ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ
Ш. ЕСЕНОВ АТЫНДАҒЫ КАСПИЙ МЕМЛЕКЕТТІК ТЕХНОЛОГИЯЛАР ЖӘНЕ ИНЖИНИРИНГ УНИВЕРСИТЕТІ

ТҰРЫМБЕТОВ Т.Ә.

СИ ТІЛІНДЕ БАҒДАРЛАМАЛАУ

Оқу құралы

АҚТАУ - 2011

ӘОЖ 004.43(075.8)
ББК 32.973-018.1я73
Т 86

Пікір жазғандар:

Ш.Есенов атындағы Каспий мемлекеттік технологиялар және инжиниринг университетінің проректоры, техн.ғ.д., профессор Ажиханов Н.Т.

Х.Досмұхамедов атындағы Атырау мемлекеттік университеті, «Физика және математика» факультетінің деканы, техн.ғ.д., профессор Кенжегулов Б.З.

Қ.А.Ясауи атындағы Халықаралық қазақ-түрік университеті, «Бағдарламалармен қамтамасыздандыру» кафедрасының меңгерушісі, техн.ғ.к., ХҚТУ профессоры Темірбеков Ә.Н.

Жауапты редактор

Ш.Есенов атындағы Каспий мемлекеттік технологиялар және инжиниринг университеті, «Физика және информатика» кафедрасының меңгерушісі, п.ғ.к., доцент Түркменбаев Ә.Б.

Тұрымбетов Т.Ә.
Т 86 Си тілінде бағдарламалау: оқу құралы. Ақтау. Ш.Есенов атындағы Каспий мемлекеттік технологиялар және инижиниринг университеті, 2011. -111 бет.

ISBN 978-601-7349-12-7

Техника мен технологияның даму кезеңінде копьютерде жұмыс жасай білу, әсіресе бағдарламалау тілдерінің бірінде бағдарлама жаза білу заман талабының бірі болып тұр. Бұл оқу құралда Си тілінде бағдарламалауды оқып үйрену жолдары қарастырылған.

Оқу құралы Ш. Есенов атындағы Каспий мемлекеттік технологиялар және инижиниринг университеті, педагогикалық технологиялар және теңіз технологиялары институтының білімгерлеріне және басқа оқу орындардың жаратылыстану және ақпараттық технологиялар бағыттарындағы мамандықтарына арналған.

ӘОЖ 004.43(075.8)
ББК 32.973-018.1я73

ISBN 978-601-7349-12-7

КІРІСПЕ

Си бағдарламалау тілін 1972 жылы Bell Laboratories фирмасының қызметкері Денис Ритч жасап шығарды. Бұл тіл Unix операторлық жүйесімен бірге жарыққа келді. Бұл тіл жүйелік бағдарламаларды жасау мақсатында қолданылатын құрылымдық бағдарлама. Тілдің локалдық құрылымы мен мүмкіндігінің жоғарылығына байланысты кең тараған. Си тілінің компиляторы қазіргі таңдағы барлық операциялық жүйелерде жұмыс жасай береді. Бұл тілдің басқа тілдерден айырмашылығы ұлттық және халықаралық талаптар негізінде жасалғандығы.

Бұл оқу құралы стандартты Си бағдарламалау тілін үйренуге арналған. Мұнда типтік есептерді практикалық шешу мен тілдің синтаксистік және семантикалық құрылымына талдау жасалынған.

Бірінші бөлімде ақпарат түсінігі, ақпараттың қасиеттері, ақпаратты өңдеу жолдры қарастырылған.

Екінші бөлімде Си бағдарламалау тілінің алфавиті, идентификаторлары, қызметші сөздері, тұрақтылар және жолдар, айнымалылар және ат берілген тұрақтылар жайлы мәліметтер келтірілген.

Үшінші бөлімде бағдарламаның мәтіні және препроцессор бағдарламалаудың қарапайым құралдары, қайталану операторлары, массивтер және қайталану операторларының сатылап орналасуы, функциялар және таңдау операторына түсініктер келтіріліп мысалдармен пысықталған.

