеткіш анық түрде қажетті типке беріледі. Malloc және онымен байланысты функциялардың сипаттамалары стандартты <stdlib. h> тақырып файлында орналасқан. Осылайша, talloc функциясын мына түрде жазуға болады:

```
#include <stdlib.h>

/* talloc: tnode жасайды */
struct tnode *talloc(void)
{
    return (struct tnode *) malloc(sizeof(struct tnode));
}
```

Strdup функциясы дәлелде көрсетілген жолды malloc көмегімен алынған орынға көшіреді:

```
char *strdup(char *s) /* s көшірмесін жасайды*/
{
    char *p;
    p = (char *) malloc(strlen(s)+1); /* +1 для '\0' */
    if (p != NULL)
        strcpy(p, s);
    return p;
}
```

Егер бос орын болмаса, malloc функциясы NULL мәнін қайтарады; strdup қате жағдайынан қоңырау шалушының қамын қалдырып, сол мәнді қайтарады.

Malloc арқылы алынған жадты бос функцияға қол жеткізу арқылы қайта пайдалануға босатуға болады (7 және 8-тарауларды қараңыз).

6.2-жаттығу. С бағдарламасының мәтінін оқып, алфавиттік тәртіпте барлық өзгермелі атаулардың барлық топтарын алғашқы 6 таңба сәйкес келеді, бірақ кейінгісі біршама ерекшеленетін программа жазыңыз. Ұсынылған жолдар мен түсініктемелердің ішкі жағын ұстамаңыз. 6 саны, пәрмен жолында көрсетілген параметрді жасаңыз.

6.3-жаттығу. «Кросс-сілтемелер» кестесін басып шығаруға бағдарлама жазыңыз, онда құжаттың барлық сөздері басылып

шығады және олардың әрқайсысы үшін табылған жол нөмірлері көрсетіледі. Бағдарлама «шу» сөздерін «және», «немесе», т.б. ескермеуі керек.

6.4-жаттығу. Кіріс ағынын құрайтын әр түрлі сөздердің жиынтығын олардың пайда болу жиілігін арттыру үшін басып шығаратын программа жазыңыз. Әр сөздің алдында қайталану саны көрсетілуі керек.

6.6. Кестелерді қарау

Бұл параграфта құрылымдарды қолданудың жаңа аспектілерін көрсету үшін, біз кестелерге элементтерді кірістіруді және оларды кестелер ішінде іздеуді жүзеге асыратын бағдарламалар пакетінің өзегін жазамыз. Бұл пакет — кез келген макропроцессор немесе компилятордағы атау кестелерімен жұмыс істейтін бағдарламалардың типтік жиынтығы. Мысалы, #define нұсқаулығын қарастырайық. Мына түрдегі жол кездескен кезде

```
#define IN 1
```

IN аты және оны алмастыратын 1 мәтіні кестеде есте сақталуы тиіс. Егер IN аты нұсқаулықта кездессе, мысалы

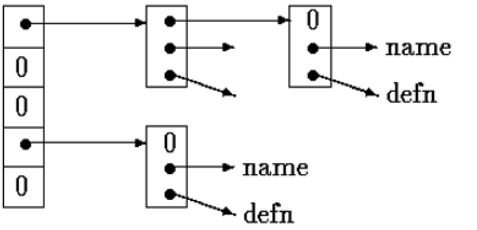
```
state = IN;
```

ол 1-ге ауыстырылуы керек.

Аттармен және олардың мәтіндерімен манипуляцияланатын екі бағдарлама бар. Бұл install(s, t), ол s атауын және оның орнын ауыстыратын мәтінді t кестеге жазады, мұндағы s және t - жолдар және lookup(s) кестені іздеу және көрсеткішті s табылған орынға қайтару, немесе NULL, егер s кестеде болмаса.

Алгоритм хэш іздеуге негізделген: кіріс атауы теріс емес санға (хэш код) жинақталады, ол кейін көрсеткіштер қатарында индекс ретінде қолданылады. Бұл алаптың әр элементі осы хэш-кодымен атауларды сипаттайтын блоктардың байланыстырылған тізімінің

басында болады. Егер массив элементі NULL болса, сәйкес хэш-коды бар атаулар жоқ екенін білдіреді.



Тізімдегі блок – бұл атау көрсеткіштері бар құрылым, ауыстыратын мәтін және тізімдегі келесі блок; келесі блоктағы көрсеткіш NULL мәні тізімнің соңын білдіреді.

```
struct nlist { /* кесте элементі */
struct nlist *next; /* келесі элементке көрсеткіш */
    char *name; /* белгілі бір атау */
    char *defn; /* ауыстыратын мәтін */
};
```

Ал көрсеткіштер массивінің анықтамасы қалай жазылады:

```
#define HASHSIZE 101
static struct nlist *hashtab[HASHSIZE]; /* көрсеткіштер кестесі */
```

Lookup және install-да қолданылатын хэширлеу функциясы жолдағы таңбалар кодын қосады және нәтиже ретінде алынған соманы көрсеткіштер массивінің мөлшеріне бөлуден қалған соманы береді. Бұл ең жақсы хэшинг функциясы емес, бірақ өте қысқа және тиімді.

```
/* hash: s жолы үшін хэш кодын алады */
unsigned hash(char *s)
{
    unsigned hashval;

    for (hashval = 0; *s != '\0'; s++)
        hashval = *s + 31 * hashval;
    return hashval % HASHSIZE;
}
```

Таңбасыз арифметика хэш коды теріс емес болады деп кепілдік береді.

Хэшрлеу массив hashtab үшін бастапқы индексті тудырады; кестеде тиісті жол бар болса, ол осы индексі бар hashtab массив элементін көрсетеді, оның басында блоктар тізімінде ғана анықталуы мүмкін. Іздеу lookup арқылы жүзеге асырылады. Егер lookup берілген жолы бар элементті тапса, онда оған көрсеткіш қайтарады, егер таппаса, онда null қайтарады.

```
/* lookup: s іздеуде*/
struct nlist *lookup(char *s)
{
    struct nlist *np;

    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s, np->name) == 0)
            return np; /* таптық */
    return NULL; /* таппадық */
}
```

Lookup функциясының for-циклында тізімді көру үшін стандартты дизайн қолданылады:

```
for (ptr = head; ptr != NULL; ptr = ptr->next)
```

Install функциясы орнатылып қойған атының бар-жоғын анықтау үшін lookup қызметіне жүгінеді. Егер солай болса, ескі анықтама жаңасына ауыстырылады. Әйтпесе, жаңа элемент пайда болады. Егер жаңа элемент үшін жад сұрауын қанағаттандыру мүмкін болмаса, install функциясы null береді.

```
struct nlist *lookup(char *);
char *strdup(char *);

/* install: кестеге аты мен мәтінін (name, defn) енгізеді */
struct nlist *install(char *name, char *defn)
{
    struct nlist *np;
    unsigned hashval;
```

```
    if ((np = lookup(name)) == NULL) { /* табылмады */
        np = (struct nlist *) malloc(sizeof(*np));
        if (np == NULL || (np->name = strdup(name)) == NULL)
            return NULL;
        hashval = hash(name);
        np->next = hashtab[hashval];
        hashtab[hashval] = np;
    } else /* бар */
        free((void *) np->defn); /* бұрынғы defn босатамыз */
    if ((np->defn = strdup(defn)) == NULL)
        return NULL;
    return np;
}
```

6.5-жаттығу. Lookup және install функциялары қолдайтын кестеден атау мен анықтаманы алып тастайтын undef функциясын жазыңыз.

6.6-жаттығу. #Define-процессордың қарапайым нұсқасын (дәлелсіз) іске асырыңыз. Ол осы параграфтың бағдарламаларын пайдаланады және С-бағдарламалар үшін жарамды болатындай болсын. Сізге getch және ungetch бағдарламалары көмектесе алады.

6.7. Typedef құралы

С тілі typedef деп аталатын құралды ұсынады, ол деректердің типтеріне жаңа аттар беруге мүмкіндік береді. Мысалы,

```
typedef int Length;
```

Length атауын int синонимі етеді. Осы сәттен бастап Length түрін хабарландыруда, келтіру операторында және т. б. қолдануға болады.:

```
Length len, maxlen;
Length *lengths[];
```

Дәл сол сияқты

```
typedef char *String;
```

String Char* синонимін жасайды, яғни char көрсеткіші және оның келесі көрсетілген түрін пайдалану бірдей болады:

```
String p, lineptr[MAXLINES], alloc(int);
int strcmp(String, String);
p = (String) malloc(100);
```

Typedef-да жарияланған түр typedef сөзінің артында бірден емес, әдеттегі хабарландыруда айнымалы атының орнында тұрғанын ескеріңіз. Синтаксис тұрғысынан typedef сөзі жады класына ұқсайды – extern, static және т.б. түрлердің аттары бас әріптерден бөлініп шығу үшін жазылған.

Typedef қолданудың күрделі мысалдарын көрсету үшін осы тарауда кездескен ағаш түйіндерін беру кезінде осы құралды қолданамыз.

```
typedef struct tnode *Treeptr;
typedef struct tnode { /* ағаш түйіні: */
    char *word; /* мәтін көрсеткіші */
    int count; /* кіру саны */
    Treeptr left; /* сол жақ торап */
    Treeptr right; /* оң жақ торап */
} Treenode;
```

Нәтижесінде екі жаңа түр атауы жасалады: Treenode (құрылым) және Treeptr (құрылым көрсеткіші). Енді talloc бағдарламасын келесі түрде жазуға болады:

```
Treeptr talloc(void)
{
    return (Treeptr) malloc(sizeof (Treenode));
}
```

Typedef жариялауы жаңа түрдегі хабарландыруларды жасамайтынын атап өту керек, ол тек қолданыстағы түрге жаңа атауды хабарлайды. Бұл жаңа атаулардың ешқандай жаңа мағынасы жоқ, олар түрдің атын өзгертпей тікелей жарияланғандай дәл сол қасиеттері бар айнымалыларды жариялайды. Шын мәнінде typedef

#define-ге ұқсас, компиляторды түсіндіргенде, ол препроцессормен өңделе алмайтын мәтіндік көшірмені жеңе алады. Мысалы:

```
typedef int (*PFI)(char *, char *);
```

PFI түрін жасайды – “Int қайтаратын функцияға көрсеткіш (char * екі аргументі)”, мысалы, 5-тарауда сипатталған сұрыптау бағдарламасында осы контексте қолдануға болады:

```
PFI strcmp, numcmp;
```

Тек эстетикалық ойлардан басқа, typedef қолдану үшін екі маңызды себеп бар. Оның бірі – төзімділік проблемасына байланысты бағдарламаның параметрленуі. Егер typedef көмегімен машинаға тәуелді болуы мүмкін деректер түрлерін жарияласа, бағдарламаны басқа машинаға ауыстырған кезде typedef анықтамаларына ғана өзгерістер енгізу қажет болады. Ортақ жағдайлардың бірі – мақсатты шамаларды өзгерту үшін typedef атауларды пайдалану. Әрбір нақты машина үшін бұл size_t және ptrdiff_t сияқты стандартты типтегі қондырғыларға ұқсас жасалатын тиісті қысқа, int немесе ұзын орнатуды қамтиды.

Typedef-ті қолдануға итермелейтін екінші себеп – бағдарламаның мәтінін анық ету. Treeptr деп аталатын түр (tree – ағаш және pointer – көрсеткіш ағылшын сөзінен), кейбір күрделі құрылымға көрсеткіш ретінде жазылған сол түрге қарағанда түсінікті.

6.8. Бірлестіктер

Бірлестік – бұл айнымалы, ол әр түрлі типтегі және өлшемдегі объектілерді қамтуы мүмкін (уақыттың әр түрлі сәттерінде). Өлшемдерге және теңестіруге қатысты барлық талаптарды компилятор орындайды. Біріктірулер әртүрлі деректерді бағдарламаға машиналық-тәуелді ақпаратты қоспай бір жад аумағында сақтауға мүмкіндік береді. Бұл құралдар Паскалдағы нұсқалық жазбаларға ұқсас.

Мысалы, егер константа int, float түрі немесе символға көрсеткіш болып табылуы мүмкін және char *түрі болуы мүмкін деп болжайтын болса, онда компилятордың өзі, символдар кестесін меңгерушісі бола алады. Әрбір нақты константаның мәні осы түрдің айнымалысында сақталуы керек. Егер мәндер көлемі бірдей жады болса және өз түріне қарамастан бір жерде есте қалса, таңбалар кестесімен жұмыс істеу әрдайым ыңғайлы. Бірлестік бағдарламасына енгізудің мақсаты — заңды негізде бірнеше түрдің мәнін сақтайтын айнымалы болуы. Бірлестік синтаксисі құрылым синтаксисіне ұқсас. Бірлестікке мысал келтірейік.

```
union u_tag {
    int ival;
    float fval;
    char *sval;
} u;
```

U айнымалысы көрсетілген үш түрдің кез келген айнымалысы сәйкес келуі үшін жеткілікті үлкен болады; оның нақты мөлшері іске асырылуға байланысты. Осы үш түрдің бірінің мәні айнымалы болып берілуі мүмкін және одан әрі, егер бұл заңды болса, яғни, ол алған мәннің түрі оған соңғы берілген мәннің түрімен сәйкес келсе, өрнектерде пайдаланылуы мүмкін. Бұл талапты орындау әрбір ағымдағы сәтте — толығымен программистің ар-ожданында. Егер бір нәрсе бір түрдің мәні ретінде есте қалса және басқа түрдің мәні ретінде алынса, нәтиже іске асырылуға байланысты.

Біріктіру элементтеріне қатынау синтаксисі:

```
аты-бірлестік.элемент
```

немесе

```
көрсеткіш-біріктіру - >элемент
```

яғни құрылымдар да дәл осындай. Егер ағымдағы мән түрін сақтау үшін және utype айнымалысын қолдансаңыз, бағдарламаның мұндай бөлігін жазуға болады:

```
if (utype == INT)
    printf("%d\n", u.ival);
else if (utype == FLOAT)
    printf("%f\n", u.fval);
else if (utype == STRING)
    printf("%s\n", u.sval);
else
    printf ("қате түр %d в utype\n", utype);
```

Бірлестіктер құрылымдар мен массивтерге, керісінше кіруі мүмкін. Құрылымдағы (бірлестіктегі құрылым сияқты) бірлестік элементіне кіру жазбасы салынған құрылымдардағыдай. Мысалы, құрылымдар массивінде

```
struct {
     char *name;
     int flags;
     int utype;
     union {
          int ival;
          float fval;
          char *sval;
     } u;
} symtab[NSYM];
```

ival-ға келесі түрде жүгінеді:

```
symtab[i].u.ival
```

ал sval жолының бірінші таңбасына келесі екі тәсілдің кез келгеніне жүгінуге болады:

```
*symtab[i].u.sval symtab[i].u.sval[0]
```

Іс жүзінде біріктіру – бұл барлық элементтері оның базалық мекен-жайына қатысты нөлдік ауытқуы бар және оның мөлшері оның ең үлкен элементіне сәйкес келуге мүмкіндік беретін құрылым, ал бұл құрылымды теңестіру бірлестіктің барлық түрлерін қанағаттандырады. Құрылымдарға қолданылатын операциялар бірлестіктер үшін де жарамды, яғни бірлестікті беру және оны біртұтас тұтас ретінде көшіру, бірлестіктен мекенжай алу және оның жекелеген элементтеріне қол жеткізуге де болады.

Біріктіруді оның бірінші элементінің түрі бар мәнмен ғана бастамалауға болады; осылайша жоғарыда аталған u айнымалысын тек int түрінің мәнімен ғана бастамалауға болады.

8-тарауда (жадты бөлуші бағдарлама мысалында) біз біріктіруді қолдану арқылы айнымалының орналасуы жадтың тиісті шекарасымен теңестірілуі үшін қол жеткізуге болатынын көрсетеміз.

6.9. Бит өрістері

Жадтың жетіспеушілігімен машинаның бір сөзіне бірнеше нысанды салу қажет болуы мүмкін. Компиляторларға арналған таңбалар кестелерін өңдеу мәселелерінде кездесетін қарапайым жағдайлардың бірі бір биттік жалаулар топтарын біріктіру. Кейбір деректердің пішімдері бізге мүлдем тәуелді болмауы мүмкін, мысалы, сыртқы құрылғылардың интерфейсімен байланысты болуы мүмкін; бұл жерде сөздің бөліктеріне қатысты қажеттілік туындайды.

Бір компилятор фрагментін елестетіп көріңіз, ол таңбалар кестесін басқарады. Бағдарламаның әр идентификаторы онымен байланысты ақпаратқа ие: мысалы, ол кілт сөзді білдіре ме, егер ол айнымалы болса, қай сыныпқа жатады: сыртқы және/немесе статикалық және т.б. Мұндай ақпаратты кодтаудың ең ықшам тәсілі – бір биттік жалаушаларды бір char немесе int түріндегі сөзге орналастыру.

Биттермен жұмыс жасаудың кең таралған әдістерінің бірі осы биттердің позицияларына сәйкес келетін “маскалар” жиынтығын анықтауға негізделген, мысалы,

```
#define KEYWORD 01 /* кілт сөз */
#define EXTERNAL 02 /* ішкі */
#define STATIC 04 /* статикалық */
```

немесе

```
enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };
```

Сандар екілік дәрежелері болуы керек. Содан кейін биттерге қол жеткізу 2-тарауында сипатталған (жылжу, жасырыну, қосымша алу) “бинарлық операциялары” ісіне айналады.

Өрнектерді жазудың кейбір түрлері жиі кездеседі. Сонымен,

```
flags |= EXTERNAL | STATIC;
```

flags айнымалы бит сәйкес 1 орнатады,

```
flags &= ~(EXTERNAL | STATIC);
```

оларды нөлдейді, ал

```
if ((flags & (EXTERNAL | STATIC)) ==0) ...
```

шартты екі бит нөлдік болса, шынайы деп бағалайды.

Мұндай өрнектерді жазуды үйрену қиын емес болса да, бинарлы логикалық операциялардың орнына С-ға берілетін тікелей анықтаудың және сөздің ішіндегі өрістерге қол жеткізудің басқа да әдісін қолдануға болады. Бит өрісі (немесе қысқа өріс үшін) - бұл “сөз”деп аталатын жад бірлігін іске асыруға байланысты бір-біріне жақын жатқан көптеген биттер. Өрістерді анықтау және оларға қол жеткізу синтаксисі құрылым синтаксисіне негізделеді. Мысалы, таңбалар кестесін орнатқанда жоғарыда көрсетілген #define жолдарын үш өрісті анықтауға ауыстыруға болады:

```
struct {
    unsigned int is_keyword : 1;
    unsigned int is_extern : 1;
    unsigned int is_static : 1;
} flags;
```

Бұл жазба үш бір биттік өрісті қамтитын flags айнымалысын анықтайды. Саны, келесі үшін қос нүкте, салықтар ені өріс. Өрістер unsigned int ретінде жарияланды, олар unsigned шамалар ретінде қабылданады.

Жеке өрістерге қарапайым құрылым элементтері сияқты сілтеме жасайды: flags.is_keyword, flags.is_extern және т.б. өрістер “өздерін ұстайды” шағын бүтін және арифметикалық өрнектерге басқа бүтін сияқты қатыса алады. Осылайша, алдыңғы мысалдар табиғи түрде жазылуы мүмкін:

```
flags.is_extern = flags.is_static = 1;
```

1-ді тиісті биттерге орнатады;

```
flags.is_extern = flags.is_static = 0;
```

оларды нөлдейді,

```
if (flags.is_extern == 0 && flags.is_static == 0)
```

оларды тексереді.

Өрістерге қатысты барлық техникалық бөлшектер, атап айтқанда, өрістің сөз шегіне өту мүмкіндігі іске асыруға байланысты. Өрістердің аты болмауы мүмкін; атаусыз өрістің (тек қос нүктемен және енмен берілетін) көмегімен разрядтардың қажетті санын өткізу ұйымдастырылады. Нөлге тең арнайы ені келесі сөздің шекарасына шығу қажет болғанда пайдаланылады.

Бір машиналарда өріс солдан оңға, екіншісінде — оңнан солға орналастырылады. Бұл дегеніміз, олармен жұмыс істеудің барлық пайдалылығы кезінде, егер біз жұмыс істейтін деректердің форматы бізге одан да көп болса, онда алаңдардың орналасу тәртібін мұқият зерттеу қажет; осындай заттарға байланысты бағдарламалар тасымалданбайды. Өрістерді тек int түрімен ғана анықтауға болады, ал төзімділікті қамтамасыз ету үшін signed немесе unsigned-ді анық көрсету керек. Олар массив болуы мүмкін емес және мекен-жайы жоқ, сондықтан оператор & оларға қолданылмайды.

7-тарау. Кіру және шығу

Енгізу және шығару мүмкіндіктері С тілінің бір бөлігі емес, сондықтан біз оларды егжей-тегжейлі қарастырмадық. Сонымен қатар, нақты бағдарламалар қоршаған ортамен бұрынғыға қарағанда әлдеқайда күрделі тәсілмен өзара іс-қимыл жасайды. Бұл тарауда біз енгізу-шығаруды, жолдармен жұмыс істеуді, жадыны басқаруды, стандартты математикалық функцияларды және түрлі сервистік С-бағдарламаларды қамтамасыз ететін функциялар жиынтығы бар стандартты кітапхананы сипаттаймыз. Бірақ енгізу-қорытындыға ерекше назар аударамыз.

Кіріс-шығыс кітапхана функциялары ANSI стандартымен анықталады, сондықтан олар С қолдайтын кез келген жүйелерде үйлесімді. Стандартты кітапхананың мүмкіндіктері шеңберінен шықпайтын жүйелік ортамен өзара әрекеттестікте бағдарламаларды өзгеріссіз бір машинадан екіншісіне ауыстыруға болады.

Кітапханалық функциялардың қасиеттері оннан астам тақырып файлдарында сипатталған; сіз олардың кейбіреулері кездесті, соның ішінде <stdio.h>, <string.h> және <ctype.h>. Біз мұнда барлық кітапхананы қарастырмаймыз, өйткені бізді кітапханалық функцияларды қолданудан гөрі С-бағдарламалар жазу қызықтырады. Стандартты кітапхана В қосымшасында егжей-тегжейлі сипатталған.

7.1. Стандартты енгізу-шығару

1-тарауда айтылғандай, кітапханалық функциялар мәтіндік енгізу-шығарудың қарапайым моделін жүзеге асырады. Мәтіндік ағын жолдар тізбегінен тұрады; әрбір жол жаңа жолдың символымен аяқталады. Егер жүйе қабылданған модельде болмаса, кітапхана бұл модель толығымен қанағаттандырылады. Мысалы, екі таңба — қайтару-кареткалар және аудару-жолдар — енгізу кезінде жаңа жолдың бір символына түрленуі мүмкін, ал шығару кезінде кері түрленуі мүмкін.

Ең қарапайым енгізу механизмі — getchar функциясымен стандартты енгізуден (әдетте пернетақтадан) бір таңбаны оқу:

```
int getchar(void)
```

Әрбір қоңыраудың нәтижесі ретінде getchar келесі енгізу таңбасын қайтарады немесе файлдың соңы анықталса, EOF. EOF деп аталатын тұрақты (end of file — файлдың соңы) <stdio деп анықталған. h>. Әдетте EOF мәні -1-ге тең, бірақ осы константаның нақты мәніне тәуелді емес, оны атаңыз (EOF).

Көптеген жүйелерде пернетақтаны < белгішесі арқылы енгізуді қайта бағыттау арқылы файлмен ауыстыруға болады. Мәселен, егер prog бағдарламасы getchar пайдаланса, командалық жол

```
prog < infile
```

prog бағдарламасы пернетақтадан емес, infile таңбаларын оқып шығады. Енгізуді ауыстыру prog бағдарламасының өзі алмастыруды байқамайды; атап айтқанда "<infile" жолы argv пәрмен жолының аргументтеріне қосылмайды. Егер енгізу басқа бағдарламадан шыққан болса және конвейерлік жолмен берілетін болса, енгізу де байқалмайды. Кейбір жүйелерде командалық жол

```
otherprog | prog
```

бұл екі бағдарламаның іске қосылуына әкеледі, otherprog және prog, және стандартты otherprog шығару стандартты prog енгізуге түседі.

```
int putchar(int)
```

функциясы шығару үшін пайдаланылады: putchar (c) с таңбасын стандартты экранға жібереді, бұл әдетте экранды білдіреді. Putchar функциясы жіберілген таңбаны немесе қате болған жағдайда EOF нәтижесін береді. Шығу үшін де дәл солай: > файл атауын жазу арқылы нәтижені файлға бағыттауға болады. Мысалы, егер prog шығару үшін putchar функциясын қолданса, онда

```
prog > outfile
```

стандартты қорытындыны экранға емес, outfile-ге бағыттайды. Командалық жол

```
prog | anotherprog
```

prog бағдарламасының стандартты шығуын anotherprog бағдарламасының стандартты енгізуімен байланыстырады.

Printf функциясы жүзеге асыратын қорытынды стандартты шығыс ағынына да жіберіледі. Putchar және printf қоңыраулары кез келген уақытта ауысып кетуі мүмкін, ал бұл мүмкіндіктердің қоңыраулары болған ретпен шығарылады.

Кез келген бастапқы енгізу-шығару кітапханасының кем дегенде бір функциясын пайдаланатын С-файлда жол болуы керек

```
#include <stdio.h>
```

және ол енгізу-шығаруға бірінші өтініш бергенге дейін орналасуы тиіс. Егер тақырып файлының атауы бұрыштық жақшада < және > болса, бұл тақырып файлын іздеу стандартты жерде (мысалы, UNIX жүйесінде әдетте /usr/include директориялары) жүргізілетінін білдіреді.

Көптеген бағдарламалар тек бір кіріс ағынынан оқиды және тек бір демалыс ағынына жазады. Мұндай бағдарламаларды енгізу-шығаруды ұйымдастыру үшін getchar, putchar және printf функциялары жеткілікті, ал бастапқы оқыту үшін осы мүмкіндіктермен танысу жеткілікті. Атап айтқанда, аталған функциялар бір бағдарламаны келесімен қосу қажет болғанда жеткілікті. Мысал ретінде, төменгі регистрге енгізетін lower бағдарламасын қарастырайық:

```
«include <stdio.h>
«include <ctype.h>

main() /* lower: төменгі регистрге енгізеді */
{
```

```
        int c;

        while ((c = getchar()) != EOF)
              putchar(tolower(c));
         return 0;
}
```

Толеранттылық функциясы <ctype.h> анықталған. Бас әріптерді кіші әріптерге ауыстырады және қалған таңбаларды өзгеріссіз қайтарады. Жоғарыда айтылғандай, <stdio.h> кітапханасынан getchar және putchar сияқты «функциялар» және <ctype.h> кітапханасындағы толтыру функциясы көбінесе әр жеке кейіпкерге арналған функцияны шақырудың үстеме шығындарын жою үшін макростар түрінде орындалады. 8.5-бөлімінде біз мұның қалай жасалатынын көрсетеміз. <Ctype.h> кітапханасының функциялары белгілі бір машинада қалай жүзеге асырылатынына қарамастан, оларды қолданатын бағдарламалар таңбаларды кодтау туралы ештеңе білмеуі мүмкін.

7.1-жаттығу. Мәтін argv [0] -ке байланысты қалай аталатынына байланысты кірісті үлкен регистрден кіші немесе кіші әріпке ауыстыратын программа жазыңыз.

7.2. Формат шығысы (printf)

Printf функциясы ішкі мәндерді мәтінге аударады.

```
int printf(char *format, argr arg2, ...)
```

Алдыңғы тарауларда біз printf формасын бейресми түрде қолдандық. Мұнда біз осы функцияны қолданудың ең типтік жағдайларын көрсетеміз; оның толық сипаттамасы В қосымшасында келтірілген.

Printf функциясы форматты түрлендіреді, форматтайды және форматты басқарумен стандартты шығыс түрінде басып шығарады. Ол басылған таңбалар санын қайтарады.

Формат жолында нысандардың екі түрі бар: шығыс ағынына тікелей көшірілетін қарапайым таңбалар және конверсия сипаттама-

лары, олардың әрқайсысы келесі printf дәлелін түрлендіруге және басып шығаруға себеп болады. Айырбастаудың кез келген сипаттамасы% -дан басталады және жіктеуішпен аяқталады. % және жіктеуіш таңбалары арасында келесі элементтерді орналастыруға болады (төменде көрсетілген ретпен):

▪ Минус белгісі түрлендірілген дәлелді өрістің сол жақ шетіне туралауды нұсқайды.

▪ Өрістің минималды енін көрсететін сан. Түрлендірілген дәлел кемінде көрсетілген енін алады. Қажет болса, сол жақтағы қосымша орындар (немесе оң жақта - сол жағымен) бос орындармен толтырылады.

▪ Өрістің енін дәлдікті орнататын мәннен бөлетін нүкте.

▪ Бір жолға басып шығарылатын таңбалардың максималды санын немесе өзгермелі нүкте сандарының ондық үтірінен кейінгі цифрларды немесе бүтін сандардың минималды санын көрсететін сан (дәлдік).

▪ h әрпі, егер басып шығарылатын тұтас short ретінде қарастырылуға тиіс болса, немесе l (ell латын әрпі), егер тұтас long ретінде қарастырылуға тиіс болса.

Арнайы таңбалар 7.1-кестеде көрсетілген. Егер % үшін спецификатор таңба қойылмаса, printf функциясының әрекеті анықталмайды.

Символдар	**Аргументтің түрі; басып шығару түрі;**
d, i	int; ондық бүтін
o	int; белгісіз сегіздік (octal) бүтін (сол жақта нөлсіз)
x, X	unsigned int; unsigned он алтылық бүтін (0x немесе 0X сол жақта), 10 үшін...15 abcdef немесе ABCDEF пайдаланылады
u	int;ондық бүтін
c	int; бір таңба
s	char *; \0 белгісіне дейін немесе дәлдікпен берілген санда орналасқан таңбаларды басып шығарады
f	double; [- + m. ddddddd, онда d сандар саны дәл орнатылады (әдепкі бойынша 6-ға тең)

e, E	double; [-] m. ddddddе+xx немесе [-] m.dddddd E±xx, онда d сандар саны дәлдікпен анықталады (әдепкі бойынша 6-ға тең))
g, G	double; %e немесе %e қолданады, егер реті -4-тен аз болса немесе дәлдікке тең болса; ал, керісінше жағдайда %f қолданады. соңғы нөлдер және соңғы ондық нүкте басылмайды
p	void *; көрсеткіш (көрініс іске асыруға байланысты)
%	Дәлел өзгертілмейді; % белгі басылады

Ені мен дәлдігін * көмегімен ерекшелеуге болады; бұл жағдайда ен (немесе дәлдік) мәні келесі аргументтен алынады (int түрі болуы тиіс). Мысалы, s жолынан max таңбаларды басып шығару үшін келесі жазба жарамды:

```
printf("%.*s", max, s);
```

Форматтық өзгерістердің басым бөлігі алдыңғы тарауларда көрсетілді. Жолдар үшін дәлдік тапсырмасы ерекше. Одан әрі ерекшеліктер тізбесі келтіріледі және олардың 12 символдан тұратын "hello, world" жолының басып шығаруына әсері көрсетіледі. Оның ұзындығы көрінуі үшін өріс қос нүктемен арнайы қоршалған.

```
:%s:          :hello, world:
:%10s:        :hello, world:
:%.10s:       :hello, wor:
:%-10s:       :hello, world:
:%.15s:       :hello, world:
:%-15s:       :hello, world   :
:%15.10s:     :     hello, wor:
:%-15.10s:    :hello, wor     :
```

Ескерту: printf функциясы аргументтердің қанша күтілетінін және олардың қандай түрін анықтау үшін алғашқы аргументті пайдаланады. Аргументтер жеткіліксіз болса немесе олар басқа түрге тиесілі болса, дұрыс нәтиже алмайсыз. Сіз сондай-ақ, келесі 2 өрнектердегі айырмашылықтарды түсіне білуіңіз қажет:

```
printf(s); /* НЕВЕРНО, егер s-да % бар болса, */
printf("%s", s); /* әрқашан ДҰРЫС */
```

Sprintf функциясы printf сияқты түрлендіруді орындайды, бірақ нәтижені шығару жолда есте сақтап қалады.

```
int sprintf(char *string, char *format, arg1, arg2, ...)
```

Бұл функция arg1, arg2 және т.б. пішімдейді, format дәлелімен берілген ақпаратқа сәйкес, біз бұрын сипаттағанымыздай, бірақ нәтиже стандартты қорытындыға емес, string-де орналастырылады. Айта кету керек, string жолы нәтижеге жету үшін жеткілікті үлкен болуы керек.

7.2-жаттығу. Кез келген енгізуді ақылға қонымды түрде басып шығаратын бағдарламаны жазыңыз. Кем дегенде ол графикалық емес таңбаларды сегіз немесе он алтылық түрінде (машинада қабылданған пішінде), ұзын мәтіндік жолдарды үзіп басып шығара алуы керек.

7.3. Айнымалы ұзындықтың аргумент тізімдері

Бұл параграф printf минималды нұсқасын іске асыруды қамтиды. Бұл функцияны ұзындығы айнымалы аргументтер тізімдерімен қалай жазу керектігін көрсету үшін келтіріледі және олар тасымалданады. Біз негізінен аргументтерді өңдеуге мүдделі болғандықтан, minprintf функциясын, ол негізінен, пішімді пішіммен және аргументтермен жұмыс істейтін етіп жазады; пішімді түрлендірулерге қатысты болса, олар стандартты printf арқылы жүзеге асырылады.

Стандартты printf функциясы:

```
int printf(char *fmt, ...)
```

Функциядағы көп нүктелер аргументтердің саны мен түрлері өзгеруі мүмкін дегенді білдіреді. Көп нүкте белгісі аргументтер

тізімінің соңында ғана тұра алады. Біздің minprintf функциясы ретінде жарияланады:

```
void minprintf(char *fmt, ...)
```

ол printf сияқты таңбалар санын бермейді.

Барлық қиындық-minprintf дәлелдер тізімінің бойымен қалай жылжуында, себебі бұл тізім тіпті аты жоқ. Стандартты <stdarg.h> тақырып файлында аргументтер тізімімен қалай өту керектігін анықтайтын макро анықтамалардың жиынтығы бар. Бұл тақырып файлын толтыру машинадан машинаға өзгеруі мүмкін, бірақ олар ұсынған интерфейс барлық жерде бірдей.

Va_list түрі аргументтердің әрқайсысына кезекпен көрсететін айнымалыны сипаттау үшін қызмет етеді; minprintf — да бұл айнымалының ар атауы бар (“argument pointer” - дәлел). Макрос va_start ар айнымалысын бірінші атаусыз аргументті көрсетеді. Va_start-ке ар алғаш пайдаланғанға дейін хабарласу керек. Аргументтер арасында, кем дегенде, біреуі атауы болуы керек; соңғы атаулы аргументтен бастап, бұл макрос бастапқы орнату кезінде “кері кетеді”.

Макрос va_arg өзінің әрбір шақыруында келесі аргументті береді, ал ар келесіге жылжытады; түрі бойынша ол қайтарылатын мәннің түрін және келесі аргументке шығу үшін қадамның өлшемін анықтайды. Ақырында, макрос va_end қажет нәрсені тазалайды. Va_end-ге функциядан шығу алдында жүгіну керек.

Бұл құралдар басып шығарудың қарапайым printf нұсқасының негізін құрайды.

```
#include <stdarg.h>

/* minprintf: аргументтің айнымалы саны бар минималды printf */
void minprintf(char *fmt, ...)
{
    va_list ap; /* келесі аты жоқ аргументті көрсетеді */
    char *p, *sval;
```

```
    int ival;
    double dval;

    va_start(ap, fmt); /* 1-ші атаусыз аргументке ар орнатады */
    for (p = fmt; *p; p++) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
        switch (*++p) {
        case 'd':
            ival = va_arg(ap, int);
            printf ("%d", ival);
            break;
        case 'f':
            dval = va_arg(ap, double);
            printf("%f", dval);
            break;
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
        }
  }
  va_end(ap); /* барлығы жасалған кезде тазалау */
}
```

7.3-жаттығу. Minprintf басқа printf мүмкіндіктерімен толықтырыңыз.

7.4. Форматты енгізу (scanf)

Енгізуді қамтамасыз ететін scanf функциясы printf аналогы болып табылады; ол көптеген түрлендірулерді орындайды, бірақ қарама-қарсы бағытта. Оның жариялауы келесідей:

```
int scanf(char *format, ...)
```

Scanf функциясы стандартты кіріс ағынынан таңбаларды оқиды, оларды формат жолының сипаттамаларына сәйкес түсіндіреді және нәтижелерін басқа дәлелдерге жібереді. Пішім аргументі кейінірек сипатталады; басқа дәлелдер, олардың әрқайсысы көрсеткіш болуы керек, дұрыс өзгертілген мәліметтер қай жерде сақталатынын анықтайды. Printf сияқты, бұл бөлімде осы функцияның ең пайдалы, бірақ барлық сипаттамалары жинақталған.

Scanf функциясы стандартты кіріс ағынынан таңбаларды оқиды, оларды формат жолының сипаттамаларына сәйкес түсіндіреді және нәтижелерін басқа дәлелдерге жібереді. Пішім аргументі кейінірек сипатталады; басқа дәлелдер, олардың әрқайсысы көрсеткіш болуы керек, дұрыс өзгертілген мәліметтер қай жерде сақталатынын анықтайды. Printf сияқты, бұл бөлімде осы функцияның ең пайдалы, бірақ барлық сипаттамалары жинақталған.

Сондай-ақ, жолдан оқылатын sscanf функциясы бар (және стандартты кірістен емес).

```
int sscanf(char *string, char *format, arg1, arg2, ...)
```

Sscanf функциясы жолды формат пішіміне сәйкес сканерлейді және алынған мәндерді arg1, arg2, т.с.с. жібереді. Соңғысы көрсеткіштер болуы керек. Пішім, әдетте, кіріс түрлендірулерін басқаруға қолданылатын сипаттамаларды қамтиды.

Ол келесі элементтерді қамтуы мүмкін:

- Елембейтін бос орындар немесе табуляциялар.
- Әдеттегі таңбалар (%қоспағанда), кіріс ағынын бөлгіш таңбалардан басқа келесі таңбаларға сәйкес келеді деп күтілуде.
- Әрқайсысы % белгісінен басталып, түрлендіру түрінің сипаттамасының символымен аяқталатын түрлендіру сипаттамалары. Осы екі таңбаның арасындағы аралықта кез-келген сипаттамада және оларды осында көрсетілген ретпен орналастыруға болады: * белгі (беруді басу белгісі); өрістің енін анықтайтын сан; алынған әріптің

мөлшерін көрсете отырып, h, l немесе L әрпі; және түрлендіру таңбасы (o, d, x).

Түрлендіру ерекшелігі келесі енгізілетін өрісті түрлендіруді басқарады. Әдетте нәтиже тиісті аргументті көрсететін айнымалыға орналастырылады. Алайда, егер түрлендіру ерекшелігінде * символы болса, онда енгізу өрісі өтіп, ешқандай меншіктеу орындалмайды. Енгізу өрісі таңбалар − бөлгішсіз жол ретінде анықталады; ол келесі таңбаға дейін созылады немесе егер берілген болса, өрістің енімен шектеледі. Жаңа жолдың символы бөлгіш таңбаларға қатысты болғандықтан, scanf оқу кезінде бір жолдан екіншісіне ауысады. (Бос орын, табуляция, жаңа жол, каретканы қайтару, тік табуляция және парақты аудару символдары болып табылады.)

Спецификатор таңбасы келесі енгізу өрісін қалай түсіну керектігін көрсетеді. Тиісті аргумент C мәнінде қабылданған параметрлер бойынша параметрлерді беру механизмі талап еткендей көрсеткіш болуы керек. Анықтамалық белгілер 7.2-кестеде келтірілген.

D, l, o, u және x классификаторының таңбалары h әрпінен бұрын болуы мүмкін, бұл сәйкес аргументтің ұзындық түрін көрсететін қысқа * (int емес) немесе l (Latin ell) түрінде болуы керек. Сол сияқты, e, f және g жіктеуіш таңбалары l әрпінің алдында болуы мүмкін, бұл аргумент типінің қос * екенін (өзгермейтін *) білдіреді.

7.2-кесте. Негізгі сканф түрлендірулері

Символдар	**Таңба енгізілген деректер; аргумент түрі**
d	ондық бүтін сан; int *
i	тұтас; int *. Бүтін сан сегіздік (0 солға) немесе он алтылық (0x немесе 0X сол жақта) болуы мүмкін
o	сегіздік бүтін сан (солға нөлмен немесе онсыз); int *
u	қол қойылмаған ондық бүтін сан; қол қойылмаған int *
x	оналтылық бүтін сан (сол жақта OX немесе OX жоқ); int *
c	таңбалар char *. Келесі енгізу таңбалары (әдепкі бойынша) көрсетілген жерге орналастырылады. Бөлгіш белгілерді қалыпты өткізіп жіберу басылған; бөлгіш таңбадан басқа келесі таңбаны оқу үшін %1s қолданыңыз

s	таңба жолдары (тырнақшасыз); char * жол үшін жеткілікті таңбалар массивін және «\ 0» аяқталатын таңбаны қосу үшін
e, f, g	қалқымалы нүкте нөмірі, белгісімен болуы мүмкін; Ондық бөлшектердің де, экспоненциалды бөліктің де, мүмкін екеуінің де болуы міндетті болып табылады; float *
%	% белгісінің өзі, ешқандай меншіктеу орындалмайды

Бірінші мысалды салу үшін, scanf функциясы арқылы енгізуді ұйымдастыратын 4-тараудан калькулятор бағдарламасына жүгініңіз:

```
#include <stdio.h>

main() /* калькулятор бағдарламасы*/
{
      double sum, v;

      sum = 0;
      while (scanf (“%lf”, &v) == 1)
            printf(“\t%.2f\n”, sum += v);
      return 0;
}
```

Көрініс деректерін қамтитын енгізу жолдарын оқу керек делік:

```
25 дек 1988
```

Scanf-ге жүгіну келесідей:

```
int day, year; /* күн, год жыл*/
char monthname[20]; /* айдың атауы */
scanf (“%d %s %d”, &day, monthname, &year);
```

& белгісі monthname алдына қажет емес, себебі массивтің атауы көрсеткіш бар.