Төртінші бөлімде препроцессордың өңдеу командалары және кезеңдері, тақырыптық файлдарға арналған мәтіндерді бағдарламаға қосу, шартты копиляция, тармақталу директивалары, препроцессор құрылымдарын макростармен ауыстыру, көмекші дрективалар жайлы түсініктер келтірілген.

Бесінші бөлімде көрсеткіштер массивтер, жолдар, динамикалық жадымен жұмыс жасау, символдық ақпараттар жайлы мәліметтер қарастырылған.

Алтыншы бөлімде функция параметрлеріне көрсеткіштерді қолдану, функция параметрлері ретінде массивтермен жолдарды қолдану тақырыптары қарастырылған.

Жетінші бөлімде Си бағдарламалау ортасында енгізу және шығару функцияларын қолдану, легтік енгізу және шығару, легтерді ашу және жабу, стандартты файлдар және олармен жұмыс жасуға арналған функциялар, дискідегі файлдармен жұмыс жасау, форматталған режимде файлдармен мәлімет алмасу, төменгі деңгейдегі енгізу-шығару, файлды ашу және жабу, мәліметтерді оқу және жазу түсініктері келтіріліп, мысалдармен пысықталған.

Сегізінші бөлімде құрылымдық типтер және құрылымдар, туынды типтер, құрылымдарды сипаттау, құрылымдарға жады бөлу, құрылымдарды инициализациялау және меншіктеу тақырыптары қарастырылған.

Барлық бөлімдер бақылау сұрақтары арқылы пысықталған.

Оқу құралын жақсарту мақсатында айтылған сын пікірлерге автор рахмет айтады.

І БӨЛІМ. АҚПАРАТ ТҮСІНІГІ МЕН АҚПАРАТТЫҢ ӨҢДЕЛУ ЖОЛДАРЫ

1.1 Ақпарат түсінігі мен қасиеттері

Ақпарат - термині латын тілінің Informatio түсіндіру, баяндау деген ұғымынан шыққан. Ақпаратты біз ауызша, не жазбаша, қимыл не қозғалыс түрінде бере аламыз. Кез-келген керекті ақпараттың мағынасын түсініп, оны басқаларға жеткізіп, соның негізінде белгілі бір ой тұжырым жасаймыз. Сонымен бірге ақпарат кез-келген түрде бізге белгілі бір мағлұматтар бере алады. Жалпы тұрғыдан алғанда, ақпарат таңбалар мен сигналдар түрінде берілген әлемнің, заттың бейнесі болып саналады.

Ақпарат алу – бізді қоршаған құбылыстар мен нысандардың өзара байланыстары, құрылымы немесе олардың бір-біріне қатысуы жөнінде нақты мағлұматтар мен мәліметер алу деген сөз.

Ақпарат істің ақиқаттық жағдайын толық ашатын болса оның дәл болғаны. Дәлдігі жоқ ақпарат оны түсінбеушілікке және соған байланысты теріс шешім қабылдауға әкеліп соқтыруы мүмкін. Егер ақпарат оны түсінуге және белгілі бір шешім қабылдауға жеткілікті болса, онда оның толық болғаны (ақпараттың толықтылығы). Ақпараттың толық болмауы ол жөнінде белгілі бір тұжырым жасауға кедергісін тигізіп, қателікке ұрындыруы мүмкін.

Ақпаратты пайдалана отырып, қандай да бір мәселелерді шеше алатын болсақ, онда ол ақпарат құнды болып есептеледі.

Егер бағалы, әрі өзекті ақпарат түсініксіз түрде берілген болса, онда ол пайдаға аспайды (бұл ақпараттың түсініктілігі).

1.2 Ақпаратты өңдеу

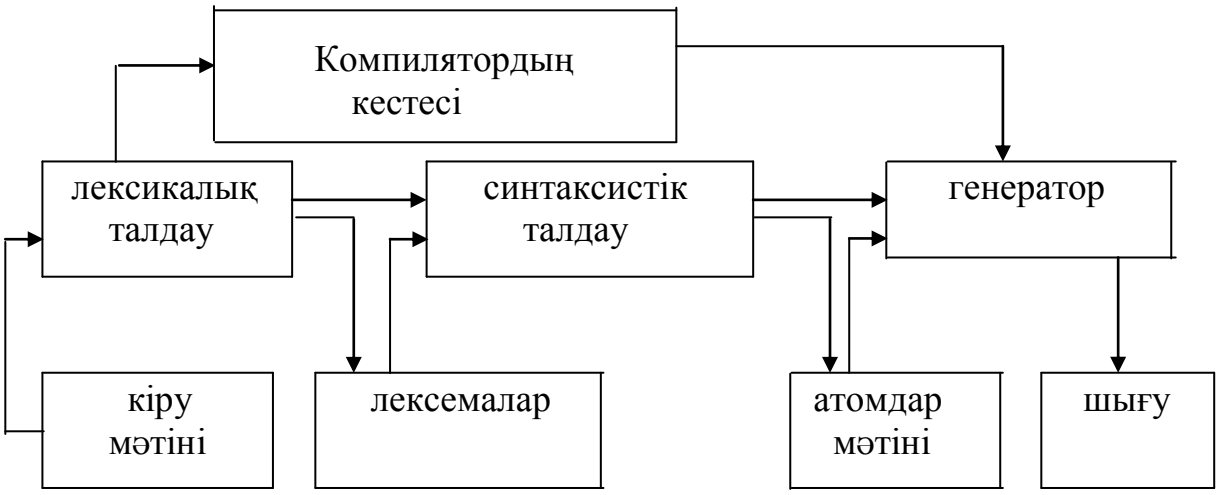
Ақпаратты өңдеу дегеніміз – бір қатар алгоритмдерді орындай отырып ақпараттық объектілерді алу. Өңдеу ақпарат орындалатын негізгі операциялардың бірі.

Ақпаратты өңдеу құралдары адамзат жасаған әртүлі құрылғылармен мен жүйелер бірінші кезекте компьютер, компьютерлер шыққан кезден бастап, олармен қарым-қатынас жасау қажеттілігі туындады. Өйткені оларға біз берген тапсырмалар мен жұмыстарды орындату керек болды. Міне, осы мақсатта арнайы тілдер ойлап шығарыла бастады. Оларды біздің табиғи тіліміздің басқа жасанды тілдер деп атады. Жасанды тілдер бір жағынан қарапайым және адамға түсінікті болу керек те, ал екінші жағынан ол тілді құрылғылардың қабылдауы керек болды. Міне, осы талаптарды бір тілде сыйғызу өте қиын мәселенің бірі болып келді, сондықтан мәтінді адамға түсінікті тілден құрылғылар тіліне айналдыратын құралдар пайда болды. Мұндай құралдарды *трансляторлар* деп атады. Транслятордың екі түрі бар:

1. Интерпретатор - бағдарламаның бастапқы мәтініне талдау жасайтын және бағдарламаны жеке-жеке оператор бойынша орындайтын аудармалаушы. Басқаша айтқанда, есептеу үрдісі кезінде бастапқы бағдарлама мәтініндегі жеке-жеке оператор бойынша, қадамдап аудармалайтын және орындайтын бағдарлама.

2. Компиллятор - машина кодына аударылған бағдарлама (exe, com, формулалар). Ол интерпретатор сияқты бағдарламадағы операторларға бір-бірлеп талдау жасамайды, сондықтан оның орындалуы интерпретаторға қарағанда өте жылдам. Осы себептерге байланысты дайын болған бағдарламаны компиляция жасаған дұрыс.

Компилятордың жұмысы бірнеше кезеңдерден тұрады. Олар бірінен кейін бірі орындалады. Бұл кезеңдер төменде көрсетілген (1.1-сурет).