Пішім жолында ешқандай ерекшеліктерге қатыспайтын таңбалар болуы мүмкін; бұл дегеніміз, бұл таңбалар енгізу кезінде пайда болуы керек. Осылай, біз келесі scanf арқылы mm/dd/yy түрінің күндерін оқып алар едік:

```
int day, month, year; /* күн, ай, жыл */
scanf("%d/%d/%d", &day, &month, &year);
```

Scanf өзінің пішімінде бос орындар мен табуляцияларды елемейді. Сонымен қатар, келесі енгізу бөлігін іздеу кезінде ол кіріс ағынында барлық таңбаларды-бөлгіштерді (бос орындар, табуляция, жаңа жолдар және т.б.) өткізеді. Тұрақты пішімі жоқ кіріс ағынын қабылдау, бүкіл жолды толығымен енгізгенде және әрбір жеке жағдайда sccanf-тың сәйкес нұсқасын таңдасаңыз, жиі ыңғайлы болады. Мысалы, жоғарыда келтірілген нысандардың кез келгенінде жазылған күндері бар жолдарды оқу керек деп есептейік. Сонда біз оны мына түрде жаза аламыз:

```
while (getline(line, sizeof(line)) > 0) {
      if (sscanf(line, "%d %s %d", &day, monthname, &year) == 3)
          printf("верно: %s\n", line); /* в виде 25 дек 1988 */
      else if (sscanf(line, "%d/%d/%d", &month, &day, &year) == 3)
          printf("верно: %s\n", line); /* mm/dd/yy түрінде*/
      else
          printf("дұрыс емес %s\n", line); /* күні дұрыс емес */
}
```

Scanf қызметіне қоңыраулар басқа кіріс функцияларына байланысты болуы мүмкін. Scanf-дан кейін енгізілген кез келген енгізу функциясы әлі оқылмаған бірінші таңбадан бастап оқуды жалғастырады.

Қорытындылай келе, scanf және sscanf функциялары үшін аргументтер көрсеткіштер болуы керек екенін тағы бір еске саламыз.

Ең көп кездесетін қателіктердің бірі – бұл жазудың орнына

```
scanf («% d», & n);
```

олар жазады

```
scanf («% d», n);
```

Компилятор осындай қате туралы хабарлайды.

7.4-жаттығу. Алдыңғы параграфтан minprintf-ге ұқсас scanf нұсқасын жазыңыз.

7.5-жаттығу. 4-тараудан кейінгі жазбаға негізделген калькулятор бағдарламасын сандарды енгізу және түрлендіру үшін scanf және/немесе sscanf пайдаланатындай етіп қайта жазыңыз.

7.5. Файлдарға кіру

Барлық алдыңғы мысалдарда бағдарлама үшін автоматты түрде белгілі бір машинаның операциялық жүйесі алдын ала анықталған стандартты енгізу және стандартты шығару ісі болды.

Келесі қадам – бағдарламаларға алдын ала қосылмаған файлдарға қол жеткізе алатын бағдарламаларды жазуды үйрену. Мұндай қажеттілік туындаған бағдарламалардың бірі – бірнеше атаулы файлдарды біріктіретін және нәтижені стандартты қорытындыға бағыттайтын cat бағдарламасы — cat функциясы жиі экранға файлдарды беру үшін, сондай-ақ аты бойынша файлға хабарласу мүмкіндігі жоқ бағдарламалар үшін файл ақпаратының әмбебап “коллекторы” ретінде қолданылады. Мысалы, команда

```
cat x.c y.c
```

x.c және y.c файлдарының мазмұнын стандартты шығаруға жібереді (және ештеңе жоқ).

Сұрақ туындайды: аталған файлдарды оқып шығу үшін не істеу керек; басқаша айтқанда, пайдаланушы ойлап тапқан сыртқы атауларды деректерді оқу нұсқаулықтарымен қалай байланыстыру керек?

Бұл ретте қарапайым ережелер бар. Файлдан оқу немесе файлға жазу үшін ол fopen кітапхана функциясымен алдын ала ашылуы керек. Fopen функциясы x.c немесе y.c типті сыртқы атауды алады, содан кейін амалдық жүйемен “келіссөздер” мен ұйымдастыру әрекеттерін жүзеге асырады (техникалық бөлшектері осында қарас-

тырылмайды) және файлға кіру үшін қолданылатын көрсеткішті қайтарады.

Файл көрсеткіші деп аталатын бұл көрсеткіш файл туралы ақпаратты қамтитын құрылымға (буфердің мекен-жайы, буферде-гі ағымдағы таңбаның орны, оқуға немесе жазуға файл ашылды, файлмен жұмыс істеу кезінде қателер болды ма және файлдың соңы кездесті ма) сілтеме жасайды. Пайдаланушы мәліметтерді білудің қажеті жоқ, себебі <stdio.h>-дан алынған анықтамалар, FILE деп аталатын құрылымның сипаттамасын қамтиды.

Файл көрсеткішін анықтау үшін қажет жалғыз нәрсе, мысалы, түрін сипаттау:

```
FILE *fp;
FILE *fopen(char *name, char *mode);
```

Бұл FP file көрсеткіш бар дейді, a fopen file көрсеткішін қайтарады. Айта кету керек, FILE-құрылымдық тег емес, int сияқты түрдің атауы. Ол typedef арқылы анықталады. (UNIX жүйесінде fopen-ді қалай жүзеге асыруға болатындығы туралы мәліметтер 8.5-параграфта келтірілген.)

Бағдарламада fopen-ге жүгіну келесідей болуы мүмкін:

```
fp = fopen(name, mode);
```

Бірінші аргумент - бұл файлдың аты бар жол. Екінші аргумент режим туралы ақпаратты береді. Бұл сонымен қатар жол: бұл файл-ды пайдаланушының қалай пайдаланғысы келетінін көрсетеді. Ке-лесі режимдер мүмкін: оқу (оқу - «r»), жазу (жазу - «w») және қосу (қосу - «a»), яғни бар файлдың соңына ақпарат жазу. Кейбір жү-йелер мәтіндік және екілік файлдарды ажыратады; соңғы жағдайда режім сызығына (бинарлық) «b» әрпін қосу керек.

Бұрын болмаған файл жазу немесе қосу үшін ашылса, ол құры-лады (егер мұндай рәсім физикалық мүмкін болса). Жазбадағы

бар файлды ашу оның ескі мазмұнын тастайды, ал файлды қосқан кезде оның ескі мазмұны сақталады. Жоқ файлды оқу әрекеті қате болып табылады. Басқа да қателер болуы мүмкін; мысалы, мәртебесі бойынша оқуға тыйым салынған файлды оқу әрекеті қате болып саналады. Кез келген қате болған жағдайда fopen null қайтарады. (Қатені нақты анықтау мүмкін; осы мәселе бойынша егжей-тегжейлі ақпарат В қосымшасының 1-параграфының соңында келтіріледі.)

Біз білуіміз керек келесі нәрсе-файлдан оқу немесе файлға жазу, коль көп ұзамай ол ашық. Мұны жасаудың бірнеше жолы бар, оның ең қарапайым-getc және putc мүмкіндіктерін пайдалану. Getc функциясы файлдан келесі таңбаны қайтарады; ол таңбаны қайдан алу керектігін білу үшін файл көрсеткішін хабарлау керек.

```
int getc(FILE *fp)
```

Getc функциясы *fp көрсететін ағынның келесі таңбасын қайтарады; файл таусылған немесе қате болған жағдайда ол EOF қайтарады.

Putc функциясы c таңбасын fp файлына жазады:

```
int putc(int c, FILE *fp)
```

және қате болған жағдайда жазылған таңбаны немесе EOF мәнін қайтарады. getchar мен putchar сияқты, getc және putc амалдары макростар ретінде емес, функциялар ретінде жүзеге асырылуы мүмкін.

С бағдарламасын іске қосқан кезде, амалдық жүйе әрқашан үш файлды ашады және оларға үш файл сілтемелерін ұсынады. Бұл файлдар: стандартты енгізу, стандартты шығару және қателіктің стандартты файлы; олардың сәйкес көрсеткіштері stdin, stdout және stderr деп аталады; олар <stdio.h> сипатталған. Әдетте stdin пернетақтамен байланысты, ал stdout және stderr экранмен байланысты. Алайда, stdin және stdout файлдарға немесе 7.1-тармақта сипат-

талғандай, басқа бағдарламаларға тікелей қосылған құбырлы механизмді қолдана отырып байланыстыруға болады.

Getc, putc, stdin және stdout, getchar және putchar функцияларын қолдана отырып, енді келесідей анықтауға болады:

```
#define getchar() getc(stdin)
#define putchar(c) putc((c), stdout)
```

Пішімді енгізу-шығару файлдарды fscanf және fprintf функцияларында құруға болады. Олар scanf және printf бірдей, олардың бірінші аргументі енгізу-шығару жүзеге асырылатын файлға көрсеткіш болып табылады, пішім екінші аргументпен көрсетіледі.

```
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)
```

Енді біз cat бағдарламасын жазу үшін жеткілікті мәліметтерді иеленеміз, ол файлдарды конкатенациялауға (дәйекті байланыс) арналған. Cat функциясының ұсынылған нұсқасы көптеген бағдарламалар үшін ыңғайлы болып шықты. Егер командалық жолда дәлелдер болса, олар

дәйекті өңделетін файлдардың аттары ретінде қарастырылады. Егер дәлелдер болмаса, стандартты енгізу өңделеді.

```
#include <stdio.h>

/* cat: файлдарды конкатенациялау, 1 нұсқасы*/
main(int argc, char *argv[])
{
      FILE *fp;
      void filecopy(FILE *, FILE *);

      if (argc == 1) /* аргументтер жоқ; стандартты енгізу көшірілуде */
          filecopy(stdin, stdout);
       else
           while (--argc > 0)
                 if ((fp = fopen(*++argv, “r”)) == NULL) {
                     printf(“cat: %s\n”, *argv файлын ашылмайды);
                     return 1;
```

```
                } else {
                    filecopy(fp, stdout);
                    fclose(fp);
                }
        return 0;
}

/* filecopy: ifp файлын ofp файлына көшіреді */
void filecopy(FILE *ifp, FILE *ofp)
{
    int c;

    while ((c = getc(ifp)) != EOF)
        putc(c, ofp);
}
```

Stdin және stdout файлдық көрсеткіштері FILE * түріндегі нысандар болып табылады. Бұл тұрақты емес, айнымалы емес, сондықтан оларға ештеңе берілмейді.

```
int fclose(FILE *fp)
```

функциясы - fopen-ге қатысты кері; ол файлдық көрсеткіш пен сыртқы атау арасындағы байланысты бұзады (ол бұрын fopen арқылы орнатылған), осылайша басқа файлдар үшін осы көрсеткішті босатады. Операциялық жүйелердің көпшілігінде бір уақытта ашылған файлдардың саны шектеулі болғандықтан, файлдық көрсеткіштер, егер олар қажет болмаса, cat бағдарламасында қалай жасалатынын босату керек. Шығару файлына eclose-ді қолданудың тағы бір себебі бар, бұл putc шығаруға арналған деректерді жинақтаған буферді "босату" қажеттілігі. Бағдарлама қалыпты аяқталған кезде әрбір ашық fclose файлына автоматты түрде шақырылады. (сіз stdin және stdout қажет болмаса, жаба аласыз. freopen кітапханалық қызметін пайдаланып, оларды қалпына келтіруге болады.)

7.6. Қателерді басқару (stderr және exit)

Cat-дағы қателерді өңдеу мінсіз деп танылмайды. Мәселе мынада, егер файл қандай да бір себеппен қол жетімсіз болса, бұл туралы

хабарламаны біз конкатенирируемой қорытынды аяқталғаннан кейін аламыз. Бұл бізді, егер шығару файлға немесе конвейерге басқа бағдарламаға емес, экранға жіберілсе, жасайды.

Бұл мәселені жақсы шешу үшін stdout стандартты шығысынан басқа бағдарламаға stderr деп аталатын тағы бір шығыс ағыны беріледі. Stderr шығысы, әдетте, stdout шығысы басқа жерге бағытталса да, экранға жіберіледі.

Қате туралы хабарлар stderr-ға жіберілетіндей етіп cat қайта жазамыз.

```
#include <stdio.h>

/* cat: файлдарды конкатенациялау, 2-нұсқасы*/
main(int argc, char *argv[])
{
      FILE *fp;
      void filecopy(FILE *, FILE *);

      char *prog = argv[0]; /* бағдарлама аты */
      if (argc ==1) /* дәлелдер жоқ; станд көшірілген. енгізу */
          filecopy(stdin, stdout);
      else
         while (--argc > 0)
               if ((fp = fopen(*++argv, “r”)) == NULL) {
                   fprintf (stderr, “%s: %s\n файлы ашылмайды”,
                        prog, *argv);
                   exit(t);
               } else {
                   filecopy(fp, stdout);
                   fclose(fp);
               }
       if (ferror(stdout)) {
            fprintf (stderr, “%s: stdout\n”, prog жазу қатесі);
            exit(2);
       }
       exit(0);
}
```

Бағдарлама қате туралы екі жолмен сигнал береді. Біріншісі-қате туралы хабар fprintf арқылы stderr-ға жіберіледі, ол экранға түсіп,

конвейерде немесе басқа шығыс файлында болмас үшін. Argv-де сақталған бағдарлама Атауы[0], біз осы бағдарлама басқалармен бірге жұмыс істеген жағдайда, қате көзі анықталуы үшін хабарға қосылдық. Хабарламада argv [0] бағдарламасында сақталған бағдарламаның атауын енгіздік, осылайша бұл бағдарлама басқалармен бірге жұмыс істеген жағдайда қатенің көзі анық болады.

Қатені келтірудің екінші жолы – бағдарламаның жұмысын аяқтайтын exit кітапханалық функциясына жүгіну. Exit функциясының дәлелі осы процесті тудырған кейбір процеске қол жетімді. Демек, бағдарламаның табысты немесе қате аяқталуын осы бағдарламаны бағынышты процесс ретінде қарайтын белгілі бір бағдарламаның көмегімен бақылауға болады. Жалпы келісім бойынша нөлді қайтару жұмыстың қалыпты болғанын білдіреді, ал нөлдік емес мәндер әдетте қателер туралы айтады. Барлық ашық шығару файлдары үшін ақпарат жинақтаған буферді босату үшін exit функциясы eclose тудырады.

Негізгі main бағдарламасының return нұсқауы exit(өр) функциясына тең. Соңғы опция (exit арқылы) басқа функциялардан шығу үшін қолайлы артықшылығы бар, сонымен қатар, exit сөзі 5-тарауда қарастырылған контекстік іздеу бағдарламасының көмегімен оңай анықталады.

Ferror функциясы fp файлында қате анықталса, нөлдік мәнді береді.

```
int ferror(FILE *fp)
```

Қорытынды кезінде сирек қателер бар болса да, олар кездеседі (мысалы, диск толып қалды); сондықтан кең қолдану бағдарламаларында олар мұқият бақылануы керек.

Feof(file *) функциясы ferror функциясына ұқсас; аргументте көрсетілген файлдың соңы болса, ол нөл емес мәнді қайтарады.

```
int feof(FILE *fp)
```

Біздің шағын иллюстрациялық бағдарламаларымызда шығу мәртебесін беруге, яғни аяқталу сәтіндегі бағдарламаның жағдайын сипаттайтын кейбір санды беруге қамқорлық жасамадық: жұмыс қалыпты аяқталды немесе қателікке байланысты үзілді ме? Егер жұмыс қате нәтижесінде үзілген болса, онда қандай? Кез келген маңызды бағдарлама шығу мәртебесін беруі керек.

7.7. Жолдарды енгізу-шығару

Стандартты кітапханада алдыңғы тарауларда қолданылған getline бағдарламасына ұқсас fgets енгізу бағдарламасы бар.

```
char *fgets(char *line, int maxline, FILE *fp)
```

Fgets функциясы келесі енгізу жолын (жаңа жол таңбасын қоса) FP файлынан line таңбалар массивіне оқиды және ол MAXLINE-1 таңбадан артық емес оқи алады. Қайта жазылған жол '\0' белгісімен толықтырылады. Әдетте fgets line қайтарады, ал файл таусылған немесе қате болған жағдайда – NULL. (Біздің getline біз пайдаланған жолдың ұзындығын және файлдың соңына қарай нөл қайтарады.)

Fputs шығару функциясы файлға жолды (жаңа жолдың символымен аяқталмауы мүмкін) жазады.

```
int fputs(char *line, FILE *fp)
```

Бұл функция қате пайда болған жағдайда EOF, ал кері жағдайда теріс емес мәнін қайтарады.

Gets және puts кітапхана функциялары fgets және fputs функцияларына ұқсас. Олар тек стандартты stdin және stdout файлдарымен жұмыс істейді, сонымен қатар, gets соңғы '\n' таңбасын шығарады, ал puts оны қосады.

Fgets және fputs сияқты функцияларға ерекше ештеңе жоқ екенін көрсету үшін, біз оларды біздің жүйеде стандартты кітапханада бар түрінде келтіреміз.

```
/* fgets: IOP-тен n таңбадан артық емес алады */
char *fgets(char *s, int n, FILE *iop)
{
      register int c;
      register char *cs;

      cs = s;
      while (--n > 0 && (c = getc(iop)) != EOF)
            if ((*cs++ = c) == '\n' )
                break;
      *cs = '\0' ;
      return (c == EOF && cs == s) ? NULL : s;
}

/* fputs: s жолын iop файлына жібереді */
 int fputs(char *s, FILE *iop)
{
     int c;

     while (c = *s++)
           putc(c, iop);
     return ferror(iop) ? EOF : 0;
}
```

Стандарт ferror функциясы қате болған жағдайда нөл емес мәнді қайтарады; fputs қате болған жағдайда EOF қайтарады немесе теріс емес мәнді қайтарады.

Fgets көмегімен біздің getline функциясын іске асыру оңай:

```
/* getline: жолды оқиды, оның ұзындығын қайтарады */
int getline(char *line, int max)
{
    if (fgets(line, max, stdin) == NULL)
        return 0;
    else
         return strlen(line);
}
```

7.6-жаттығу. Екі файлды салыстыратын және олар бір-бірінен өзгеше болатын бірінші жолды басып шығаратын бағдарлама жазыңыз.

7.7-жаттығу. Іздеу бағдарламасын 5-тараудағы модельге сәйкес өзгертіңіз, сонда ол әртүрлі аталған файлдардан мәтін алады, ал егер дәлелдерде файл атаулары болмаса, онда стандартты енгізуден. Сәйкес келетін жол табылған файлдың атауы басыла ма?

7.8-жаттығу. Бірнеше файлды басып шығаратын программа жазыңыз. Әрбір файл жаңа беттен басталуы керек, оның тақырыбы алдында болуы керек және өзіндік бетбелгісі болуы керек.

7.8. Басқа кітапханалық функциялар

Стандартты кітапханада әртүрлі функциялардың кең спектрі бар. Осы параграф олардың ең пайдаларының қысқаша шолуын қамтиды. Осы және басқа да функциялар B қосымшасында сипатталған.

7.8.1. Жолдармен операциялар

Біз бұған дейін <string.h> -де сипатталған strlen, strcpy, strcat және strcmp функциялары туралы айтқанбыз. Әрі қарай, абзацтың соңына дейін s және t - char *, c және n типтері int типіне жатады.

strcat(s, t) — s соңында t жазады.
strncat(s, t, n) — t-дан s соңына n таңбаларды қосады.
strcmp(s, t) — s < t, s = t немесе s > t үшін теріс санды қайтарады.
strncmp(s, t, n) — ол strcmp сияқты жасайды, бірақ салыстырылатын таңбалардың саны n-ден аспайды.
strcpy(s, t) — t-ны s-ға көшіреді.
strncpy(s, t, n) — t-дан s-ға n таңбадан артық емес көшіреді.
strlen(s) — s ұзындығын қайтарады.
strchr(s, c) — көрсеткіш s-да c символының бірінші пайда болуына қайтарады, ал егер c s-да болмаса, NULL-де қайтарады.
strrchr(s, c) — көрсеткіш s-да c символының соңғы пайда болуына қайтарады, ал егер c s-да болмаса, NULL-де қайтарады.

7.8.2. Символдар класын талдау және символдарды түрлендіру

<Ctype.h> кітапханасынан бірнеше функциялар таңбаларды тексеру және түрлендіру жұмыстарын атқарады. Әрі қарай, тармақтың соңына дейін, айнымалы c-unsigned char немесе EOF мәнімен ұсы-

нылуы мүмкін int типті айнымалы. Барлық осы функциялар int типті мәндерді қайтарады.

isalpha (c) — егер с-әріп болса, 0 емес; ал керісінше жағдайда 0.
isupper (c) — егер с-жоғарғы регистр әріпі болса, 0 емес; ал керісінше жағдайда 0.
islower (c) — егер с-төменгі регистр әріпі болса 0 емес; ал керісінше жағдайда 0.
isdigit (c) — егер с-цифр болса, 0 емес; ал керісінше жағдайда 0.
isalnum(c) — егер isalpha(c) немесе isdigit (c) ақиқат болса емес; ал керісінше жағдайда 0.
isspace (c) — бос орын, табуляция, жаңа жол, каретканы қайтару, беттің аудармасы, тік табуляция белгісі болса, 0 емес.
toupper (c) - жоғарғы регистрге келтірілген с-ны қайтарады.
tolower (c) - төменгі регистрге келтірілген с-ны қайтарады.

7.8.3. Ungetc функциясы

Стандартты кітапханада ungetch функциясының шектеулі нұсқасы бар, біз 4-тарауында жазғанымызбен салыстырғанда ол ungetc деп аталады. Бұл прототипі бар функция:

```
int ungetc(int c, FILE *fp)
```

таңбаны кері FP файлына жібереді және С қайтарады, ал EOF қатесі болған жағдайда. Әрбір файл үшін бір таңбадан артық емес қайтару кепілдігі бар. Ungetc функциясын scanf, getc, getchar және т.б. сияқты кез келген енгізу мүмкіндіктерімен бірге пайдалануға болады.

7.8.4. Операциялық жүйе командаларын орындау

System(char *s) функциясы s жолында бар жүйе командасын орындайды, содан кейін ағымдағы бағдарламаны орындауға қайтарылады. S мазмұны, қатаң айтқанда, нақты операциялық жүйеге байланысты. Қарапайым мысал қарастырайық: UNIX жүйесінде нұсқаулық

```
system(“date”);
```

күн мен уақытты стандартты шығаруға бағыттайтын date бағдарламасын шығарады. Функция орындалған команданың жүйеге тәуелді күйін қайтарады. UNIX жүйесінде қайтарылатын мәртебе-exit функциясымен берілген мән.

7.8.5. Жадты басқару

Malloc және calloc функциялары динамикалық бос жад блоктарын сұрайды. Malloc функциясы

```
void *malloc(size_t n)
```

егер сұранымды қанағаттандыру мүмкін болмаса, n байт-ға немесе NULL-ге көрсеткіш қайтарады. Calloc функциясы

```
void *calloc(size_t n, size_t size)
```

көрсеткіні көрсетілген өлшемдегі (size) нысандарынан (size) немесе null массивін сақтау үшін жеткілікті аймаққа қайтарады. Бөлінген жады нөлденеді.

Malloc және calloc функцияларымен қайтарылатын көрсеткіш нысанның көрсетілген түріне сәйкес жасалған туралауды ескере отырып беріледі. Дегенмен, бағдарламаның келесі фрагментінде жасалғандай, тиісті түріне келтіру операциясы қолданылуы керек:

Сондай-ақ, босатылған жад аймақтарын да қолдануға болмайды. Келесі мысал тізімнің элементтерін босататын циклдегі әдеттегі қатені көрсетеді.

```
for (p = head; p != NULL; p = p->next) /* ДҰРЫС ЕМЕС */
    free(p);
```

Егер сіз босатқанға дейін келесі цикл сияқты қажет нәрсені сақтасаңыз, дұрыс болады:

```
for (p = head; p != NULL; p = q) {
    q = p->next;
    free(p);
}
```

8.7-параграфта біз кез келген ретпен арнайы жад блоктарын босатуға мүмкіндік беретін malloc сияқты жады басқару бағдарламасын іске асыруды қарастырамыз.

7.8.6. Математикалық функциялар

В <math.h> жиырмадан астам математикалық функциялар сипатталған. Осында келтірілген жиі қолданылатын функциялар. Олардың әрқайсысы double типті бір немесе екі дәлелге ие және нәтижені double типті қайтарады.

```
sin (x) — синус x, радиандағы x.
cos (x) — косинус x, радиандағы x.
atan2 (y, x) — радиандағы арктангенс y/X, y және X.
exp (x) — ex экспоненциалды функциясы.
log (x) — табиғи (E негізі бойынша) логарифм x (x>0).
log10 (x) — қарапайым (10 негізі бойынша) логарифм x (x > 0).
pow(x, y) — xy.
sqrt (x) — шаршы x түбірі(x > 0).
fabs (x) — x абсолюттік мәні.
```

7.8.7. Кездейсоқ сандар генераторы

Rand() функциясы <stdlib.h>-те анықталған, нөлден RAND_MAX тұрақты деп аталатын мәнге дейінгі жалған кездейсоқ бүтін тізбекті есептейді. Кездейсоқ сандарды 0-ге тең немесе 1-ден кем өзгермелі нүктелі мәндерге келтіруге болады

```
#define frand() ((double) rand() / (RAND_MAX+1.0))
```

(Егер сіздің кітапханада өзгермелі нүктелі кездейсоқ сандарды алу функциясы бар болса, оның статистикалық сипаттамалары көрсетілгеннен жақсы болуы мүмкін.)

Srand(unsigned) функциясы rand үшін тұқым орнатады. Стандартты ұсынылған rand және srand жүзеге асыру және, демек, әр түрлі машиналарға тасымалданатын, 2.7-параграфында қарастырылған.

7.9-жаттығу. Isupper сияқты функцияларды іске асыру арқылы жадты немесе уақытты үнемдеуге болады. Функцияның екі нұсқасын жазыңыз.

8-тарау. UNIX жүйесінің интерфейсі

UNIX операциялық жүйесі өз қызметтерін жүйелік қоңыраулар жиынтығы ретінде ұсынады, олар іс жүзінде оның ішкі функциялары болып табылады және оларға пайдаланушының бағдарламаларынан жүгінуге болады. Осы тарауда С-бағдарламаларда кейбір маңызды сын-қатерлерді қалай қолдануға болатынын сипатталған. Егер сіз UNIX жүйесінде жұмыс істейтін болсаңыз, онда бұл ақпарат сізге тікелей пайдалы болады және жұмыс тиімділігін арттыруға немесе кітапханада жоқ мүмкіндіктерге қол жеткізуге мүмкіндік береді. Егер сіз басқа операциялық жүйеде С тілін пайдалансаңыз да, мұнда қарастырылған мысалдар сізді si-ге бағдарламалау түсінігіне жақындатады; ұқсас бағдарламалар (тек бөлшектермен ерекшеленетін) сіз кез келген операциялық жүйеде кездестіресіз. ANSI стандарты ретінде бекітілген Си-бағдарламалар кітапханасы негізінен UNIX жүйесінің мүмкіндіктерін көрсетеді, ұсынылған бағдарламалар сізге кітапхананы да жақсы түсінуге көмектеседі.

Тарау үш негізгі бөліктен тұрады: енгізу-шығару, файлдық жүйе және жадты басқаруды ұйымдастыру. Алғашқы екі бөлікте оқырманның UNIX жүйесінің сыртқы сипаттамаларымен танысуы болжанады.

7-тарауда біз барлық операциялық жүйелер үшін бірыңғай енгізу-шығару интерфейсін қарастырдық. Стандартты кітапхана бағдарламасының кез келген нақты жүйесінде дәл осы нақты жүйенің құралдарын пайдалану арқылы жазылады. Келесі бірнеше параграфта біз UNIX жүйесінің енгізу-шығару бойынша шақыруларын сипаттаймыз және олардың көмегімен стандартты кітапхананың кейбір бөлімдерін іске асыруға болатынын көрсетеміз.

8.1. Файлдар дескрипторлары

UNIX жүйесінде кез келген енгізу-шығару операциялары файлдарды оқу және жазу арқылы орындалады, өйткені барлық сыртқы

құрылғылар, соның ішінде пернетақта мен экран файлдық жүйе нысандары ретінде қарастырылады. Бұл дегеніміз, бағдарлама мен сыртқы құрылғылар арасындағы барлық байланыстар бірыңғай біртекті интерфейс шеңберінде жүзеге асырылады.

Жалпы алғанда, оқу немесе жазу алдында, сіз файлға қатысты істегіңіз келетін әрекеттер туралы жүйені хабардар етуіңіз керек; бұл рәсім файлды ашу деп аталады. Егер файлға жазғыңыз келсе, оны қайта жасау немесе сақталған ақпараттан тазарту қажет болуы мүмкін. Жүйе сіздің осы әрекеттерге құқықтарыңызды тексереді (файл бар ма? сіз оған қол жеткізе аласыз ба? d) егер дескриптор файл деп аталатын, егер бәрі дұрыс болса, бағдарламаны теріс емес бүтін қайтарады. Енгізу-шығару жүзеге асырылған кезде, файлды сәйкестендіру аты бойынша емес, оның дескрипторы бойынша орындалады. (Файл дескрипторы стандартты кітапханада қолданылатын файлдық көрсеткішке немесе MSDOS-дағы хэндлға (handle) ұқсас.) Ашық файл туралы барлық ақпарат операциялық жүйемен сақталады және өңделеді; пайдаланушы бағдарламасы тек оның дескрипторы арқылы файлға жүгінеді.

Пернетақтадан енгізу және экранға шығару жиі қолданылады, сондықтан олармен жұмыс істеу ыңғайлы болуы үшін арнайы келісімдер жасалады. Бағдарламаны іске қосқан кезде командалық интерпретатор (Shell) стандартты енгізу, стандартты шығару және стандартты қате файлы деп аталатын 0, 1 және 2 дескрипторлары бар үш файлды ашады. Егер бағдарлама 0 файлынан оқыса, 1 және 2 файлдарына (мұнда сандар — файлдар дескрипторлары) жазса, онда ол оларды ашуға қамқорлық жасамай, енгізу мен шығаруды жүзеге асыра алады.

Бағдарлама пайдаланушысы енгізу-шығаруды файлға немесе файлдан < және > таңбашалары арқылы жүзеге асыра алады, мысалы, файлға

```
prog <infile >outfile
```

Бұл жағдайда командалық интерпретатор 0 және 1 дескрипторларының стандартты қондырғыларын атау файлдарына ауыстырады. Әдетте 2 файлының дескрипторы қателерді жіберу үшін экранға қосылған болып қалады. Конвейерге байланысты енгізу-шығару үшін де айтылған. Барлық жағдайларда файлды ауыстыру бағдарлама емес, командалық интерпретатор жүзеге асырады. Бағдарлама, егер ол 0 файлына (енгізген жағдайда) және 1 және 2 файлдарына (шыққан жағдайда) сілтеме жасаса, оны енгізу қайдан келгенін, оның шыққан жерін білмейді.

8.2. Енгізу-шығарудың төменгі деңгейі (read и write)

Енгізу-шығару read және write жүйелік қоңырауларына негізделген, оған С-бағдарлама read және write атаулары бар функцияларды пайдаланады. Екі бірінші дәлел дескриптор файл. Екінші аргументте деректер жіберілетін немесе қайдан алынатын бағдарламаңыздың символдар массиві көрсетіледі. Үшінші дәлел-жіберілетін байттардың саны.

```
int n_read = read(int fd, char *buf, int n);
int n_written = write(int fd, char *buf, int n);
```

Екі функция берілген байттар санын қайтарады. Оқу кезінде оқылған байттардың саны үшінші аргументте көрсетілген саннан аз болуы мүмкін. Нуль файлдың соңын білдіреді, а -1 қате туралы сигнал береді. Жазғанда, жазылған байттардың санын қайтарады және егер бұл сан талап етілгенмен сәйкес келмесе, жазба орын алған жоқ деп есептеу керек.

Бір қоңырау үшін байттардың кез келген санын оқуға немесе жазуға болады. Әдетте бұл сан 1-ге тең, яғни «буферизациясыз» немесе сыртқы құрылғының физикалық блогының мөлшеріне сәйкес келетін 1024 немесе 4096 сияқты бірдеңе дегенді білдіреді. Көптеген байттарды тиімді алмасу, себебі ол жүйелік қоңырауларды аз қажет етеді. Алынған мәліметтерді пайдалана отырып, біз өз кірістеріңізді көшіретін қарапайым бағдарламаны және 1-тарауда сипатталған файлды көшірудің эквивалентті бағдарламасын жаза аламыз. Бұл

бағдарлама кез келген жерден және кез келген жерден көшіруге болады, өйткені әрқашан кез келген файлға немесе құрылғыға енгізу мүмкіндігі бар.

```
#include “syscalls.h”

main() /* енгізуді шығысқа көшіру */
{
        char buf[BUFSIZ];
        int n;
        while ((n = read(0, but, BUFSIZ)) > 0)
                write(i, but, n);
        return 0;
}
```

Жүйелік қоңырауларды қамтамасыз ететін функциялар прототиптері syscalls.h файлында жинадық, Бұл бізге осы тараудың бағдарламаларына қосуға мүмкіндік береді. Дегенмен, бұл файлдың атауы стандартта тіркелмеген.

BUFSIZ параметрі де <syscalls.h>-да анықталған; әрбір нақты жүйеде оның мәні бар. Файл өлшемі BUFSIZ еселенген емес болса, онда қандай да бір оқу операциясы BUFSIZ қарағанда аз мәнді қайтарады, және read келесі үндеу нәтиже ретінде нуль береді.

Getchar, putchar және т.б. сияқты жоғары деңгейдегі бағдарламаларды жазу кезінде read және write қалай пайдаланылатынын қарастыру пайдалы.

```
#include “syscalls.h”

/* getchar: бір символды муферленбеген енгізу */
int getchar(void)
{
    char c;
    return (read(0, &c, 1) == 1) ? (unsigned char) c : EOF;
}
```

С айнымалы char түрі болуы керек, себебі read char көрсеткішін талап етеді. С-ді unsigned char-ге келтіру оны нәтиже ретінде қай-

тармас бұрын, белгіні таратуға байланысты қандай да бір проблемаларды жояды.

Getchar екінші нұсқасы үлкен бөліктермен енгізіледі, бірақ әрбір айналымда тек бір таңбаны береді.

```
#include “syscalls.h”

/* getchar: буферлеудің қарапайым нұсқасы */
int getchar(void)
{
    static char buf[BUFSIZ];
    static char *bufp = buf;
    static int n = 0;

    if (n == 0) { /* буфер пуст ‘*/
        n = read(0, buf, sizeof buf);
        bufp = buf;
    }
    return (--n >= 0) ? (unsigned char) *bufp++ : EOF;
}
```

Егер мұнда келтірілген getchar функциясының нұсқалары <stdio.h> тақырып файлын қосу арқылы құрастырылады және осы getchar тақырып файлы макрос ретінде анықталған, онда getchar атымен #undef жолын орнату керек.

8.3. Жүйелік қоңыраулар open, creat, close, unlink

Стандартты енгізу, шығару және әдепкі бойынша ашық қателер файлдарынан айырмашылығы, қалған файлдарды анық ашу керек. Ол үшін екі жүйелік қоңырау бар: open және creat.

Open функциясы 7-тарауында қарастырылған fopen-ге сәйкес келеді. Олардың арасындағы айырмашылық – біріншісі файлдық көрсеткіш емес, int файлының дескрипторы. Кез келген қате open -1 қайтарады.

```
#include <fcntl.h>
int fd;
```

```
int open(char *name, int flags, int perms); f
d = open(name, flags, perms);
```

Fopen сияқты, name аргументі – файл атауы бар жол. Екінші дәлел, flags, int түрі бар және файлды қалай ашу керек екенін анықтайды. Оның негізгі мәндер болып табылады:

```
O_RDONLY-тек оқу үшін ашу;
O_WRONLY - тек жазбаға ашу;
O_RDWR- оқу және жазба үшін де.
```

System V UNIX-де бұл константалар <fcntl.h>-да анықталған, ал Berkeley (BSD) нұсқаларында - <sys/file.h>.

Бар файлды оқу үшін ашуға болады

```
fd = open(name, O_RDONLY, 0);
```

Бұдан әрі біз open функциясын пайдаланатын барлық жерде оның perms аргументі нөлге тең.

Жоқ файлды ашу әрекеті қате болып табылады. Жаңа файлды жасау немесе ескісін қайта жазу creat жүйелік қоңырауымен қамтамасыз етіледі. Мысалы

```
int creat(char *name, int perms);
fd = creat(name, perms);
```

Creat функциясы файл жасалған болса, файл даскрипторын қайтарады және -1, егер қандай да бір себептермен файл жасау мүмкін болмаса. Егер файл бар болса, creat оны нөлдік ұзындыққа дейін кеседі, бұл файлдың алдыңғы мазмұнын лақтыруға тең; бұрыннан бар файлды жасау қате емес.

Егер шын мәнінде жаңа файл салынса, онда creat оны pегms аргументінде сипатталған кіру құқықтарымен жасайды. UNIX жүйесінде әр файлмен тоғыз бит байланыстырылған, онда үш санаттағы тұлғаларға оқу, жазу және орындау үшін осы файлды пайдалану құқығы туралы ақпарат бар: жеке тұлғалар тобы мен барлық қалған файл иесі. Осылайша, қол жеткізу құқықтарын үш сегіз

санмен ерекшелеуге ыңғайлы. Мысалы, 0755 файлдың меншік иесіне оқу, жазу және орындау құқығын, сондай-ақ оқу және орындау құқығын топқа және басқаларға бөледі.

Үшін иллюстрациялар келтірейік оңайлатылған нұсқасын ср жүйесін UNIX, ол көшіреді де, бір файлды басқа. Біздің нұсқада тек бір файл көшіріліп, екінші аргументте директорияларды (каталогты) көрсетуге рұқсат етілмейді және қол жеткізу құқықтары көшірілмейді, бірақ тұрақты түрде орнатылады.

```
#include <stdio.h>
#include <fcntl.h>
#include “syscalls.h”
```

#define PERMS 0666 /* RW меншік иесі, топ және басқалары үшін */

```
void error(char *, ...);

/* cp: копирование f1 в f2 */
main(int argc, char *argv[])
{
      int f1, f2, n;     char buf[BUFSIZ];

      if (argc != 3)
          error(“ср қай жерден басталып, қай жерде аяқталсын”);
      if ((f1 = open(argv[1], 0_RDONLY, 0)) == -1)
          error (“ср: файл ашылмайды %s”, argv[1]);
      if ((f2 = creat(argv[2], PERMS)) == -1)
           error (“ср: файлды жасау мүмкін емес %s, режим %03o”,
                argv[2], PERMS);
      while ((n = read(f1, buf, BUFSIZ)) > 0)
           if (write(f2, buf, n) != n)
                error (“ср: файлды жазу кезінде қате %s”, argv[2]);
      return 0;
 }
```

Бұл бағдарлама 0666 кодымен анықталған тіркелген кіру құқықтары бар шығыс файлын жасайды. 8.6 параграфында сипатталатын stat жүйелік қоңырауы арқылы бар файлды пайдалану режимін анықтап, сол көшірме режимін орната аламыз.

Әр түрлі аргументтер санымен туындаған error функциясы көбінесе printf-ге ұқсас. Error іске асыру printf отбасының басқа бағдарламаларын қалай пайдалануға болатынын көрсетеді. Vprintf кітапхана функциясы printf-ге ұқсас, аргументтер тізімінің айнымалы бөлігі va_start макросымен инициалданған бір аргументпен ауыстырылды. Сондай-ақ, vfprintf функциялары fprintf және vsprintf sprintf-пен сәйкес келеді.

```
#include <stdio.h>
#include <stdarg.h>

/* error: қате туралы хабарды басып шығарады да, жоғалады*/
void error(char *fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        fprintf(stderr, “ошибка: “);
        vfprintf(stderr, fmt, args);
        fprintf(stderr, “\n”);
        va_end(args);
        exit(1);
}
```

Бағдарламада бір уақытта ашылған файлдардың санына шектеу бар (әдетте олардың саны 20-ға жуық). Сондықтан көптеген файлдармен жұмыс істегісі келетін кез келген бағдарлама олардың дескрипторларын қайта пайдалануға дайын болуы керек. Close(int fd) функциясы файлдық дескриптор мен ашық файл арасындағы байланысты бұзады және дескрипторды басқа файлмен қолдану үшін босатады. Ол кітапхана функциясы ұқсас, бірақ тек айырмашылығы бар, буферді тазалау жоқ. Негізгі бағдарламада exit немесе return көмегімен бағдарламаны аяқтау барлық ашық файлдарды жабады.

Unlink (char *name) файлдық жүйеден файл атауын жояды. Ол стандартты кітапхананың қашықтан басқару функциясына сәйкес келеді.

8.1-жаттығу. Read, write, open және close мүмкіндіктерін пайдаланып, 7-тарауынан cat бағдарламасын қайта жазыңыз. Стандартты

кітапхананың тиісті функцияларын ауыстырыңыз. Екі нұсқаның жылдамдығын салыстыру үшін тәжірибе жасаңыз.

8.4. Еркін кіру (lseek)

Енгізу-шығару әдетте дәйекті болып табылады, яғни әрбір жаңа оқу-жазу операциясы алдыңғы операциядан (оқу-жазу) кейінгі файл позициясымен айналысады. Дегенмен, қаласаңыз, файлды еркін түрде оқуға немесе жазуға болады. Lseek жүйелік қоңырауы деректерді оқымай және жазбай файлға жылжу тәсілін ұсынады. Осылай, функция

```
long lseek(int fd, long offset, int origin);
```

FD дескрипторы бар файлда ағымдағы позицияны орнатады, оны origin мәнімен берілген орынға қатысты offset шамасына жылжытады. Origin 0, 1 немесе 2 параметрінің мәндері offset шамасына сәйкес басынан, ағымдағы позициясынан немесе файлдың соңынан шегінетінін білдіреді. Мысалы, файлға бірдеңе қосу қажет болса (UNIX жүйесінің shell командалық интерпретаторында енгізу оператормен >> файлға бағытталғанда немесе fopen-ге “a” аргументі орнатылғанда), бірдеңе жазу алдында функцияны шақыру арқылы файлдың соңын табу керек

```
lseek(fd, 0L, 2);
```

0L аргументіне назар аудару керек: 0Ll орнына (long) 0 немесе lseek функциясы дұрыс жарияланса, тек 0 деп жазсақ болады.

Lseek-тің арқасында файлдармен үлкен массивтер сияқты жұмыс істеуге болады, бірақ баяу қол жеткізу. Мысалы, келесі функция файлдың кез келген жерінен байттардың кез келген санын оқиды. Ол қате болған жағдайда оқылған байттар санын немесе -1 қайтарады.

```
#include “syscalls.h”
```

```
/* get: роз позициясынан n байт оқиды */
int get(int fd, long pos, char *buf, int n)
{
    if (lseek(fd, pos, 0) >= 0) /* установка позиции */
        return read(fd, buf, n);
    else
        return -1;
}
```

Lseek функциясы қайтарылатын ұзын түрі бар және файлдағы жаңа позиция немесе қате жағдайында -1 тең. Стандартты кітапханадан fseek функциясы lseek-ге ұқсас; соңғысы қате болған жағдайда кейбір нөл емес мәнді қайтарады, ал оның бірінші аргументі FILE*түріне ие.