1.1-сурет. Компилятор жұмысының кезеңдері

Компилятордың бірінші кезеңі *лексикалық талдау* деп аталады, ал оны жүзеге асыратын бағдарлама – *лексикалық талдағыш* деп аталады. Лексикалақ талдағышқа кіру мәтінінің символдар тізбегі болады. Бұл тізбектегі лексикалық талдау тілдің ең қарапайым құрылымдарын бөледі, олар лексикалық бірліктер деп аталады.

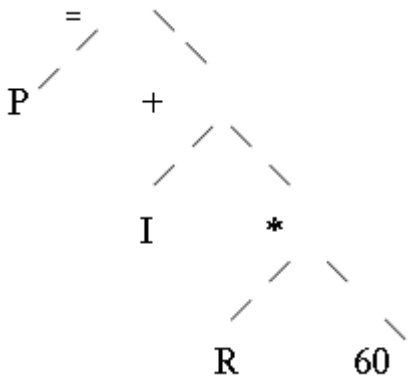
Лексикалық бірліктердің мысалы ретінде идентификаторлар, сандар, таңбалар және операцияларды алуға болады. Лексикалық талдау бастапқы мәтінді, лексикалық бірліктерді лексемаларға ауыстыра отырып өзгертеді. Лексемалар лексикалық бірліктердің класы және оның мәні туралы ақпаратты алып қалады.

Лексикалық талдау. Лексикалық талдау кезінде таңбалар немесе символдар легі тізбектеліп бірінен кейін бірі оқылады да, лексикалық бірліктерге немесе лексемаларға бөлінеді. Мысал ретінде келесі операторының талдауын қарастырайық:

P= i+r*60
Бұл операторды келесі лексемаларға бөлуге болады:
P идентификаторы
= меншіктеу таңбасы
i идентификатор
+ белгісі
r идентификатор

* белгісі
60 саны.

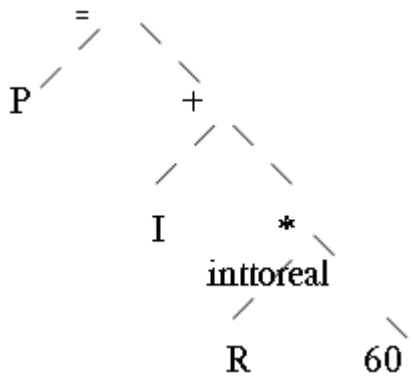
Компиляторды құру үшін кіру және шығу тілдері біркелкі және дәл анықталуы керек. Мұнда бағдарламалау тілінің әртүрлі құрылымдарын түзу ережелері анықталуы қажет. Мұндай ережелер жиынтығын тілдің синтаксисі деп атайды. Бұдан басқа бағдарламалау тілінің әрбір құрылымдарының мағынасы және міндеті анықталу қажет. Компилятордың екінші кезеңі *синтаксистік талдау* деп аталады, ал оған сәйкес бағдарламаны – *синтаксистік талдағыш* деп атайды. Синтаксистік талдағышқа лексемалар тізбегі беріледі де, олар қандай да бір аралық кодқа айналдырылады. Бұл аралық код атомдар немесе іс-әрекет символдарының тізбегін көрсетеді. Әрбір атом, орындалатын операциялардан тұрады және олар қолданылатын операндалар бойынша орындалады. Мұнда атомдардың орналасу тізбегі, лексемаларға қарағанда, операциялардың орындалу кезегіне байланысты орналасады. Лексемалар легі ирархиялық құрылымға біріктіріледі де, ол компиляторға шығу кодын синтездеуге көмек береді немесе синтаксистік ағаш тұрғызуға көмек береді:



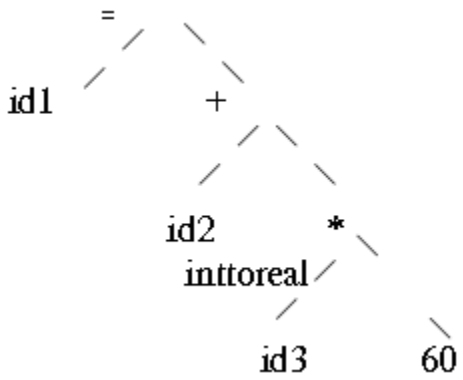
Мұндай ағаштарды *синтаксистік ағаш* деп атайды. Мұнда бірінші 60-қа көбейту орындалады, онан кейін қосу, сонан кейін меншіктеу операторы орындалады.