8.5. Мысал. fopen және getc функцияларын іске асыру

Енді стандартты кітапханадан fopen және getc функциялары мысалында жоғарыда сипатталған бөліктер бір-бірімен келісіледі.

Стандартты кітапханада файлдар дескрипторлармен емес, файлдық көрсеткіштермен сипатталады. Файл көрсеткіші – бұл файл туралы ақпаратты қамтитын құрылым көрсеткіші: үлкен бөліктермен файлды оқуға мүмкіндік беретін буферге көрсеткіш; буфердің бос байттарының саны; буфердегі келесі позицияға көрсеткіш; файл дескрипторы; режимді (оқу/жазу) сипаттайтын жалауша, қате күйлер және т. б.

Файлды сипаттайтын деректер құрылымы <stdio.h>-да бар, ол стандартты енгізу-шығару болса, кез келген бастапқы файлға (#include арқылы) қосылуы керек. Сол тақырып файлы енгізу-шығару кітапханасының бастапқы мәтіндеріне қосылған.

Келесі фрагментте <stdio.h> файлына тән, тек кітапхана функцияларында пайдаланылатын атаулар асты сызудан басталады. Бұл пайдаланушының бағдарламасында көрсетілген аттармен кездейсоқ сәйкес келмеуі үшін жасалған. Мұндай келісім стандартты кітапхананың барлық бағдарламаларында сақталады.

```
#define NULL O
#define EOF (-1)
#define BUFSIZ 1024
#define OPEN_MAX 20 /* бір уақытта ашық файлдардың max саны */

typedef struct _iobuf {
      int cnt; /* қалған таңбалар саны */
      char *ptr; /* келесі таңбаның орны */
      char *base; /* буфердің адресі */
      int flag; /* кіру режимі */
      int fd; /* файл дескрипторы */
} FILE;
extern FILE _iob[OPEN_MAX];

#define stdin (&iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])

enum _flags {
      _READ =01, /* файл оқуға ашық */
      _WRITE = 02, /* файл жазбаға ашық */
      _UNBUF = 04, /* файл буферленбейді */
      _EOF = 010, /* бұл файлда EOF кездесті */
      _ERR = 020 /* бұл файлда қате кездесті */ };

int _fillbuf(FILE *);
int _flushbuf(int, FILE *);

#define feof(p) (((p)->flag & _EOF) != 0)
#define ferror(p) (((p)->flag & _ERR) != 0)
#define fileno(p) ((p)->fd)
#define getc(p) (--(p)->cnt >= 0 \
      ? (unsigned char) *(p)->ptr++ : _fillbuf(p))
#define putc(x.p) (--(p)->cnt >= 0 \
      ? *(p)->ptr++ = (x) : _flushbuf((x),p))
#define getchar() getc(stdin)
#define putchar(x) putc((x), stdout)
```

Getc макросы әдетте буфердегі таңбалар санының есептегішін азайтады және таңбаны қайтарады, содан кейін көрсеткіш бірлікке ұзартады. (Ұзақ #define келесі жолдарда кері көлбеу сызық арқылы жалғастыруға болады еске саламыз.) Санауыш мәні теріс болған кезде, getc буферді қайтадан толтыру, құрылым мазмұнын инициа-

лизациялау және таңба беру үшін _fillbuf шақырады. Қайтарылатын таңбалардың түрлері unsigned-ге келтіріледі; бұл олардың барлығы оң болатындығының кепілі.

Мұнда енгізу-шығару егжей-тегжейлі қарастырылмаса да, біз әлі де putc толық анықтамасын алып келдік. Бұл буфер толы кезде _flushbuf функциясын тудыратын getc сияқты әрекет ететінін көрсету үшін жасалды. Мәтінде қате және файл соңына, сондай-ақ оның дескрипторына кіруге мүмкіндік беретін макростар бар.

Енді fopen функциясын жазуға болады. Fopen нұсқауларының басым бөлігі файлды ашуға, оның тиісті орналасуы мен ағымдағы күй индикациясына арналған құсбелгі биттерін орнатуға жатады. Fopen өзі буферге орын бермейді; бұл файлды бірінші оқыған кезде _fillbuf жасайды.

```
#include <fcntl.h>
#include “syscalls.h”

#define PERMS 0666 /* RW меншік иесі, топ және басқалар үшін.*/

/* fopen: файлды ашады, файл көрсеткішін қайтарады */
FILE *fopen(char *name, char *mode) {
      int fd;
      FILE *fp;

      if (*mode != ‘r’ && *mode != ‘w’ && *mode != ‘a’)
          return NULL;
      for (fp = _iob; fp < _iob + OPEN_MAX; fp++)
          if ((fp->flag & (_READ | _WRITE)) == 0)
              break; /* бос орын табылды */
          if (fp >= _iob + OPEN_MAX) /* бос орын жоқ */
              return NULL;
          if (*mode == ‘w’ )
               fd = creat(name, PERMS);
          else if (*mode == ‘a’ ) {
               if ((fd = open(name, O_WRONLY, 0)) == -1)
                   fd = creat(name, PERMS);
               lseek(fd, 0L, 2);
          } else
               fd = open(name, O_RDONLY, 0);
```

```
        if (fd == -1) /* name аты бойынша кіру мүмкін емес */
            return NULL;
        fp->fd = fd;
        fp->cnt = 0;
        fp->base = NULL;
        fp->flag = (*mode == 'r' ) ? _READ : _WRITE;
        return fp;
}
```

Мұнда fopen нұсқасы стандартта айтылған барлық кіру режимдерін іске асырмайды; бірақ біз оларды толық көлемде іске асыру Бағдарламаның ұзындығын арттырады деп ойлаймыз. Біздің fopen екілік енгізу-шығару туралы сигнал беретін b әріптерін танымайды (UNIX жүйелерінде бұл мағынасы жоқ) және бір мезгілде оқу және жазу мүмкіндігін көрсететін + белгісін танымайды.

Кез келген файл үшін оған бірінші рет жүгінген кезде getc макровы көмегімен CNT есептегіші нөлге тең. Осының салдарынан _fillbuf шақыруы болады. Егер оқу файлы ашылмаса, _fillbuf функциясы дереу EOF қайтарады. Әйтпесе, ол буферге арналған жадты сұрауға тырысады (егер оқу буферленуі керек болса).

Буферге арналған жад аймағын алғаннан кейін _fillbuf оны толтыру үшін read-ға жүгінеді, есептегіш пен сілтегіштерді орнатады және буфердің бірінші таңбасын қайтарады. Келесі кезекте _fillbuf буфер жадының таңдалғанын анықтайды.

```
#include "syscalls.h"

/* _fillbuf: жад сұрау және буферді толтыру */
int _fillbuf(FILE *fp)
{
    int bufsize;
    if ((fp->flag&(_READ | _EOF | _ERR)) != _READ)
        return EOF;
    bufsize = (fp->flag & _UNBUF) ? 1 : BUFSIZ;
    if (fp->base == NULL) /* буфер әлі жоқ */
        if ((fp->base = (char *) malloc(bufsize)) == NULL)
            return EOF; /* буферді алу мүмкін емес */
    fp->ptr = fp->base;
```

```
    fp->cnt = read(fp->fd, fp->ptr, bufsize);
    if (--fp->cnt < 0) {
        if (fp->cnt == -1)
            fp->flag |= _EOF;
        else
            fp->flag |= _ERR;
        fp->cnt = 0;
        return EOF;
     }
     return (unsigned char) *fp->ptr++;
}
```

Шоттың басталуын қалай ұйымдастыру керектігі түсініксіз болып қалды. _Iob массивін анықтап, инициализациялау керек, сонда бағдарлама іске қосылмас бұрын stdin, stdout және stderr файлдары туралы ақпарат болады.

```
FILE _iob[OPEN_MAX] =
      { /* stdin, stdout, stderr: */
      { 0, (char *) 0, (char *) 0, _READ, 0 },
      { 0, (char *) 0, (char *) 0, _WRITE, 1 },
      { 0, (char *) 0, (char *) 0, _WRITE | _UNNBUF, 2 }
};
```

Құрылымның бір бөлігі ретінде flag инициализациялау stdin оқуға ашық екенін, жазу үшін stdout және буферлеусіз жазуға арналған stderr екенін білдіреді.

8.2-жаттығу. fopen және _fillbuf функцияларын жалаушаларға өрістер ретінде қарау үшін, нақты бинарлық амалдарды қолданудан гөрі қайта жазыңыз. Бағдарламаның екі нұсқасының өлшемдері мен жылдамдықтарын салыстырыңыз.

8.3-жаттығу. _flushbuf, fflush және eclose мүмкіндіктерін жасаңыз және жазыңыз.

8.4-жаттығу. Стандартты кітапхана функциясы

```
int fseek(FILE *fp, long offset, int origin)
```

lseek-ке ұқсас, айырмашылығы fp файл дескриптор емес, файл сілтемесі болып табылады және оның күйін емес, интеллектуалды күйді білдіретін int береді. Fseek-тің нұсқасын жазыңыз. Fseek-тің буферлеу әрекеті кітапхананың басқа функциялары пайдаланатын буфермен үйлесетініне көз жеткізіңіз.

8.6. Мысал. Каталогтарды басып шығару

Файл жүйесімен әр түрлі өзара әрекеттестікте кейде оның мазмұны емес, файл туралы ақпаратты алу қажет. Мұндай қажеттілік, мысалы, UNIX жүйесінің ls командасына ұқсас жұмыс істейтін файлдар каталогын басып шығару бағдарламасында пайда болады. Ол каталог файлдарының аттарын және пайдаланушының қалауы бойынша басқа қосымша ақпаратты (өлшемдер, кіру құқықтары және т. б.) басып шығарады. MS-DOS-дегі ұқсас команда-dir.

Каталог UNIX-та файл болғандықтан, файлдар атына ену үшін тек функциялар оны оқу керек. Бірақ файл туралы басқа ақпаратты алу үшін (мысалы, оның өлшемін білу) жүйелік қоңырауды орындау керек. Басқа жүйелерде (MS-DOS-да, мысалы) жүйелік қоңырауды тіпті файл аттарына қол жеткізу үшін де қолдануға тура келеді. Біздің мақсатымыз – ақпаратқа қол жеткізуді қамтамасыз ету іске асыру айтарлықтай жүйелі-тәуелді болғанына қарамастан, мүмкіндігінше жүйелі-тәуелсіз тәсілмен.

Мұны fsize бағдарламасын жазу арқылы суреттейік. Fsize функциясы - ls бағдарламасының ерекше жағдайы; ол пәрмен жолында тізімделген барлық файлдардың өлшемдерін басып шығарады. Егер кез келген файлдың өзі каталог болса, онда fsize ол туралы ақпарат алу үшін өзіне қарайды. Егер пәрмен жолында дәлелдер болмаса, ағымдағы каталог өңделеді.

UNIXе файлдық жүйесінің құрылымын еске түсіруден бастайық. Каталог дегеніміз – бұл файл атаулары мен олардың орналасқан жері туралы кейбір ақпараттарды қамтитын файл. «Орын» – бұл

«инодтар тізімі» деп аталатын басқа кестеге қол жеткізуді қамтамасыз ететін индекс. Әрбір файлдың жеке inode бар, оның атауынан басқа файл туралы барлық ақпарат бар. Каталогтың әр жазбасында екі бөлік бар: файл атауы және inode нөмірі.

Өкінішке орай, каталогтың форматы мен нақты мазмұны жүйенің түрлі нұсқаларында бірдей емес. Сондықтан, тасымалданатын компонентті көтерілмейтін бөліктен бөлу үшін, біздің міндетімізді екіге бөліңіз. Сыртқы деңгей Dirent деп аталатын құрылымды және opendir, readdir және closedir үш кіші бағдарламасын анықтайды. Біз fsize бағдарламасын жазамыз, осындай интерфейске есептей отырып, содан кейін version 7 және System V UNIX сияқты бірдей каталог құрылымын пайдаланатын жүйелер үшін көрсетілген функцияларды қалай іске асыруға болатынын көрсетеміз. Басқа опциялар жаттығулар үшін қалдырылады.

Dirent құрылымы inode нөмірін және атауын қамтиды. Файл атауының максималды ұзындығы NAME_MAX-ға тең-бұл жүйелік-тәуелді мән. Opendir функциясы readdir және closedir функцияларымен пайдаланылатын DIR (FILE-ге ұқсас) деп аталған құрылымның көрсеткішін қайтарады. Бұл ақпарат dirent.h тақырып файлында шоғырланған.

```
#define NAME_MAX 14 /* файл атауының максималды ұзындығы; */
/* жүйелік-тәуелді шама */

typedef struct { /* универс. каталог элементінің құрылымы: */
    long ino; /* inode нөмірі */
    char name[NAME_MAX+1]; / * аты + соңы '\0' */
} Dirent;

typedef struct { /* минималды DIR: буферизациясыз және т. б.*/
    int fd; /* каталогтың файлдық дескрипторы */
    Dirent d; /* каталог элементі */ } DIR;

DIR *opendir(char *dirname);
 Dirent *readdir(DIR *dfd);
void closedir(DIR *dfd);
```

Stat жүйелік қоңырауы файл атын алады және inode торабында қамтылған толық ақпаратты қайтарады, немесе-1 қате болған жағдайда. Сонымен,

```
char *name;
struct stat stbuf;
int stat(char *, struct stat *);
stat(name, &stbuf);
```

stbuf құрылымын inode торабынан name аты бар файл туралы ақпаратпен толтырады. Stat функциясы қайтарылатын мәнді сипаттайтын құрылым <sys/stat.h> шамамен былай көрінеді:

```
struct stat { /* stat қайтарылатын inode ақпарат */
dev_t st_dev; /* құрылғы */
ino_t st_ino; /* inode нөмірі */
short st_mode; /* режимдік биттер */
short st_nlink; /* файл байланыстарының саны */
short st_uid; /* пайдаланушы-меншік иесінің аты */
short st_gid; /* меншік иесі тобының аты */
dev_t st_rdev; /* арнайы файлдар үшін */
off_t st_size; /* таңбалардағы файл өлшемі */
time_t st_atime; /* соңғы пайдалану уақыты */
time_t st_mtime; /* соңғы өзгерту уақыты */
time_t st_ctime; /* inode соңғы өзгерту уақыты */
};
```

Бұл мағыналардың көпшілігі түсініктемелерде түсіндірілген. Dev_t және ino_t сияқты типтер <sys / types.h> файлында анықталған, ол да # қамтуы керек.

St_mode элементінде файл туралы қосымша ақпарат беретін жалаулар жиынтығы бар. Жалаулар анықтамалары <sys/stat.h> ішінде де кездеседі; бізге тек файл түріне қатысты бөлік қажет:

```
#define S_IFMT 0160000 /* файл түрі */
#define S_IFDIR 0040000 / * каталог */
#define S_IFCHR 0020000 / * символдық бағытталған */
#define S_IFBLK 0060000 / * блок-бағдарланған */
#define S_IFREG 0100000 / * қалыпты */
```

Енді біз fsize бағдарламасын жазуға дайынбыз. Егер stat-тан алынған режимдік биттер (st_mode) файл каталог емес екенін көрсетсе, оның өлшемін (st_size) алып, басып шығаруға болады. Алайда, егер файл каталог болса, онда біз оның барлық файлдарын өңдеуіміз керек, олардың әрқайсысы өз кезегінде каталог болуы мүмкін. Каталогты өңдеу – рекурсивті процесс.

Main бағдарламасы пәрмен жолының параметрлерін қарап, fsize функциясының әрбір аргументін жібереді.

```
#include <stdio.h>
#include <string.h>
#include “syscalls.h”
#include <fcntl.h> / * оқу және жазу ұяшықтары */
#include <sys/types.h> /* типін анықтау */
#include <sys/stat.h> / * stat қайтарылатын құрылым */
#include “dirent.h”

void fsize(char *);

/* файл өлшемдерін басып шығарады */
main(int argc, char **argv)
{
    if (argc == 1) /* әдепкі бойынша ағымдағы каталог */
        fsize(“.”);
    else
        while (--argc > 0)
            fsize(*++argv);
    return 0;
}
```

Fsize функциясы файл өлшемін басып шығарады. Дегенмен, файл каталогы болса, ол алдымен dirwalk-ді барлық файлдарды өңдеу үшін тудырады. S_IFMT және S_IFDIR құсбелгілерінің атаулары < sys/stat.h> файл каталог болып табылады ма тексеру кезінде. Бұл жерде жақшалар қажет, себебі & оператордың басымдығы == оператордың басымдығынан төмен.

```
int stat(char *, struct stat *);
void dirwalk(char *, void (*fcn)(char *));

/* fsize: “ name” файл өлшемін басып шығарады */
```

```
void fsize(char *name)
{
    struct stat stbuf;
    if (stat(name, Sstbuf) == -1) {
        fprintf(stderr, “fsize: %s\n”, name қолжетімділік жоқ);
        return;
    }
    if ((stbuf.stjnode & S_IFMT) == S_IFDIR)
        dirwalk(name, fsize);
    printf(“%81d %s\n”, stbuf.st_size, name);
}
```

Dirwalk функциясы – әрбір каталог файлына кейбір функцияны қолданатын әмбебап бағдарлама – Ол каталогты ашады, циклдің көмегімен ондағы файлдарды іріктейді, олардың әрқайсысына көрсетілген функцияны қолданады, содан кейін каталогты жабады және қайтаруды жүзеге асырады. Себебі fsize әрбір каталогта dirwalk тудырады, бұл екі функцияда жанама рекурсия салынған.

```
#define MAX_PATH 1024

/* dirwalk: FCN барлық dir файлдарына қолданады */
void dirwalk(char *dir, void (*fcn)(char *))
{
    char name[MAX_PATH];
    Dirent *dp;
    DIR *dfd;

    if ((dfd = opendir(dir)) == NULL) {
        fprintf(stderr, “dirwalk: ашылмайды %s\n”, dir);
        return;
    }
    while ((dp = readdir(dfd)) != NULL) {
        if (strcmp(dp->name, “.”) == 0
            || strcmp(dp->name, “..”) == 0)
            continue; /* өзін және ата-ананы жіберіп алу */
        if (strlen(dir)+strlen(dp->name)+2 > sizeof(name))
            fprintf (stderr, “dirwalk: өте ұзын ат %s/%s\n”,
                dir, dp->name);
        else {
            sprintf(name, “%s/%s”, dir, dp->name);
            (*fcn)(name);
        }
```

```
        }
        closedir(dfd);
}
```

Әрбір readdir қоңырауы барлық файлдар өңделсе, келесі файл немесе NULL туралы ақпаратты көрсеткіні қайтарады. Кез келген каталог әрқашан өзі туралы ақпаратты файлда "." атымен сақтайды және аты жазылған файлда өз ата-анасы туралы ".." атымен; оларды жіберіп алу керек, әйтпесе бағдарлама жабылады. Назар аударыңыз: осы деңгейдегі бағдарлама коды каталогтардың пішімделуіне байланысты емес. Келесі қадам – кейбір нақты жүйе үшін opendir, readdir және closedir ең аз нұсқаларын ұсыну. Мұнда Version 7 және System V UNIX жүйелері үшін бағдарламалар бар. Олар <sys/dir.h> тақырып файлында сақталған каталог туралы ақпаратты пайдаланады, ол келесідей:

```
#ifndef DIRSIZ
#define DIRSIZ 14
#endif
struct direct /* каталог элементі */
{
    ino_t d_ino; /* inode нөмірі  */
    char d_name[DIRSIZ]; /* ұзын атауы жоқ '\0' */
};
```

Жүйенің кейбір нұсқалары ұзын атауларға жол береді және каталогтың күрделі құрылымына ие.

Ino_t түрі typedef көмегімен орнатылған және inode тораптар тізімінің индексін сипаттайды. Біз пайдаланатын жүйеде бұл түрі unsigned short бар, бірақ басқа жүйелерде ол басқаша болуы мүмкін, сондықтан оны typedef арқылы анықтау жақсы. "Жүйелік" типтердің толық жиынтығы <sys/types.h>.

Opendir функциясы каталогты ашады, ол шын мәнінде каталог болып табылатынын тексереді (бұл жағдайда stat ұқсас fstat жүйелік қоңырау арқылы жасалады, бірақ файлдың дескрипторы қолданылады), каталог құрылымы үшін кеңістік сұрайды және ақпаратты жазады.

```
int fstat(int fd, struct stat *);
```

```
/* opendir: readdir қоңырау каталогын ашады */
DIR *opendir(char *dirname)
{
    int fd;
    struct stat stbuf;
    DIR *dp;

    if ((fd = open(dirname, 0_RDONLY, 0)) == -1
        || fstat(fd, Sstbuf) == -1
        || (stbuf.stjnode & S_IFMT) != S_IFDIR
        || (dp = (DIR *) malloc(sizeof(DIR))) == NULL)
        return NULL;
    dp->fd = fd;
    return dp;
}
```

Closedir функциясы каталогты жабады және бос орынды босатады.

```
/* closedir: opendir ашық каталогты жабады */
void closedir(DIR *dp)
{
    if (dp) {
        close(dp->fd);
        free(dp);
    }
}
```

Соңында, readdir read арқылы каталогтың әрбір элементін оқиды. Егер каталогтың белгілі бір элементі қазіргі уақытта пайдаланылмаса (оған сәйкес файл жойылған), онда inode торабының нөмірі нөлге тең және бұл позиция жіберіледі. Олай болмаған жағдайда inode нөмірі мен аты статикалық (static) құрылымға орналастырылады және оның көрсеткіші нәтиже ретінде беріледі. Әрбір келесі айналымда жаңа ақпарат алдыңғы орынға ие.

```
#include <sys/dir.h> /* каталог құрылымының орналасқан жері */

/* readdir: каталог элементтерін дәйекті оқиды */
Dirent *readdir(DIR *dp)
{
```

```
        struct direct dirbuf; /* осы жүйедегі каталог құрылымы */
        static Dirent d; /* біртұтас құрылымды қайтарады */

        while (read(dp->fd, (char *) &dirbuf, sizeof (dirbuf ))
              == sizeof (dirbuf)) {
              if (dirbuf.d_ino == 0) /* бос элемент пайдаланылмайды */
                  continue;
              d.ino = dirbuf.d_ino;
              strncpy(d.name, dirbuf.d_name, DIRSIZ);
              d.name[DIRSIZ] = '\0'; /* соңғы таңба '\0' */
              return &d;
        }
        return NULL;
}
```

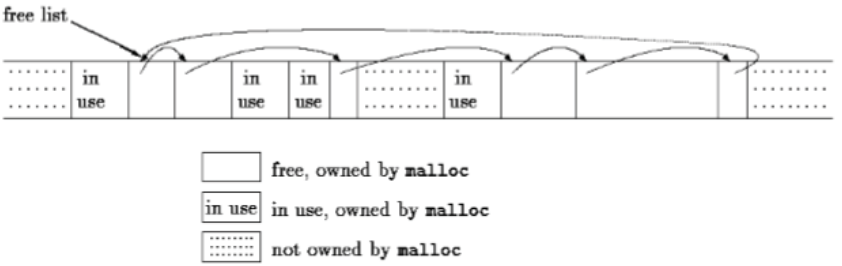
Fsize бағдарламасы өте мамандандырылған болса да, ол екі маңызды фактіні суреттейді. Бірінші: көптеген бағдарламалар "жүйелік" емес; олар операциялық жүйе сақтайтын ақпаратты ғана пайдаланады. Мұндай бағдарламалар үшін ақпаратты ұсыну тек стандартты тақырып файлдарында шоғырланған. Бағдарламалар өзінде хабарландырулар ұстап емес, осы файлдарды қамтиды. Екінші бақылау жүйелік-тәуелді объектілерге күш салғанда, өздері жүйелік-тәуелді болмайтын интерфейстерді құруға болады. Оған жақсы мысалдар-стандартты кітапхананың функциялары бола алады.

8.5-жаттығу. Inode торабында қалған ақпаратты басып шығару үшін fsize түрлендіріңіз.

8.7. Мысал. Жад таратқыш

5-тарауда стек принципіне негізделген қарапайым жад таратқышы сипатталды. Мұнда жазатын нұсқада шектеулер жоқ: malloc және free қоңыраулары кез-келген тәртіпте орындалуы мүмкін; malloc қажет болған кезде операциялық жүйеге жады бөлуге сұраныс жасайды. Бұл бағдарламалар машинаға тәуелді кодты салыстырмалы машинаға тәуелді тәсілмен алуға мүмкіндік беретін тәсілдерді бейнелейді, сонымен қатар, олар құрылым, біріктіру және typedef сияқты тіл құралдарын қолданудың мысалы бола алады.

Жад бөліктері бөлінетін бекітілген өлшемді бұрын құрастырылған массив жоқ. Malloc функциясы қажет болған жағдайда операциялық жүйеден жад сұрайды. Өйткені бағдарламаның басқа әрекеттері осы жад таратқышқа қарамастан жауап беретін жад сұрауларын тудыруы мүмкін. Сондықтан бос жад блоктар тізімі ретінде сақталады. Әрбір блок келесі блоктың өлшемін, көрсеткішін және кеңістіктің өзін қамтиды. Тізімде блоктар жад адрестерінің өсу ретімен сақталады, ал соңғы блок (ең үлкен мекенжай) бірін көрсетеді.



Жад сұралса, жеткілікті үлкен блок табылғанға дейін бос блоктардың тізімі көрінеді. Мұндай алгоритм “ең жақсы қолайлы іздеу” алгоритміне қарағанда “бірінші қолайлы іздеу” деп аталады, ол сұранысты қанағаттандыратын саннан ең аз блокты іздейді. Егер блок мөлшері дәл талаптарға сәйкес келсе, онда мұндай блок тізімнен шығарылады және пайдалануға беріледі. Егер блоктың өлшемі талап етілгеннен көп болса, ол қажетті бөлікті кесіп тастайды – ол пайдаланушыға беріледі, ал қажетсіз бос блоктар тізімінде қалады. Егер блоктар жеткілікті болмаса, онда операциялық жүйе бос блоктар тізіміне қосылатын тағы бір үлкен жады сұралады.

Босату рәсімі бос блоктар тізімі бойынша өтумен байланысты, себебі бос блок үшін қолайлы орын табу керек. Егер босатылуға жататын блок қандай да бір жағынан бос блоктардың біріне жанаса, онда ол мүмкіндігінше жадтың бөлшектенуін (фрагменттілігін) азайту үшін онымен үлкен көлемдегі бір блокқа біріктіріледі. Блок-

тардың бір-біріне жанасуын тексеру қиын емес, өйткені бос блоктардың тізімі әрдайым адрестердің өсуі бойынша реттелген.

5-тарауда айтылғандай, malloc функциясы беретін жады сақталатын нысандарды ескере отырып, тиісті түрде теңестірілуі керек. Бір-бірінен айырмашылығы бар, бірақ олардың әрқайсысы үшін теңестіруге ең үлкен талаптар қоятын үлгі бар, және егер қандай да бір мекен-жай бойынша осы түрдегі объектіні орналастыруға рұқсат етілген болса, онда ол бойынша барлық басқа түрдегі объектілерді орналастыруға болады. Кейбір машиналарда мұндай" талапшыл" түрі екі, басқалары бұл int немесе ұзын болуы мүмкін.

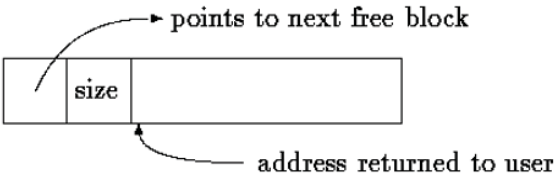
Бос блок тізімде келесі блоктың көрсеткішін, оның мөлшерін және нақты бос кеңістігін қамтиды. Көрсеткіш пен өлшем басқарушы ақпаратты білдіреді және "тақырып"деп аталады. Туралауды жеңілдету үшін барлық блоктар тақырыптың қысқаша өлшемімен жасалады және тақырып тиісті түрде туралanады. Осыған сәйкес тақырып құрылымы мен теңестіру туралы ең талап етілетін түрін қамтитын бірлестікті құрастыру арқылы қол жеткізуге болады. Нақтылық үшін біз long ұзын түрін таңдадық.

```
typedef long Align; / * long шекарасы бойынша туралау үшін */
union header { /* блок тақырыбы: */
    struct {
        union header *ptr; /* із. бос тізімдегі блок */
        unsigned size; / * осы блоктың өлшемі */
    } s;
    Align x; / * мәжбүрлі блокты теңестіру */
};
typedef union header Header;
```

Align өрісі еш жерде пайдаланылмайды; әрбір тақырып шекараның ең "ең нашар" нұсқасына туралануы үшін ғана қажет.

Талап етілген таңбалар саны malloc-қа тақырып өлшеміндегі жад бірліктерінің толық санына дейін дөңгелектенеді (дәл осы сан және тақырып size (өлшем) өрісінде жазылады); сонымен қатар, блокта жадтың тағы бір бірлігі – тақырып өзі кіреді. Malloc функциясы

қайтарылатын көрсеткіш тақырыпқа емес, бос кеңістікті көрсетеді. Еркін кеңістікпен пайдаланушы кез келген нәрсені істей алады, бірақ егер ол одан тыс нәрсе жазса, онда тізім бұзылады.



Malloc функциясымен басқарылатын жадтың байланысы жоқ болғандықтан, блоктардың өлшемдерін көрсеткіштер бойынша есептеуге болмайды, сондықтан өлшемді сақтайтын өріссіз біз жасай алмаймыз.

Жұмысты бастау үшін base айнымалы қолданылады. Егер freep NULL болса (malloc-қа алғаш рет жүгінгенде), бос кеңістіктің "азғантай" тізімін жасайды; ол өзіне арналған көрсеткішпен бір нөлдік өлшемді блокты қамтиды. Қолайлы өлшемнің бос блогын іздеу осы көрсеткіштен (freep) басталады, яғни соңғы табылған блоктан басталады; мұндай стратегия тізімді біркелкі ұстауға көмектеседі. Егер табылған блок тым үлкен болса, пайдаланушыға оның құйрығы беріледі; бұл жағдайда табылған бос блоктың тақырыбында оның өлшемін нақтылау қажет. Кез келген жағдайда пайдаланушыға қайтарылатын көрсеткіш блокта тікелей тақырыптан тыс орналасқан бос кеңістіктің мекен-жайы болып табылады.

```
static Header base; /* бастау үшін бос тізім */
static Header *freep = NULL; /* бос блоктар  тізімінде бастау */

/* malloc: әмбебап жад таратқыш */
void *malloc(unsigned nbytes)
{
      Header *p, *prevp;
      Header *morecore(unsigned);
      unsigned nunits;

      nunits = (nbytes + sizeof (Header) - 1) / sizeof (Header) + 1;
```

```
    if ((prevp = freep) == NULL) { /* бос жадтар тізімі жоқ */
        base.s.ptr = freep = prevp = &base;
        base.s.size = 0;
    }
    for (p = prevp->s.ptr; ; prevp = p, p = p->s.ptr) {
        if (p->s.size >= nunits) { /* жеткілікті үлкен */
            if (p->s.size == nunits) /* дәл қажетті өлшем */
                prevp->s.ptr = p->s.ptr;
            else { /* біз құйрықты кесеміз */
                p->s.size -= nunits;
                p += p->s.size;
                p->s.size = nunits;
            }
            freep = prevp;
            return (void *)(p+1);
        }
        if (p == freep) /* тізім бойынша толық цикл өтті */
            if ((p = morecore(nunits)) == NULL)
                return NULL; /* басқа жад жоқ */
    }
}
```

Morecore функциясы операциялық жүйеден жад алады. Бұл қалай жасалатыны туралы мәліметтер әртүрлі жүйелерде сәйкес келмеуі мүмкін. Жүйеден жад сұрауы-салыстырмалы түрде қымбат операция болғандықтан, біз бұл үшін malloc-ке хабарласқымыз келмейді. Сондықтан morecore функциясы пайдаланылады, ол кемінде NALLOC жады бірліктерін сұрайды; бұл үлкен жады қажет болған жағдайда “кесіледі”. Тиісті мәнді өлшем өрісіне орнатқаннан кейін morecore функциясы тегін функцияны тудырады және алынған бөлікті бос жад аймақтары тізіміне қосады.

UNIXе-де sbrk (n) жүйелік шақыру меңзерді n байт жадына қайтарады, немесе егер қажет бос орын болмаса -1, алайда, егер ол NULL-ді қайтарса жақсы болар еді. 1 тұрақтысын қайтарылатын мәнмен салыстыру үшін char * түріне келтіру керек. Бұл түрін келтіру операциясы функцияны әртүрлі машиналарда нақты көрсетулерге қатысты тәуелсіз етеді. Дегенмен, sbrk функциясы беретін әртүрлі блоктарға арналған көрсеткіштерді салыстыратын бір” қате “ бар. Мұндай салыстыру стандартты емес, ол көрсеткіштерді бір

массивте ғана салыстыруға мүмкіндік береді. Осылайша, бұл malloc нұсқасы кез келген көрсеткіштерді салыстыруға болатын машиналарда ғана дұрыс.

```
#define NALLOC 1024 /* сұрау үшін жад бірліктерінің минималды саны */

/* morecore: жүйеден қосымша жад сұрайды */
static Header * morecore( unsigned nu)
{
    char *cp, *sbrk(int);
    Header *up;

    if (nu < NALLOC)
       nu = NALLOC;
    cp = sbrk(nu * sizeof(Header));
    if (cp == (char *) -1) /* басқа жад жоқ */
        return NULL;
    up = (Header *) cp;
    up->s.size = nu;
    free((void *)(up+1));
    return freep;
}
```

Қорытындылай келе, free функциясын қарастырайық. Ол кірістірілген блоктың орнын табу үшін freep-тен бастап бос жады тізімін қарайды. Талап етілетін орын блоктар арасында немесе тізімнің басында немесе оның соңында болуы мүмкін. Кез келген жағдайда, егер босатылуға жататын блок көрші блок, ол онымен бір блокқа біріктіріледі. Келесісі — бұл көрсеткіштер қажетті орындар мен блоктардың өлшемдері дұрыс екенін көрсетеді.

```
/* free: бос жады тізіміне блокты қамтиды */
void free(void *ap)
{
     Header *bp, *p;

     bp = (Header *)ap -1; /* блоктың айдарына меңзер */
     for (p=freep; !(bp > p && bp < p->s.ptr); p = p->s.ptr)
         if (p >= p->s.ptr && (bp > p || bp < p->s.ptr))
             break; /* блокты босату басында немесе соңында */
     if (bp + bp->s.size == p->s.ptr) { /* жоғарғы жағымен біріктіріңіз */
         bp->s.size += p->s.ptr->s.size; /* көршілеріне */
```

```
        bp->s.ptr = p->s.ptr->s.ptr;
    } else
        bp->s.ptr = p->s.ptr;
    if (p + p->s.size == bp) { /* төменгі көршімен біріктіру */
        p->s.size += bp->s.size;
        p->s.ptr = bp->s.ptr;
    } else
         p->s.ptr = bp;
    freep = p;
}
```

Жадты бөлу көбінесе машинаға байланысты проблема болса да, оны шешуге болады, өйткені төмендегі бағдарлама суретте көрсетілгендей, машинаға тәуелділік оның кішкене бөлігінде жасырылған. Ортаға дәл келтіру мәселесіне келетін болсақ, біз оны typedef және кәсіподақпен шештік (sbrk түзетуге сәйкес көрсеткішті ұсынады деп болжанады). Кастинг жұмыстары бізге түрлендірулерді анық жасауға және тіпті нашар құрастырылған жүйелік интерфейспен айналысуға мүмкіндік береді. Біздің ойымыз жадты бөлу туралы болса да, бұл жалпы тәсіл басқа жағдайларға да қатысты.

8.6-жаттығу. Calloc(n, size) стандартты функциясы N өлшемімен толтырылған size жад элементтеріне көрсеткішін қайтарады. Malloc функциясын пайдаланып немесе соңғысын өзгерту арқылы callos нұсқасын жазыңыз.

8.7-жаттығу. Malloc функциясы кез келген өлшемге рұқсат береді, оны сенімді түрде тексермейді; free босатылатын блоктың өлшемі дұрыс деп болжайды. Бұл бағдарламаларды қателерді мұқият бақылау үшін жақсартыңыз.

8.8-жаттығу. Bfree(p, n) бағдарламасын N таңбадан тұратын еркін p блогын босатыңыз, оны malloc және free функцияларымен қолдайтын бос жад тізіміне қосу арқылы жазыңыз. Bfree көмегімен пайдаланушы кез келген уақытта бос жад тізіміне статикалық немесе сыртқы массив қосу мүмкіндігі болуы керек.

А. Анықтамалық нұсқаулық

А 1. Кіріспе

Бұл нұсқаулық ANSI-де вкпараттық жүйелер үшін Американдық ұлттық стандарт ретінде бекітілген жобаға сәйкес 1989 жылдың 31-қазанында анықталған С бағдарламалау тілін сипаттайды: С бағдарламалау тілі, X3.159-1989 (“American National Standard for Information Systems-Programming Language C, X3.159-1989”). Бұл сипаттама стандарттың өзі емес, ұсынылған стандарттың бір нұсқасы ғана, бірақ біз оны сенімді тілдік нұсқаулыққа айналдыруға ерекше назар аударды қ.

Бұл құжат негізінен стандарттың жалпы сипаттамалық сызбасына сәйкес келеді (жарияланым осы кітаптың бірінші басылымына негізделген), бірақ ұйымдастырушылық айырмашылықтар бар. Бірнеше өнім атауларындағы ауытқулардан және лексемалар мен препроцессордың ресми анықтамаларының жоқтығынан басқа, мұндағы тілдің грамматикасы мен стандарттағы грамматика баламалы.

Әрі қарайғы жазбалар (осы сияқты) парақтың сол жақ шетінен шегініспен жазылады. Негізінен, бұл жазбалар осы кітаптың бірінші басылымында сипатталған стандарт пен тілдің нұсқасы арасындағы айырмашылықтар мен әртүрлі компиляторлардағы кейінгі жаңашылдықтар туралы.

А 2. Лексика туралы келісімдер

Бағдарлама файлдар түрінде сақталатын бір немесе бірнеше трансляция бірлігінен тұрады. Әрбір осындай бірлік А12-да сипатталған трансляцияның бірнеше фазасынан өтеді. Бастапқы фазалар төменгі деңгейдің лексикалық түрленуін жүзеге асырады, # белгісінен басталатын жолдармен бағдарламада берілген директиваларды орындайды, макроанықтауды өңдейді және макросширлеуді жүргізеді. Препроцессор (А12) жұмысы аяқталғаннан кейін бағдарлама лексеманың реттілігі түрінде ұсынылады.

А 2.1. Лексемалар (tokens)

Белгілердің алты класы бар (немесе таңбалауыштар): идентификаторлар, кілт сөздер, тұрақтылар, жол әріптері, операторлар және басқа бөлгіштер. Бос орындарды, көлденең және тік қойындыларды, жаңа жолдарды, парақтарды және ескертулерді (жиынтықты бөлгіш белгілер деп атайды) компилятор тек токендер бөлгіші ретінде қарастырады және басқаша аударма нәтижесіне әсер етпейді. Бөлгіш белгілердің кез келгенін көршілес сәйкестендіргіштерді, кілт сөздерді және тұрақты мәндерді бір-бірінен ажырату үшін пайдалануға болады.

Егер кіріс ағыны белгілі бір таңбаларға дейін таңбаланған болса, келесі таңбалауыш лексема болуы мүмкін ең ұзын жол болады.

А 2.2. Түсініктеме

/ * таңбалары түсініктемені ашады және * / таңбалары оны жабады. Түсініктемелер бір-біріне кірістірілмейді, оларды жолдарға немесе мәтіндік әріптерге орналастыруға болмайды.

А 2.3. Идентификаторлар

Идентификатор — әріптер мен сандар тізбегі. Бірінші символ әріп болуы тиіс; астын сызу белгісі әріп болып саналады. Төменгі және жоғарғы регистрлердің әріптері әртүрлі. Идентификаторлар кез келген ұзындыққа ие болуы мүмкін; ішкі идентификаторлар үшін алғашқы 31 символ маңызды; кейбір іске асыруларда маңызды символдар саны көп. Ішкі идентификаторларға макростардың аттары және сыртқы байланыстары жоқ барлық басқа есімдер (А11.2) жатады. Сыртқы байланыстары бар идентификаторларға үлкен шектеулер қойылуы мүмкін: кейде бірінші алты таңбадан артық емес қабылданады және жоғарғы және төменгі регистрлердің әріптері ажырамауы мүмкін.