Семантикалық талдау. Бұл кезеңде бағдарлама семантикалық дұрыстыққа тексеріледі және келесі кезеңдерге ақпарат жиналынады. Сементикалық талдаудың ең маңызды бөлігі – бұл типтерді тексеру: мұнда компилятор типтерді өзгертуді қоя отырып, синтаксистік ағаштарды өзгерте алады.

Жоғарыдағы мысалда, егер r:real (нақты) типті болса, онда бүтін 60 саны да real типтіне түрленеді:



Бұл жерде 60 саны бүтін типтен нақты типке айналдырады. Бірінші үш кезең қарастырылды, бұл кезеңдер талдау кезеңін құрайды. Оларда тек синтаксистік ағаштар құрылып қана қоймай, әр түрлі ақпараттар жинап, олар сәйкес кестелерге енгізіледі. Компилятордың үшінші кезеңінде бағдарламаның шығу мәтінін құру орындалады. Бұл кезеңді орындайтын бағдарламаны шығу мәтінінің генераторы деп атайды. Генератор өзіне келген әрбір іс-әрекет символына, оған сәйкес шығу тілінің бір немесе бірнеше бұйрықтарын (командаларын) қояды. Шығу тілі болып құрылғылар бұйрығы, ассемблер бұйрықтары, болмаса қандайда бір басқа тілдің операторлары пайдаланылады. Жоғарыда қарастырылған мысалда, p, i, r айнымалылары қатысты. Көп тілдедерде, олар бұл кезеңге дейін анықталып, айнымалылар кестесіне енгізілуі керек. Шығу мәтінінің генераторына айнымалылардың аты маңызды емес. Генераторға олардың адрестері маңызды. Сондықтан синтаксистік ағаш төменднгідей көрсетіледі:



Бұл жерде id1, id2, id3, p, i, r айнымалыларының адрестерінің шартты белгіленуі. Аралық код әртүрлі болуы мүмкін. Көп жағдайда “төрттіктер” қолданылады, яғни олар командалар кодынан, екі операнданың адресінен және нәтиженің адресінен тұратын командалар. Біз қарастырған мысалда мынандай кодқа трансляция жасауға болады. Ол оқуға да ыңғайлы:

Temp1 ⟵ inttoreal (60)
Temp2 ⟵ id3*temp1
Temp3 ⟵ id2+temp2
Id1 temp3

Бұл жерде оптимизация (оңтайландыру) қолданып, inttoreal (60) 60-санының типін ауыстыруды компиляция кезеңінде орындап кетуге болады:

```
Temp1 ⟵ 60.0
Temp2 ⟵ id3*temp1
Temp3 ⟵ id2+temp2
Id1        temp3
```

Ары қарай қарастырсақ Temp1 өзінің мәнін тек бір рет қана алады да, ары қарай кодтың оң жағында ғана кездеседі. Оны алып тастауға болады:

```
Temp2 ⟵ id3*60.0
Temp3 ⟵ id2+temp2
Id1        temp3
```

Осындай әдіспен Temp3-ті да алып тастап ең негізгі оптималды (оңтайландырылған) код аламыз:

```
Temp2 ⟵ id3*60.0
Id1        id2 temp2
```

Әртүрлі компилятор әртүрлі оптимизациялау (оңтайландыру) жасайды. Шығу мәтінін генеризациялау компиляцияның соңғы кезеңі. Айнымалыларға нақты адрестер беріліп, регистрлерге орналастырылады. Одан кейін аралық код, мақсаты кодқа трансляцияланады.