А 2.4. Кілт сөздер

Келесі идентификаторлар кілт сөздер ретінде сақталған және басқа мағынада қолданыла алмайды:

```
auto        double      int         struct
break       else        long        switch

case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

Кейбір іске асыруларда fortran және asm сөздері сақталады.

Const, signed және volatile кілт сөздері алғаш рет ANSI стандартында пайда болды; enum және void-кітаптың бірінші басылымына қатысты жаңалары, бірақ қолданылған; бұрын сақталған entry еш жерде пайдаланылмады және сондықтан бұдан былай сақталмайды.

А 2.5. Тұрақтылар

Тұрақтылардың бірнеше түрі бар. Әрқайсысының өзіндік деректер түрі бар; негізгі түрлері А4.2-де қарастырылған.

тұрақтылар:
- бүтін тұрақты
- символьдік-тұрақты
- өзгермелі нүктелі тұрақтылар
- тұрақты-тізімдеу

А 2.5.1. Бүтін тұрақты

Сандардың реттілігінен тұратын бүтін константа, егер ол 0-ден (нөл сандары) басталса, сегіздік деп қабылданады және олай бол-

маған жағдайда ондық деп қабылданады. Сегіздік тұрақтыда 8 және 9 сандар жоқ. 0X немесе 0X тұрған сандар тізбегі он алтылық бүтін ретінде қарастырылады. Он алтылық санға 10-нан 15-ке дейінгі мәндерімен a (немесе A) - дан f (немесе F) - ға дейінгі әріптер енгізілген.

Толық тұрақты әріптермен u (немесе U) жұрнақсыз тұрақты ретінде сипатталуы үшін жазылуы мүмкін. Ол сондай-ақ L (немесе L) жұрнағы ұзын long түрі бар екенін көрсету үшін болуы мүмкін.

Бүтін константалардың түрлері тілдің бірінші редакциясымен салыстырғанда елеулі дамуға ие болды, онда үлкен тұтас ұзын түрі болды. U және u жұрнақтары алғаш рет енгізілді.

А 2.5.2. Символьдік-тұрақты

Символьдік константа – бір немесе бірнеше таңбалардың тізбегі, бір тырнақшаға (мысалы, ‘x’). Егер бір тырнақшаның ішінде бір таңба орналасқан болса, тұрақтының мәні осы машинада қабылданған кодтауда осы символдың сандық мәні болып табылады. Бірнеше таңбалы константаның мәні іске асырылуға байланысты.

Символдық константа жалғыз тырнақшаны немесе жаңа жолдың белгісін қамтуы мүмкін емес; оларды және кейбір басқа таңбаларды бейнелеу үшін эскейп-реттілік қолданылуы мүмкін:

жаңа жол (newline, linefeed)	NL (LF)	\n
көлденең табуляция (horisontal tab)	HT	\t
тік табуляция (vertical tab)	VT	\v
қадамға оралу (backspace)	BS	\b
каретканы қайтару (carriage return)	CR	\r
беттің аудармасы (formfeed)	FF	\f
сигнал қоңырау (audible alert, bell)	BEL	\a
кері көлбеу сызық (backslash)	\	\\
сұрақ белгісі (question mark)	?	\?
бір тырнақша (single quote)		\’
қос тырнақша (double quote)		\”
сегіздік код (octal number)	ooo	\ooo
он алтылық код (hex number)	hh	\xhh

Эскейп-бірізділік \ooo қалаған символдың мәнін сипаттайтын бір, екі немесе үш сегіз санды құрайтын кері көлбеу сызықтан тұрады. Мұндай конструкцияның ең жиі мысалы болып табылады \0 (одан кейін сан болмайды); ол null-символын сипаттайды. Эскейп тізбегі \xhh x әрпімен кері көлбеу сызықтан тұрады, содан кейін қалаған символдың мәнін сипаттайтын он алтылық сандар. Сандар санына шектеу жоқ, бірақ алынған таңбаның мәні рұқсат етілген таңбалардың ең "үлкен" мәнінен асып кетсе, нәтиже анықталмайды. Егер осы іске асыруда char түрі белгісі бар сан ретінде түсіндірілсе, онда мәні сегіздік және он алтылық және эскейп-бірізділікте "белгіні тарату" көмегімен, Егер char түріне келтіру операциясы орындалғандай алынады. Егер жоғарыда аталған таңбалардың ешқайсысы болмаса, нәтиже анықталмайды.

Кейбір іске асыруларда char түрін қамтымайтын кеңейтілген таңбалар жиынтығы бар. Мұндай теру үшін тұрақты l әріпімен жазылады (мысалы, L' X') және кеңейтілген символ тұрақты деп аталады. Бұл тұрақты wchar_t түрі бар (стандартты тақырып файлында анықталған бүтін түрі <stddef.h>). Кәдімгі символдық константалар сияқты, мұнда сегіз және он алтылық эскейп-реттілік болуы мүмкін; егер арнайы мән wchar_t түрінен асып кетсе, нәтиже анықталмайды.

Келтірілген эскейп тізбектерінің кейбірі жаңа (атап айтқанда, он алтылық). Жаңа таңбалар үшін кеңейтілген түрі. Әдетте пайдаланылатын таңбалар Америка және Батыс Еуропа, char түрі қолайлы және wchar_t түрі негізінен Азия тілдері үшін қосылды.

А 2.5.3. Өзгермелі нүктелі тұрақтылар

Өзгермелі нүктелі тұрақтылар бүтін бөліктен, ондық нүктеден, бөлшек бөліктен, е немесе Е және бүтіннен (мүмкін, белгімен) тұрады, және, мүмкін, F, F, L немесе L әріптердің біреуімен берілетін типті жұрнақ тұрады. Бүтін бөлік немесе бөлшек бөлік (бірақ екеуі бірге емес) болмауы мүмкін; сондай-ақ ондық нүкте немесе тәртіппен (бірақ екеуі бір мезгілде емес) болмауы мүмкін. Түрі жұрнақпен

анықталады; F немесе f F F float түрін, L немесе l — L-L-L-түрін анықтайды; жұрнақ болмаған жағдайда double түрін білдіреді.

Өзгермелі нүктелі тұрақтылар үшін жұрнақтар жаңалық болып табылады.

A 2.5.4. Тұрақты-тізімдеу

(A8.4) санау элементтері ретінде жарияланған идентификаторлар int типті тұрақтылар болып табылады.

A 2.6. Жол литерлары

Сондай-ақ, жол тұрақты деп аталатын жол литерал – бұл қос тырнақшаға салынған символдар тізбегі (мысалы,"..."). Жол "таңбалар массиві" және static (A4) класының жадына ие, ол берілген таңбалармен инициаланады. Бірдей жол литерлары бір көшірмемен немесе бірнеше рет ұсынылма, іске асыруға байланысты. Жол литералын өзгертуге тырысатын бағдарламаның мінез-құлқы анықталмаған.

Қатар жазылған жол литерлары бір жолға біріктіріледі (конкатенирленеді). Кез келген конкатенациядан кейін жолға null-байт (\0) қосылады, бұл жолды қарайтын бағдарламаға оның соңын табуға мүмкіндік береді. Жол литералдары жаңа жолдың символын немесе қос тырнақшаны қамтымайды; оларда символдық константаларда бірдей эскейп-реттілікті пайдалану керек.

Символьдік константалар сияқты, кеңейтілген жиынтықтан символдары бар жол литерал l әрпінен басталуы тиіс (мысалы, l"..."). Кеңейтілген жиынтықтағы жол литерал "wchar_t массиві" түріне ие. Бір-бірімен кәдімгі және "кеңейтілген" жол литералдарының конкатенациясы анықталмаған.

> Жол литералдары міндетті түрде әр түрлі көшірмелермен, олардың модификациясына тыйым салу, сондай-ақ көршілес жол литералдарын конкатенациялау-ANSI-стандартының жаңа енгізілімі емес. "Кеңейтілген" жол литерлары да алғаш рет жарияланды.

А 3. Синтаксистік белгі

Осы оқулықта қолданылатын синтаксистік белгілерде синтаксистік ұғымдар курсивпен, ал сөзбе-сөз қабылданған таңбалар қалыпты шрифтпен жазылады. Балама конструкциялар, әдетте, бағанда тізімделеді (әр жол бөлек балама); сирек жағдайларда кішкентай балама нұсқалардың ұзын тізімдері біреуіне «біреуі» деген жазумен орналастырылады. Қосымша сөзжасамдық немесе терминдік емес «индекстелген» индекстеледі. Сонымен, жазба

{өрнек$_6$}

жалпы жағдайда болмауы мүмкін фигуралы жақшаларға жасалған өрнек білдіреді. Синтаксистік құрылымдардың толық тізімі А13 келтірілген.

Бұл кітаптың бірінші басылымында берілген грамматикадан айырмашылығы, мұнда келтірілген грамматика үлкендік пен операцияларды орындау тәртібі айқын сипаттайды.

А 4. Идентификаторлар нені білдіреді?

Идентификаторлар, немесе аттар, әр түрлі объектілерге сілтеме жасайды: функциялар; құрылым тегтері, бірлестіктер және тізімдер; құрылымдар немесе бірлестіктер элементтері; typedef-атаулар; белгілер және объектілер. Объект (кейде айнымалы деп аталатын) жадтың бір бөлігі болып табылады, оның түсіндірілуі екі негізгі сипаттамаға байланысты: жад класы және оның түрі. Жад класы сәйкестендірілетін объектімен байланысты жадтың өмір сүру уақыты туралы хабарлайды; түрі объектіде қандай мәндерді анықтайды. Кез келген атаумен өз көріну аймағы (яғни. бұл атау белгілі бағдарлама бөлігі) және бұл атауды басқа файлда бірдей нысан немесе функцияны білдіретін байланыс атрибуты. Көріну аймағы мен байланыс атрибуты А11-де талқыланады.

А 4.1. Жад класы

Екі жад сыныбы бар: автоматты және статикалық. Объектілердің жарнама мәтінмәнімен бірге бірнеше кілт сөздер осы нысандар үшін жад сыныбын көрсетеді.

Автоматты объектілер блокта (А9.3) жергілікті, одан шыққан кезде олар ”жоғалады”. Блок ішінде орнатылған хабарландыру, егер ол жад класс ерекшелігі болмаса немесе auto спецификаторы көрсетілсе, автоматты нысанды жасайды. Хабарламада register сөзімен белгіленген нысан автоматты болып табылады және мүмкіндігінше машина тіркеліміне орналастырылады.

Статикалық объектілер блокта жергілікті болуы немесе блоктан тыс орналасуы мүмкін, бірақ екі жағдайда да олардың мәндері блоктан (немесе функциядан) шыққаннан кейін оған қайта кіргенге дейін сақталады. Блоктың ішінде (соның ішінде функция денесін құрайтын блокта) хабарландырулардағы статикалық объектілер static сөзбен белгіленеді.

Оларды статистикалық кілт сөзді қолдана отырып, құрастырылған блок ішінде локализациялауға болады (бұл жағдайда олар ішкі сілтеме төлсипатын алады) және егер сіз нақты сақтау класын қалдырсаңыз немесе сыртқы кілт сөзін қолдансаңыз (бұл жағдайда олар атрибутты алатын болса), олар бүкіл бағдарлама үшін ғаламдық болады. сыртқы байланыс).

А 4.2. Негізгі түрлері

Бірнеше негізгі түрлері бар. Стандартты тақырып файлы <limits.h >. В қосымшасында сипатталған осы нақты іске асырудағы әрбір түр үшін ең үлкен және ең кіші мәндерді анықтайды. В қосымшасында ең аз ықтимал шамалар келтірілген.

Таңбалар ретінде жарияланатын нысандардың көлемі машинада қабылданған таңбалар жиынтығынан кез келген таңбаны сақтауға мүмкіндік береді. Егер char типті нысан шын мәнінде осы жиын-

тықтан символды сақтаса, онда оның мәні осы символдың коды, яғни кейбір теріс емес бүтін болып табылады. Char типті айнымалылар басқа мәндерді сақтай алады, бірақ олардың мәндерінің диапазоны, әсіресе бұл мәндер немесе беймәлім мәндер іске асырылуға байланысты.

Unsigned char сөздерінің көмегімен жарияланған unsigned char таңбалар әдеттегі таңбалар сияқты, бірақ теріс емес мәндерді білдіреді; signed char сөздерінің көмегімен әдеттегі таңбалар сияқты орын алатын таңбаларды анық жариялауға болады.

Unsigned char түрі тілдің бірінші басылымында
айтылмаған,бірақ бәрі қолданған. Signed char түрі болса, жаңа.

Char-дан басқа тұтас түрлердің арасында үш өлшем болуы мүмкін: short int, int және long int. Қарапайым int типті объектілер осы машинаның архитектурасында қабылданған табиғи өлшемге ие, басқа өлшемдер арнайы қажеттіліктерге арналған. Ұзынырақ, кем дегенде, барлық мәндерді қысқа бүтін қамтиды, бірақ кейбір іске асыруларда қарапайым тұтас қысқа (қысқа) немесе ұзын (ұзын) тұтас эквивалентке тең болуы мүмкін. Int барлық түрлері егер қарсы деп айтылмаса, белгісі бар мәндерді білдіреді.

Жариялауларда unsigned кілт сөзі қолданылады. Мұндай бүтін 2n модулі бойынша арифметикаға бағынады, мұнда N – Сан көрінісіндегі биттер саны, демек, белгісіз бүтін арифметикада ешқашан толып кетпейді. Белгісі бар объектілерде сақталуы мүмкін теріс емес мәндердің көптігі тиісті объектілерде белгісі жоқ сақталуы мүмкін мәндердің жиыны болып табылады; әрбір осындай мәннің таңбалы және белгісіз көрінісі сәйкес келеді.

Өзгермелі нүктелі кез келген екі түр: бір дәлдікпен (float), екі дәлдікпен (double) және жоғары дәлдікпен (long double) синоним болуы мүмкін, бірақ осы тізімнің әрбір келесі түрі кем дегенде алдыңғы дәлдікті қамтамасыз етуі керек.

long double-жаңа түрі. Тілдің алғашқы редакциясында double long float болды, енді соңғысы айналымнан алынды.

Тізімдер – бұл интегралдық мәндері бар ерекше типтер; аталған тұрақтылар жиынтығы әр тізіммен байланысты (А8.4). Тізімдер бүтін сандар сияқты әрекет етеді, бірақ компилятор, әдетте, белгілі бір санау объектісіне оның тұрақтыларының бірінен немесе оның түрінің өрнегінен басқа нәрсе берілген кезде ескерту береді.

Тізімдеу нысандарын сан ретінде қарастыруға болатындықтан, тізімдеу арифметикалық түрге жатады. Барлық өлшемдегі char және int түрлері, олардың әрқайсысы белгімен немесе белгісіз болуы мүмкін, сондай-ақ тізімдер бүтін (integral) түрлер деп аталады. Float, double және long double түрлері өзгермелі нүкте түрлері деп аталады.

Void түрі бос мәндерді анықтайды. Ол “функциямен қайтарылатын мәннің түрі” ретінде пайдаланылады, ол ешқандай нәтиже бермейді.

А 4.3. Туынды түрлері

Базалық түрлерден басқа, бұрыннан бар және келесі дизайндарды сипаттайтын туынды түрлерінің шексіз сыныбы бар:

- берілген үлгідегі объектілердің массивтері;
- берілген түрдегі нысандарды қайтаратын функциялар;
- берілген үлгідегі объектілерге көрсеткіштер;
- объектілердің тізбектілігін қамтитын құрылымдар, әр түрлі типтерден берілуі мүмкін;
- әр түрлі берілген түрдегі бірнеше объектілердің кез келгені болуы мүмкін біріктірулер.

Жалпы жағдайда объектілерді құрастырудың келтірілген әдістері рекурсивті түрде қолданылуы мүмкін.

A 4.4. Үлгілердің біліктілігі

Объектінің түрі біліктілікпен жабдықталуы мүмкін. Const біліктілігі бар объектіні жариялау оның мәні бұдан әрі өзгермейтіндігін көрсетеді; объектіні volatile ретінде жариялай отырып (өзгермелі, тұрақсыз (ағыл.)), біз компилятор оңтайландыру үшін оның ерекше қасиеттерін көрсетеміз. Бір де бір біліктілік мәндер диапазонына және нысандардың арифметикалық қасиеттеріне әсер етпейді. Квалификаторлар A8. 2-де талқыланады.

A 5. Нысандар және Lvalues

Нысан, объект – белгілі бір жад аймағы; lvalue-нысанды білдіретін өрнек. Lvalue-тің айқын мысалы-тиісті түрі мен жад сыныбы бар идентификатор. Lvalue жасайтын операциялар бар. Мысалы, егер E-көрсеткіш түрінің өрнегі болса, онда * E-е-деп көрсететін нысанды білдіретін lvalue үшін өрнек бар. “lvalue” термині сол жақ (left-сол жақ (ағыл.), осыдан L әрпі, value – мән) операнд E1 өрнегі lvalue болуы керек. Әрбір операторды сипаттай отырып, біз lvalue Операндтар ретінде күте ме және lvalue нәтиже ретінде беретіндігін хабарлаймыз.

A 6. Түрлендіру

Кейбір операторлар операндықтарға байланысты олардың мәндерін бір түрден екіншісіне түрлендіруі мүмкін. Бұл параграфта мұндай өзгерістерден не күтуге болады. A6. 5-те көптеген қарапайым операторлар үшін түрлендірулер орындалатын ережелер тұжырымдалады. Әрбір жеке операторды қарау кезінде бұл ережелер нақтылануы мүмкін.

A 6.1. Толық арттыру

Тізбектегі объект тізімдеу, символ, қысқа бүтін, биттік өрісте бүтін – олардың барлығы белгімен немесе онсыз бүтін қолданылуы мүмкін жерде қолданыла алады. Егер int түрі операндтың бастапқы

түрінің барлық мәндерін “қамтуға “ мүмкіндік берсе, онда операнд Int-ге келтіріледі, әйтпесе ол unsigned Int-ге келтіріледі. Бұл рәсім бүтін арттыру деп аталады.

А 6.2. Бүтін түрлендіру

Кез келген тұтастық ең аз теріс емес мәннің конгруэнтті (яғни, екілік көрінісі бар) іздеу және nmax + 1 – ге бөлуден қалдықты алу арқылы кейбір берілген белгісіз түрге келтіріледі, онда nmax – бұл белгісіз түрдегі ең көп сан. Қосымша кодта екілік ұсыну үшін бұл, егер бастапқы типтегі “қазірдің өзінде” бытырасыз тип болса, артық үлкен разрядтарды лақтырып тастауды, не болмаса жетпейтін үлкен разрядтарды нөлдермен (белгісіз мән үшін) немесе белгінің мәнімен (белгімен мән үшін) толтыруды білдіреді, егер қол қойылмаған түрі түпнұсқаға қарағанда «кеңірек» болса.

Кез келген бүтіннің таңбалық түрге келтірілуі нәтижесінде, егер ол осы жаңа түрде көрсетілсе, түрлендірілетін мән өзгермейді, әйтпесе нәтиже іске асырылуға байланысты болады.

А 6.3. Бүтін сандар және қалқымалы нүкте

Өзгермелі нүктелі түрден бүтін бөлшек бөлікке өзгергенде, мәннің алынып тасталынады; егер бұл ретте алынған мәнді берілген бүтін түрде ұсынуға болмайтын болса, онда нәтиже анықталмайды. Атап айтқанда, өзгермелі нүктемен теріс мәндерді бүтіндей бөлшектерге түрлендіру нәтижесі анықталмайды.

Егер мән бүтін саннан өзгермелі нүктелік мәнге түрленсе және ол дұрыс диапазонда болса, бірақ жаңа типте дәл көрсетілмесе, онда нәтиже түпнұсқаға жақын жаңа типтің екі мәнінің бірі болады. Егер нәтиже жарамды мәндер ауқымынан тыс болса, бағдарламаның әрекеті анықталмайды.

А 6.4. Қалқымалы нүктенің түрлері

Төменгі дәлдіктегі өзгермелі нүкте түрінен жоғары дәлдіктегі өзгермелі нүкте түріне ауысу мәнді өзгертпейді. Егер, керісінше, жоғары дәлдіктен төменгі дәлдікке ауысу орындалса және мән жаңа типтің қабылданатын шегінде қалса, онда нәтиже жаңа типтің екі жақын мәндерінің бірі болады. Егер нәтиже жарамды мәндер ауқымынан тыс болса, бағдарламаның әрекеті анықталмайды.

А 6.5. Арифметикалық түрлендіру

Көптеген операцияларда операндтардың түрін түрлендіру және нәтиженің түрін анықтау бірдей ережелерге сәйкес жүзеге асырылады. Олар операндтардың белгілі бір типке, яғни нәтиже түріне айналуы фактісінен тұрады. Бұл ережелер тұрақты арифметикалық түрленулер деп аталады.

▪ Операндтардың кез-келген түрі long double болса, екіншісі long double беріледі.

▪ Олай болмаған жағдайда, операндтардың кез келген түрі double болса, екіншісі double-ге келтіріледі.

▪ Ал егер, кез келген операндалар float түрі болса, онда екіншісі float келтіріледі.

▪ Егер, екі операндылар үшін де бүтін арттыру жүзеге асырылса, содан кейін операндтардың біреуі unsigned long int түрі болса, екіншісі unsigned long int-ға айналады.

▪ Ал, егер, операндтардың біреуі long int түріне, ал екіншісі — unsigned Int түріне тиесілі болса, онда нәтиже long int барлық мәндерін жабатындығына байланысты, егер солай болса, онда unsigned Int long int-ге келтіріледі; егер жабылмаған болса, онда екі операндтар unsigned long int-ге айналады.

▪ Операндтардың біреуі ұзын int түріне ие болса, екіншісі ұзын int.

▪ Операндтардың бірі unsigned int болса, содан кейін екіншісі unsigned int келтіріледі.

▪ Керісінше жағдайда, екі операнда да int түрі бар.

Мұнда екі өзгеріс бар. Біріншіден, өзгермелі нүктесі бар операндалар арифметикасы енді тек екі есе ғана емес, бір дәлдікпен жүргізілуі мүмкін; тілдің бірінші редакциясында өзгермелі нүктесі бар барлық арифметика екі есе дәлдікпен жасалған. Екіншіден, ұзын белгілік типті комбинациядағы қысқа сырғымасыз тип сырғыманың қасиетін нәтиже түріне таратпайды; бірінші редакцияда сырғымасыз тип әрдайым басым болды. Жаңа ережелер сәл қиын, бірақ белгілі және белгісіз шамалардың комбинацияларында күтпеген әсерлердің ықтималдығын азайтады. Бірдей өлшемдегі таңбалы өрнектерді салыстырған кезде күтпеген нәтиже пайда болуы мүмкін.

A 6.6. Көрсеткіштер және бүтін сандар

Көрсеткішке бүтін санды түрдің өрнегін қосуға (және одан алып тастауға) болады; соңғысы бұл жағдайда қосу операторын қарау кезінде A7.7 сипатталған түрлендіруге ұшырайды.

Бір массивке тиесілі бір үлгідегі объектілерге екі көрсеткішке азайту операциясы қолданылуы мүмкін; нәтиже азайту операторын қараған кезде A7.7 сипатталған түрлендіру арқылы мақсатты көрсеткішке келтіріледі.

0 мәні бар немесе ондағы бүтін санның тұрақты өрнегі, бірақ void * типіне көшірілген болса, оны кез-келген түрдегі көрсеткішке меншіктеу және салыстыру операторлары түрлендіре алады. Нәтиже – NULL көрсеткіші, ол кез келген басқа типтегі NULL типіне ұқсас, бірақ нақты объектіге немесе функцияға кез келген көрсеткішке тең емес.

Көрсеткіштер үшін басқа өзгертулерге де рұқсат етіледі, бірақ соған байланысты нәтиженің орындалуына тәуелділігі бар. Бұл түрлендірулер нақты конверсия операторы немесе оператор (A7.5 және A8.8) арқылы көрсетілуі керек.

Меңзерді сақтау үшін жеткілікті үлкен санға түрлендіруге болады; талап етілетін мөлшері іске асыруға байланысты. Айырбастау функциясы да іске асырылуына байланысты.

Бүтін сан түріндегі объект анық көрсеткішке түрлендірілуі мүмкін. Егер бүтін сан көрсеткіштен алынған болса және үлкен болса, бұл түрлендіру сол көрсеткішті береді; әйтпесе, нәтиже іске асыруға байланысты болады.

Бір типтегі меңзерді басқа типке сілтегіш түрлендіруге болады. Егер бастапқы көрсеткіш жадтағы сөздердің шекараларында дұрыс туралanбаған объектіге сілтеме жасаса, онда адресацияның қатесі болуы мүмкін. Егер жаңа типтегі туралау талаптары бастапқы типтегі туралау талаптарына аз немесе сәйкес келсе, онда меңзерді басқа түрге және артқа ауыстыру оны өзгертпейтініне кепілдік беріледі; ортаға дәл келтіру іске асыруға тәуелді, бірақ кез-келген іске асыруда char объектілері теңестірудің минималды талаптарын алады. А6.8-де сипатталғандай, меңзердің мәнін өзгертпестен * босқа * және керісінше түрлендіруге болады.

Сілтегіш осы типтегі объект түріне квалификаторларды (А4.4, А8.2) қосу немесе алып тастау арқылы сол типтегі басқа көрсеткішке айналдыруға болады. Классификаторды қосу арқылы алынған жаңа көрсеткіш бірдей мағынаға ие, бірақ жаңа жіктеушілер енгізген қосымша шектеулермен. Нысаннан жіктеуішті алып тастау операциясы осы объектіні декларацияда көрсетілген оның бастапқы жіктеуіштерінің әсері қалпына келтірілетіндігіне әкеледі.

Соңында, функция сілтегішін басқа типтегі функция сілтегішіне айналдыруға болады. Түрлендірілген сілтегіште функцияны шақыру іске асыруға байланысты; алайда, егер көрсеткіш тағы бір рет өзінің бастапқы түріне айналдырылса, нәтиже түпнұсқа көрсеткішіне қоңырау шалумен бірдей болады.

А 6.7. Void түрі

Мәні (жоқ) объектінің типін void пайдалануға болмайды, сондай-ақ оны анық емес түрде немесе жойылмайтын void түрге жіберілмейді. Void өрнегі ешқандай мағынаны білдірмейді, оны тек ешқандай мән қажет емес жерде пайдалануға болады. Мысалы,

“үтір” (А7.18) операторында нұсқау (А9.2) немесе сол жақ операндтың өрнегі ретінде қолданылады.

Өрнек түрі келтіру операциясымен void түріне әкелуі мүмкін. Мысалы, өрнек рөлінде қолданылатын функцияның шақыруына қатысты-нұсқаулар, void-ке келтіру операциясы функцияның нәтижесі алынып тасталатынын айқын көрсетеді.

Void түрі осы кітаптың алғашқы басылымында көрінбеді, бірақ өткен уақыт ішінде жалпы қолдануға айналды.

А 6.8. Void көрсеткіштері

Нысанның кез келген көрсеткіші ақпаратты жоғалтпай void * түріне әкелуі мүмкін. Егер нәтиже кері түрлендірілсе, онда біз бұрынғы көрсеткішті аламыз. Түрлендірулерден айырмашылығы (А6.6-да қарастырылған), олар түрге келтірудің нақты операторларын талап етеді, меншіктеулер мен салыстыруларда кез келген түрдегі көрсеткіш void * типті нұсқағышпен шыға алады. Сонымен қатар, алдын-ала түрлендірулерсіз әрекет ете алады.

Мұндай интерпретация көрсеткіштері void * - жаңа; бұрын жалпыланған көрсеткіш рөлі char * типті көрсеткішке берілген. ANSI стандарты void * көрсеткіштерді тапсырмалар мен салыстыруларда басқа көрсеткіштермен бірге пайдалануға мүмкіндік береді; басқа көрсеткіштер комбинациясында стандарт нақты түрлендіруді талап етеді.

А 7. Өрнектер

Сипатталған операторлардың басымдықтары осы бөлімдегі элементтермен бірдей (ең жоғарыдан төменге). Мысалы, А7.7-де сипатталған + операторы үшін «операндтар» термині «А7.1-тен А7.6-ге дейін анықталған өрнектерді» білдіреді. Әр тармақта бірдей басымдылыққа ие операторлар сипатталады, және олардың ассоциативтілігі көрсетілген (солға немесе оңға). Барлық операторлардың артықшылығы мен ассоциациясы А13-те берілген грамматикада көрінеді.

Басымдылықтар мен ассоциативтілік толық анықталған, бірақ өрнек бағалану тәртібі кейбір ерекшеліктермен, тіпті жанама әсерлердің әсерінен де анықталмаған. Бұл дегеніміз, егер оператордың анықтамасында оның операндаларын есептеу реті нақты көрсетілмеген болса, онда сіз кез келген есеп айырысу тәртібін еркін таңдап, тіпті оң және сол тәртіпті ауыстыра аласыз. Алайда кез-келген оператор өзінің операндтарының мәндерін ол пайда болатын өрнектің грамматикасына қатаң түрде қолданады.

Бұл ереже математикалық коммутативті және ассоциативті болып табылатын, бірақ есептеулер барысында болмауы мүмкін операцияларды орындау тәртібін таңдауда бұрын берілген бостандықты жояды. Бұл өзгеріс тек «дәлдікте» болатын құбылмалы нүктелер есептеулеріне және толып кету мүмкін жағдайларға ғана әсер етеді.

Тіл толып кетуді, нөлге бөлуді және өрнекті бағалау кезінде пайда болатын басқа ерекшеліктерді анықтамайды. Қолданыстағы С енгізулерінің көпшілігі қол қойылған бүтін сандарды бағалау және тағайындау кезінде толып кетуді елемейді, бірақ мұндай есептеулердің нәтижесі анықталмайды. Бөлуді нөлдік және барлық өзгермелі нүктелер бойынша түсіндіру әртүрлі енгізулерде бірдей болмауы мүмкін; кейде ерекше жағдайларды басқару үшін стандартты емес кітапхана функциясы ұсынылады.

A 7.1. Көрсеткіш генерациясы

Егер өрнектің немесе түрленудің түрі “T массиві” болса, онда бұл өрнектің мәні массивтің бірінші элементіне көрсеткіш болып табылады және мұндай өрнектің түрі “T-ға көрсеткіш”түріне ауыстырылады. Егер өрнек унарлы оператордың операндасы&, немесе операция операндасы++,..., sizeof немесе сол жақ тағайындау операндасы немесе оператордың операндасы болса, мұндай түрді ауыстыру жасалмайды . (нүкте). Сол сияқты, “т қайтаратын функция” түрі, ол үшін операндтан басқа,”т қайтаратын функцияға көрсеткіш” түріне айналады.

А 7.2. Бастапқы өрнектер

Бастапқы өрнектер – бұл идентификаторлар, тұрақтылар, жолдар және жақшадағы өрнектер.

```
бастапқы-өрнек:
        идентификаторлар
        тұрақтылар
        жол
        (өрнек)
```

Идентификатор, егер ол дұрыс жарияланса (бұл қалай жасалатыны туралы төменде айтылса), — бастапқы өрнек. Идентификатордың түрі хабарландыруда эго-ға тән. Идентификатор арифметикалық түрдегі нысанды (А5) немесе құрылым, біріктіру немесе көрсеткіш түріндегі нысанды білдіретін болса, lvalue болып табылады.

Тұрақты-бастапқы өрнек. Оның түрі А2.5-те қаралған жазба түріне байланысты.

Жол литерал – бастапқы өрнек. Бастапқыда оның түрі - “ char массиві “(кеңейтілген терімнің таңбалар жолы үшін wchar_t массиві), бірақ А7.1-де келтірілген ережеге сәйкес, аталған түр әдетте” charға көрсеткіш “(“wchar_t-ке көрсеткіш”)” жолдың бірінші символына көрсеткіш “деген нәтижелік мәнге айналады. Кейбір инициализаторлар үшін бұл түрді ауыстыру жасалмайды. (А8.7 тақырыбында.)

Жақшадағы өрнек – түрі мен мәні жақшасыз осы өрнектің түрі мен мәніне ұқсас бастапқы өрнек. Жақшаның болуы немесе болмауы бұл lvalue өрнегі бар немесе жоқ екеніне әсер етпейді.

А 7.3. Постфиксті өрнектер

Постфиксті өрнектерде операторлар солдан оңға қарай орындалады.

```
Постфиксті өрнектер:
        бастапқы-өрнек
```

```
        постфиксті өрнек [ өрнек ]
        постфиксті өрнек ( тізім-дәлелдер-өрнектер.)
        постфиксті өрнек . идентификаторлар
        постфиксті өрнек -> идентификаторлар
        постфиксті өрнек ++
        постфиксті өрнек –
аргументтер тізімі:
        меншіктеу өрнегі
        аргументтер тізімі, меншіктеу өрнегі
```

А 7.3.1. Жиым элементтеріне үндеу

Төртбұрышты жақшада өрнек болатын постфиксті өрнек индекстелетін массивке үндеу білдіретін постфиксті өрнек бар. Осы Екі өрнектің бірі “Т – ға көрсеткіш” түріне жатуы тиіс, мұнда т – кейбір тип, ал екіншісі-бүтін тип; индекстеу нәтижесінің түрі бар, яғни бірдей өрнегі бойынша Е1[Е2] өрнегі бар *(Е1)+(Е2)). Толығырақ А8.6.2 қараңыз.

А 7.3.2. Функцияны шақыру

Функцияны шақыру постфиксті сөз тіркесі бар (ол функцияның атауы деп аталады – function designator), одан кейін осы функцияның дәлелдері болып табылатын (А7.17) үтірмен бөлінген өрнектердің тізімі бар жақшалар. Егер постфиксті өрнек-ағымдағы көріну аймағында жарияланбаған идентификатор болса, онда бұл идентификатор хабарландырумен анық сипатталғандай болып саналады.

```
extern int идентификаторлар();
```

сәйкес блокқа қоңырау салынған ішкі блокқа орналастырылған. Постфикс өрнегі (мүмкін, анықтағыштың сипаттамасы мен буыны, А7.1 қараңыз) 1) «Т функциясын қайтаратын функцияға сілтегіш» түріне ие болуы керек, мұндағы Т - қайтару мәні.

Тілдің бірінші нұсқасында функцияны атау үшін тек” функция “ түріне рұқсат етілген және функцияны көрсеткіш арқылы шақыру үшін айқын оператор *қажет болады. ANSI - стандарт функцияға және көрсеткішпен ерекшеленетін функцияға жүгіну үшін бірдей синтаксиске рұқсат беретін

кейбір компиляторлардың тәжірибесін қолдайды. Ескі синтаксисті қолдану
мүмкіндігі қалады.

Аргумент термині функцияны анықтау немесе жариялау кезінде алатын объектіні (немесе оның идентификаторын) белгілеу үшін қолданылады. Осы ұғымдардың орнына кейде мағыналық айырмашылықтары бар "нақты дәлел (параметр)" және "формальды дәлел (параметр)" терминдері кездеседі.

Функцияны шақырғанда оның әрбір аргументі көшіріледі; аргументтерді беру олардың мәндері арқылы жүзеге асырылады. Функциялар өз параметрлерінің мәндерін өзгертуге рұқсат етіледі, олар тек дәлелдердің көшірмелері болып табылады, бірақ бұл өзгерістер дәлелдердің мәндеріне әсер ете алмайды. Алайда, болады осы көрсеткіш көрсететін нысанның мәнін өзгерту мүмкіндігін беру үшін көрсеткішті жіберу.

Функцияны жариялаудың екі жолы бар. Жаңа тәсілде параметрлер түрлері анық белгіленеді және функция түрінің бөлігі болып табылады; мұндай хабарландыру функция прототипі деп аталады. Ескі әдіс кезінде параметрлер түрлері көрсетілмейді. Әдістері функцияларын талқыланады А8.6.3 және А10.1.

Егер қоңырау ескі жолмен жазылған мәлімдеменің аясында болса, онда оның әр дәлелі алға тартылады: бүтін сандар жоғарылату бүтін санға арналған (А6.1), ал қалқымалы дәлелдер үшін олар екі есеге өзгертіледі. Егер аргументтер саны функцияны анықтаудағы параметрлер санына сәйкес келмесе немесе алға қойылғаннан кейін дәлелдердің түрлері сәйкес параметрлердің түрлеріне сәйкес келмесе, қоңырау нәтижесі анықталмайды. Түрдің сәйкестігі критерийі функцияның қалай анықталатынына байланысты (ескі немесе жаңа). Ескі әдіс қоңыраудағы дәлелдің ұсынылған түрін және сәйкес параметрдің ұсынылған түрін салыстырады; жаңа әдіспен, ұсынылған аргумент түрі мен параметр түрі (жылжытусыз) бірдей болуы керек.

Егер қоңырау жаңадан жазылған мәлімдеме шеңберіне кірсе, дәлелдер тиісті прототиптік параметрлердің типтері бар айнымалыларға тағайындалғандай түрлендіріледі. Параметрлер тізімі эллипспен (, ...) аяқталмаса, дәлелдер саны нақты сипатталған параметрлердің санына сәйкес келуі керек. Әйтпесе, дәлелдер саны параметрлер санынан үлкен немесе оған тең болуы керек; эллипс астындағы «жасырылған» аргументтер түрін жылжытуға болады (алдыңғы абзацта сипатталғандай). Егер функцияның анықтамасы ескі болса, онда прототиптегі қоңырауларда анық көрінетін параметрлердің типтері олар алға жылжытылғаннан кейін функция анықтамасындағы параметрлер түрлеріне сәйкес келуі керек.

Бұл ережелер әсіресе күрделі, өйткені олар анықталатын функциялардың аралас тәсілімен (ескі және жаңа) қызмет етуге арналған. Мүмкіндігінше оны болдырмау керек.

Аргументтерді есептеу кезегі анықталмайды, әр түрлі компиляторларда ол әртүрлі. Дегенмен, дәлелдер мен функцияның өрнегі оған кіргенге дейін толық (соның ішінде жанама әсерлер) есептелетініне кепілдік беріледі. Кез келген функция рекурсивті айналымға жол береді.

A 7.3.3. Құрылымдарға үндеу

Кейінгі идентификаторы бар нүкте тұратын постфиксті өрнек постфиксті өрнек болып табылады. Бірінші операндтың өрнегі құрылым немесе бірлестік, ал идентификатор – құрылым немесе бірлестік элементінің атауы болуы керек. Мән – құрылымның немесе бірлестіктің атауы, ал мәннің түрі – құрылымның немесе бірлестіктің элементінің түрі. Өрнек lvalue, егер бірінші өрнек lvalue болса және екінші өрнектің түрі “массив” болмаса.

Кейін көрсеткі тұратын постфиксті өрнек (таңбадан жасалған - және >), кейіннен идентификатормен постфиксті өрнек болып табылады. Бірінші операндтың өрнегі құрылымның (бірлестік) көрсеткіші, ал идентификатор – құрылымның (бірлестік) элементінің аты болуы керек. Нәтиже – көрсеткіш, ал мән түрі –

құрылым (біріктіру) элементінің түрі көрсететін құрылым (біріктіру) деп аталатын элемент; нәтиже-lvalue, егер тип “массив” болмаса.

Осылайша, E1->MOS өрнегі (*E1) өрнегі сияқты. MOS. Құрылымдар мен бірлестіктер A8. 3 қарастырылады.

Кітаптың бірінші басылымында элементтің аты постфиксті түрде айтылған құрылымға немесе бірлестікке тиесілі болатын ереже келтірілген. Алайда, ол қатаң міндетті емес деп айтылды. Соңғы компиляторлар мен ANSI оны міндетті етеді.

A 7.3.4. Инкремент және декементтің постфиксті операторлары

Постфиксті білдіретін өрнек ++ немесе - постфикс өрнегі. Мұндай өрнектің мәні оның операндының мәні болып табылады. Мәнді қабылдағаннан кейін, операнд 1-ге көбейтіледі (++) немесе кішірейтіледі (-); операндтық шектеулер мен операциялардың егжей-тегжейлері туралы ақпарат алу үшін қосымша операторларды талқылау үшін A7.7 және тапсырма үшін A7.17 қараңыз. Көбею немесе азайту нәтижесі бағаланбайды.

A 7.4. Унарлы операторлар

Бірыңғай операторлармен өрнектер оңнан солға қарай орындалады

Унарлы операторлар:

```
постфиксті-өрнек
++ унарлы өрнек
 --  унарлы өрнек
унарлы-оператор өрнек типке келтірілген
sizeof унарлы өрнек
sizeof ( аты-типі )
```

біреуі унарлы оператор:
& * + ~ !

А 7.4.1. Инкремент және декемент префиксті операторлары

Бірыңғай өрнек, оның алдында ++ немесе..., бірыңғай өрнек бар. Операнд ұлғаяды (++) немесе 1-ге азаяды (--).

Өрнектің мәні — оның операндтың ұлғаюынан (азаюынан) кейінгі мәні. Операнд әрқашан lvalue болуы тиіс; операндқа шектеулер туралы және операцияның егжей-тегжейі туралы ақпарат А7.7, онда аддитивті операторлар талқыланады, онда және беру қаралатын А7.17 болады. Инкрементациялау және декрементациялау нәтижесінде lvalue жоқ.

А 7.4.2. Мекенжай алу операторы

Унарлы оператор & операндтың мекенжайын алу операциясын білдіреді. Операнд lvalue, не бит өрісіне, register деп жарияланған нысанға немесе «функция»түріне сілтеме жасамауы керек. Нәтиже – осы lvalue мекен-жайы бар нысанның (немесе функцияның) көрсеткіші. Егер операндтың түрі Т болса, онда нәтиженің түрі «т көрсеткіш» болып табылады.

А 7.4.3. Жанама қол жеткізу операторы

Унарлы оператор * объектіні (немесе функцияны) қайтаратын жанама қол жеткізу операциясын (көрсеткішті ашу) білдіреді, оған операнды көрсетеді. Нәтиже lvalue болып табылады, егер операнд – арифметикалық нысанға немесе «құрылым», «біріктіру» немесе «Көрсеткіш»түріндегі нысанға көрсеткіш болса. Егер өрнек түрі - «т-ға көрсеткіш» болса, онда нәтиже түрі-Т.

А 7.4.4. Унарлы қосу операторы

Унарлы + операндының арифметикалық түрі болуы керек, нәтижесі – операндтың мәні. Бүтін санды операнд бүтін сандармен

өтеді. Нәтиже түрі – операндтың жоғары түрі. Унарлы + - унарлы симметрия үшін қосылды.

А 7.4.5. Унарлы минус операторы

Унарлы операндтың арифметикалық түрі болуы керек, нәтижесі – қарама-қарсы белгісі бар операндтың мәні. Бүтін санды операнд бүтін сандармен өтеді. Сырғымасыз шаманың теріс мәні жоғары операндтың түріне келтірілген nmax+1 – ден шегерумен есептеледі, мұнда NMAX-жоғары типті ең жоғары сан; алайда минус нөл бар. Нәтиже түрі операндтың жоғары түрі болады.

А 7.4.6. Бинарлық терістеу операторы

~ операторының операндында бүтін сан болуы керек, нәтижесі операндты барлық сандармен толықтырады. Операнд түрін бүтін түрде жылжыту жүзеге асырылады. Егер операнд қол қойылмаған болса, нәтиже оның ең үлкен арттырылған санынан оның мәнін алу арқылы алынады. Егер операндқа қол қойылса, онда нәтиже «алға қойылған операндты» қол қойылмаған түрге аударып, ~ әрекетін орындап, қол қойылған түрге айналдыру арқылы есептеледі. Нәтиже түрі – операндтың ұсынылған түрі.

А 7.4.7. Логикалық терістеу операторы

Операнд операторы ! арифметикалық типті немесе сілтегіш болуы керек. Егер операндты 0-мен салыстыру дұрыс болса, нәтиже 1-ге тең, ал басқасы 0-ге тең болады. Нәтиже түрі - int.

А 7.4.8. Көлем өлшемі операторы

Sizeof операторы оның операндты бар нысанды сақтау үшін қажетті байттар санын береді. Операнд-не өрнек (есептелмейтін) немесе жақшада жазылған түрдің атауы. Char-ға қолданылған sizeof операторы 1 береді. Алап үшін нәтиже массивте байттардың жалпы санына, құрылымға немесе біріктіруге – объектідегі байттар саны-

на, сонымен қатар, егер элементтерден алап құрастырылса, қажет болатын байттар-толтырғыштарды қоса алғанда, тең болады. N элементтердің массивінің өлшемі әрқашан оның жеке элементінің өлшеміне көбейтіледі. Бұл операторды «функция» типті операндқа, аяқталмаған типке және биттік өріске қолдануға болмайды. Нәтижесі-unnopless бүтін константа; оның нақты түрі іске асыруға байланысты. Стандартты тақырып файлында <stddef.h> (В қолданбасын қараңыз) бұл түрі size_t деп аталады.

А 7.5. Типті келтіру операторы

Жақшадағы унарлы өрнектің алдында жазылған түрдің атауы осы өрнектің мәнін көрсетілген түрге келтіруге әкеледі.

өрнекке-келтірілген-түрі:
    унарлы өрнек
    (аты-типі) өрнекке-келтірілген-түрі:

Бұл дизайн келтірілген деп аталады. Типтердің атаулары А8.8 толығырақ жазылған. Қайта құру нәтижесі А6-да сипатталған. Үлгі өрнегі lvalue емес.

А 7.6. Мультипликативтік операторлар

Мультипликативті операторлар *, / және % солдан оңға қарай орындалады.

мультипликативті-өрнек:
    өрнекке-келтірілген-түрі
    мультипликативті-өрнек * өрнекке-келтірілген-түрі
    мультипликативті-өрнек / өрнекке-келтірілген-түрі
    мультипликативті-өрнек % өрнекке-келтірілген-түрі

Операторлар операндылары * және /арифметикалық түрдегі, оператор % — бүтін түрдегі болуы тиіс. Операндалар үстінде олардың мәндерін нәтиже түріне әкелетін қарапайым арифметикалық түрлендірулер жүзеге асырылады.

Бинарлық оператор * көбейтуді білдіреді .

Бинарлы оператор/жеке алады, ал % - бірінші операндты екіншіге бөлуден қалған қалдық; егер екінші операнд 0 болса, онда нәтиже анықталмады. Егер екі операнда да теріс болмаса, онда қалдық теріс емес және бөлгіштен аз; ал олай болмаған жағдайда стандарт тек бір ғана кепілдік береді: қалдық абсолюттік мәні бөлгіштің абсолюттік мәнінен аз.

А 7.7. Аддитивті операторлар

Аддитивті операторлар + және - солдан оңға қарай орындалады. Егер операндалардың арифметикалық түрі болса, онда әдеттегі арифметикалық түрлендірулер жүзеге асырылады. Әрбір оператор үшін бірнеше қосымша комбинациялар бар.

аддитивті-өрнек:

мультипликативті-өрнек
аддитивті-өрнек + мультипликативті-өрнек
аддитивті-өрнек-мультипликативті-өрнек

Оператордың орындалу нәтижесі + оның операндтарының сомасы бар. Объект көрсеткіш массивте бүтін мәнмен жинақтауға болады. Бұл ретте соңғысы оны көрсеткіш сілтеме жасалатын объектінің өлшеміне көбейту арқылы адрестік ығысуға айналады. Бұл көрсеткіш сол түрдегі нысанның көрсеткіші болып табылады; тек осы көрсеткіш сол массивтің басқа объектісіне сілтеме жасайды, ол бастапқы есептелінген ығысудан артта қалады. Мысалы, егер р – массивтегі объектіге көрсеткіш болса, онда Р+1 – сол массивтегі келесі объектіге көрсеткіш. Егер жиынтықтау нәтижесінде алынған көрсеткіш массивтің шекарасынан тыс көрсетілсе, онда ол массивтің соңында тікелей тұрған орынды көрсеткен жағдайдан басқа, нәтиже белгісіз болады.

Көрсеткіш үшін массивтің соңында бірден орналасқан элементті көрсету мүмкіндігі жаңа болып табылады. Осылайша, массив элементтерінің циклдік іріктеуін ұйымдастырудың жалпы қабылданған тәжірибесі заңдастырылды.

Оператордың орындалу нәтижесі – (минус) операндтардың айырмашылығы бар. Көрсеткіштен кез келген бүтін санды түрдің мәнін сол түрлендірулермен және қосудағы жағдайлармен алып тастауға болады.

Егер бір типті объектілерге екі көрсеткішке шегеру операторын қолданса, онда нәтижесінде осы көрсеткіштер көрсететін объектілер арасындағы қашықтықты білдіретін белгісі бар бүтін мәнді алады; келесі объектіге көрсеткіш алдыңғы объектіге көрсеткіштен 1 артық. Нәтиже түрі іске асырылуға байланысты; стандартты тақырып файлында <stddef.h> ол ptrdiff_t деп аталады. егер көрсеткіштер бір массивтің емес нысандарын көрсетсе, мән анықталмады; бірақ егер P массивтің соңғы элементін көрсетсе, онда P+1-P 1-ге тең мәнге ие.

A 7.8. Жылжыту операторлары

Жылжу операторлары < және > солдан оңға қарай орындалады. Екі оператор үшін әрбір операндтың бүтін түрі болуы керек және олардың әрқайсысы бүтін өсуге ұшырайды. Нәтиже түрі сол жақ операндтың жоғары түріне сәйкес келеді. Егер оң операнд теріс болса немесе оның мәні сол жақ өрнектегі биттер санынан асып кетсе немесе оған тең болса, нәтиже анықталмады.

Жылжыту өрнектері:

```
жылжыту өрнектері
жылжыту өрнектері >> аддитивті-өрнек
жылжыту өрнектері << аддитивті-өрнек
```

E1<<E2 мәні E2 битке солға жылжыған E1 мәніне тең (биттер тізбегі ретінде қарастырылатын); толып кету болмаған кезде мұндай операция 2E2-ге көбейтуге тең. E1>>E2 мәні E2 биттік позицияға оңға жылжыған E1 мәніне тең. Егер E1 — белгісіз немесе теріс емес мәнге ие болса, онда оң өзгеріс 2E2-ге бөлуге баламалы, олай болмаған жағдайда нәтиже іске асыруға байланысты.

А 7.9. Қатынас операторлары

Қатынас операторлары солдан оңға қарай орындалады, алайда бұл қасиет пайдалы болуы мүмкін; тілдің грамматикасына сәйкес a<b<c өрнегі (a<b)<c сияқты түсіндіріледі, ал a<b есептеу нәтижесі тек 0 немесе 1 болуы мүмкін.

Қатынас өрнектері:

жылжымалы өрнек
қатынас өрнектері < жылжымалы өрнек
қатынас өрнектері > жылжымалы өрнек
қатынас өрнектері <= жылжымалы өрнек
қатынас өрнектері >= жылжымалы өрнек

Операторлар: < (аз), > (көп), < = (аз немесе тең) және > = (көп немесе тең) - егер спецификирленетін қатынас жалған болса, барлығы 0, және егер ол шын болса, 1 береді. Нәтиже түрі-int. Арифметикалық операндалар үстінде қарапайым арифметикалық түрлендірулер орындалады. Бір түрдегі нысандардың көрсеткіштерін салыстыруға болады (біліктіліктерді есепке алмағанда); нәтиже жадтағы салыстырмалы орналасуына байланысты болады. Алайда, көрсеткіштерді бір объектінің әртүрлі бөліктеріне салыстыруға жол беріледі: егер екі көрсеткіш бір қарапайым объектіні көрсетсе, онда олар бірдей; егер олар бір құрылымның элементтерін көрсетсе, онда құрылымда неғұрлым кеш хабарландыруы бар элементке көрсеткіш көбірек; егер көрсеткіштер бір біріктірудің элементтерін көрсетсе, онда олар тең; егер көрсеткіштер кейбір массивтің элементтерін көрсетсе, онда осы көрсеткіштерді салыстыру олардың индекстерімен салыстырғанда эквивалентті. Егер Р массивтің соңғы элементін көрсетсе, онда Р+1 р-дан артық, бірақ Р+1 массивтің шекарасынан тыс көрсетеді. Басқа жағдайларда салыстыру нәтижесі анықталмаған.

Бұл ережелер тілдің бірінші редакциясында белгіленген шектеулерді әлсіретді.

Олар құрылым мен бірлестіктің әр түрлі элементтеріне көрсеткіштерді салыстыруға мүмкіндік береді және массивтің соңында тікелей орналасқан орынға көрсеткішпен салыстыруды заңдастырады.

А 7.10. Теңдік операторлары

Теңдік өрнектері:
    теңдік өрнектері
    теңдік өрнектері == теңдік қатынастары
    теңдік өрнектері != теңдік қатынастары

Операторлар = = (тең) және != (тең емес) төмен басымдыққа ие айырмашылықпен қарым-қатынас операторларына ұқсас. (Осылайша, a<b = c<d 1 сонда ғана a<b және c<d қатынасы шынайы немесе екеуі де жалған болған кезде ғана бар).

Теңдік операторлары қатынас операторлары сияқты ережелерге бағынады. Сонымен қатар, олар көрсеткішті бүтін тұрақты өрнегімен салыстыруға мүмкіндік береді, оның мәні нөлге тең және void көрсеткішімен (А6.6. тақырыбын қараңыз).

А 7.11. Бинарлы ЖӘНЕ опеаторы

ЖӘНЕ өрнегі:
    теңдік өрнегі
    ЖӘНЕ өрнегі & теңдік өрнегі

Кәдімгі арифметикалық түрлендірулер орындалады; нәтиже операндалардың биттік және қисынды нәтижелері. Оператор тек бүтін санды операндтарға қолданылады.

А 7.12. Бинарлы НЕМЕСЕ ерекше операторы

Бинарлы НЕМЕСЕ ерекше өрнегі:
    ЖӘНЕ өрнегі
    бинарлы НЕМЕСЕ ерекше өрнегі ^ ЖӘНЕ өрнегі

Кәдімгі арифметикалық түрлендірулер орындалады; нәтиже - операндтардың бинарлы НЕМЕСЕ ерекше операторы. Оператор тек бүтін санды операндтарға қолданылады.

А 7.13. Бинарлы НЕМЕСЕ операторы

НЕМЕСЕ өрнегі:
бинарлы НЕМЕСЕ ерекше өрнегі
НЕМЕСЕ өрнегі | бинарлы НЕМЕСЕ ерекше өрнегі

Кәдімгі арифметикалық түрлендірулер орындалады; нәтиже - операндтардың биттік немесе НЕМЕСЕ. Оператор тек бүтін санды операндтарға қолданылады.

А 7.14. Логикалық ЖӘНЕ операторы

Логикалық ЖӘНЕ өрнегі:
НЕМЕСЕ өрнегі
логикалық ЖӘНЕ өрнегі && НЕМЕСЕ өрнегі

&& операторлары солдан оңға қарай орындалады. && операторы егер 1 операнд нөлге тең болмаса, 0-ді қайтарады. &, & & Айырмашылығы && есептеулердің солдан оңға қарай жүргізілуіне кепілдік береді: бірінші операнд жанама әсерлерімен бағаланады; егер ол 0 болса, онда өрнектің мәні 0-ге тең болады. Әйтпесе, дұрыс операнд есептеледі, ал 0-ге тең болса, онда өрнектің мәні 0-ге тең, әйтпесе ол 1-ге тең болады.

Операндалар әр түрлі болуы мүмкін, бірақ олардың әрқайсысы арифметикалық немесе сілтегіш түрінде болуы керек. Нәтиже түрі - int.

А 7.15. Логикалық НЕМЕСЕ операторы

Логикалық НЕМЕСЕ өрнегі:
логикалық ЖӘНЕ өрнегі
логикалық НЕМЕСЕ өрнегі || логикалық ЖӘНЕ өрнегі

Операторлар || солдан оңға қарай орындалады. Оператор || егер операндалардың кем дегенде біреуі нөлдік емес болса, 1 шығарады, әйтпесе 0. | Айырмашылығы, || есептеулердің солдан оңға қа-

рай жүргізілуін қамтамасыз етеді: бірінші операнд бағаланады, оның ішінде барлық жанама әсерлер; егер ол 0 болмаса, онда өрнектің мәні 1-ге тең. Әйтпесе, дұрыс операнд есептеледі, ал егер ол 0 болмаса, онда өрнектің мәні 1-ге тең, әйтпесе ол 0-ге тең болады.

Операндалар әр түрлі болуы мүмкін, бірақ операнд арифметикалық немесе сілтегіш түрінде болуы керек. Нәтиже түрі - int.

А 7.16. Шартты оператор

Шартты өрнек:

логикалық НЕМЕСЕ өрнегі
логикалық НЕМЕСЕ өрнегі ? өрнек : шартты өрнек

Барлық жанама әсерлерді қоса алғанда, бірінші өрнек есептеледі; егер ол 0-ге тең болмаса, нәтиже екінші өрнектің мәні, әйтпесе үшінші өрнектің мәні. Соңғы екі операндтың біреуі ғана есептеледі: екінші немесе үшінші. Егер екінші және үшінші операндалар арифметикалық болса, онда нәтиженің түрі болатын кейбір жалпы түрге әкелетін қарапайым арифметикалық түрлендірулер орындалады. Егер екі операндта void түрі болса, немесе сол түрдегі құрылымдар немесе бірлестіктер болса, немесе сол түрдегі объектілерге көрсеткіштер болса, онда нәтиже операндалар сияқты бірдей түрге ие болады. Егер операндтардың біреуінде «көрсеткіш» түрі болса, екіншісі 0 тұрақты болса, онда 0 «көрсеткіш» түріне келтіріледі, сол түрдің нәтижесі болады. Егер бір операнд void көрсеткіші болса, екіншісі – басқа түрдегі көрсеткіш болса, онда соңғысы void көрсеткішіне айналады, ол нәтиженің түрі болады.

Көрсеткіштер түрлерін салыстыру кезінде көрсеткіштер сілтейтін объектілер типтерінің (а 8.2) квалификаторлар көрсеткіштері назарға алынбайды, бірақ нәтиже түрін шартты өрнектің екі тарауының да квалификаторлары иеленеді.

А 7.17. Меншіктеу өрнегі

Бірнеше меншіктеу операторлары бар; олар оңнан солға қарай орындалады.

Меншіктеу өрнегі:
шартты өрнек
унарлы өрнек операторы-меншіктеу өрнегі-меншіктеу
меншіктеу операторы осылардың біреуі:
/= %= += -= <<= >>= &= ^= |=

Сол жақ операнд ретінде меншіктеу операторлары lvalue-ті талап етеді, бұл дегеніміз, ол массив бола алмайды немесе аяқталмаған түрге ие болмайды немесе функция болуы мүмкін. Сол жақ операндтың түрі, бұдан басқа, const біліктілігі болмайды; және егер ол құрылым немесе бірлестік болып табылса, онда const біліктілігі бар элементтер немесе кіші элементтер (салынған құрылымдар немесе бірлестіктер үшін) болмауы тиіс.

Меншіктеу өрнегінің түрі оның сол жақ операндының түріне сәйкес келеді және оның сол жақ операндының мәні меншіктеу аяқталғаннан кейін сол жақ операндының мәніне тең.

Оператормен қарапайым меншіктеу = өрнектің мәні lvalue қолданылатын нысанды алмастырады. Бұл ретте мынадай шарттардың бірі орындалуы тиіс: екі операндтың да арифметикалық түрі болады (егер операндтардың типтері әртүрлі болса, оң жақ операндтың сол жақ операндының типіне келтірілсе); екі операндтың да сол бір типтегі құрылымы немесе біріктірілуі бар; бір операндтың көрсеткіші бар, ал екіншісі – void – ке көрсеткіш; сол жақ операнд – көрсеткіш, ал оң жақ операнд-0 мәні бар константалық өрнек.; екі операнда да бірдей түрі бар функцияларға немесе объектілерге көрсеткіштер (оң операнда const немесе volatile болмау мүмкіндігін қоспағанда).

E1 ор = E2 өрнегі бір ерекшелік бар E1 = E1(E2) өрнегіне тең: E1 тек бір рет есептеледі.

А 7.18. Үтір операторы

өрнек:
    меншіктеу өрнегі
    өрнек, меншіктеу өрнегі

Үтірмен бөлінген екі өрнек солдан оңға қарай есептеледі және сол жақ өрнектің мәні алынып тасталады. Нәтиженің түрі мен мәні оң операндтың түрі мен мәніне сәйкес келеді. Сол жақ операндтың барлық жанама әсерлерін есептеу оң жақ операндты есептеу алдында аяқталады. Мысалы, функция аргументтерінің тізімдерінде (А7.3.2) немесе бастамашылардың тізімдерінде (А8.7) (синтаксистік бірліктер ретінде меншіктеу өрнектері пайда болады), мысалы, үтір операторы тек топталған жақшаларда пайда болуы мүмкін. Мысалы,

```
f(a, (t=3, t+2), c)
```

үш дәлел, оның ішінде екіншісінің 5 мәні бар.

А 7.19. Тұрақты өрнектер

Синтаксистік, тұрақты өрнек-операторлардың шектеулі ішкі жиыны бар өрнек:

Тұрақты өрнек:
    шартты өрнек

Case-белгіні ауыстырып қосқышта көрсеткенде, массивтердің шекарасын және бит өрістерінің ұзындығын көрсеткенде, аталған константтар мен инициализаторлар мәндерінің орнында, сондай-ақ, кейбір өрнектерде препроцессор үшін өрнектер қажет, оларды есептеу тұрақтыға әкеледі.

Тұрақты өрнектерде меншіктеу, инкрементациялау және декрементациялау операторлары, функциялар мен операторлардың-үтірлі шақырулары болмайды; аталған шектеулер sizeof операторының

операндына қолданылмайды. Егер бүтін тұрақты өрнек алу қажет болса, онда оның операндалары бүтін, тізбектелген (enum), символдық тұрақты және өзгермелі нүктелі тұрақты тұрады; келтіру операциялары бүтін санды түрге, ал өзгермелі нүктелі кез келген тұрақты – тұтас сипатқа келтірілуі тиіс. Осыдан тұрақты түрде массивтер, жанама айналым операциялары (көрсеткішті ашу), мекен-жайды алу және құрылым өрістеріне қол жеткізу мүмкін емес. (Алайда, sizeof үшін кез келген түрдегі операндалар мүмкін.)

Тұрақты өрнектер үшін инициализаторларда үлкен еркіндікке рұқсат етіледі; операндалар кез келген түрдегі константтар болуы мүмкін, ал сыртқы немесе статикалық объектілер мен тұрақты өрнектермен индекстелетін сыртқы және статикалық массивтерге бірыңғай оператор & қолданылуы мүмкін. Унарлы оператор & сондай-ақ дәлелдер тізімі жоқ индекссіз немесе функциясыз массив пайдалану кезінде болмауы мүмкін. Инициализаторды есептеу бұрын жарияланған сыртқы немесе статикалық объектінің плюс-минус константасына немесе мекен-жайын беруі тиіс.

Аз еркіндік #if кейін қолданылатын бүтін тұрақты өрнектер үшін рұқсат етіледі: sizeof-өрнектер, enum түр тұрақтылары және типті келтіру операциялары рұқсат етілмейді. (12.5 қараңыз).

А 8. Хабарландыру

Әрбір идентификатор қалай түсіндірілетіндігі хабарландырумен сипатталады; олар әрдайым олар сипатталатын идентификаторлар үшін жадты сақтамайды. Жадты сақтайтын хабарландырулар анықтамалар деп аталады және келесі түрге ие:

хабарландыру:
    спецификаторлар - тізімі - бастамашы-хабарландырушылар

Хабарландырушылар инициализатор-хабарландырушылардың тізімінде хабарланатын сәйкестендіргіштер болады; спецификатор-хабарландырулар түрі мен жады класының спецификаторларынан тұратын реттілік болып табылады.

спецификаторлар-хабарландырулар:
спецификатор-сынып-жад спецификаторлар-хабарландырулар
спецификатор-типті спецификаторлар-хабарландырулар
спецификаторлар - хабарландырулар

бастамашы-хабарландырушылардың тізімі:
бастамашы-хабарлаушы
бастамашы- тізімі, бастамашы-хабарландырушы

бастамашы-хабарлаушы:
хабарлаушы
хабарлаушы = бастамашы

Хабарландырушылар хабарландыруға жататын аттарды қамтиды. Біз оларды кейінірек 8.5-те қарастырамыз Хабарландыруда кем дегенде бір хабарландырушы, не оның үлгі спецификаторы құрылымның немесе бірлестіктің тегін анықтауы, не санамалау элементтерін беруі тиіс; бос хабарландыруға жол берілмейді.

А 8.1. Жад класс спецификаторлары

Жад класы келесідей ерекшеленеді:

жад-класс спецификаторы:
auto
register
static
extern
typedef

Жад кластарының мағынасы А4 тақырыбында қозғалған болатын.

Auto және register спецификаторлары хабарланатын нысанмен Автоматты жад классын береді және бұл спецификаторларды тек функцияның ішінде қолдануға болады. Auto және register хабарландырулары бір уақытта жадты анықтау және сақтау болып табылады. Спецификатор register баламасы auto, бірақ құрамында ақпарат жазылып тұрады және бағдарламасында жарияланған объектілер пайдаланылады. Тіркелімдерде объектілердің аз ғана саны ғана орнала-

стырылуы мүмкін, бұл ретте белгілі бір түрдегі; көрсетілген шектеулер іске асыруға байланысты. Кез келген жағдайда register-объектіге унарлы оператор & қолдануға болмайды.

register класындағы нысанның мекенжайын есептеуге болмайды, ал auto класын есептеуге болады.

Static спецификаторы жарияланған объектілерге статикалық жады класы береді, ол ішкі және функциялардан тыс пайдаланылуы мүмкін. Функцияның ішінде бұл спецификатор жадты ерекшелеуді тудырады және анықтау болып табылады; оның функциялардан тыс рөлі А11.2-де түсіндіріледі.

Функцияның ішінде пайдаланылатын extern спецификаторымен жарияланған, хабарланатын объект үшін бір жерде жады бөлінгенін хабарлайды; оның функциялардан тыс рөлі туралы А11.2-де айтылатын болады.

Typedef спецификаторы ешқандай жадты резервтемейді және синтаксис стандарты тұрғысынан жад класс спецификаторы деп аталады; бұл спецификатор туралы А8.9-да айтылған.

Хабарландыруда жад сыныбының бір спецификатордан артық болмауы мүмкін. Егер ол хабарландыруда болмаса, онда мынадай ережелер қолданылады: функциялардың ішінде жарияланатын объектілердің auto класы бар деп есептеледі; функциялардың ішінде жарияланатын функциялар – extern класы; функциялардан тыс жарияланатын объектілер мен функциялар – статикалық және сыртқы байланыстары бар (А10, А11-ді қараңыз).

А 8.2. Типті спецификаторлар

Типті спецификаторлар келесідей анықталады:

Типті спецификатор:
    void
    char

short
int
long
float
double
signed
unsigned
құрылым-немесе-бірлестік-спецификатор
аудару спецификаторы
typedef-ат

Int-мен бірге басқа бір сөзді пайдалануға болады-ұзын немесе қысқа; және long int тіркесімі жай ұзын сияқты мағынасы бар; short int ұқсас — short сияқты. Long сөзі double - мен бірге қолданыла алады. Int және оның басқа да модификацияларымен (short, long немесе char) signed немесе unsigned сөздерінің бірін пайдалануға рұқсат етіледі. Соңғысының кез келгені дербес пайдаланылуы мүмкін, бұл жағдайда int түсініледі.

Signed спецификаторлары char типті нысандардың белгісі болуын қамтамасыз ету қажет болғанда пайдалы болады; оны басқа бүтін түрлерге де қолдануға болады, бірақ бұл жағдайларда ол артық.

Жоғарыда сипатталған жағдайларды қоспағанда, хабарландыру біреуден артық үлгі спецификаторын қамтымайды. Егер хабарландыруда бір де бір үлгі спецификаторы болмаса, онда int түрі көрсетіледі.

Хабарланатын объектілердің ерекше қасиеттерін көрсету үшін квалификаторлар арналады:

квалификатор-типі:
const
volatile

Үлгі біліктілігі кез келген үлгі спецификаторымен қолданылуы мүмкін. Const-объектіні инициализациялауға рұқсат етіледі, бірақ одан әрі оған бірдеңе беруге тыйым салынады. Volatile біліктілік мағынасы іске асыруға байланысты.

Const және volatile (өзгермелі) құралдары ANSI стандартымен енгізілген. Const
біліктілігі объектілерді тек оқуға (ROM) ашылған жадқа орналастыру үшін
немесе ықтимал оңтайландыруға ықпал ету үшін қолданылады. Volatile
біліктілік тағайындау — бұл нұсқаусыз жүргізілуі мүмкін оңтайландыруды басу.
Мысалы, енгізу-шығару регистрлерінің мекенжайлары жадтың адрестік
кеңістігіне көрсетілген машиналарда, кейбір құрылғының регистрлеріне
көрсеткіш құрастырушыға тыйым салу үшін volatile ретінде жариялануы мүмкін
көрсеткіш арқылы артық сілтемені үнемдейді. Компилятор көрсетілген
біліктіліктерді елемеуге болады, бірақ constобъектілердің мәнін өзгертуге нақты
әрекет туралы сигнал беруге міндетті.

А8.3. Құрылымдар мен бірлестіктердің хабарландырулары

Құрылым – әртүрлі типтегі атаулы элементтердің тізбектерінен тұратын объект. Біріктіру – әр уақытта әр түрлі бірнеше элементтердің бірін қамтитын объект. Құрылымдар мен бірлестіктердің хабарландырулары бірдей түрге ие.

```
құрылым-немесе-бірлестік-спецификатор:
        құрылым-немесе-біріктіру идентификаторы { хабарландыру құрылымы тізімі }
    құрылым-немесе-біріктіру идентификаторы

құрылым-немесе-бірлестік:
    struct
    union
```

Тізім-хабарландырулар-құрылымдар-құрылым немесе біріктіру элементтері хабарландыру дәйектілігі:

```
хабарландыру тізімі-құрылымдар:
    хабарландыру-құрылымдар
    хабарландыру тізімі-құрылымы хабарландыру құрылымы

хабарландыру-құрылымдар:
    тізім-спецификаторлар-тізім-құрылым-хабарландырулар ;
біліктілік маманданушылар тізімі:
```

```
        үлгі спецификатор-спецификатор-біліктілік
        үлгі-квалификатор біліктілік-мамандануышылар тізімі

хабарландыру-құрылым-тізімі:
        құрылым-хабарлаушы
        тізім-құрылым-хабарлаушы, құрылым-хабарлаушы
```

Әдетте хабарландыру-құрылымның немесе бірлестіктің элементтері үшін жай ғана хабарландыру болып табылады. Құрылым элементтері, өз кезегінде, берілген разряд (бит) санынан тұруы мүмкін. Мұндай элемент бит өрісі немесе жай өріс деп аталады. Оның өлшемі қос нүкте өрісінің атынан бөлінеді:

```
құрылым-хабарлаушы:
    хабарлаушы
    хабарлаушы: тұрақты өрнек

Түрі бар типті спецификатор

құрылым-немесе-біріктіру id {тізім-хабарландырулар-құрылымдар }
```

тізіммен ерекшеленетін құрылым немесе бірлестік тег идентификаторын жариялайды. Сол немесе ішкі көріну аймағындағы кейінгі хабарландыру спецификаторда тізімсіз тег пайдалана отырып, сол түрге жүгіне алады:

```
құрылым-немесе-біріктіру идентификаторы
```

Егер тек бір ғана спецификатор бар болса, бірақ тізім жоқ, тег жарияланбаған жерде пайда болса, аяқталмаған түрі ерекшеленеді. Құрылымның немесе бірлестіктің аяқталмаған түрі бар объектілер олардың өлшемін білу талап етілмейтін контекстте — мысалы, көрсеткішті сипаттау немесе typedef құру үшін хабарландыруларда (бірақ анықтамаларда емес), бірақ өзге жағдайларда емес айтылуы мүмкін. Хабарландыру тізімі бар осы тегпен кейінгі спецификатор пайда болған кезде, түрі аяқталады. Тіпті тізімдегі спецификаторларда жарияланатын құрылымның немесе бірлестіктің түрі тізімнің ішінде аяқталмаған және спецификаторды аяқтайтын} белгісі пайда болғаннан кейін ғана аяқталады.

Құрылымда аяқталмаған элементтердің болуы мүмкін емес. Демек, өздері бар құрылымды немесе бірлестікті жариялау мүмкін емес. Алайда, құрылым немесе бірлестік түріне ат беруден басқа, тег өзі айналатын құрылымдарды анықтауға мүмкіндік береді; құрылым немесе бірлестік өзіне арналған көрсеткіштерді қамтуы мүмкін, өйткені аяқталмаған түрлерге арналған көрсеткіштерді жариялауға болады.

Ерекше ереже түр хабарландыруларына қолданылады.
құрылым-немесе-біріктіру идентификаторы ;

олар құрылымды немесе бірлестікті жариялайтын, бірақ хабарландыру мен хабарландырушының тізімі жоқ. Егер идентификатор сыртқы көріну аймағында (A 11.1) құрылым немесе біріктіру тегіне ие болса да, бұл хабарландыру идентификаторды жаңа құрылым немесе ішкі көріну аймағында аяқталмаған типті біріктіру тегіне айналдырады.

Бұл түсінбеушілік ереже — ANSI-де жаңа. Ол ішкі көріну аймағында жарияланған
өзара рекурсивті құрылымдарға арналған, бірақ олардың тегтері сыртқы көріну
аймағында жариялануы мүмкін.

Құрылымның немесе тізіммен біріктірудің спецификаторы, бірақ тегсіз бірегей түр жасайды, оған тек бір бөлігі болып табылатын хабарландыруда ғана жүгінуге болады.

Элементтер мен тег атаулары бір-бірімен немесе әдеттегі айнымалылармен қақтығыспайды. Элемент аты бірдей құрылымда немесе бірлестікте екі рет пайда болмайды, бірақ сол элементті әр түрлі құрылымдарда немесе бірлестіктерде қолдануға болады.

Бұл кітаптың бірінші редакциясында құрылым мен бірлестік элементтерінің атаулары өз ата-аналарымен байланыспаған. Дегенмен, компиляторларда бұл байланыс ANSI стандарты пайда болғанға дейін әдеттегі болды.

Өріс болып табылмайтын құрылымның немесе бірлестіктің элементі объектінің кез келген түріне ие болуы мүмкін. Өрістің (хабарландырушысы жоқ және, демек, аты жоқ болуы мүмкін) int, unsigned int немесе signed int түрі бар және бит ұзындығы көрсетілген бүтін түрдегі нысан ретінде түсіндіріледі. Int өрісі таңбалы немесе беймәлім болып саналады ма, іске асыруға байланысты. Көрші элемент-өріс іске асыруға байланысты бағытта іске асыруға байланысты жад ұяшықтарына оралады. Өрістің артынан басқа өріс ішінара толтырылған жад ұяшығына кірмесе, ол екі ұяшық арасында бөлінуі мүмкін немесе ұяшық балластпен бітелуі мүмкін. Нөлдік еннің аты жоқ өрісі осындай бітеуге әкеледі, сондықтан келесі өріс келесі жад ұяшығының шетінен басталады.

ANSI стандарты кітаптың бірінші редакциясына қарағанда өрістерді одан да тәуелді етеді. Бит өрістерін «іске асыруға байланысты» түрде квалификациясыз сақтау үшін тіл ережелерін оқып шығу керек. Биттік өрістері бар құрылымдар жадтың көлемін құрылымға (бағдарлама коды мен өрістерге қол жеткізу уақытын ұлғайту бағасы) азайту үшін тасымалданатын әдіс немесе биттік деңгейде жадтың таралуын сипаттау үшін көтерілмейтін әдіс ретінде қызмет етуі мүмкін. Екінші жағдайда жергілікті іске асыру ережелерін түсіну қажет.

Құрылым элементтері мекенжай элементтерінің жариялануына қарай ұлғаятын болады. Өріс болып табылмайтын құрылым элементтері өз түріне байланысты мекенжай шекарасы бойынша теңестіріледі; осылайша құрылымда атаусыз тесіктер болуы мүмкін. Егер құрылымдағы көрсеткіш оның бірінші элементінің көрсеткіш түріне келтірілсе, нәтиже бірінші элементті көрсетеді.

Біріктіруді барлық элементтері 0 ығысуымен басталатын және кез келген элементтерді сақтау үшін жеткілікті болатын құрылым ретінде елестетуге болады. Кез келген уақытта бірлестікте бір элементтен артық емес сақталады. Біріктіру көрсеткіші бір элементтің көрсеткіш түріне келтірілсе, нәтиже осы элементті көрсетеді.

Мұнда құрылымын жариялау қарапайым мысал:

```
struct tnode {
```

```
        char tword[20];
        int count;
        struct tnode *left;
        struct tnode *right;
};
```

Бұл құрылым 20 таңбадан тұратын массив, int түрі және ұқсас құрылымға екі көрсеткіш бар. Егер осындай хабарландыру берілсе, онда

```
struct tnode s, *sp;
```

s берілген түрдің құрылымы ретінде, ал sp — бұл құрылымға көрсеткіш ретінде жариялайды. Келтірілген анықтамаларға сәйкес өрнек

```
sp->count
```

sp көрсететін құрылымдағы count элементіне жүгінеді;

```
s.left
```

- s құрылымында сол жақ тұтқаны көрсеткіш; a

```
s.right->tword[0]
```

— бұл оң жақтағы s tword элементінің алғашқы символы.

Жалпы айтқанда, соңғы рет мән берілген бірлестік элементі пайдаланылатынын бақылау мүмкін емес. Алайда, біріктіру элементтерімен жұмыс істеуді жеңілдететін ережелердің орындалуына кепілдік беріледі: егер бірлестікте олар үшін деректердің жалпы реттілігімен басталатын бірнеше құрылымдар болса және егер бірлестікте ағымдағы сәтте осы құрылымдардың біреуі болса, онда деректердің жалпы бөлігіне көрсетілген құрылымдардың кез келгені арқылы жүгінуге рұқсат етіледі. Сонымен, Бағдарламаның келесі фрагменті заңды:

```
union {
      struct {
```

```
            int type;
        } n;
        struct {
            int type;
            int intnode;
        } ni;
        struct {
            int type;
            float floatnode;
        } nf;
} u;
...
u.nf.type = FLOAT;
u.nf.floatnode = 3.14;
...
if (u.n.type == FLOAT)
... sin(u.nf.floatnode) ...
```

А 8.4. Есептеулер

Есептеулер – бұл санды санағыштар деп аталатын көптеген тұрақты константаларымен жабылатын ерекше тип. Санақ спецификаторының түрі құрылымдар мен кәсіподақтардан алынады.

```
есептеулер-спецификатор:
        enum идентификаторы, { есептеулер тізімі }
        enum идентификаторы

есептеулер тізімі:
        есептеулер
        есептеулер тізімі, есептеулер

есептеулер:
        идентификатор
        идентификатор = тұрақты өрнек
```

Тізімдер тізіміне кіретін идентификаторлар int типті константалармен жарияланады және тұрақты қажет болған жерде қолданылуы мүмкін. Егер осы тізімде = белгісімен бірде-бір тізбек болмаса, онда константаның мәні 0 басталады және солдан оңға қарай хабарландыру оқылуына қарай 1-ге артады. = белгісімен аударғыш

тиісті идентификаторға мән береді; келесі идентификаторлар берілген мәннен прогресті жалғастырады.

Бір көріну аймағында қолданылатын тізімдеу аттары бір-бірінен және әдеттегі айнымалылардың аттарынан ерекшеленуі тиіс, бірақ олардың мәндері сәйкес болуы мүмкін.

Тізім-спецификатордағы идентификатордың рөлі структ-спецификатордағы құрылым тегінің рөліне ұқсас: ол белгілі бір тізбектің аты болып табылады. Тізімдер мен тізбек-спецификаторларға (тегтермен және тегтерсіз) арналған ережелер, құрылымдар мен бірлестіктердің спецификаторларына арналған сияқты, тізімдеу элементтері аяқталмаған тип болмайды деген ескертпемен; тізімдеу тізімі жоқ тізім-спецификатордың тег көріну аймағы шегінде тізіммен спецификатор болуы тиіс.

Аударма тілінің бірінші нұсқасында жоқ болғанымен, олар бірнеше жыл бойы қолданылады.

А 8.5. Хабарлаушылар

Жариялаушыларда келесі синтаксис бар:

```
хабарлаушы:
        өзін-өзі хабарлау көрсеткіші

өзін-өзі хабарлаушы:
        идентификатор
        ( хабарлаушы )
        өзін-өзі хабарлаушы [ тұрақты өрнек ]
        өзін-өзі хабарлаушы ( параметр-тип-тізім)
        өзін-өзі хабарлаушы ( идентификаторлар тізімі)

көрсеткіш:
        *квалификатор типті тізім
        *көрсеткіш типті квалификатор тізімі
```

квалификатор типті тізім:
квалификатор типті тізім
квалификатор тізімі - квалификатор типті тізім

Хабарлаушының құрылымында шағылысу құрылымымен ұқсас белгілер көп, өйткені хабарлаушыға жанама айналыс операциялары, функцияға жүгіну және массив элементін алу (сол қолдану тәртібімен) рұқсат етіледі.

А 8.6. Хабарландырушылар нені білдіреді

Хабарландырушылардың тізімі түрі спецификаторларынан және жад сыныбының көрсеткішінен кейін бірден орналасады. Кез келген хабарландырушының басты элементі – ол жариялаған идентификатор; қарапайым жағдайда оның бір хабарлаушысы және грамматика өнімдерінің бірінші жолында өзіндік хабарландырушы атауымен көрсетілген. Жад класс спецификаторлары тікелей сәйкестендіргішке жатады және оның түрі хабарлаушының түріне байланысты. Хабарландырушыны бекіту ретінде қабылдау керек: егер айқындамада идентификатор хабарландырушыдағы сияқты контексте пайда болса, онда ол ерекше түрдегі объектіні білдіреді.

Егер (А8.2) түрге жататын хабарландырудың спецификаторларын және кейбір нақты хабарландырушыны қоссаңыз, бұл хабарландыру Т түрін қабылдайды», мұнда Т – түр, ал D – хабарландырушы. Бұл жазба индуктивті кез келген хабарландырушының идентификаторына түр береді.

Т D хабарландыруында, Д-жай идентификатор болса, идентификатор түрі Т болады.

Т D хабарландыруда D

(D1)

D1-де сәйкестендіргіштің түрі D-мен бірдей болады. Жақша түрін өзгертпейді, бірақ олар оны күрделі декларациядағы идентификаторларға «байланыстыру» нәтижелеріне әсер етуі мүмкін.

А 8.6.1. Көрсеткіштер хабарландырушылары

Хабарландыру T D, D түрі

* D1 типті квалификатор тізімі

декларация идентификаторының типі T D1 - «тип-түрлендіргіш T», D сәйкестендіргіштің түрі - «тип-түрлендіргіш-классификатор-тізім типінің сілтемесі». Келесі біліктіліктер * сілтейтін нысанды емес, меңзердің өзін білдіреді. Мысалы, декларацияны қарастырайық:

```
int *ap[];
```

Мұнда ap [] D1 рөлін атқарады; int ap [] декларациясы «int жиымы» деп шешілуі керек (төменге қараңыз); тип классификаторларының тізімі мұнда бос, ал түрлендіргіш «массив» болып табылады. Осылайша, іс жүзінде ap декларациясында: «интегралға дейінгі көрсеткіштер жиыны» делінген. Жарнамалардың бірнеше мысалдары:

```
int i, *pi, *const cpi = &i;
const int ci = 3, *pci;
```

Олар і мен бүтін санды рі санына апарады. срі мәні өзгермейді; срі мәні өзгеретін болса да, әрқашан бір орынға нұсқайды. Бүтін сан тұрақты болып табылады, өзгере алмайды (бірақ бұл жағдайда инициализациялауға болады). Рсі көрсеткіші «pointer to const int» деп айтылады; көрсеткіштің өзін өзгертуге болады; ол басқа орынға нұсқайды, бірақ оны көрсететін мәнді рсі арқылы өзгерту мүмкін емес.

А 8.6.2. Массивтер хабарландырушылары

D түрі бар T D хабарландыруларында

```
D1 [ тұрақты өрнек ]
```

және онда хабарландыру идентификаторының түрі T D1 бар «модификатор-Т типті», идентификатордың түрі D бар «модификатор-т массив түрі». Егер тұрақты өрнек болса, онда ол бүтін және 0-ден көп болуы керек. Егер массивте элементтердің санын сипаттайтын тұрақты өрнек жоқ болса, массивтің аяқталмаған түрі бар.

Массивті арифметикалық түрдегі объектілерден, сілтегіштерден, құрылымдар мен бірлестіктерден, сондай-ақ басқа массивтерден (көп өлшемді массивтерді жасай отырып) құрастыруға болады. Массивтің кез келген түрі аяқталған болуы керек, мысалы, аяқталмаған типтегі құрылым немесе массив болуы мүмкін емес. Бұл дегеніміз, көп өлшемді массив үшін тек бірінші өлшемділік бос болуы мүмкін. Массивтің аяқталмаған түрі осы массивтің басқа хабарландыруында (10.2) немесе оның инициализациясы кезінде (А8.7) аяқталады. Мысалы, жазу

```
float fa[17], *afp[17];
```

float түріндегі сандардың массивін және float түріндегі көрсеткіштердің массивін жариялайды. Оған ұқсас

```
static int x3d[3][5][7];
```

3x5x7 өлшемді статикалық үш өлшемді массивін жариялайды. Шын мәнінде, егер дәл болса, x3d үш элементтің массиві болып табылады, олардың әрқайсысы 7 int типті мәндерден тұратын бес элементтің массиві бар.

E1[E2] индекстеу операциясы *(E1+E2) операцияларына ұқсас болып анықталды. Демек, жазбаның асимметриясына қарамастан, индекстеу-коммутативті операция. Оператор + және массивтер (А6.

6, А7. 1, А7. 7) үшін қолданылатын түрлендіру ережелерін ескере отырып, Егер Е1-массив, ал Е2-бүтін болса, онда Е1 [Е2] Е1 массивінің Е2-ші элементін білдіреді.

Мысалы, x3d[i] [j] [k] *(x3d[i] [j]+k) дегенді білдіреді. Бірінші түрленуі, x3d[i][j], А7.1 сәйкес, «бүтін массивіне көрсеткіш» түріне келтіріледі; А7.7 тақырыбында айтылғандай қосу int типті объектінің өлшеміне көбейтуді қамтиды. Осы ережелерден, массивтер «жол» (соңғы индекстер жиі өзгереді) есте болады және хабарландырудағы бірінші өлшемділік массивтің жадының мөлшерін анықтауға көмектеседі, бірақ массивтің элементінің мекен-жайын есептеу қатыспайды.

А 8.6.3. Функцияларды жариялаушылар

Жаңа әдіспен d түрі бар T D функциясын жариялау

```
D1 (тізім-түрлері-параметрлері)
```

және хабарландыру идентификаторының түрі T D1 бар «модификатор-Т типті», идентификатордың түрі D бар «модификатор-т қайтаратын тізім-типтері-параметрлерінің аргументтері бар функция түрі-модификатор». Параметрлерде келесі синтаксис бар:

```
тізім-түрі-параметрлері:
    параметрлер тізімі
    параметрлер тізімі , ...

параметрлер тізімі:
    хабарландыру-параметр
    параметрлер тізімі, хабарландыру-параметр

хабарландыру-параметр:
    спецификаторлар-хабарландырулар хабарландырушы
    спецификаторлар-хабарландырулар абстрактілі-хабарландыру
```

Жаңа тәсілі хабарландырулар функцияларының тізімі параметрлерін специфицирует олардың түрлері, ал егер функциясы жоқ

параметрлердің орнында тізімнің типі көрсетіледі бір сөз – void. Егер параметрлер түрлерінің тізімі көп нүктемен аяқталса, »...» функцияның анық сипатталған параметрлердің санынан артық дәлелдері болуы мүмкін. (7.3.2 қараңыз.)

Массивтер мен функциялар болып табылатын параметрлердің түрлері параметрлерді өзгерту ережелеріне (А10.1) сәйкес көрсеткіштерге ауыстырылады. Параметрді хабарландыруда қолдануға болатын жалғыз жад класс спецификаторы-register, бірақ егер функция хабарландырушысы оны анықтаудың тақырыбы болмаса, ол елемейді. Сол сияқты, егер хабарландыруларда параметрлер хабарландыруларында идентификаторлар болса, ал функция хабарландырушысы функцияны анықтаудың тақырыбы болып табылмаса, онда бұл идентификаторлар сол мезетте ағымдағы көріну аймағынан шығарылады.

Ескі әдіспен d түрі бар T D функциясын жариялау

D1 (идентификаторлар тізімі)

және хабарландыру идентификаторының түрі t D1 бар «модификатор-t типті», идентификатордың түрі D бар «модификатор-T қайтаратын спецификацияланбаған аргументтерден функция түрі». Параметрлер, егер олар бар болса, келесі түрге ие:

идентификаторлар тізімі:
    идентификатор
    идентификаторлар тізімі, идентификатор

Ескі әдіс кезінде, егер функция хабарландырушысы функцияны анықтау тақырыбы ретінде пайдаланылмаса (а 10.1), идентификаторлар тізімі болмауы керек. Хабарландыруларда параметрлер түрлері туралы ешқандай ақпарат жоқ.

Мысалы:

```
int f(), *fpi(), (*pfi)();
```

F функциясын, int түрі санын, арі функциясын, Int түрі санына сілтегішті және int түрі санын қайтаратын функцияға pfi сілтегішін жариялайды. Хабарландыруда ешбір функция параметр түрі көрсетілмеген; барлық функциялар ескі жолмен сипатталған.

Жаңа жазбада хабарландыру төмендегідей болады:

```
int strcpy(char *dest, const char *source), rand(void);
```

Мұнда strcpy-int түрінің мәнін қайтаратын екі аргументі бар функция; бірінші аргумент-char түрінің мәніне көрсеткіш, ал екіншісі-char түрінің өзгермейтін мәніне көрсеткіш. Параметрлер атаулары жақсы пікірлердің рөлін атқарады. Екінші функция, rand, аргументтері жоқ және int қайтарады.

Параметрлер прототипі бар функцияларды жариялаушылар-ANSI-стандартының ең маңызды жаңалығы — Тілдің бірінші редакциясында қабылданған ескі әдіспен салыстырғанда, олар барлық шақыруларда дәлелдерді тексеруге және қажетті түрге келтіруге мүмкіндік береді. Алайда, оларды енгізу тілге кейбір сумятицаны және екі нысанды келісу қажеттілігін әкелгенін атап өткен жөн. Үйлесімділікті қамтамасыз ету үшін кейбір» синтаксистік ұсқынсыздықтар « қажет болды, атап айтқанда void, параметрлердің болмауын айқын көрсету үшін.
Көпнүкте «, ...»аргументтердің әртүрлі санымен функцияларға қатысты —
стандартты макростардың тақырып файлымен бірге <stdarg.h> ресми емес пайдаланылатын, бірақ ресми түрде бірінші редакцияда тыйым салынған механизмді қалыптастырады.
Бұл жазу әдістері C++тілінен алынған.

А 8.7. Инициализация

Иниц-хабарландырушының көмегімен хабарланатын объектінің бастапқы мәнін көрсетуге болады. Фигуралы жақшада жасалған өрнек немесе бастамашылардың тізімі болып табылатын бастамашыға = белгісі қойылады. Бұл тізім үтірмен аяқталуы мүмкін; оның мақсаты – пішімдеуді анық ету.

инициализатор:
меншіктеу өрнегі
{ инициализаторлар тізімі }
{ инициализаторлар тізімі, }

инициализаторлар тізімі:
инициализатор
инициализаторлар тізімі, инициализатор

Статикалық объектінің немесе массивтің бастамашысында барлық өрнектер тұрақты болуы тиіс (7.19). Егер auto-және register-объект немесе массив инициализаторы фигуралық жақшаларға салынған тізімде болса, онда оған кіретін өрнектер де тұрақты болуы тиіс. Алайда автоматты жағдайда бір өрнегі бар объект инициализатор тұрақты өрнегі болуға міндетті емес, ол тек нысанға сәйкес тип болуы керек.

Бірінші редакцияда автоматты құрылымдарды, бірлестіктер мен массивтерді инициаландыруға рұқсат етілмеген. ANSI стандарты бұған мүмкіндік береді; алайда, егер инициализатор бір қарапайым өрнектермен ұсынылмаса, инициализация тұрақты конструкциялармен ғана жасалуы мүмкін.

Инициализациясы анық көрсетілмеген статикалық нысан оған (немесе оның элементтеріне) 0 тұрақтысы берілгенде инициалданады. Автоматты нысанның бастапқы мәні айқын инициалдалмаған, яғни, анықталмаған.

Көрсеткіш немесе арифметикалық түрдегі объектінің бастамашысы – бұл объектіге берілетін бір өрнек (мүмкін, фигуралық жақшамен жасалған).

Құрылым инициаторы – сол құрылымдық түрдегі өрнек немесе ретімен берілген оның элементтерінің инициализаторларының фигуралық жақшаларына жасалған. Атаусыз бит өрістері еленбейді және инициалданбайды. Егер инициализаторлар тізімдегі элементтерден аз болса, қалған элементтер нөлмен инициаланады. Инициализаторлар элементтердің санынан артық болмауы керек.

Массивтің инициализаторы – бұл фигуралық жақшаларға салынған оның элементтерінің инициализаторларының тізімі. Егер массив көлемі белгісіз болса, онда ол инициализаторлар санына тең болып саналады, сонымен қатар оның түрі аяқталады. Егер массивтің көлемі белгілі болса, онда инициализаторлар саны оның элементтерінің санынан аспауы тиіс; егер инициализаторлар аз болса, қалған элементтер нөлденеді.

Символдар массивін инициализациялау ерекше жағдай ретінде. Соңғысы жол литералымен инициалдауға болады; таңбалар жол литералында орнатылған тәртіпте массив элементтерін инициализациялайды. Сонымен қатар, Softswitch-тың көмегімен (2.6) wchar_t типті массивін инициалдауға болады. егер массивтің көлемі белгісіз болса, онда ол жолдағы символдар санымен анықталады, соның ішінде соңғы NULL-символ; егер массивтің көлемі белгілі болса, онда жолдағы символдар саны соңғы NULL-символды есептемегенде, оның өлшемінен аспауы тиіс.

Біріктірудің бастамашысы сол түрдегі өрнек немесе оның бірінші элементінің бастамашысының жақшасына жасалған өрнек болуы мүмкін.

Тілдің бірінші нұсқасында бірлестіктерді бастамалауға мүмкіндік болмады.

«Бірінші элемент» ережесі талғампаздықпен ерекшеленбейді, бірақ жаңа синтаксисті талап етпейді. ANSI стандарты анық бірлестіктердің бастамашылық емес семантикасын түсіндіреді.

Құрылым мен массив үшін жалпылама атау енгіземіз: агрегат. Егер Агрегатта агрегаттық типті элементтер болса, инициализация ережелері рекурсивті түрде қолданылады. Кейбір инициализация жағдайларында фигуралы жақшаларды төмендетуге болады. Егер агрегат элементінің бастамашысы өзі болып табылатын сол жақ фигуралы жақшадан басталса, онда бұл қосалқы агрегат кейіннен бөлінген үтірлі бастамашылардың тізімімен бастамаланады; Егер қосалқы агрегат бастамашысының саны оның элементтерінің санынан асып кетсе, қате деп саналады. Егер субагрегаттық бап-

тағыш сол жақ бұйралаудан басталмаса, оны баптау үшін тізімдегі элементтердің тиісті санын санау керек; ал қалған элементтер агрегаттың келесі инициализаторларымен интегралданған, олар осы субегрегат болып табылады.

Мысалы:

```
int x[] = { 1, 3, 5 };
```

х-ті үш элементі бар бір өлшемді массив ретінде жариялайды және инициалдайды, себебі ешқандай өлшем көрсетілмеген және тізім үш инициализатордан тұрады.

```
float y[4][3] = {
    { 1, 3, 5 },
    { 2, 4, 6 },
    { 3, 5, 7 }, };
```

инициализация — бұйра жақшалардың толық жиынтығымен: 1, 3 және 5 массивтегі бірінші жолды инициализациялау у [0], яғни у [0] [0], у [0] [1] және у [0] [2]. Келесі екі жол осылай басталады: у [1] және у [2]. Барлық массивке инициализаторлар жеткіліксіз болды, сондықтан у [3] жолының элементтері бос болады. Дәл осындай нәтижеге келесі мәлімдеме арқылы қол жеткізіледі:

```
float y[4][3] = {
    1, 3, 5, 2, 4, 6, 3, 5, 7
```

Y үшін инициализатор сол жақ бұдырлы жақшадан басталады, бірақ [0] үшін жақша жоқ, сондықтан тізімнен үш элемент алынады. Сол сияқты үш элемент у [1], содан кейін у [2] үшін алынады. IN };

```
float y[4][3] = {
    { 1 }, { 2 }, { 3 }, { 4 }
};
```

у матрицасының бірінші бағанына инициализация жасалады, қалған барлық элементтер нөлге тең болады.

Соңында

```
char msg[] = « Синтаксистік қате %s\n»;
```

элементтері жолмен инициалданған таңбалар массивінің мысалы; оның өлшемі ақырғы NULL түртіндісін қамтиды.

А 8.8. Түрлер атаулары

Сонымен қатар, деректер түрінің атын қолдану қажеттілігі туындайды (мысалы, түрге нақты келтіргенде, хабарландыру ішіндегі типтік параметрлерді көрсету кезінде sizeof операторының аргументінде болады). Бұл қажеттілік сол түрдегі объектінің жариялануымен синтаксистік түрде сәйкес келетін түрдің атауымен жүзеге асырылады. Ол хабарландырудан тек объектінің атауы жоқ болғандықтан ерекшеленеді.

```
аты-типті:
    список-спецификатор- квалификатор абстрактілі-хабарлаушы

абстрактілі-хабарлаушы:
    көрсеткіші
    көрсеткіш нақты-абстрактілі-хабарлаушы

шын мәнінде-абстрактілі-хабарлаушы:
    ( абстрактілі-хабарлаушы)
    шын мәнінде-абстрактілі-хабарлаушы [ тұрақты өрнек ]
    шын мәнінде-абстрактілі-хабарландыру ( тізім-түрі-параметр)
```

Егер бұл құрылым толыққанды хабарлаушы болса, идентификатор болуы мүмкін абстрактілі хабарландырушыдағы жалғыз орынды көрсетуге болады. Аталған түрі осы «көрінбейтін идентификатор» түріне сәйкес келеді. Мысалы

```
int int *
int *[3]
int (*)[]
int *()
int (*[])(void)
```

тиісінше int, «int-ға көрсеткіш», «int-ға үш көрсеткіш массиві», «int-ға белгісіз көрсеткіш массиві», «int-ға сілтегішті қайтаратын белгісіз параметрлер санының функциясы», «параметрлерсіз функцияларға сілтегіштердің белгісіз санының массиві, олардың әрқайсысы int-ны қайтарады»типтерін көрсетеді.

А 8.9. Typedef хабарландыру

Typedef жадының спецификаторы бар хабарландырулар объектілерді жарияламайды — олар түрлердің аттарын білдіретін идентификаторларды анықтайды. Бұл идентификаторлар typedef-атаулары деп аталады.

```
typedef-аты:
    идентификатор
```

Typedef хабарландыруы өзінің хабарландырушысының әрбір атының түрін әдеттегі тәсілмен жазады (8.6 қараңыз.). Осы сәттен бастап typedef-атау синтаксистік түр спецификаторының негізгі сөзіне тең, онымен байланысты түрін білдіреді. Мысалы,

```
typedef long Blockno, *Blockptr;
typedef struct { double r, theta; } Complex;
```

келесі хабарландыруларға жол беріледі:

```
Blockno b;
extern Blockptr bp;
Complex z, *zp;
```

b long түріне, bp – «long көрсеткішіне» тиесілі, z – бұл берілген түрдің құрылымы, а zp - «осындай құрылымға көрсеткіш» түріне жатады.

Typedef хабарландыруы жаңа түрлерді енгізбейді, ол тек басқа жолмен ерекшеленуі мүмкін түрлерге атаулар береді. Мысалы, b ұзын типті кез келген басқа нысан сияқты бірдей түрі бар.

typedef-аттар ішкі көріну аймағында басқа анықтамалармен жабылуы мүмкін, бірақ оларда типін көрсету болған жағдайда. Мысалы

```
extern Blockno;
```

Blockno-ны жарияламайды, бірақ

```
extern int Blockno;
```

жариялайды.

А 8.10. Типтердің эквиваленттілігі

Егер оларда атаулардың синонимиялығын ескере отырып (мысалы, long және long int бірдей типтегі болып саналады) типті спецификаторлардың бірдей жиынтығы бар болса, типті спецификаторлардың екі тізімі баламалы. Әр түрлі тегтермен құрылымдар, бірлестіктер және тізімдер әр түрлі болып саналады, ал әрбір тегсіз бірлестік, құрылым немесе тізімдеу бірегей түрі болып табылады.

Егер олардың абстрактілі хабарлағыштары (А 8.8) барлық typedef аттарын олардың типтерімен ауыстырғаннан кейін және функция параметрлерінің аттарын тастағаннан кейін типтердің спецификаторларының эквивалентті тізімін құрастырса, екі түр сәйкес келеді деп саналады. Салыстыру кезінде массивтердің өлшемдері мен функциялар параметрлерінің типтері ескеріледі.

А 9. Нұсқаулық

Айтылған жағдайларды қоспағанда, нұсқаулықтар олар қалай жазылған тәртіппен орындалады. Нұсқаулар маңызды емес және белгілі бір әрекеттерді орындау үшін орындалады. Нұсқаулардың барлық түрлерін бірнеше топқа бөлуге болады:

нұсқаулық:
    белгіленген-нұсқаулық
    өрнек нұсқаулығы
    құрама нұсқаулық
    таңдау нұсқаулығы
    циклдік нұсқаулық
    көшу нұсқаулығы

А 9.1. Белгіленген нұсқаулар

Нұсқаулар белгінің алдында болуы мүмкін.

белгіленген-нұсқаулық:
  идентификатор : инструкция
  case тұрақты өрнек : нұсқаулық
  default : нұсқаулық

Идентификатордан тұратын белгі осы идентификаторды жариялау болып табылады. Белгі идентификаторының жалғыз мақсаты-goto үшін өту орнын көрсету. Id-белгі көріну аймағы ағымдағы функция болып табылады. Tages өз атауларының кеңістігіне ие болғандықтан, олар басқа идентификаторлармен «қақтығыспайды» және жабу мүмкін емес (А11.1 қараңыз).

case-tages және default-tages switch (А9.4) нұсқауларында қолданылады. Case-дағы тұрақты өрнек бүтін болуы керек.

Белгілердің өздері есептеу тәртібін өзгертпейді.

А 9.2. Өрнек нұсқаулығы

Нұсқаулықтың ең көп қолданылатын түрі – бұл өрнек нұсқаулығы.

өрнек нұсқаулығы:
    өрнек;

Ең жиі өрнек нұсқаулығы – функцияны меншіктеу немесе шақыру болып табылады. Өрнектің жанама әсерін жүзеге асыратын бар-

лық әрекеттер келесі Нұсқаулық орындалғанға дейін аяқталады. Егер нұсқаулықта өрнек түсірілсе, онда ол бос деп аталады; бос нұсқаулық циклдік нұсқаулықтың бос денесін белгілеу үшін немесе белгі орны ретінде жиі қолданылады.

А 9.3. Құрама нұсқаулық

Бір нұсқаудың синтаксисіне сүйенетін жерлерде кейде бірнешеуін орындау қажеттілігі пайда болғандықтан, құрама нұсқауды (блок деп аталады) тапсыру мүмкіндігі қарастырылады. Функцияны анықтау денесі құрамдас нұсқаулық бар:

```
құрама нұсқаулық:
        { тізім-хабарландырулар тізімі-нұсқаулар }

хабарландыру тізімі:
        хабарландырулар
        хабарландыру тізімі хабарландыру

нұсқаулар тізімі:
        нұсқаулық тізім-нұсқаулық
```

Егер хабарландыру тізімінен идентификатор көлемді блоктың көріну аймағында болса, онда сыртқы хабарландырудың әрекеті осы блоктың ішіне кіргенде тоқтатылады (А11. 1), ал одан шыққаннан кейін қайта басталады. Блоктың ішінде id тек бір рет жариялануы мүмкін. Әрбір жеке атаулар кеңістігі үшін бұл ережелер тәуелсіз әрекет етеді (А 11); әр түрлі атаулар кеңістіктеріндегі идентификаторлар әр түрлі.

Автоматты объектілерді инициализациялау блокқа әрбір кіруде жүзеге асырылады және хабарландырушылар бойынша жылжу шамасына қарай жалғасады. Басқару блоктың ішіне берілгенде ешқандай инициализация орындалмайды. Статикалық нысандарды инициализациялау бағдарламаны іске қосар алдында бір рет жүзеге асырылады.

А 9.4. Таңдау нұсқаулары

Таңдау нұсқаулары нұсқауларды орындау тәртібін айқындайтын бірнеше баламалардың бірін іріктеуді жүзеге асырады.

таңдау нұсқаулары:
if (өрнек) нұсқаулық
if (өрнек) else нұсқаулығы
switch (өрнек) нұсқаулық

If-нұсқаулардың екі түрі де арифметикалық түрі немесе көрсеткіш түрі болуы керек өрнек бар. Алдымен оның барлық жанама әсерлері бар өрнек есептеледі, нәтиже 0-мен салыстырылады. 0-ге сәйкес келмеген жағдайда бірінші қосымша нұсқаулық орындалады. 0-ге сәйкес келген жағдайда if екінші түрі үшін екінші нұсқауға сәйкес орындалады. Else сөзімен байланысты бір белгісіздік else сөзі осы else блогында және блоктар тіркемесінің бір деңгейінде орналасқан соңғы else if-нұсқаулығымен сәйкес келеді.

Switch нұсқаулығы тұтас түрі болуы тиіс өрнектің мәніне байланысты бірнеше нұсқаулардың біріне басқаруды жібереді.

Нұсқаулық бойынша switch арқылы басқарылатын әдетте құрамдас. Бұл нұсқаудың ішіндегі кез-келген нұсқаулық бір немесе бірнеше case-таңбаларымен (А 9.1) белгіленуі мүмкін. Басқару өрнегі бүтін жоғарылауға ұшырайды (а 6.1), ал case тұрақтылары жоғары түрге келтіріледі. Мұндай түрлендіруден кейін, бір switch нұсқаулығында екі case тұрақтысы бірдей мәндерге ие болмауы керек. Switch-нұсқаулықпен бір default-тегінен артық болмауы мүмкін. Switch конструкциялары бір-біріне салуға рұқсат етіледі; case және default-белгілер ішкі switch-нұсқауларды қамтиды.

Switch нұсқаулығы келесідей орындалады. Барлық жанама әсерлері бар өрнек есептеледі және нәтиже әр case-тұрақтымен салыстырылады. Егер case тұрақтысының біреуі өрнектің мәніне тең болса, Басқару тиісті case белгімен нұсқауға ауысады. Егер case константасының ешқайсысы сәйкес келмесе, басқару default-белгімен

нұсқауға беріледі, егер ондай болса, әйтпесе switch аспабының ешқайсысы орындалмайды.

Тілдің бірінші нұсқасында switch өрнек және case константы int түрі болды.

А 9.5. Циклдік нұсқаулар

Циклдік нұсқаулар циклдарды сипаттайды.

циклдік нұсқаулар:
```
while ( өрнек ) нұсқаулық
do нұсқаулық while ( өрнек )
for ( өрнек ; өрнек ; өрнек ) нұсқаулық
```

While және do операторларында қосымша нұсқаулықтардың орындалуы өрнектің мәні нөлге айналғанша қайталанады. Өрнек арифметикалық немесе сілтегіш типінде болуы керек. Сонымен қатар, барлық жанама әсерлері бар өрнек әр орындалғанға дейін және кейін орындалады.

Мәлімдемеде бірінші өрнек бір рет бағаланады, осылайша цикл басталады. Бұл өрнектің түріне ешқандай шектеулер жоқ. Екінші өрнек арифметикалық немесе сілтегіш типінде болуы керек; ол әр итерация алдында есептеледі. Оның мәні 0-ге тең болған кезде, ол тоқтап қалады. Үшінші өрнек әр итерациядан кейін бағаланады, сондықтан циклді қайта қосады. Оның түріне ешқандай шектеулер жоқ. Барлық үш өрнектің жанама әсерлері бағаланған кезде аяқталады. Егер қосымша нұсқаулықта жалғастырылмаса, онда

```
for ( өрнек1 ; өрнек 2 ; өрнек 3 ) нұсқаулық
```

эквалиентті -

```
өрнек 1;
while ( өрнек2 ) {
      нұсқаулық
      өрнек3;
}
```

Үш цикл өрнегінің кез келгенін қалдыруға болады. Екінші өрнектің болмауы нөлдік тұрақты мәнді нөлмен салыстыруға тең деп есептеледі.

А 9.6. Ауысу нұсқаулары

Ауысу нұсқаулары басқаруды сөзсіз беруді жүзеге асырады.

Ауысу нұсқаулары:

```
goto идентификаторы ;
continue ;
break ;
return өрнек ;
```

Goto-нұсқаулықта id ағымдағы функцияда орналасқан (А9.1) белгісі болуы керек. Басқарма белгіленген нұсқаулыққа беріледі.

Continue нұсқауын цикл ішінде ғана орналастыруға болады. Ол өзінің ішкі циклінің келесі итерациясына көшуді тудырады. Әр конструкциялар үшін дәлірек айтқанда

```
while (...) {        do {                 for (...) {
  ...                  ...                  ...
contin: ;            contin: ;            contin: ;
}                    } while (...);       }
```

continue нұсқаулығы, егер ол одан да ішкі циклға қойылмаса, goto continue сияқты жасайды.

Break нұсқаулығы циклдік немесе switch-нұсқауларда кездеседі және оларда ғана. Ол осы break нұсқаулығын қамтитын ішкі циклдік немесе switch нұсқауларын аяқтайды, содан кейін Басқару келесі нұсқаулыққа ауысады.

Return арқылы функция басқаруды туындаған бағдарламаға қайтарады. Егер return үшін өрнек болса, онда оның мәні осы функция-

ны тудырған бағдарламаға қайтарылады. Өрнектің мәні функция сияқты бірдей түрге ие айнымалыға берілсе, түрге келтіріледі.

Есептеулердің «жолы» функцияның соңына (яғни соңғы жабатын жақшаға) әкелетін жағдай return-нұсқауларды білдірмей орындауға тең. Бұл ретте, сондай-ақ return тапсырмасы айқын болған жағдайда қайтарылатын мән анықталмады.

А 10. Сыртқы хабарландырулар

С-компилятор үшін енгізу ретінде дайындалған нәрсе хабар тарату бірлігі деп аталады. Ол сыртқы жарнамалардың реттілігінен тұрады, олардың әрқайсысы жарнама немесе функцияның анықтамасы.

трансляция бірлігі:
    сыртқы хабарландыру
    трансляция бірлігі сыртқы хабарландыру

сыртқы хабарландыру:
    анықтау-функциялары
    хабарландыру

Сыртқы жарнамалардың көріну аймағы блоктағы жарнамалардың көріну аймағы осы блоктың екі соңына дейін таратылады, сол сияқты олар жарияланған хабар бірлігінің екі соңына дейін созылады. Сыртқы жарнаманың синтаксисі кез келген басқа жарнаманың синтаксисінен айырмашылығы жоқ: функция кодын тек сыртқы жарнаманың көмегімен анықтауға болады.

А 10.1. Функцияны анықтау

Функцияны анықтаудың келесі түрі бар:

Функцияны анықтау:
    спецификаторлар-хабарландыру хабарлаушы тізім-хабарландыру
      құрама-нұсқаулық

Жад класс спецификаторларында - хабарландыруларда тек extern және static болуы мүмкін; соңғыларының арасындағы айырмашылықтар А11.2-де қарастырылады.

Функциямен қайтарылатын мәннің түрі арифметикалық түр, құрылым, біріктіру, көрсеткіш және void болуы мүмкін, бірақ «функция» және «массив»емес. Хабарлаушы функцияны хабарландыруда өзі сипаттаған идентификатордың «функция» түрі бар екенін анық көрсетуі керек, яғни келесі екі нысанның бірі болуы керек (А8.6.3):

жеке хабарландырушы (тізім-түрі-параметрлері)
жеке хабарландырушы (тізім-идентификатор)

нақты хабарландырушы жақшада орналасқан идентификатор немесе идентификатор бар. Typedef арқылы «функция» түрін алуға болмайды.

Бірінші нысан — функцияны жаңа тәсілмен анықтауға сәйкес келеді, ол үшін параметрлерді тізімде-түрлер-параметрлерді олардың типтерімен бірге жариялау тән; хабарлағыштың артында жарияланымдардың тізімі болмауы тиіс. Егер тізім-түрлер-параметрлер бір-жалғыз void сөзінен тұрса, функция жоқ екенін көрсететін, онда әрбір хабарландырушыда тізім-түрлер-параметрлердің идентификаторы болуы тиіс. Егер тізім-түрлер-параметрлер таңбаларымен аяқталатын болса»,...»функцияның шақыруы параметрлерден артық дәлелдер болуы мүмкін; бұл жағдайда қосымша дәлелдерге жүгіну үшін, <stdarg.h> тақырып файлынан va_arg макрос тетігін пайдалану керек аргументтердің айнымалы саны бар функциялардың кем дегенде бір атаулы параметрі болуы керек.

Екінші нысан — функцияны ескі жолмен анықтау. Идентификаторлар тізімі параметрлердің аттарын қамтиды және хабарландыру тізімі оларға түрлерді қосады. Хабарландыру тізімінде тек аталған параметрлерді жариялауға рұқсат етіледі, бастамаландыруға тыйым салынады және жад класс спецификаторларынан тек register ғана мүмкін.

Функцияны анықтаудың сол және басқа тәсілінде барлық параметрлер функцияның денесін құрайтын құрамдас нұсқаудың ең басында жарияланғандай және олармен сәйкес келетін атаулар мұнда жарияланбауы тиіс (кез келген идентификаторлар сияқты, оларды ішкі блоктарда қайта жариялауға болады). «Түрден массив» параметрін хабарлауды «түрге көрсеткіш» деп түсіндіруге болады; «түрді қайтаратын функция» параметрін хабарландыруға ұқсас «түрді қайтаратын функцияға көрсеткіш» деп түсіндіруге болады. Шақыру кезінде оның дәлелдері тиісті түрде өзгертіліп, параметрлерге беріледі (7.3.2-ні қараңыз).

Функцияларды анықтаудың жаңа әдісі ANSI-стандартымен енгізілді. Сондай-ақ,
түрін арттыру операциясында шағын өзгерістер бар; тілдің бірінші нұсқасында
float түрінің параметрлері double ретінде оқылуы керек. Float және double
арасындағы айырмашылық функцияның ішінде параметрге көрсеткіш жасалған
кезде ғана байқалды.

Төменде функцияны жаңа әдіспен анықтау мысалы келтірілген:

```
int max(int a, int b, int c)
{
      int m;
      m = (a > b) ? a : b;
      return (m > c) ? m : c;
}
```

Мұнда int-спецификаторлар-хабарландырулар; max (int a, int b, int c) - функцияны хабарлаушы, a,. . . - - оның кодын беретін блок. Сол функцияны ескі жолмен анықтау келесідей:

```
int max(a, b, c)
int a, b, c;
{
     /* ... */
}
```

мұнда max (a, b, c) — хабарландырушы, a int a, b, c-параметрлер үшін хабарландыру тізімі.

А 10.2. Сыртқы хабарландырулар

Сыртқы хабарландырулар объектілердің, функциялардың және басқа да сәйкестендіргіштердің сипаттамаларын ерекшелейді. “Сыртқы” термині жарнамалар функциялардан тыс орналасқан фактіні атап көрсету үшін пайдаланылады; тікелей сөз extern (“сыртқы”) ол байланысты емес. Сыртқы хабарландыруы бар объект үшін жад класы не мүлдем көрсетілмейді, не extern немесе static ретінде ерекшеленеді.

Бір идентификатор үшін хабар таратудың бір бірлігінде бірнеше сыртқы хабарландыру болуы мүмкін, егер олар бір-бірімен байланыс түрі мен тәсілі бойынша келіссе және егер бұл идентификатор үшін бір анықтамадан артық болмаса.

Объектінің екі хабарландыруы немесе функциялары А8.10-да қаралған ережелерге сәйкес түрі бойынша келісілген болып саналады. Сонымен қатар, егер хабарландыру олардың біреуінде құрылымның, бірлестіктің немесе тізімнің (А8.3) түрі аяқталмаған, ал екіншісінде сол тегпен оған сәйкес тип аяқталған болса, онда мұндай тип келісілген болып саналады. Егер массивтің екі түрі (8.6.2) бір аяқталған, ал екіншісі аяқталмаған болса, онда мұндай түрлер де келісілген болып саналады. Ақырында, егер бір түрі функцияны ескі жолмен, ал екіншісі – бірдей функцияны жаңа жолмен (параметрлерді жариялаумен) сипаттаса, онда мұндай түрлер де келісілген болып саналады.

Егер функцияның немесе объектінің бірінші сыртқы хабарландыруы static спецификаторымен белгіленген болса, онда жарияланған идентификатор ішкі байланысқа ие; әйтпесе – сыртқы байланысқа ие болады. Байланыс тәсілдері 11.2-де талқыланады.

Объектінің сыртқы хабарландыруы, егер оның бастамашысы болса, анықтама болып саналады. Бастамашы жоқ және extern спецификаторы жоқ сыртқы хабарландыру байқау анықтамасы болып саналады. Егер хабар тарату бірлігінде Объектінің анықтамасы пайда болса, онда оның барлық сынақ анықтамалары артық хабарландырулар болады. Егер бұл объект үшін хабар тарату бірлігінде ешқандай анықтама анықталмаса, онда оның барлық сынақ анықтамалары 0 инициализаторымен бір анықтама ретінде түсіндіріледі.

Әрбір объектінің дәл бір анықтамасы болуы керек. Ішкі байланыс объектісі үшін бұл ереже трансляцияның әрбір жеке бірлігіне жатады, өйткені әрбір бірліктегі ішкі байланыс объектілері бірегей. Сыртқы байланыстары бар объектілер жағдайында көрсетілген ереже тұтастай бүкіл бағдарламаға қатысты қолданылады.

Бір анықтаудың ережесі тілдің бірінші нұсқасына қарағанда басқаша тұжырымдалса да, ол бұрынғыға сәйкес келеді. Кейбір іске асыру сынамалық анықтау ұғымын кеңінен түсіндіре отырып, әлсірейді. Аталған Ереженің басқа нұсқасында UNIX жүйелерінде таралған және стандартты жалпы қабылданған кеңейту ретінде танылған, бағдарламаның барлық трансляцияланатын бірліктерінен сыртқы байланыстары бар объектілердің барлық сынамалы анықтамалары әр бірлікте бөлек емес, бірге қаралады. Егер бағдарламада бір жерде анықтама анықталса, онда сынақ анықтамалары жай ғана хабарландыру болып табылады, бірақ егер ешқандай анықтамалар кездеспесе, онда барлық сынақ анықтамалары 0 бастамашысымен бір бірлік анықтама болып табылады.

А 11. Көріну және байланыс саласы

Барлық бағдарламаны құрастырудың қажеті жоқ. Бастапқы мәтінді хабар алмасу бірліктері болып табылатын бірнеше файлдарда сақтауға болады. Бұрын құрастырылған бағдарламалар кітапханалардан жүктелуі мүмкін. Бағдарлама функциялары арасындағы байланыстар шақырулар мен сыртқы деректер арқылы жүзеге асырылуы мүмкін.

Сондықтан ауқымның екі түрі бар: біріншісі — идентификатордың лексикалық аясы: яғни бағдарламалық сипаттамада оның барлық сипаттамалары мағынасы бар аймақ; екінші аймақ — бұл бөлек құрастырылған аударма бірліктерінен идентификаторлар арасында сыртқы байланысы бар нысандармен және функциялармен байланысты аймақ.

А 11.1. Көрінудің лексикалық аймағы

Әрбір идентификатор бірнеше атау кеңістігінің біріне түседі. Бұл кеңістіктер бір-бірімен байланысты емес. Бір идентификатор әр түрлі атаулар кеңістігіне тиесілі болса, көрінудің бір аймағында да әр түрлі мағынада пайдаланылуы мүмкін. Төменде үтірлі нүкте арқылы аталғанаттары жеке тәуелсіз кеңістіктер болып табылатын объектілер кластары: объектілер, функциялар, typedef-аттары және enum-константалары; Нұсқаулық белгілері; құрылым, бірлестік және тізімдеу тегтері; әрбір жеке құрылым немесе бірлестік элементтері.

Тұжырымдалған ережелер бірінші басылымда сипатталған бұрынғы ережелерден біршама ерекшеленеді. Нұсқаулық белгілері өз кеңістігінен бұрын болмаған; құрылым тегтері мен бірлестіктер тегтері (ал кейбір іске асыруларда және тізім тегтері) жеке кеңістіктерге ие болды. Бір ортақ кеңістікте құрылым, бірлестіктер және тізімдер тегтерін орналастыру-бұл бұрын болмаған қосымша шектеу. Бірінші редакциядан неғұрлым маңызды ауытқу-әрбір жеке құрылым (немесе бірлестік) өз элементтері үшін өзінің Жеке атаулар кеңістігін құруда.

Осылайша, сол атау бірнеше түрлі құрылымдарда пайдаланылуы мүмкін. Бұл ереже бірнеше жыл бойы кеңінен қолданылады.

Сыртқы хабарландыруда жарияланған объект идентификаторының (немесе функцияның) көрінуінің лексикалық аймағы оның хабарлаушысы аяқталатын жерден басталады және ол жарияланған

трансляция бірлігінің соңына дейін созылады. Функцияны анықтаудағы параметрдің көріну аймағы функция денесін білдіретін блоктың басынан басталады және барлық функцияға таралады; функцияны сипаттауда параметрдің көріну аймағы осы сипаттаманың соңында аяқталады. Блоктың басында жарияланған идентификатордың көріну аймағы оның хабарландырушысы аяқталатын жерден басталады және осы блоктың соңына дейін жалғасады. Белгінің көріну аймағы — бұл белгі табылған барлық функция. Құрылым, біріктіру немесе тізімдеу тегінің көріну аймағы оның түрі спецификаторында пайда болуынан басталады және сыртқы деңгейді жариялау үшін трансляция бірлігінің соңына дейін және функцияның ішінде хабарландыру үшін блоктың соңына дейін жалғасады.

Егер идентификатор блоктың басында анық жарияланса (функцияның денесін қоса алғанда), осы блоктың сыртында орналасқан сол идентификатордың кез келген хабарландыруы блоктың соңына дейін уақытша әрекет етпейді.

А 11.2. Байланыстар

Егер бірдей идентификаторы бар және объектіні (немесе функцияны) сипаттайтын бірнеше хабарландыру болса, онда сыртқы байланыс жағдайында барлық осы жарнамалар бүкіл бағдарлама үшін бірегей бір объектіге (функцияға) жатады; егер ішкі байланыс болса, бірегейліктің қасиеті тек трансляция бірлігіне ғана таралады.

А10. 2-де айтылғандай, егер бірінші сыртқы хабарландыруда static спецификаторы бар болса, онда ол ішкі байланыс идентификаторын сипаттайды, егер мұндай спецификатор болмаса, онда — сыртқы байланыс. Егер хабарландыру блоктың ішінде болса және extern болмаса, онда тиісті идентификатор осы функция үшін еш нәрсемен байланысты емес және бірегей. Егер хабарландыруда extern және блок осы сәйкестендіргіштің сыртқы хабарландыруының көріну аймағында болса, онда соңғысы бірдей байланысқа ие және сол объектіге (функцияға) қатысты болады. Алайда, егер бұл идентификатор үшін ешқандай сыртқы хабарландыру болмаса, онда ол сыртқы байланысқа ие.

А 12. Алдын ала өңдеу

Препроцессор макро-қалпына келтіруді, шартты компиляцияны, аталған файлдарды қосуды орындайды. # белгісінен басталатын жолдар (оның алдында бөлгіш таңбалар болуы мүмкін), препроцессормен байланыс орнатады. Олардың синтаксисі тілдің қалған бөлігіне байланысты емес; олар кез келген жерде пайда болуы және трансляцияланатын бірліктің соңына дейін әсер етуі мүмкін. Жолдардың шекаралары назарға алынады; әрбір жол Жеке талданады (алайда жолдарды "желімдеу" мүмкіндігі бар, А12.2 қараңыз). Мысалы, #include (А12.4) директивасында файл аттарын беретін тілдің барлық лексемалары және символдар реттілігі препроцессор үшін лексемдер болып табылады. Сонымен қатар, кез келген басқа тәсілмен анықталмаған кез келген символ лексема ретінде қабылданады.

Бос орындар мен көлденең табуляциядан ерекшеленетін бөлгіш символдардың әсері препроцессор жолдарының ішінде анықталмаған.

Алдын ала өңдеу өзі бірнеше логикалық тізбекті фазаларда өтеді. Жекелеген іске асыруларда кейбір фазалар біріктірілген.

1. А12. 1-де сипатталған үш таңбалы бірізділік олардың баламаларымен ауыстырылады. Амалдық жүйе қажет болса, жолдар арасында жаңа жолдың таңбалары енгізіледі.

2. Келесі жаңа жолдың символымен кері көлбеу сызықтан тұратын символдардың жұптары шығарылады; сол арқылы (А12.2) жолдарды "желімдеу" жүзеге асырылады.

3. Бағдарлама бөлгіш-символдармен бөлінген лексемаларға бөлінеді. Пікірлер бір бос орындармен ауыстырылады. Содан кейін препроцессор мен макро-қалпына келтіру директивалары орындалады (А 12.3-12.10).

4. Символдық константалар мен жолдық литералдардағы Эскейп-реттілік (А 2.5.2, 2.6) олар белгілейтін символдарға ауыстырылады. Көрші жол литералдары конкатенирленеді.

5. Нәтиже таратылады. Содан кейін қажетті бағдарламалар мен

деректерді жинау және сыртқы функциялар мен объектілерге сілтемелерді олардың анықтамаларымен қосу арқылы басқа бағдарламалармен және кітапханалармен байланыс орнатылады.

А 12.1. Үш таңбалы реттілік

Бастапқы С бағдарламалары жазылатын таңбалар жиынтығы жеті биттік ASCII кодына негізделген. Алайда, бұл ISO 646-1983 инвариантты коды жиынтығына қарағанда кеңірек. Қысқартылған таңбалар жиынтығын пайдалану үшін төмендегі барлық үш таңбалы тізбектер сәйкес келетін жеке таңбалармен ауыстырылады. Ауыстыру кез келген басқа өңдеуден бұрын жүзеге асырылады.