Бақылау сұрақтары

1. Ақпарат дегеніміз не?
2. Ақпаратты өңдеу дегеніміз не?
3. Транслятор не қызмет атқарады.
4. Транслятордың неше түрі бар.
5. Интерпретатордың қызметі не?
6. Компилятор дегеніміз не?
7. Компилятордың жұмыс жасау кезеңдерін атңыз.
8. Лексикалық талдау дегеніміз не, мысал келтіріңіз.
9. Синтаксистік талдау дегеніміз не, мысал келтіріңіз.
10. Семантикалық талдауға мысал келтіріңіз.

II БӨЛІМ. СИ БАҒДАРЛАМАЛАУ ТІЛІНІҢ НЕГЗГІ ТҮСІНІКТЕРІ

2.1 Си тілінің алфавиті, идентификаторлары, қызметші сөздері

Си тілінің алфавитіне:

- латын алфавитінің бас және кіші әріптері (A, B, …, Z, a, b, … , z);
- цифрлар - өзімізге белгілі араб цифрлары 0,1,2,3,4,5,6,7,8,9;
- арнайы символдар: “ , { } [ ] ( ) + - / % ; . : < = > _ ! & * # ~ ^;
- көрінбейтін символдар, олар лексемаларды бір-бірінен бөліп тұрады (мысалы: бос орын, табуляция, жаңа қатарға өту). Сонымен қатар Си тілінде жазылған бағдарламалар түсінікті болуы үшін арнайы драйверлердің көмегімен орыс, қазақ алфавиттерінің әріптерін пайдалану мүмкіндіктері бар. Мысалы түсіндірмелерде (комментарияларда) қолданылады. Түсіндірмелер сол жағында /* таңбаларымен оң жағында */ таңбаларының арасына жазылады.

/* Түсіндірмелік мәтін */

Си тілінде лексемалардың алты класы пайдаланылады:

- идинтификаторлар;
- қызметші сөздер;
- тұрақтылар;
- қатарлар;
- операциялар (операциялар таңбалары);
- бөлгіштер (пунктуация таңбалары).

Әріптен немесе төменгі сызықша белгісімен басталатын әріптер, цифрлар, символдар тізбегі Си тілінің *идентификаторы* болып есептеледі.

a, b, c, KOM_16, _MIN, TAME, tame Си бағдарламалау тілінде үлкен және кіші әріптер бөлінеді, сондықтан жалпы идентификатордың ұзындығы шексіз, бірақ Си компиляторында идентификатор ұзындығы 31 символдан аспауы керек.

Қызметші сөздер – алдын-ала мағынасы анықталған тілдің құрамының бір бөлігі болып табылатын сөздер. Қызметші сөздер мәліметтердің типтерін, жадының кластарын, типтердің квалификаторларын, модификаторларын, псевдоайнымаларды және операторларды анықтайды. Мағанасына байланысты қызметші сөздерді мынандай топтарға бөлуге болады. Мәліметтердің типін анықтау үшін Си тілінде типтердің спецификаторлары және типтердің квалификаторлары қолданылады. Типтердің спецификаторларына мыналар жатады:

Char – символдық тип;
double – жылжымалы нүктелі екілік дәлдікті нақты тип;
Enum – атап өту типі әр қайсына жеке-жеке ат және мән енгізілетін, бүтін санды тұрақтыларды анықтау;
Float – жылжымалы нүктелі нақты тип;
Int – бүтін тип;
Long – ұзындығы ұлғайтылған бүтін тип (ұзын бүтін тип);
short – ұзындығы қысқартылған бүтін тип (қысқа бүтін тип);
struct – құрылымдық тип (структура);
signed – таңбалы яғни бүтін таңбалы тип (үлкен битті таңбалы болып есептеледі);

union – біріктіру (біріктіру тип);
unsigned – таңбасыз бүтін таңбасыз тип (үлкен биті таңбасыз болып есептеледі);
Void – мәнің болмауы;
typedef – типті анықтау синонимін енгізеді;

Типтердің квалификаторларына мыналар жатады:
const – тек қана оқылатын, тұрақты мән қабылдайтын объектінің квалификаторы;
volatile – программистің нұсқауынсыз-ақ мәнін өзгертуге болатын, объектінің квалификаторы;