```
??=   #          ??(   [          ??<   {
??/   \          ??)   ]          ??>   }
??'   ^          ??!   |          ??-   ~
```

Көрсетілгендерден басқа ешқандай ауыстыру жасалмайды.

Үш таңбалы тізбектер ANSI стандартымен енгізілген.

А 12.2. Жолдарды біріктіру

Кері көлбеу сызықпен аяқталатын жол келесімен қосылады, өйткені \ символы және одан кейінгі жаңа жолдың символы шығарылады. Бұл мәтінді лексемаға «бөлу» алдында жасалады.

А 12.3. Макро анықтау және макро кеңейту

Түрдің басқару жолы

```
# define лексем тізбегінің  идентификаторы
```

препроцессорды сәйкестендіргішті идентификатор тізбегіне ауыстыруға мәжбүр етеді; таңбалауыш тізбегінің басында және соңында бөлгіш таңбалар алынып тасталады. Бірдей идентификаторы бар қайталанатын # қате, егер таңбалауыштардың тізбегі бірдей

болмаса, қате деп саналады (салыстыру кезінде бөлгіш белгілердегі сәйкессіздіктер ескерілмейді). Көру жолы

```
# define идентификатор( идентификатор тізімі ) лексем тізімі
```

мұнда бірінші идентификатор мен белгінің арасында (бір де бір бөлгіш таңба болмауы тиіс, идентификаторлар тізімімен берілетін параметрлермен макроанықтама болып табылады. Бірінші нұсқадағы сияқты, бөлгіш таңбалар лексаның басында және соңында шығарылады, және макрос тек сол параметрлер тізімімен және сол тізбектілігімен қайта анықталуы мүмкін. Түрдің басқару жолы

```
# undef идентификатор
```

идентификаторға берілген анықтаманы «ұмытып» деп белгілейді. Белгісіз идентификаторға #undef қолдану қате деп саналмайды.

Егер макроанықтама екінші тәсілмен берілсе, онда оның идентификаторынан тұратын мәтіндік дәйектілік, мүмкін, келесі бөлгіш символдармен-белгімен, белгімен (үтірмен бөлінген лексемалардың тізімімен және белгімен) макростың шақыруын білдіреді. Макростың шақыруының дәлелдері үтірмен бөлінген (үтір, «жабық» тырнақшалармен немесе ішкі жақшалармен, аргументтердің бөлінуіне қатыспайды) лексемалар болып табылады. Оларды бөлу кезінде аргументтер макроширленуге ұшырамайды. Макрос шақыруында аргументтер саны макротанықтау параметрлерінің санына сәйкес келуі керек. Аргументтерді таңдағаннан кейін, оларды қоршаған таңбалар-бөлгіштер лақтырылады. Содан кейін макростың лексемасының алмастырушы жүйелігінде идентификаторлар-параметрлер (егер олар жойылмаса) оларға сәйкес аргументтерге ауыстырылады. Егер ауыстыратын ретпен параметр алдында # белгісі болмаса және одан кейін # белгісі болмаса, онда Аргументтің лексемалары тексеріледі: олар макровузов бар ма, егер бар болса, онда аргумент ауыстырылмас бұрын, оған сәйкес макросширлеу жүргізіледі.

Қалпына келтіру процесіне екі арнайы оператор әсер етеді. Біріншісі – параметр алдында орнатылған # операторы. Ол параметрдің

орнына және # (оның алдында) белгісі қос тырнақшаға салынуын талап етеді. Бұл ретте, әрбір қос тырнақшаның алдында жол литерларында және символдық константаларында « (соның ішінде жиектелген жолдар), сондай-ақ әрбір кері көлбеу сызықтың алдында \ кірістіріледі \.

Екінші оператор ## ретінде жазылады. Егер лексемдердің жүйелілігі макроанықтаудың кез келген түрлерінде ## операторы болса, онда параметрлерді қойғаннан кейін ол оны қоршаған символдармен бірге тастайды, соның арқасында көрші лексемалар «желімделеді», осылайша жаңа лексеманы қалыптастырады. Нәтиже дұрыс емес лексеманы алған кезде немесе генерацияланатын мәтін ## операторларын қолдану тәртібіне байланысты болған кезде анықталмады. Сонымен қатар, ## басында да, соңында да тұра алмайды.

Екі түрдің макростарында лексаның алмастырушы тізбегі жаңа define-атауларды табу үшін қайта қаралады. Дегенмен, егер кейбір идентификатор осы кеңейтуде ауыстырылса, мұндай идентификатордың қайта пайда болуы оны алмастырмайды.

Егер алынған кеңейтім # белгісінен басталса, ол препроцессор директивасы ретінде қабылданбайды.

ANSI-стандартта макросширлеу процесі кітаптың бірінші басылымына қарағанда дәл сипатталған. Ең маңызды өзгерістер # және # # операторларын енгізуге қатысты. Кейбір жаңа ережелер, әсіресе конкатенацияға қатысты бірнеше қызықты көрінуі мүмкін. (Төмендегі мысалдарды қараңыз.)

Сипатталған мүмкіндіктерді тұрақтылардың мағыналық мәнін көрсету үшін қолданыла алады, мысалы,

```
#define TABSIZE 100
int table[TABSIZE];
#define ABSDIFF(a, b) ((a)>(b) ? (a)-(b) : (b)-(a))
```

макрос оның аргументтерінің айырмашылығының абсолюттік мәнін қайтарады. Бірдей функциядан айырмашылығы, дәлелдер мен қайтарылатын мән мұнда кез келген арифметикалық түр болуы мүмкін және тіпті көрсеткіштер болуы мүмкін. Сонымен қатар, әрқайсысы жанама әсер етуі мүмкін дәлелдер екі рет есептеледі: бір рет – тексеру кезінде, екінші рет – нәтижені есептеу кезінде.

Егер

```
#define tempfile(dir) #dir «/%s»
```

анықтамасы болса,

```
tempfile(/usr/tmp) макросы
```

«/usr/tmp» «/%s»

нәтижесін береді.

Әрі қарай бұл екі жол бір сызыққа айналады. Макро бойынша

```
#define cat(x, y) x ## y
```

cat(var, 123) шақыруы var123 құрады. Алайда, Cat (Cat (1, 2), 3) қалаған мүмкіндік бермейді, өйткені # операторы cat сыртқы шақыруы үшін дұрыс аргументтерді алуға кедергі жасайды. Нәтижесінде келесі тізбек лексем беріледі:

```
cat (1 , 2 )3
```

онда) 3 (екінші аргументтің бірінші лексемасының соңғы лексемасының «біріктіру» нәтижесі) дұрыс лексема емес.

Егер келесі макроанықтама түрі мынадай болса:

```
#define xcat(x.y) cat(x.y)
```

мұнда ешқандай коллизия болмайды; xcat(xcat(1, 2), 3) нәтижес 123 болады, себебі xcat # операторын қолданбайды.

ABSDIFF(ABSDIFF(a, b), c) де іске қосылады, солай біз дұрыс нәтижені аламыз.

А 12.4. Файлды қосу

Басқару жолы

```
# include <файл атауы>
```

Файл атауы бар файл мазмұнына ауыстырылады. Аты-файлды құрайтын таңбалардың арасында > белгісі және жаңа жолдың символы болмауы керек. Егер файл атында»,', \ немесе бірнеше /* таңба бар болса, нәтиже анықталмады. Көрсетілген файлды іздеу тәртібі іске асырылуға байланысты.

Сол сияқты басқару жолы орындалады

```
# include « файл атауы «
```

Бірінші іздеу компилятор бастапқы файлды іздейтін ережелер бойынша жүзеге асырылады (бұл іздеу механизмі іске асырылуға байланысты), ал сәтсіз болған жағдайда бірінші типті #include қабылданған іздеу әдісімен жүзеге асырылады. Егер файл атында «, \ немесе / * болса, нәтиже белгісіз болып қалады; > таңбаны пайдалану рұқсат етіледі.

Соңында, директив

```
# include лексем тізбегі
```

алдыңғы формалардың ешқайсысына сәйкес келмейтін лексеманың дәйектілігін барлық макропотүсірілімдердің нәтижесінде #include <беруі керек мәтін ретінде қарастырады...> немесе #include «...». Осылайша жасалған директива бұдан әрі алынған нысанға сәйкес түсіндірілетін болады.

#Include арқылы салынған файлдар өздері #include директиваларын қамтуы мүмкін.

А 12.5. Шартты компиляция

Бағдарламаның бөліктері шартты түрде құрастырылуы мүмкін, егер олар келесі схемалық бейнеленген синтаксиске сәйкес ресімделсе:

условная-конструкция-препроцессора: if-строка текст elif-части else-частьнеоб #endif

```
if-жолы: #
        if тұрақты өрнек # ifdef идентификатор # ifndef идентификатор

elif-бөлігі:
        elif-мәтін жолы elif-бөлігі

elif-жолы: #
        elif тұрақты өрнек

else-бөлігі:
        else- мәтін жолы

else-строка: #
         else
```

Директивалардың әрқайсысы (if-жол, elif-жол, else-жол және #endif) жеке жолға жазылады. #If және келесі жолдардағы тұрақты өрнектер нөлдік емес (шынайы) мәні бар өрнек анықталғанша ретретімен есептеледі; нөлдік мәні бар жолдан кейінгі мәтін шығарылады. Нөл емес мәндегі директиваның артында орналасқан мәтін әдеттегі түрде өңделеді. «Мәтін» сөзінің астында мұнда кез келген жолдар тізбегі, соның ішінде шартты құрылымның бөлігі болып табылмайтын препроцессор жолдары; Мәтін бос болуы мүмкін. Егер #if немесе #elif жолы өрнектің нөл емес мәнімен табылған және оның мәтіні өңделсе, келесі #elif және #else жолдары мәтіндермен бірге шығарылады. Егер барлық өрнектер нөлдік мәнге ие болса және #else жолы болса, онда келесі мәтін әдеттегі түрде өңделеді. Шартты конструкциялардың» белсенді емес « тармақтарының мәтіндері еленбейді.

#If және #elif тұрақты өрнектер әдеттегі макро қалпына келтіру үшін объектілер болып табылады. Сонымен қатар, көрініс өрнектерін көрмес бұрын

```
defined идентификаторы
```

және

```
defined (идентификаторы)
```

оларда макровызовтың болуы үшін, олар 1L немесе 0l-ге ауыстырылады, себебі ол идентификаторда көрсетілген прецессормен анықталмады. Ақырында, кез келген бүкіл тұрақты әрқашан l жұрнағы бар деп болжанады, яғни барлық арифметика тек ұзын немесе unsigned long түрі операндами ісі.

Тұрақты өрнек (7.19) мұнда шектеулермен қолданылады: ол бүтін болуы керек, онда тізімделген константаларды, түрді және sizeof операторларын қамтуы мүмкін емес.

Басқару жолдары

```
#ifdef идентификаторы
#ifndef идентификаторы
```

эквивалентті

```
# if defined идентификаторы
# if ! defined идентификаторы
```

#elif жолдары тілдің бірінші нұсқасында болмады, бірақ ол кейбір препроцессорларда пайдаланылды. Defined prepressor операторы – жаңа.

А 12.6. Жолдардың нөмірленуі

С-бағдарламаны генерациялайтын басқа препроцессорлармен жұмыс істеуге ыңғайлы болу үшін келесі директивалардың бірін пайдалануға болады:

```
# line тұрақтысы «файл аты»
# line тұрақтысы
```

Бұл директивалар компиляторға көрсетілген ондық бүтін және идентификатор келесі жолдың нөмірі және ағымдағы файлдың атауы деп саналады. Егер файл атауы жоқ болса, бұрын есте қалған атау өзгермейді. #Line директивасында макровызовтарды кеңейту соңғысы түсіндірілгенге дейін орындалады.

А 12.7. Қате туралы хабарды жариялау

Түрдің пропроцессор жолы

```
# error лексем тізбегі
```

оған лексеманың берілген жүйелілігін қамтитын диагностикалық хабарлама беруді бұйырады.

А 12.8. Прагма

Түрдің басқару жолы

```
# pragma лексем тізбегі
```

пропроцессорды іске асыруға байланысты әрекеттерді орындауға шақырады.

Танылмаған прагма елемейді.

А 12.9. Бос директива

Түрдің пропроцессор жолы

#

ешқандай іс-қимыл тудырмайды.

А 12.10. Алдын ала анықталған атаулар

Алдын ала анықталған бірнеше идентификаторларды «түсінеді»; ол оларды арнайы ақпаратпен ауыстырады. Бұл идентификаторлар (және defined propressor операторы, соның ішінде) қайта анықтауға болмайды, оларға #undef директивасын қолдануға болмайды. Бұл келесі идентификаторлар:

LINE Бастапқы мәтіннің ағымдағы жолының нөмірі, ондық тұрақты.

FILE Компиляцияланатын файлдың аты, жол.

DATE Компиляция күні «Ммм дд гггг» түріндегі, жол.

TIME «чч: мм: ее» түріндегі компиляция уақыты, жол.

STDC Тұрақты 1. Бұл идентификатор стандартты ұстанатын іске асыруларда 1 ретінде анықталған.

#Error және #pragma жолдары алғаш рет ANSI стандартымен енгізілді. Алдын
ала анықталған пропроцессор макростары әлі күнге дейін сипатталмаған, бірақ
кейбір іске асыруларда пайдаланылды мүмкін.

А 13. Грамматика

Төменде осы қосымшада қарастырылған грамматикалық ережелер бар. Олардың мазмұны бірдей, бірақ басқа тәртіппен берілген.

Мұнда келесі символдар-терминдердің анықтамасы келтірілмейді: бүтін-константа, символды тұрақты, с-өзгермелі-нүкте тұрақты-

сы, идентификатор, жол және тізімдеу тұрақтысы. Кәдімгі латын қарпімен (курсив емес) терілген сөздер және белгілер терминдер ретінде қарастырылады және қалай жазылғандай түрде пайдаланылады. Бұл грамматиканы грамматикалық тануды автоматты түрде генерациялау жүйесіне түсінікті мәтінге механикалық түрлендіруге болады. Бұл үшін баламалы өнімдерді көрсетуге арналған кейбір синтаксистік белгілерді қосудан басқа, құрылымның «бірі» деген сөздермен шифрын ашу және необ индексі бар символды пайдаланатын әрбір өнімнің қайталануы қажет, өнімнің бір нұсқасы осы символмен жазылуы керек, ал екіншісі-онсыз. Бір өзгеріспен, атап айтқанда typedef – атауын алып тастау-атауы: идентификатор және typedef-атауын атау символы-терминмен жариялау, бұл грамматика УАСС грамматикалық тану генераторына түсінікті болады. Ол if-else дизайнының екіталай еместігінен туындаған бір ғана қарама-қайшы.

```
трансляция бірлігі:
        сыртқы-хабарландыру
        трансляция бірлігі сыртқы-хабарландыру

сыртқы-хабарландыру:
        функцияны анықтау
        хабарландыру

функцияны анықтау:
        спецификаторлар-хабарландыру
        хабарландырулар тізімі құрылым нұсқаулығы

хабарландыру
        спецификаторлар-бастамашы-хабарландырушылардың тізімі

хабарландырулар тізімі:
        хабарландыру
        хабарландырулар тізімі хабарландыру

спецификаторлар-хабарландырулар:
        спецификатор-класс-жад спецификаторлар-хабарландыру
        спецификатор-спецификатор-хабарландыру

жады спецификаторы осылардың біреуі:
        auto register static extern typedef
```

тип спецификатор осылардың біреуі:
void char short int long float double signed unsigned
құрылым немесе біріктіру спецификаторы
есептеу спецификаторы
typedef-аты

квалификатор типі осылардың біреуі:
const volatile

құрылым немесе біріктіру спецификаторы:
құрылым немесе біріктіру идентификаторы { құрылым хабарландыруының тізімі }
құрылым немесе біріктіру идентификаторы

құрылым немесе біріктіру осылардың біреуі:
struct union

құрылым хабарландыруының тізімі:
құрылым хабарландыруы
құрылым хабарландыруының тізімі құрылым хабарландыруы

инициализатор хабарландыруының тізімі:
инициализатор хабарландыруы
инициализатор хабарландыруының тізімі, инициализатор хабарландыруы

инициализатор хабарлаушы:
хабарлаушы
хабарлаушы = инициализатор

құрылым хабарландыру:
спецификаторлар тізімі-квалификаторлар тізімі-объявителей-стрқұрылым хабарлаушылары

спецификаторлар мен квалификаторлар тізімі :
спецификатор типті тізім-спецификатор-квалификатор типті
спецификаторлар тізімі-квалификаторлар

хабарландыру құрылымының тізімі:
хабарландыру құрылымы
құрылым тізімі-хабарлаушы, хабарлаушы құрылымы

хабарлаушы құрылымы:
хабарлаушы
хабарлаушы : тұрақты өрнек

есептеу спецификаторы:
enum идентификаторы{ есептеу тізімі }
enum идентификаторы

есептеу тізімі:
есептеу
есептеу тізімі есептеу

есептеу:
идентификатор
жеке хабарландырушының көрсеткіші

жеке хабарландырушы:
идентификатор
(хабарландырушы)
жеке хабарландырушы [тұрақты өрнек]
жеке хабарландырушы (тізім-түрі-параметрлері)
жеке хабарландырушы (идентификаторлар тізімі)

көрсеткіш:
* квалификатор типті тізім
* квалификатор типті тізім көрсеткіш

квалификатор типті тізім:
квалификатор типті
квалификатор типті тізім квалификатор типті

параметрлер типті тізім:
параметрлер тізімі
параметрлер тізімі, ...

параметрлер тізімі:
параметрлер хабарландыруы
параметрлер тізімі, параметрлер хабарландыруы

параметрлер хабарландыруы:
спецификаторлар-хабарландырулар хабарландырушы
спецификаторлар-хабарландырулар абстрактілі-хабарландыру

идентификаторлар тізімі:
идентификатор
идентификаторлар тізімі, идентификатор

инициализатор:
меншіктеу өрнегі

{ инициализаторлар тізімі }
{ инициализаторлар тізімі, }

инициализаторлар тізімі:
инициализатор
инициализаторлар тізімі, инициализатор

тип аты:
абстрактілі-хабарландырушы-мамандануышылар тізімі

абстрактілі-хабарландырушы:
көрсеткіш
жеке көрсеткіш - абстрактілі-хабарландырушы

жеке - абстрактілі-хабарландырушы:
(абстрактілі-хабарландырушы)
жеке - абстрактілі-хабарландырушы [тұрақты өрнек]
жеке - абстрактілі-хабарландырушы (параметр типті тізім)

typedef-аты:
идентификатор

нұсқаулық:
белгіленген нұсқаулық
өрнек нұсқаулығы
құрама нұсқаулық
таңдау нұсқаулығы
циклдік нұсқаулық
ауысу нұсқаулығы

белгіленген нұсқаулық:
идентификатор : нұсқаулық
case тұрақты өрнек: нұсқаулық
default : нұсқаулық

өрнек нұсқаулығы:
өрнек ;

құрама нұсқаулық:
{ хабарландырулар тізімі нұсқаулықтар тізімі }

нұсқаулық тізімі:
нұсқаулық
нұсқаулықтар тізімі нұсқаулық

таңдау нұсқаулығы:
if (өрнек) нұсқаулық:
if (өрнек) нұсқаулық: else нұсқаулық:
switch (өрнек) нұсқаулық:

циклдік нұсқаулық:
while (өрнек) нұсқаулық:
do нұсқаулық: while (өрнек)
for (өрнек ; өрнек ; өрнек) нұсқаулық:

ауысу нұсқаулығы:
goto идентификатор ;
continue ;
break ;
return өрнек ;

өрнек:
меншіктеу өрнек
өрнек , меншіктеу өрнек

меншіктеу өрнек:
шартты өрнек
унарлы өрнек меншіктеу операторы меншіктеу өрнегі

меншіктеу операторы:
*= /= %= += -= <<= >>= &= ^= |=

шартты өрнек:
логикалық НЕМЕСЕ өрнегі
логикалық НЕМЕСЕ өрнегі? өрнек : шартты өрнек

тұрақты өрнек:
шартты өрнек

логикалық НЕМЕСЕ өрнегі:
логикалық ЖӘНЕ өрнегі
логикалық НЕМЕСЕ өрнегі || логикалық ЖӘНЕ өрнегі

логическое-И-выражение:
НЕМЕСЕ өрнегі
логикалық ЖӘНЕ өрнегі && НЕМЕСЕ өрнегі

НЕМЕСЕ өрнегі:
қосылмайтын НЕМЕСЕ өрнегі
НЕМЕСЕ өрнегі | қосылмайтын НЕМЕСЕ өрнегі

қосылмайтын НЕМЕСЕ өрнегі:
ЖӘНЕ өрнегі
қосылмайтын НЕМЕСЕ өрнегі ^ ЖӘНЕ өрнегі

ЖӘНЕ өрнегі:
теңдік өрнегі
ЖӘНЕ өрнегі & теңдік өрнегі

теңдік өрнегі:
қатынастар өрнегі
теңдік өрнегі == қатынастар өрнегі
теңдік өрнегі!= қатынастар өрнегі

қатынастар өрнегі:
ығысу өрнегі
қатынастар өрнегі < ығысу өрнегі
қатынастар өрнегі > ығысу өрнегі
қатынастар өрнегі <= ығысу өрнегі
қатынастар өрнегі >= ығысу өрнегі

ығысу өрнегі:
аддитивті өрнек
ығысу өрнегі >> аддитивті өрнек
ығысу өрнегі << аддитивті өрнек

аддитивті өрнек:
мультипликативті өрнек
аддитивті өрнек + мультипликативті өрнек
аддитивті өрнек - мультипликативті өрнек

мультипликативті өрнек:
типке келтірілген өрнек
мультипликативті өрнек * типке келтірілген өрнек
мультипликативті өрнек / типке келтірілген өрнек
мультипликативті өрнек % типке келтірілген өрнек

типке келтірілген өрнек:
унарлы өрнек
(аты, типі) типке келтірілген өрнек

унарлы өрнек:
постфиксті өрнек
++ унарлы өрнек
-- унарлы өрнек
унарлы оператор типке келтірілген өрнек

```
        sizeof унарное-выражение
        sizeof ( имя-типа )

унарлы оператор:
        &  *  +  -  ~  !

постфиксті өрнек:
        бастапқы өрнек
        постфиксті өрнек [ өрнек ]
        постфиксті өрнек ( өрнек аргументтерінің тізімі )
        постфиксті өрнек. идентификатор
        постфиксті өрнек -> идентификатор
        постфиксті өрнек ++
        постфиксті өрнек --

бастапқы өрнек:
        идентификатор
        тұрақты
        жол
        ( өрнек )

өрнек аргументтерінің тізімі:
        меншіктеу өрнегі
        өрнек аргументтерінің тізімі, меншіктеу өрнегі

тұрақты:
        бүтін тұрақты
        символдік тұрақты
        қалқымалы нүктелі тұрақты
        есептеу тұрақтысы
```

Төменде басқару жолдары құрылымының тізімі түрінде препроцессор тілінің грамматикасы келтірілген. Грамматикалық талдау бағдарламасын механикалық алу үшін ол жарамсыз. Грамматика әдеттегі бағдарламаның мәтінін, прпроцессордың сөзсіз басқару жолдарын және оның аяқталған шартты құрылымдарын білдіретін мәтін символын қамтиды.

```
Басқарушы жол:
        # define идентификатор тізбегі-лексем
        # define идентификатор ( идентификатор ... идентификатор )
                тізбек-лексем
        # undef идентификатор
```

```
        # include <аты-файл>
        # include « аты-файл «
        # include тізбегі-лексем
        # line константа «идентификатор»
        # line константа
        # error идентификатор тізбегі
        # pragma лексем тізбегі
        #
        шартты-конструкция-препроцессор

шартты-конструкция-препроцессор:
        if-мәтін жолы elif-бөлігі else-бөлігі # endif

if-жолы:
        # if тұрақты өрнек
        # ifdef идентификатор
        # ifndef идентификатор

elif-бөлігі:
        elif- мәтін жолы
        elif- бөлігі

elif-жолы:
        # elif тұрақты өрнек

else- бөлігі:
        else- мәтін жолы

else-жолы:
        # else
```

В стандартты кітапхана

Осы қосымша ANSI стандарт ретінде бекітілген кітапхананың қысқаша мазмұны болып табылады. Кітапхана өзі тілдің бір бөлігі болып табылмайды, бірақ онда құрылған функциялар жиынтығы, сондай-ақ макростардың типтері мен анықтамалары Си стандартын қолдайтын жүйелік ортаны құрайды. Біз мұнда шектеулі қолдану саласы бар бірнеше функцияны келтірмейміз – басқа функциялардан оңай синтезделген, сондай-ақ тіл, ұлттық ерекшеліктер мен мәдениетке байланысты көпбайттық символдар мен ерекшеліктерге қатысты барлық нәрселерді төмендетеміз.

Функциялар, түрлері және макростар келесі стандартты тақырып файлдарында жарияланады:

```
<assert.h>  <float.h>   <math.h>    <stdarg.h>  <stdlib.h>
<ctype.h>   <limits.h>  <setjmp.h>  <stddef.h>  <string.h>
<errno.h>   <locale.h>  <signal.h>  <stdio.h>   <time.h>
```

Тақырып файлына кіру пропроцессор жолы арқылы жүзеге асырылады:

```
#include <тақырып файлы>
```

Тақырып файлдарын кез келген рет және кез келген рет қосуға болады. Жол #include тиіс ішінде сыртқы хабарландыру немесе анықтау тиіс кездесіп, бұрын ештеңе из енгізілетін басты файл сұранысқа ие болады. Нақты іске асыруда тақырып файлы бастапқы файл болмауы мүмкін.

Астын сызу белгісінен басталатын сыртқы идентификаторлар, сондай-ақ астын сызу белгісінен немесе астын сызу мен бас әріппен басталатын барлық басқа идентификаторлар кітапханада пайдалану үшін сақталған.

В 1. Кіріс-шығыс: <stdio.h>

<Stdio.h> анықталған енгізу-шығару функциялары, типтері және макрослары кітапхананың шамамен үштен бірін құрайды.

Ағын – бұл деректердің көзі немесе баратын орны; оны дискіге немесе басқа сыртқы құрылғымен байланыстыруға болады. Кітапхана екі түрлі ағымды қолдайды: мәтіндік және екілік, бірақ кейбір жүйелерде, атап айтқанда UNIXе, олар бір-бірінен ерекшеленбейді. Мәтіндік ағын – бұл жолдар тізбегі; әр жол нөл немесе одан көп таңбадан тұрады және '\ n' аяқталады. Жұмыс ортасы мәтін ағынын түзетуді қажет етуі мүмкін (мысалы, тасымалдауды қайтаруға және жол берілісіне '\ n').

Екілік ағын дегеніміз – өзгермейтін байттардың тізбегі, бұл кейбір аралық мәліметтерге ие, егер олар жазылып, сол кіріс-шығыс жүйесінде оқылса, біз түпнұсқаға сәйкес келетін ақпаратты аламыз.

Ағын файлға немесе құрылғыға оны ашу арқылы қосылады, көрсетілген байланыс ағынды жабу арқылы бұзылады. Файлды ашу осы ағымды басқаруға қажетті барлық ақпаратты қамтитын FILE нысанын көрсеткішке қайтарады. Егер түсініксіздік туындамаса, біз «файл сілтемесі» және «ағын» терминдерін бір-бірімен алмастырамыз.

Бағдарлама басталған кезде үш ағын ашылады: stdin, stdout және stderr.

В 1.1. Файл операциялары

Төменде файлдармен жұмыс істейтін функциялар берілген. Size_t түрі-sizeof операторының нәтижесін сипаттау үшін пайдаланылатын толық бүтін түрі.

```
FILE *fopen(const char *filename, const char *mode)
```

fopen берілген аты бар файлды ашады және ашу әрекеті сәтсіз болса, ағынды немесе NULL қайтарады. Mode режимі келесі мәндерге жол береді:

“r” – мәтіндік файл оқу үшін ашылады (read (ағылш.) – оқу);

“w” – мәтіндік файл жазу үшін жасалады; ескі мазмұн (егер ол болса) шығарылады (write (ағылш.)- жазу);

“a” – мәтіндік файл файлдың соңына жазу үшін ашылады немесе құрылады (append (ағылш.)- қосу);

“r +” – мәтіндік файл түзету үшін ашылады (яғни оқу және жазу үшін);

“w +” – мәтіндік файл түзету үшін жасалады; ескі мазмұн (егер ол болса) шығарылады;

“a +” – мәтіндік файл бұрыннан бар ақпаратты түзету және файлдың соңына жаңасын қосу үшін ашылады немесе құрылады.

“Түзету” режимі бір файлға оқуға және жазуға мүмкіндік береді; оқу операцияларынан жазба операцияларына және кері қарай ауысқанда fflush немесе файлды орналастыру функциясына жүгіну керек. Егер режим көрсеткіші b (мысалы, “rb” немесе “w+b”) әрпімен толықтырылса, онда бұл екілік файл. Файл атының ұзындығына шектеу filename_max тұрақты. Тұрақты FOPEN_MAX бір уақытта ашық файлдар санын шектейді.

FILE *freopen(const char *filename, const char *mode, FILE *stream)

fileopen көрсетілген режиммен файлды ашады және оны stream ағынымен

байланыстырады. Ол stream қайтарады немесе қате болған жағдайда, NULL. Әдетте

freopen stdin, stdout немесе stderr, басқа файлдармен байланысты файлдарды

ауыстыру үшін қолданылады.

int fflush(FILE *stream)

Шығыс ағынына қолданылатын fflush функциясы буферде қалған барлық

деректердің (әлі жазылмаған) мөлшерін жазады; кіріс ағыны үшін бұл функция

анықталмаған. Қате немесе нөл жазғанда пайда болған жағдайда EOF қайтарады.

fflush (NULL) түрін өңдеу барлық шығыс ағындары үшін көрсетілген операцияларды орындайды.

int fclose(FILE *stream)

fclose әлі жазылмаған буферлік деректерді қосады, оқылмаған буферлі кірісті тазартады, автоматты түрде сұралған барлық буферлерді шығарады, содан кейін ағымды жабады. EOF қате туралы және нөлге қайтарады.

int remove(const char *filename)

жойылған файлды көрсетілген атпен алып тастайды; келесі атаумен файлды ашуға арналған келесі әрекеті қатеге әкеледі. Сәтсіз әрекетке нөлді қайтарады.

int rename(const char *oldname, const char *newname)

rename файл атауын ауыстырады; атын өзгерту әрекеті сәтсіз болған жағдайда нөлдік емес мәнді қайтарады. Бірінші параметр ескі атау, ал екіншісі жаңа.

FILE *tmpfile(void)

tmpfile “WB+” кіру режимі бар уақытша файлды жасайды, ол жабылғанда немесе бағдарлама өз жұмысын қалыпты аяқтағанда автоматты түрде жойылады. Бұл функция ағынды қайтарады немесе файл жасай алмаса, NULL.

char *tmpnam(char s[L_tmpnam])

tmpnam (NULL) бар файлдардың аттарының ешқайсысына сәйкес келмейтін жолды жасайды және көрсеткішті ішкі статикалық массивке қайтарады.

tmpnam (s) S жолында жолды есте сақтайды және оны функцияның мәні ретінде қайтарады; s ұзындығы l_tmpnam кем болмауы керек. Әрбір tmpnam шақыруында жаңа атау пайда болады; бұл ретте бағдарлама жұмысының бір сеансы үшін

TMP_MMMHAH артық емес әр түрлі атауларға кепілдік беріледі. Tmpnam файлды емес, атауды жасайды.

int setvbuf(FILE «stream, char *buf, int mode, size_t size)

setvbuf ағынның буферленуін басқарады; Оқу, жазу немесе басқа операция
орындалмас бұрын оған жүгіну керек. _iofbf мәнімен mode толық буферлеуді
тудырады, _IOLBF — мәтіндік файлдың “жолдан кейінгі” буферлеуін, _ionbf
мәнімен a mode кез келген буферлеуді болдырмайды. Егер параметр buf жоқ
NULL, онда оның мәні-буферге көрсеткіш, әйтпесе буферге жад сұралады. Size
параметрі буфердің өлшемін анықтайды. Setvbuf функциясы қате болған жағдайда нөлдік емес мән береді.

void setbuf(FILE *stream, char *buf)

Егер buf NULL болса, онда stream ағыны үшін буферизация өшіріледі. Бұл
жағдайда setbuf шақыру (void) setbuf (stream, buf, _IOFBF, BUFSIZ) сияқты
әрекеттерге әкеледі.

В 1.2 Форматтық қорытынды

Printf функциялары формат бойынша ақпаратты шығарады.

int fprintf(FILE *stream, const char *format, ...)

fprintf format басқарылатын stream ағынын түрлендіреді және жазады.
Қайтарылатын мән-жазылған таңбалар саны немесе қате болған жағдайда
теріс мән береді.

Формат жолында екі түрлі нысандар бар: Шығыс ағынына көшірілетін қарапайым таңбалар және басқа дәлелдерді түрлендіруге және басып шығаруға әкелетін түрлендіру ерекшеліктері. Әрбір түрлендірудің ерекшелігі %-дан басталады және түрлендірудің символ-спецификаторымен аяқталады. % және спецификатор сим-

волы арасында олар осында тізімделген тәртіпте ақпараттың келесі элементтері орналасуы мүмкін:

▪ Тулар (кез келген тәртіппен):

- келесі шығару нысандарының бірін көрсетеді: о үшін бірінші цифр 0 болуы тиіс; X немесе X нөл емес нәтиженің алдында 0X немесе 0X болуы тиіс; E, E, F, G және G үшін қорытындыда міндетті түрде ондық нүкте болуы тиіс; G және G үшін аяқтаушы нөлдер шығарылмайды.

- - түрлендірілген аргумент өрістің сол жақ шетіне басу керек екенін көрсетеді;

+ - санды әрдайым белгімен басып шығаруды ұйғарады;

бос орын - егер бірінші таңба-белгі болмаса, онда санның алдында бос орын болуы тиіс;

0-сандар өрістің барлық еніне дейін сол жақтағы нөлдермен толықтырылуы керек екенін көрсетеді;

▪ Өрістің ең аз енін сипаттайтын сан. Түрлендірілген дәлел өлшемі көрсетілген еннен кем емес өріске басылады, ал қажет болса, үлкен өріске басылады. Түрлендірілген аргументтің таңбалар саны өрістің енінен аз болса, өріс солға (немесе солға басылса, оңға) толықтырылады. Әдетте өріс бос орындармен толықтырылады (немесе нөлдермен толықтыру туы болса, нөлдермен).

▪ Өріс енінің көрсеткішін дәлдік көрсеткішінен бөліп тұратын нүкте.

▪ Жолдан басылатын символдардың ең көп санын немесе e, e немесе f түрлендірулеріндегі ондық нүктеден кейін сандар санын немесе g немесе G-түрлендіруге арналған маңызды сандар санын немесе бүтін басып шығару кезінде сандардың ең аз санын (өрістің қажетті еніне дейін сан сол жақтағы нөлдермен толықтырылады) ерекшелейтін дәлдік беретін сан.

▪ h, l (ell әрпі) немесе L. “h” тиісті дәлел short немесе unsigned short ретінде басылуы тиіс екенін көрсетеді; “l” дәлел long немесе unsigned long түрі бар екенін хабарлайды; “L” дәлел long double түріне тиесілі екенін хабарлайды.

Ені, немесе дәлдігі, немесе осы сипаттамалардың екеуі де * көмегімен ерекшеленуі мүмкін; бұл жағдайда қажетті сан int түрі болуы тиіс келесі аргументтен “алынады” (екі жұлдызша жағдайда екі дәлел қолданылады).

Арнайы таңбалар және олардың мағынасын түсіндіру В-1 кестесінде келтірілген. % Үшін дұрыс спецификатор белгісі болмаса, нәтиже анықталмады.

Таңба	**Аргументтің түрі; басып шығару түрі**
d, i	int; таңбалы ондық жазба
o	int; таңбасыз сегіздік жазба (сол жақта 0 жоқ)
x, X	unsigned int; таңбасыз он алтылық жазу (0x немесе 0x сол жақта жоқ), 10-нан 15-ке дейінгі сандар ретінде x және X үшін ABCDEF үшін abcdef қолданылады
u	int; таңбасыз ондық жазба
c	int; unsigned char-ға түрлендіруден кейінгі бірлік таңба
s	char *; жол таңбалары '\0' кездескенше басылады немесе дәлме-дәл көрсетілген таңбалар саны таусылмайды
f	double; ондық көрініс жазбасы *-]mmm.ddd саны дәлдікпен ерекшеленетін d. Әдепкі дәлдік 6-ға тең; нөлдік дәлдік ондық нүктені басып шығаруды басады
e, E	қос; * -] m.dddddde ± xx сияқты ондық белгілеу немесе * -] m.ddddddE ± xx сияқты белгілеу, мұндағы d саны дәл келтірілген. Әдепкі дәлдік - 6; нөлдік дәлдік ондық бөлшектерді басып шығаруды болдырмайды
g, G	қос; % e және% E пайдаланылады, егер тапсырыс -4-тен кем болса немесе дәлдікке тең немесе оған тең болса; әйтпесе% f пайдаланылады. Нөлдер мен кейінгі кезеңдер басып шығарылмайды
p	жарамсыз *; көрсеткіш ретінде басып шығарады (ұсыну іске асыруға байланысты)
n	int *; осы қоңырауға дейін басып шығарылған таңбалар саны дәлелге жазылады. Басқа дәлелдер өзгертілмейді
%	ешқандай дәлелдер түрлендірілмейді; % басылған

int printf(const char *format, ...)

printf(...) эквалиентті fprintf(stdout,...).

int sprintf(char *s, const char *format, ...)
sprintf printf сияқты жұмыс істейді, тек s жолына шығатыннан басқа, '\ 0' дегеннен басқа. S жолы шығысты қамту үшін үлкен болуы керек. '\ 0' таңбасын қоспағанда, жазылған таңбалардың санын береді.

int vprintf (const char *format, va_list arg)
int vfprintf (FILE *stream, const char *format, va_list arg)
int vsprintf (char *s, const char *format, va_list arg)

Vprintf, vfprintf және vsprintf функциялары тиісті printf функцияларына тең, аргументтердің айнымалы тізімі va_start макросымен инициалданған arg параметрімен және va_arg қоңырауларымен (B7 сипаттамасынан қараңыз <stdarg. h>).

B 1.3. Форматтық енгізу

Scanf функциялары енгізу кезінде форматты түрлендірумен айналысады.

int fscanf(FILE *stream, const char *format, ...)

scanf format басқаруындағы stream ағынынан деректерді оқиды және түрлендірілген шамалар рет-ретімен дәлелдер береді, олардың әрқайсысы көрсеткіш болуы тиіс. Егер формат таусылса, жұмысты аяқтайды. EOF файл таусылғанда немесе қате орын алса, кез келген түрлендірудің алдында береді;
басқа жағдайларда функция түрлендірілген және енгізілген элементтердің санын қайтарады.

Формат жолында әдетте енгізуді басқару үшін пайдаланылатын түрлендіру ерекшеліктері бар. Пішімді жолға кіруі мүмкін:

- ескерілмейтін бос орындар мен табуляциялар;
- қарапайым таңбалар (% басқа), олар бөлгіш таңбалардан өзгеше таңбалар арасында енгізу ағынында күтеді;
- %-дан тұратын түрлендіру спецификациясы; меншіктелген * белгіден міндетті емес белгісі; өрістің ең жоғары енін ерекшелейтін

міндетті емес саны; берілетін мәннің мөлшерін көрсететін міндетті емес h, l немесе L және түрлендіру символының спецификаторы.

Түрлендіру ерекшелігі келесі енгізу өрісін түрлендіруді анықтайды. Әдетте нәтиже тиісті аргументті көрсететін айнымалыға орналастырылады. Дегенмен, Егер меншіктеу мысалы, % * s сияқты*белгісінің көмегімен басылады, онда енгізу өрісі жай өтіп, ешқандай тағайындау болмайды. Енгізу өрісі таңбалар бөлгіштерден айырмашылығы бар таңбалар жолы ретінде анықталады; бұл ретте жолды енгізу екі шарттың кез келгені орындалғанда тоқтатылады: егер символ-бөлгіш болса немесе өрістің ені (ол көрсетілген жағдайда) таусылса. Осыдан келесі өріске өту кезінде scanf жол шекарасы арқылы “аттап” алады, себебі жаңа жолдың символы-бөлгіш болып табылады. (Бөлгіш таңбалар деп бос орын, табуляция, жаңа жол, каретканы қайтару, тік табуляция және бетті ауыстыру символдары түсініледі.)

Таңба-спецификатор енгізу өрісін түсіндіру әдісін көрсетеді. Тиісті дәлел көрсеткіш болуы тиіс. Рұқсат етілген таңба-спецификаторлар тізімі В-2 кестесінде келтірілген.

D, i, n, o, u және x символ-спецификатормен, егер аргумент short (int емес) немесе l (әрпі l) көрсеткіш болса, егер аргумент long көрсеткіш болса, h алдында болуы мүмкін. E, f және g спецификатор таңбалары l-дан бұрын болуы мүмкін, егер аргумент жұпқа (өзгермелі емес) көрсеткіші болса, немесе L - егер дәлел ұзақ жұпқа сілтегіш болса.

Таңба	**Мәліметтерді енгізу; аргумент түрі**
d	ондық бүтін; int *
i	бүтін; int *. Бүтін сегіздік (сол жақта нөлдік) немесе он алтылық (сол жақта 0x немесе 0X)
o	сегіз бүтін (сол жақтан нөлмен немесе онсыз); int *
u	таңбасыз ондық бүтін; unsigned int *
x	он алтылық бүтін (0X немесе 0X сол жақта немесе олар-сыз); int *

c	таңбалар; char *. Енгізу символдары өрістің енімен берілген санда көрсетілген массивке орналастырылады; әдепкі бойынша бұл сан 1-ге тең. '\0' белгісі қосылмайды. Мұнда бөлгіш таңбалар әдеттегі таңбалар ретінде қарастырылады және дәлелге түседі. Келесі бөлгіш таңбаны оқу үшін %1s қолданыңыз
s	бөлгіш таңбалардан өзгеше таңбалар жолы (тырнақшасыз жазылады); char*, жолды сыйдыру үшін жеткілікті Өлшем массивін және оған қосылатын '\0' таңбасын көрсететін
e, f, g	жылжымалы нүкте саны; float *. Float үшін енгізу пішімі міндетті емес белгіден, сандар жолынан, ондық нүктеден болуы мүмкін және E немесе e және бүтіннен тұратын міндетті емес тәртіптен тұрады, мүмкін белгімен
p	көрсеткіш printf ("%p") басып шығаратын; void *
n	осы шақыруда оқыған таңбалар санын аргументке жазады; int *. Енгізу оқылмайды. Енгізілген элементтер санының санауышы арттырылмайды
[…]	кірістен тік жақшадағы таңбалардан тұратын ең ұзын бос емес жолды таңдайды; char *. Жолдың соңына '\0' қосылады. Спецификатордың []...] түрі таңдалған таңбалар жиынтығына кіреді
[^…]	енгізуден ең ұзын бос емес жолды таңдайды, ол жақшаның ішінде көп емес таңбалардан тұрады. Соңында '\0' қосылады. Түрі спецификаторы [^]...] таңдалған таңбалар жиынтығына кіреді
%	қарапайым % символы; меншіктеу жасалмайды

int scanf (const char *format, . . .)

scanf (...) тура fscanf (stdin, ...) орындаған әрекеттерді жасайды.

int sscanf (const char *s, const char *format, ...)

sscanf (s, ...) тура scanf (...)орындаған әрекеттерді жасайды, бірақ, енгізу s жолынан басталады.

B 1.4. Таңбаларды енгізу-шығару функциялары

int fgetc(FILE *stream)

fgetc stream ағынынан келесі таңбаны unsigned char (int аударылған) немесе

EOF файл таусылған немесе қате анықталған жағдайда қайтарады.

char *fgets(char *s, int n, FILE *stream)

fgets s массивінде n-1 таңбадан артық емес оқиды, егер массивке қосылатын
жаңа жолдың символы болса; сонымен қатар '\0' массивіне жазылады. Fgets
функциясы s қайтарады немесе файл таусылса немесе қате анықталса, NULL.

int fputc(int c, FILE *stream)

fputc c таңбасын жазады (unsigned char аударылған) stream. Қате болған жағдайда жазылған таңбаны немесе EOF қайтарады.

int fputs(const char *s, FILE *stream)

fputs stream-ге s жолын жазады (Ол '\n' болмауы мүмкін); қате болған жағдайда
теріс емес бүтін немесе EOF қайтарады

int getc(FILE *stream)

getc fgetc сияқты жасайды, бірақ соңғы айырмашылығы, егер ол макрос болса,
stream бір реттен артық болуы мүмкін.

int getchar(void)

getchar() getc(stdin) жасағанды да жасайды.

char *gets(char *s)

gets s массивіне келесі енгізу жолын оқиды, жаңа жолдың таңбасын '\0' деп
ауыстырады. S қайтарады немесе файл таусылған немесе қате анықталған
болса, NULL.

int putc(int c, FILE *stream

putc fputc сияқты жасайды, бірақ соңғы айырмашылығы putc-макрос болса,
stream мәні бір реттен артық болуы мүмкін.

int putchar(int c)

putchar(c) putc (c, stdout) сияқты жасайды.

int puts(const char *s)

puts s жолын және stdout жаңа жолдың белгісін жазады. Қате болған жағдайда
EOF қайтарады немесе жазба қалыпты болса теріс емес мәнді қайтарады.

int ungetc(int c, FILE *stream)

ungetc c таңбасын (unsigned char аударылған) stream-ға жібереді; келесі stream-
нан оқу кезінде ол қайтадан алынады. Әрбір ағын үшін бір таңбадан артық емес
қайтаруға болады. EOF қайтаруға болмайды. Нәтиже ретінде ungetc кері
жіберілген таңбаны немесе қате болған жағдайда EOF береді.

В 1.5. Тікелей енгізу-шығару функциялары

size_t fread(void *ptr, size_t size, size_t nobj, FILE *strem)

fread stream ағынынан ptr массивіне nobj өлшемінен артық емес size
нысандарын оқиды. Ол оқылған нысандардың санын қайтарады, ол
мәлімделгеннен аз болуы мүмкін. Оқылғаннан кейін күй индикациясы үшін
feof және ferror пайдаланыңыз.

size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE «stream)

fwrite PTR массивінен stream nobj өлшеміндегі нысандарды жазады; қате
болған жағдайда nobj-ден аз жазылған нысандардың санын қайтарады.

В 1.6. Файлды орналастыру функциялары

int fseek(FILE *stream, long offset, int origin)

fseek stream үшін позицияны орнатады; кейінгі оқу немесе жазу осы позициядан
жасалады. Бинарлық файл жағдайында позиция offset — ығысумен белгіленеді-

егер origin SEEK_SET-ке тең болса; егер origin SEEK_CUR-ге тең болса,
ағымдағы позицияға қатысты; және origin SEEK_END-ке тең болса, файлдың
соңына қатысты. Offset мәтіндік файлы үшін нөлдік немесе ftell функциясына
қоңырау шалу арқылы алынған мән болуы керек. Мәтін файлымен жұмыс
істеген кезде origin әрқашан SEEK_SET-ке тең болуы керек.

long ftell(FILE *stream)

ftell ағынының ағымдағы орнын немесе-1L, қате болған жағдайда қайтарады.

void rewind(FILE *stream)

rewind(fp) fseek(fp, 0L, SEEK_SET); clearerr (fp) сияқты жасайды.

int fgetpos(FILE *stream, fpos_t *ptr)

fgetpos ағынының ағымдағы орнын келесі fsetpos-да пайдалану үшін *PTR-ге
жазады. Fpos_t түрі осы мәндерді сақтауға мүмкіндік береді. Қате болған
жағдайда fgetpos нөл емес мәнді қайтарады.

int fsetpos(FILE *stream, const fpos_t *ptr)

fsetpos stream позициясын орнатады,
оны оқу *ptr, ол бұрын fgetpos арқылы жазылған. Қате болған жағдайда fgetpos
нөл емес мәнді қайтарады.

В 1.7. Қателерді өңдеу функциялары

Файлдың соңында немесе қатесінде кітапхананың көптеген функциялары күй индикаторларын орнатады. Бұл көрсеткіштерді тексеруге және өзгертуге болады. Сонымен қатар, бүтін өрнек errno (жарияланған <errno.h>) табылған қателердің соңғы туралы қосымша ақпарат беретін қате нөмірін қамтуы мүмкін.

void clearerr(FILE *stream)

clearerr файлдың соңғы көрсеткіштерін және stream ағынының қателерін

тазартады.

int feof(FILE *stream)

feof файлдың соңғы индикаторы орнатылған болса, нөл емес мәнді қайтарады.

int ferror(FILE *stream)

ferror stream ағыны үшін қате индикаторы орнатылған болса, нөлдік емес мәнді
қайтарады.

void perror(const char *s)

perror(s) s-ны басып шығарады және errno-да мақсатты мәнге сәйкес келетін
қате туралы хабар, яғни, fprintf функциясына жүгіну сияқты әрекеттерді орындайды
fprintf(stderr, “%s: %s\n”, s, “ қате туралы хабарлама”)
strerror туралы В3 тақырыбынан қараңыз.

B 2. Таңбаның класын тексеру: <ctype.h>

Тақырып файлы <ctype.h> таңбаларды тексеруге арналған функцияларды жариялайды. Олардың әрқайсысының дәлелінде int түрі бар және EOF немесе int-ға келтірілген unsigned char мәні болуы керек; қайтарылатын мән де int түрі бар. Функциялар c аргументі сипатталған шартты қанағаттандыратын немесе көрсетілген таңбалар класына жататын болса, нөл емес мәнді (“ақиқат”) қайтарады.

isalnum(c) — isalpha (c) немесе isdigit (c) шындық бар
isalpha(c) — isupper(c) немесе islower (c) шындық бар
iscntrl (c) — басқарушы таңба
isdigit (c) — ондық сан
isgraph (c) — бос орындан басқа басып шығарылатын таңба
islower (c) — төменгі регистр әрпі
isprint (c) — бос орынды қоса басып шығарылатын таңба
ispunct(c) — бос орын, әріп немесе саннан басқа басып шығарылатын таңба
isspace(c) — бос орын, бетті ауыстыру, жаңа жол, каретканы қайтару, табуляция, тік табуляция

isupper — c) - жоғарғы регистр әрпі
isxdigit(c) — он алтылық сан

Жиынтығында жетібитті ASCII-таңбалар басып шығарылатын таңбалар 0x20 (‘) бастап Ox7E (‘ — ‘); басқару таңбалар — 0 (nul) бастап 0x1f (us) және Ox7F (del).

Функционалдық жолдың екі еселенгенінен басқа, оның бір бөлігі

int tolower(int c) — төменгі регистрге аударады;
int toupper(int c) — жоғары регистрге аударады.

Егер c — әріп жоғарғы регистрде болса, онда topper(c) бұл әріпті төменгі регистрде береді; әйтпесе ол c-ны қайтарады. егер c-әріп төменгі регистрде болса, онда toupper(C) бұл әріпті жоғарғы регистрде береді; олай болмаған жағдайда ол c-ны қайтарады.

B 3. Жолдармен жұмыс жасайтын функциялар: <string. h>

Екі топ функциялары бар. Олар тақырып файлында анықталған <string.h>. Бірінші топ функцияларының аттары str әріптерінен, екіншісі – mem-ден басталады. Егер көшіру ісі бар объектілермен, перекрывающимися бойынша жад, онда қоспағанда, memmove, мінез-құлық функциялары анықталмаған. Салыстыру функциялары дәлелдерді unsigned char типті элементтердің массиві ретінде қарастырады.

B 3 кестесінде s және t айнымалылары char *, cs және ct — const char *, n — size_t түріне, ал c — char түріне келтірілген int түрінің мәні.

char *strcpy (s, ct) ct жолын s жолына көшіреді, соның ішінде ‘\0’; slmgr қайтарады

char *strncpy(s, ct, n) CT жолының n таңбасын s-ге көшіреді; қайтарады s. CT белгісі n-ден аз болса, нәтижені ‘\0’ таңбаларымен толықтырады

char *strcat(s, ct) ce s-ге жатқызады; s қайтарады

char *strncat(s, ct, n) s '\0' таңбасын аяқтай отырып, ct N таңбасын s-ге жатқызады; slmgr қайтарады

char strcmp(cs, st) cs және ct салыстырады; CS <ct, 0, егер cs=ct, және >0, cs>ct15 болса, <0 қайтарады

char strncmp(cs, ct) cs және ct N таңбаларын салыстырады; CS <ct, 0, егер cs=ct, және >0, егер cs>ct

char *strchr(cs, скөрсеткіш cs-те бірінші кіру үшін қайтарады немесе, ол NULL болмаса

char *strrchr(cs, c) CS-да соңғы кіру үшін меңзерді қайтарады немесе, ол NULL болмаса

size_t strspn(cs, ct) ct жолына кіретін таңбалардан тұратын cs бастапқы сегментінің ұзындығын қайтарады

size_t strcspn(cs, ct) ct жолына кірмейтін таңбалардан тұратын cs бастапқы сегментінің ұзындығын қайтарады

char * strpbrk(CS, ct) ct -ға кіретін таңбалардың біріне сәйкес келетін бірінші таңбаға cs көрсеткішін қайтарады, немесе ол NULL болмаса

char *strstr (cs, ct) ct -ның бірінші кіруіне немесе, ол болмаса, NULL қайтарады

size_t strlen (cs) cs ұзындығын қайтарады

char * strerror (n) n қате нөміріне сәйкес келетін іске асыруға байланысты жолға көрсеткішті қайтарады

char * strtok (s, ct) ct таңбаларымен шектелген s лексеманы іздейді; Бұл функцияның толық сипаттамасы төменде қараңыз

Strtok-қа жүйелі қоңыраулар жолдарды таңбаларға бөледі. Белгілеу шектегіші – ct жолындағы кез келген таңба. Бірінші қоңырауда s NULL емес. Функция ct-ге кірмеген таңбалардан тұратын бірінші таңбаны s жолынан табады; ол келесі таңбаның үстіне жазылып, таңбалауышқа қайтарылған '\ 0' аяқталады. Әрбір келесі қоңырау, онда NULL болады, сілтегіш келесі таңбалауышқа қайтарылады, функция алдыңғы жолдың соңынан кейін бірден іздейді. Strtok

функциясы NULL мәнін қайтарады, егер басқа таңбалауыш табылмаса. Ct параметрі қоңыраудан қоңырауға дейін өзгеруі мүмкін.

mem... функциялары объектілерді таңбалар массиві сияқты басқаруға арналған; олардың мақсаты тиімді бағдарламаларға интерфейстермен қамтамасыз ету. Төмендегі кестеде s және t - жарамсыз *; cs және ct - const void * типі; n - мөлшері_t; және c - бұл char-ке арналған.

void *memcpy(s, ct, n) ct-ден s-ге n таңбаларды көшіреді және S қайтарады

void *memmave(s, ct, n) memcpy сияқты жасайды, бірақ» жабылатын « Нысандар жағдайында да жұмыс істейді.

int memcmp (cs, ct, n) cs және ct таңбаларының алғашқы n салыстырады; strcmp функциясы сияқты нәтижені береді

void *memchr (cs, c, n) CS-дағы таңбаның бірінші кірісіне көрсеткішті қайтарады немесе, егер бірінші N таңбалар арасында NULL кездеспесе

void *memset (s, C, n) c таңбасын s жолының бірінші n позицияларында орналастырады және S қайтарады

B 4. Математикалық функциялар: <math. h>

Тақырып файлында <math.h> математикалық функциялар сипатталады және макростар анықталады.

EDOM және ERANGE макростары (<errno.h>-да орналасқан) аймақ қателерін және ауқымдағы қателерді бекіту үшін пайдаланылатын тұтас константаларды нөлден ерекшелендіреді; HUGE_VAL double түрінің оң мәні ретінде анықталады. Егер дәлел функция анықталған мәндер аймағынан тыс болса, аймақ қатесі пайда болады. Аймақ қатесін белгілеу errno мәндерін беру арқылы жүзеге асырылады; қайтарылатын мән іске асырылуға байланысты. Функция нәтижесі double түрінде ұсынылуы мүмкін болмаған кезде ауқым қатесі пайда болады. Толып кеткен жағдайда функция қайтарады.

Дұрыс белгісі бар HUGE_VAL және errno-да ERANGE мәні орнатылады. Егер нәтиже осы түрге қарағанда аз болса, функция нөл қайтарады, ал бұл жағдайда errno ERANGE-де орнатыла ма, іске

асыруға байланысты. Әрі қарай x және y double түрі, n — int түрі бар және барлық функциялар double түрінің мәнін қайтарады. Тригонометриялық функциялардың бұрыштары радиандарда беріледі.

sin(x) синус x
cos(x) косинус x
tan(x) тангенс x
asin(x) арксинус x диапазонында *-□/2, □/2], x□[-1, 1]
acos(x) арккосинус x диапазонында *0, □], x□[-1, 1]
atan(x) арктангенс x диапазонда *-□/2, □/2]
atan2(y, x) арктангенс y/x x диапазонда *-□, □]
sinh(x) гиперболалық синус x
cosh(x) гиперболалық косинус x
tanh(x) гиперболалық тангенс x
exp(x) экспоненциалды функция ex
log(x) қарапайым логарифм ln(x), x > 0
log10(x) ондық логарифм log10(x), x > 0
pow(x, y) xy. Аймақтық қате, егер x = 0 және y < 0 немесе x < 0 және y —бүтін болмаса.
sqrt(x) x квадраттың түбірі, x > 0
ceil(x) double түріндегі ең аз бүтін, x-тан көп емес
floor(x) double түріндегі ң үлкен бүтін, x-тан көп емес
fabs(x) абсолюттік мәні |x|
ldexp(x, n) x * 2n
frexp(x, int *exp) x екі топқа бөлінеді, оның біріншісі - *1/2, 1 аралығында нормаланған бөлшек), ол қайтарылады, ал екіншісі-екілік дәрежесі, бұл дәреже * exp-да есте қалады . Егер x — нөл болса, нәтиженің екі бөлігі нөлге тең.
modf(x, double *ip) тұтас және бөлшек бөліктерге бөлінеді, екеуінің де x сияқты белгісі бар.
fmod(x, y) өзгермелі нүктесі бар сан түрінде x-ға бөлуден қалған қалдық. Егер y нөлге тең болса, нәтиже іске асыруға байланысты.

B 5. Жалпы мақсаттағы функциялар: <stdlib. h>

Тақырып файлы <stdlib.h> сандарды, жад сұрауын және басқа тапсырмаларды түрлендіруге арналған функцияларды жариялайды.

double atof(const char *s)

atof s-ны double-ға аударады; эквивалиентті strtod(s, (char**) NULL).

int atoi(const char *s)

s-ны int-ке аударады; эквивалиентті (int) strtol (s, (char**)NULL, 10).

int atol(const char *s)

s-ны long-қа аударады ; эквивалиентті strtol(s, (char**) NULL, 10).

double strtod(const char *s, char **endp)

strtod жетекші бөлгіштерді елемей, жолдың алғашқы таңбаларын екі есеге
түрлендіреді; * endp-де өзгермейтін соңына сілтегішті есіне алады (егер endp
NULL болмаса), толып кету жағдайында HUGE_VAL анық таныс болады, егер нәтиже деректер түрі көрсететіннен аз болса, 0 қайтарылады; екі жағдайда да ERANGE қате орнатылған.

long strtol(const char *s, char **endp, int base)

strtol s жолдарының бірінші таңбаларын ұзындыққа түрлендіреді, жетекші бөлгіштерді елемейді; * endp-дегі өзгермейтін соңына меңзерды есіне алады
(егер endp NULL болмаса). Егер база 2-ден 36-ға дейінгі диапазонда болса,
онда кіру базалық базада жазылған сан болып табылады. Егер база нөлге тең
болса, онда санның негізі 8, 10 немесе 16 болады; 0-ден басталатын сан сегіздік, 0x немесе 0X он алтылық деп саналады. 10-дан бастап 1-ге дейінгі
цифрлар кез-келген жағдайда латын әліпбиінің алғашқы әріптерімен жазылады. Егер радиус 16 болса, санның басында 0x немесе 0X орналастыруға болады. Толып кету жағдайында функция LONG_MAX немесе LONG_MIN (белгіге байланысты) қайтарады және қате ERANGE күйіне орнатылады.

unsigned long strtoul(const char *s, char **endp, int base)

strtoul strtol сияқты жұмыс істейді, тек айырмашылығы - қол қойылмаған ұзақ нәтиже береді, ал толып кеткен жағдайда ол ULONG_MAX қайтарады.

int rand(void)
rand жалған кездейсоқ санды 0-ден RAND_MAX-ге дейін шығарады; RAND_MAX - кем дегенде 32767.

void srand(unsigned int seed)

srand жалған кездейсоқ сандардың жаңа тізбегі үшін тұқым ретінде тұқым қолданады. Бастапқыда тұқым параметрі 1-ге тең.

void *calloc(size_t nobj, size_t size)

calloc сілтегішті nobj нысандарының массиві үшін бөлінген жад кеңістігіне қайтарады, өлшемнің әрқайсысы, немесе егер сұралған сома болмаса, NULL. Бөлінген жад аумағы тазаланады.

void *malloc(size_t size)
malloc сілтегішті өлшемі бар объект үшін жад орнына қайтарады немесе сұралған өлшем болмаса, NULL. Бөлінген жад аймағы іске қосылмаған.

void *realloc(void *p, size_t size)

realloc өлшемін б-мен көрсетілген нысанның өлшемімен алмастырады. Өз
кезегінде, оның мөлшері ескі және жаңа өлшемдердің ең кішісіне тең,
мазмұны өзгермейді. Егер жаңа өлшем ескіден үлкен болса, қосымша орын
іске қосылмаған, қайта орналастыру көрсеткішті жаңа жад орнына
қайтарады, немесе талаптар орындалмаса NULL (* өзгермейді).

void free(void *p)

p-мен көрсетілген жадтың бос орнын босатады; p функциясы NULL болса,
бұл функция ештеңе істемейді. P-де функциялардан бұрын бөлінген жад
аймағына сілтегіш болуы керек: calloc, malloc немесе realloc.

void abort(void *p)

abort бағдарламаның авариялық аяқталуын тудырады, оның әрекеттері
raiise(SIGABRT) шақыруына тең.

void exit(int status)

exit бағдарламаның қалыпты аяқталуын тудырады. Atexit арқылы
тіркелген функциялар оларды тіркеуге кері ретпен орындалады. Ашық
файлдар буферлері босатылады, ашық ағындар жабылады және басқару
бағдарлама іске қосылған ортаға қайтарылады. Мәні status берілетін
сәрсенбі, байланысты жүзеге асыру, алайда, табысты аяқталғаны

бағдарламасы қабылданды беруге нөл. Сондай-ақ, EXIT_SUCCESS (сәтті
аяқталған жағдайда) және EXIT_FAILURE (қате болған жағдайда)
мәндерін пайдалануға болады.

int atexit(void (*fcn)(void))

atexit FCN-ді бағдарлама қалыпты аяқталған кезде туындайтын функция ретінде тіркейді; егер тіркеу орындалмаса, нөл емес мәнді қайтарады.

int system(const char *s)

system s жолын операциялық ортаға орындау үшін жібереді. Егер s NULL болса және командалық процессор болса, жүйе нөл емес мәнді қайтарады.
Егер s null болмаса, онда қайтарылатын мән іске асыруға байланысты.

char *getenv(const char *name)

getenv name байланысты орта жолын қайтарады немесе ешқандай жол болмаса, NULL. Бөлшектер жүзеге асыруға байланысты.

void *bsearch(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void *keyval, const void *datum))

b search base арасында іздейді...base [n-1] сәйкес кілті бар элемент *key. Cmp функциясы бірінші аргументті (іздеу кілтін) өзінің екінші аргументімен (кестедегі кілттің мәнімен) салыстыруы және салыстыру нәтижесіне байланысты теріс сан, нөл немесе оң мән. Base массив элементтері өсу ретімен реттелуі тиіс, bsearch сәйкес кілті элементке сілтегіш қайтарады немесе егер ол болмаса, NULL.

void qsort(void *base, size_t n, size_t size, int (*cmp)(const void *, const void *))

qsort base массивін сұрыптайды...base [n-1] size өлшемінің нысандары өсу
тәртібімен. Cmp салыстыру функциясы bsearch сияқты.

int abs(int n)

abs int түріндегі Аргументтің абсолютті мәнін қайтарады.

long labs(long n)

abs ұзын типті аргументтің абсолютті мәнін қайтарады.

div_t div(int num, int denom)

div numer-дің denom-ге бөлінуінің жеке және қалдықты есептейді. Int түріндегі нәтижелер « div_t құрылымының rem элементтерінде сақталады.

ldiv_t ldiv(long num, long denom)

div numer-дің denom-ге бөлінуінің жеке және қалдықты есептейді. Long типті нәтижелер « және ldiv_t құрылымының rem элементтерінде есте қалады.

B 6. Диагностика: <assert. h>

Макрос assert диагностикалық хабарлар бағдарламасына қосу үшін пайдаланылады.

void assert (int өрнек)
Егер өрнек мәні нөл болса, онда

assert (өрнек)

келесі түрдегі stderr хабарламасын басып шығарады:

Assertion failed: өрнек, file файл аты, line nnn

содан кейін есептеуді аяқтайтын abort функциясы пайда болады. Бастапқы файлдың аты мен жол нөмірі макростардан __FILE және__ LINE _алынады.

Егер файлды қосу кезінде <assert.h> NDEBUG атауы анықталды, онда макрос assert елемейді.

B 7. Ұзындығы айнымалы аргументтер тізімі: <stdarg.h>

Тақырып файлы <stdafx.h> саны мен түрлері алдын ала белгісіз аргументтерді таңдау үшін құралдар береді.

Кейінг-аргументтердің айнымалы саны бар f функциясының соңғы атаулы параметрі болсын. F ішінде va_list типті айнымалы ар жарияланады, көрсеткішті келесі аргументке сақтауға арналған:

```
va_list ap;
```

Атаусыз дәлелдерге қол жеткізу мүмкін болмас бұрын, va_start макросына жүгіну арқылы ар бір рет инициалдануы керек:

```
va_start(va_list ap, посларг);
```

Осы сәттен бастап әрбір макросқа жүгіну:

```
типі va_arg(va_list ap, типі);
```

келесі аты-жөні жоқ аргументтің мәнін береді және әрбір осындай өтініш келесі аргументті көрсету үшін ар көрсеткішінің автоматты түрде өсуін шақырады. Аргументтерді ауыстырғаннан кейін бір рет, бірақ f-дан шыққанға дейін макросқа жүгіну керек.

```
void va_end(va_list ap);
```

B 8. Ұзақ өтулер: <setjmp. h>

Хабарландыру <setjmp.h> әдеттегі «шақыру — қайтару» ретті лігінен бас тартуға жол береді; типтік жағдай - «терең салынған» шақырудан жоғарғы деңгейге оралу қажеттілігі.

```
int setjmp(jmp_buf env)
Макрос setjmp env-де ағымдағы қоңырау туралы ақпаратты оны
longjmp-де пайдалану үшін сақтайды . Егер қайтару setjmp — дан
тікелей жүзеге асырылса, және longjmp-дан кейінгі шақырудан болса,
нөл қайтарады. Setjmp-ге тек белгілі бір контекстерде ғана жүгіну
мүмкін; негізінен бұл if, switch және циклдарда тексерулер, және тек
қарапайым қарым-қатынас өрнектерінде.

if (setjmp() == 0)
/* тікелей қайтарылғаннан кейін */
else
```

/* longjmp қайтару кейін */
void longjmp(jmp_buf env, int val)

longjmp env ақпараты бойынша setjmp ең соңғы қоңырауында сақталған ақпаратты қалпына келтіреді; setjmp функциясы тек жұмыс істеген және Val нөл емес мәнін қайтарған сияқты бағдарламаны орындау жаңартылады. Егер longjmp-ге жүгінген кезде setjmp шақыруы бар функция «жұмыс істеп» және қайтаруды жүзеге асырса, нәтиже күтпеген болады. Оған қол жетімді Нысандар longjmp-ге жүгінген кезде болған мәндерге ие; setjmp мәндерді сақтамайды.

B 9. Сигналдар: <signal. h>

Тақырып файлы <signal.h > бағдарламаны орындау кезінде пайда болатын ерекше жағдайларды өңдеу үшін сыртқы көздер немесе есептеулердегі қате тудыратын үзіліс сияқты құралдарды ұсынады.

void (*signal(int sig, void (*handler)(int)))(int)

signal келесі сигналдарды қалай өңдейтінін анықтайды. Егер handler параметрі SIG_DFL мәніне ие болса, онда «Әдепкі өңдеу» қолданысына тәуелді қолданылады; егер handler мәні SIG_IGN тең болса, онда сигнал елемейді; қалған жағдайларда сигнал түрі үшін handler көрсететін функцияға жүгіну орындалады. Сигналдардың рұқсат етілген түрлері:

SIGABRT-апаттық аяқтау, мысалы, abort;
SIGFPE-арифметикалық қате: 0 немесе толып кету;
SIGILL-қате функция коды (жарамсыз команда);
SIGINT-үзу сияқты өзара әрекеттесу сұрауы;
SIGSEGV-шетелге шығу сияқты жады дұрыс емес;
SIGTERM-бағдарламаға жіберілген аяқтау талабы.
signal арнайы сигнал болған жағдайда алдыңғы handler мәнін немесе қате болған жағдайда SIG ERR қайтарады.

Одан кейін sig сигналы пайда болғанда, алдымен «әдепкі» мінез-құлықтың дайындығы қалпына келтіріледі, содан кейін handler параметрінде көрсетілген функция, яғни қоңырау (*handler) (sig) қалай орындалады. Егер handler функциясы кері басқаруды қайтарса, онда есептеулер бағдарламаға келген сигнал қойылған жерден басталады.

Сигналдардың бастапқы жағдайы іске асыруға байланысты.
int raise(int sig)

raise бағдарламаға sig сигналын жібереді. Сәтсіз болған жағдайда нөлдік емес қайтарады.

В 10. Күн мен уақыт функциялары: <time.h>

Тақырып файлы <time.h> күн мен уақытқа байланысты түрлері мен функцияларын жариялайды. Кейбір функциялар жергілікті уақытпен байланысты, мысалы, уақытты бөлу сияқты күнтізбеден өзгеше болуы мүмкін. Clock_t және time_t түрлері уақытты көрсету үшін арифметикалық түрлер, ал struct tm күнтізбелік уақыт компоненттерін қамтиды:

```
int tm_sec; - минут басынан секунд (0,61);
int tm_min; - сағат басынан минут (0,59);
int tm_hour; - түн ортасынан сағат (0,23);
int tmjnday; - айдың күні (1,31);
int tmjnon; - қаңтардан бастап айлар (0,11);
int tm_year; - 1900 жылдан бері;
int tm_wday; - жексенбі күндері (0,6);
int tm_yday; - 1 қаңтардан бастап күндер (0,365);
int tm_isdst; - жаз мезгілінің белгісі.
```

Tm_isdst мәні оң, егер уақыт тәулік уақыты 1 сағатқа алға жылжытылатын мезгілге түссе, нөл болады, ал ақпарат қол жетімді болмаса теріс болады.

clock_t clock(void)

clock бағдарламаның орындалуынан бастап процессор белгілеген уақытты қайтарады, немесе-1, егер ол белгісіз болса. Бұл уақытты секундта көрсету үшін clock()/CLOCKS_PER_SEC формуласы қолданылады.

time_t time(time_t *tp)

уақыт белгісіз болса, time ағымдағы күнтізбелік уақытты немесе -1 қайтарады. Егер tp NULL-ге тең болмаса, онда қайтарылатын мән *tp-ге жазылады.

double difftime(time_t time2, time_t timel)

difftime time2-time1 секундта көрсетілген айырмашылықты қайтарады.

time_t mktime(struct tm «tp)

mktime *tp құрылымымен берілген жергілікті уақытты күнтізбеге айналдырады, оны time функциясы сияқты түрде береді. Компоненттер көрсетілген ауқымдарда маңызды болады. Mktime функциясы көрінбесе, күнтізбелік уақытты немесе -1 қайтарады.

Келесі төрт функция көрсеткіштерді статикалық нысандарға қайтарады, олардың әрқайсысы басқа қоңыраулармен өзгертілуі мүмкін.

char *asctime(const struct tь *tp)

asctime *tp құрылымындағы уақытты түр жолына аударады

Sun Jan 3 15:14:13 1988\n\0 char *ctime(const time_t *tp)

ctime күнтізбелік уақытты жергілікті уақытқа ауыстырады, бұл asctime(localtime(TP)) орындауға баламалы.

struct tm *gmtime(const time_t *tp)

gmtime күнтізбелік уақытты Дүниежүзілік үйлестірілген уақытта (Coordinated Universal Time — UTC) аударады. UTC белгісіз болса, null береді. Бұл функцияның аты, gmtime, Greenwich Mean Time (Гринвич меридиан бойынша орташа уақыт).

struct tm *localtime(const time_t *tp)

localtime күнтізбелік уақытты *tp жергілікті уақытқа ауыстырады.

size_t strftime(char *s, size_t smax, const char *fmt, const struct tm *tp)

strftime printf функциясында берілген пішіммен көптеген ортақ ерекшеліктері бар fmt пішіміне сәйкес s жолына *tp күні мен уақытын пішімдейді. Әдеттегі таңбалар (‘\0’ таңбасын қоса алғанда) s-ге көшіріледі. S көп емес smax таңбалар орналастырылады, strftime ‘\0’ таңбалар санын немесе егер құрылған таңбалар саны smax артық болса, нөлдерді ескермей қайтарады.

%a апта күнінің қысқартылған атауы
%A апта күнінің толық атауы

%b айдың қысқартылған атауы
%B Айдың толық атауына.
%c күн мен уақыттың жергілікті көрінісі
%d айдың күні (01-31)
%H сағ (24 сағаттық уақыт) (00-23)
%I сағат (12 сағаттық уақыт) (01-12)
%j жыл басынан бастап күні (001-366)
%m ай (01-12)
%M минут (00-59)
%p жергілікті Am немесе PM көрінісі (түске дейін немесе одан кейін)
%S секунд (00-61)
% U жыл басынан апта (жексенбі-аптаның 1-ші күні деп есептегенде) (00-53)
%w Апта күні (0-6, жексенбі нөмірі-0)
%W жыл басынан бастап апта (дүйсенбі-аптаның 1-күні деп есептегенде) (00-53)
%x күннің жергілікті көрінісі
%X Жергілікті уақыт көрінісі
%y ғасыр көрсетілмеген жыл (00-99)
%Y ғасыр көрсетілген жыл
%Z егер бар болса, уақытша аймақтың атауы
%% %

В 11. Іске асыруға байланысты шектеулер: <limits.h> және <float.h>

Тақырып файлы <limits.h> бүтін түрлердің өлшемдеріне тұрақты мәндерді анықтайды. Төменде ең төменгі қолайлы шамалар көрсетілген, бірақ нақты іске асыруларда Үлкен мәндер де қолданылуы мүмкін.

CHAR_BIT	8	char мәніндегі биттер
SCHAR_MAX	UCHAR_MAX немесе SCHAR_MAX	максималды мәні char
CHAR_MIN	0 немесе SCHAR_MIN	минималды мәні char
INT_MAX	+32767	максималды мәні int
INT_MIN	-32767	минималды мәні int
LONG_MAX	+2147463647	максималды мәні long
LONG_MIN	-2147483647	минималды мәні long
SCHAR_MAX	+127	максималды мәні signed char
SCHAR_MIN	-127	минималды мәні signed char
SHRT_MAX	+32767	максималды мәні short

SHRT_MIN	-32767	минималды мәні short
UCHAR_MAX	255	максималды мәні unsigned char
UINT_MAX	65535	максималды мәні unsigned int
ULONG_MAX	4294967295	максималды мәні unsigned long
USHRT_MAX	65535	максималды мәні unsigned short

Келесі кестеде келтірілген атаулар <float.h> алынған және өзгермелі нүктелі арифметикаға қатысы бар тұрақтылар. Мәндер (егер олар болса) тиісті шамалар үшін ең аз мәндерді білдіреді. Әрбір іске асыруда өз мәні белгіленеді.

FLT_RADIX	2	тәртіпті ұсыну үшін негіз, мысалы: 2, 16
FLT_ROUNDS		қалқымалы нүктелі сандарды қосу кезінде дөңгелектеу тәсілі
FLT_DIG	6	
FLT_EPSILON	1E-5	минималды x, 1.0 + x □ 1.0
FLT_MANT_DIG		мантиссадағы FLT_RADIX негізі бойынша сандар саны
FLT_MAX	1E+37	қалқымалы нүктелі ең көп сан
FLT_MAX_EXP		максималды n, FLT_RADIXn-1
FLT_MIN	1E-37	өзгермелі нүктемен ең аз нормаланған сан
FLT_MIN		минималды n, 10n п қалыпты сан түрінде ұсынылады
DBL_DIG	10	double түрі үшін дұрыс ондық сандар саны
DBL_EPSILON	1E-9	ең төменгі x, 1.0 + x 1.0, онда x double түріне тиесілі
DBL_MANT DIG		double типті сандар үшін мантиссадағы FLT_RADIX негізі бойынша сандар саны
DBL_MAX	1E+37	double типті қалқымалы нүктелі ең көп саны
DBL_MAX_EXP		FLT_Radix -1 деген максималды n double түрі ретінде ұсынылады
DBL_MIN	1E-37	double типті өзгермелі нүктемен ең аз нормаланған сан DBL_MIN_EXP минималды n, 10N сияқты double типті нормаланған сан түрінде ұсынылады

С өзгерістер тізбегі

Бұл кітаптың алғашқы басылымы жарияланған сәттен бастап С тілінің анықтамасы өзгеріске ұшырады. Барлық дерлік жаңалықтар-қолданыстағы бағдарламалармен үйлесімділікті сақтау үшін жасалған тілдің бастапқы нұсқасының кеңейтілуі; кейбір өзгерістер бастапқы сипаттаманың екі мағыналарын жоюға қатысты, ал кейбіреулері қолданыстағы практикамен Енгізілген өзгерістер болып табылады. Кейінірек С-компиляторлардың басқа әзірлеушілері қабылдаған жаңа мүмкіндіктерінің көпшілігі бастапқыда компиляторларға қоса берілген құжаттарда жарияланды. ANSI комитеті тіл стандартын дайындап, осы өзгерістердің көпшілігін қосып, басқа да маңызды өзгерістерді енгізді. Кейбір коммерциялық компиляторлар оларды ресми С-стандартты шығарғанға дейін жүзеге асырды.

Бұл қосымшада оның бірінші редакциясында анықталған тіл мен оның стандарт ретінде қабылданған нұсқасы арасындағы айырмашылықтар біріктірілген. Мұнда тек тілдің өзі ғана қаралады; оның қоршаған ортасы мен кітапханасына қатысты мәселелер қозғалмайды. Соңғысы стандарттың маңызды бөлігі болса да, бірақ бірінші басылымда орта мен кітапхананы сипаттау әрекеті жасалмағандықтан, тиісті стандартты элементтермен салыстыруға ештеңе жоқ.

▪ Стандартта бірінші басылыммен салыстырғанда аса мұқият, прецессирлеу анықталды және кеңейтілді: оның негізіне лексемалар анық қойылған; лексеманы «желімдеу» (# #) және символдық жолдарды (#) жасау үшін жаңа операторлар енгізілді, сондай-ақ #elif және #pragma сияқты жаңа басқару жолдары енгізілді; лексемдердің бір жүйелілігі бар макросты қайта анықтауға рұқсат етілген; жолдардың ішінде параметрлерді қою тоқтатылды. Жолдарды кез келген жерде \ белгісі көмегімен, тек мерзім мен макроанықтамаларда ғана емес, «желімдеуге» рұқсат етіледі (А 12 қараңыз).

▪ Барлық ішкі сәйкестендіргіштердің маңызды символдарының ең аз саны 31-ге дейін жеткізілді; сыртқы байланысы бар сәйкестен-

діргіштер үшін ол 6-ға тең болып қалады; төменгі және жоғарғы регистрлердің әріптері ажыратылмайды. (Көптеген іске асыру маңызды символдардың көп санына жол береді.)

▪ #, \, X белгілері үшін, [,], { , } , ! . 0, кейбір таңбалар жиынтығында болмауы мүмкін, үш таңбалы тізбектермен басталатын ?? (A12.1 қараңыз). Үш белгілік тізбекті енгізу бар жолдардың ?? мәндеріне зиян келтіруі мүмкін екенін атап өткен жөн.

▪ Жаңа кілт сөздер енгізілді (void, const, volatile, signed, enum) және өлі туылған сөздердің жазбасы айналымнан алынып тасталды.

▪ Символдық константтар мен жол литерлері үшін жаңа эскейп-реттілік анықталды. Қабылданған эскейп тізбектерінен \ символдардың пайда болуы күтпеген нәтижеге әкеледі деп жарияланды (A2.5.2 қараңыз).)

▪ Барлық тривиальды өзгеріс заңдастырылды: 8 және 9 сегіз сандар емес.

▪ Тұрақты типін айқын көрсету үшін кеңейтілген жұрнақтар жинағы енгізілді: U және L — бүтін және F және L — қалқымалы нүктелі типтер үшін. Сондай-ақ, константалар үшін суффикстерсіз (A2.5) типті анықтау ережелері нақтыланды.

▪ Көрші жолдар конкатенирленеді деп жарияланды.

▪ (A2.6) кеңейтілген таңбалар жиынтығынан жол литералдары мен символдық константаларды жазуға мүмкіндік беретін құралдар берілген.

▪ Char түріндегі нысандарды (басқа нысандар сияқты) белгімен немесе белгісіз ажыратуға болады. Double мағынасында long float сөз тіркестерін қолдануға болмайды, бірақ жоғары дәлдіктегі өзгермелі нүктелі сандар үшін long double түрі енгізіледі.

▪ Кейбір уақыттан бері unsigned char түрі бар. Стандарт char немесе басқа бүтін түрдегі нысанның белгісі бар екенін айқын көрсету үшін signed кілт сөзін енгізеді.

▪ Бірнеше жыл бойы көптеген іске асыруларда void түрі бар. Стандартты void * жалпылама көрсеткіш түрі ретінде енгізеді; бұрын осы мақсат үшін char * пайдаланылды. Бір уақытта «араластыруға» типті нұсқағыштар мен бүтін немесе әр түрлі нұсқағыштар өзгертусіз тыйым салынатын ережелер күшіне енеді.

▪ Стандарт арифметикалық түрлердің ауқымының ең аз шектерін белгілейді, <limits.h> тақырып файлдарын қарастырады және <float.h> бұл сипаттамалар әрбір нақты іске асыру үшін орналастырылған.

▪ Тізбектеу — бірінші редакцияда болмаған жаңа түрі.

▪ Стандарт C++ үлгісінің квалификаторын, атап айтқанда const (A 8.2) квалификаторын жазу әдісін қолданады.

▪ Жолдарды өзгертуге тыйым салынады; яғни оларды тек оқуға болатын жадта орналастыруға рұқсат етіледі.

▪ «Қарапайым арифметикалық түрлендірулер «өзгертілген; шын мәнінде,» тұтас үшін әрқашан unsigned басым; құбылмалы нүкте үшін әрқашан double «» ең аз сыйымдылыққа дейін көтеру « принципіне қолданылады (A6.5 қараңыз).

▪ =+ сияқты ескі тағайындау операторлары жойылды. Әрбір тағайындау операторы енді бір жеке лексема болып табылады. Бірінші басылымда тағайындау операторы бөлгіш-символдармен бөлінген екі символды бейнелеуі мүмкін.

▪ Компиляторларға операторлардың математикалық ассоциациясын есептеу ассоциативтілігі ретінде түсіндіруге рұқсат етілмейді.

▪ - унарлы операторымен симметрия болуы үшін + унарлы операторы енгізілді .

▪ Сілтегішті функцияға нақты операторсыз (а

A7.3.2 қараңыз) деп аталатын өрнек ретінде пайдалануға рұқсат етіледі.

▪ Құрылымдармен операция жасауға рұқсат етілген, тағайындау кезінде құрылымдарды аргументтер ретінде функцияларға беруге және оларды функциялардан нәтиже ретінде алуға болады.

▪ Адресті қабылдау операторын қолдануға болады & массивке; нәтиже массивке сілтегіш.

▪ Бірінші басылымда sizeof операциясының нәтижесі int түрі болды; көптеген іске асыруларда ол unsigned-ға ауыстырылды. Стандарт оны ресми түрде іске асыруға тәуелді деп жариялайды, бірақ ол <stddef.h> тақырып файлында анықталуын талап етеді. size_t деп аталады. ұқсас өзгеріс енді ptrdiff_t деп аталатын көрсеткіштер айырмашылығының түріне қатысты жасалды (A7.4.8 және A7.7 қараңыз).

▪ Бұл компилятор регистрде болмаса да, register типті объектіге адресті алу операторын қолдануға тыйым салынады .

▪ Жылжу операциясы нәтижесінің түрі оның сол жақ операндының түрі болып табылады; оң жақ операндтың түрі нәтиже түрінің жоғарылауына әсер етпейді (A7.8 қараңыз).

▪ Стандарт массивтің соңғы элементінің артында бірден жатқан орынға сілтегіштердің көмегімен жіберуге рұқсат береді және әдеттегі сілтегіштермен жұмыс істеуге мүмкіндік береді, А7.7 қараңыз.

▪ Стандарт (С++-тен алынған) параметр типтерін және оларды өзгерту мүмкіндігін нақты көрсетумен функцияның прототипін жазу әдісін енгізеді және өзгермелі дәлелдер тізімімен жұмыс жасау әдісін рәсімдейді. (А7.3.2, А8.6.3, В7 қараңыз). Ескі жазу әдісі кейбір шектеулермен қол жетімді.

▪ Стандарт бос декларацияларға тыйым салады, яғни декларациялары жоқ және ешқандай құрылым, одақ немесе тізім жарияланбаған. Алайда, бір құрылым (немесе одақ) тегі бар декларация, егер ол сыртқы ауқымда жарияланған болса да, қайта жарияланады.

▪ Спецификаторлары мен жіктеуіштері жоқ сыртқы мәлімдемелерге (яғни, бір «жалаңаш» декларациямен) мәлімдеме жасауға тыйым салынады.

▪ Кейбір іске асыруларда, сыртқы декларация ішкі блокта орналасқан кезде, оның көлемі файлдың қалған бөлігіне таралады. Стандарт бұл жағдайды нақтылайды және мұндай мәлімдеме тек блокпен шектелгенін мәлімдейді.

▪ Параметр ауқымы функцияның негізгі бөлігі болып табылатын құрмалас сөйлемде «кірістірілген», сондықтан функцияның жоғарғы деңгейіндегі мәлімдемелер оларды «көлеңкеге түсіре» алмайды.

▪ Анықтауыштардың аттар кеңістігі сәл өзгертілген. Стандарт барлық құрылымға, одаққа және санау тегтеріне бір ат кеңістігін бөледі; нұсқаулық жапсырмалар үшін жеке аттар кеңістігі енгізілген (AIL қараңыз). Сонымен қатар, элемент атаулары олар құрамына кіретін құрылыммен немесе одақпен байланысты. (Бұл біраз уақыттан бері кең таралған тәжірибе.)

▪ Бірлестіктің бастамашылығына жол беріледі; инициализатор одақтың бірінші мүшесін білдіреді.

▪ Автоматты құрылымдарды, бірлестіктер мен массивтерді бастауға рұқсат етіледі, бірақ кейбір шектеулер бар.

▪ Жол таңбаларын массивті инициализациялауға әріптік белгілердің нақты саны рұқсат етіледі («\ 0» таңбалары жоқ).

▪ Басқару өрнегі және case-tages switch-та кез келген тұтас түрде болуы мүмкін.

B.W. Kernighan
D.M. Ritchie

С бағдарламалау тілі

Басуға 03.09.2020. Пішімі 60×90$^{1}/_{16}$.
Көлемі 24,5 б.т. Есептік б.т. 22,7 . Шартты б.т. 18,6.
Таралымы 810 дана.

«Фортуна полиграф» баспасы»ЖШС
050063, Алматы қаласы, 1-ықшам ауданы, 81-үй /
Fpolygraf@bk.ru
Тел: +7 701 787 32 92, +7 771 574 57 05,
+7 701 940 76 86