Жадының кластарын сипаттау үшін мыналар қолданылады:
Auto – автоматтық;
extern – сыртқы;
register – регистрлік;
Static – статикалық (өзгермейтін, тұрақты);

Операторларды сипаттау үшін келесі қызметші сөздер қолданылады:
break – циклдан шығу;
continue – циклдың ағымдағы итерациясын бітіріп, келесі итерацияға өтіп циклды жалғастыру;
Do – орындау (тұрақты шарты бар циклдық оператордың басы);
For – параметрлі циклдық операторлың басы;
Goto – шартсыз өту;
If – егерде - шартты оператор;
return – оралу (функциядан оралу);
switch – ауыстырғыш;
while – алдын-ала шарт бойынша жұмыс істейді;

Қызметші сөздерге келесі идентификаторлар жатады:
default – switch – операторында керек вариант болмай қалған кездегі іс-әрекетті анықтайды.
case - switch – операторындағы вариантты анықтайды;
sizeof – операнданың көлемін (размерін) анықтайтын операция;

2.2 Тұрақтылар және жолдар

Тұрақтылар дегеніміз мәндері белгілі және бағдарламаның орындалу барысында өзгермейтін шамалар. Си тілінің синтаксисі тұрақтылардың бес типін анықтайды:

- символдар;
- атап өту типінің тұрақтылары;
- нақты сандар;
- бүтін сандар;

- нөлдік көрсеткіш (сілтеме, нулевой указатель). Нөлдік көрсеткіштен басқа тұрақтылар Си тілінде арифметикалық тұрақтыларға жатқызылған.

Символдар немесе символдық тұрақтылар. Жеке ішкі кодтары бар, бөлек өзгеше белгілерді сипаттау үшін, символдық тұрақтылар қолданылады. Әрбір символдық тұрақты – бұл екі жағынан апострофқа алынған символдан тұратын лексема. Мысалы. ’A’, ’a’, ’B’, ’8’, ’0’, ’+’, ’;’ және т.б.

Си тілінде бұдан басқа бірнеше символдар құрамынан (комбинациясынан) тұратын литерлер жиынтықтары бар. Мұндағы әрбір құрам (комбинация) ’\’ кері көлбеу сызық символынан басталады. Мұндай литерлер жиынтығын Си тілінде *басқару тізбектері* деп атайды. Олар мыналар:

’\n’ - келесі жолға өту;
’\t’ - көлденең табулация;
’\r’ – курсорды жолдың басына қайтару;
’\\’ - кері көлбеу сызық \ ;
’\’’ - апостроф;
’\”’ - жақша;
’\0’ - нөлдік символ;
’\a’ - сигнал – қоңырау;
’\b’ - бір позицияға (бір символға) кері қайтару;
’\f’ - келесі бетке өткізу;
’\v’ - тік (вертикальды) табулация;
’\?’ - сұрау белгісі;

Бұлардан басқа басқару тізбектеріне ’\ddd’ және ’\xhh’ немесе ’\Xhh’ лексемалары да кіреді. ’\ddd’- кез-келген символдық тұрақтының сегіздік түрде бейнеленуі. Бұл жерде d-сегіздік сан (0-7). Мысалы, ’\017’ немесе ’\233’. ’\xhh’ немесе ’\Xhh’- кез-келген символдық тұрақтының он алтылық түрдегі бейнеленуі. Бұл жерде он алтылық сан (0-F). Мысалы: ‘\x0b’, ‘\x1A’ және т.б.

Бүтін тұрақтылар. Си тілінің синтаксисінде мынадай бүтін тұрақтылар анықталған: сегіздік, ондық және он алтылық. Негізгі тұрақтылар жазбасындағы префикспен анықталады. Ондық тұрақтылар үшін префикс қолданылмайды. Ондық бүтін тұрақтылықтар 0-ден басталмайтын, ондық цифрлардың тізбегінен тұрады, (егер ол сан 0 - дің өзі болмаса): 44, 684, 0, 1024. Нөлден басталып 7-ден үлкен сан болмайтын, цифрлар тізбегін, сегіздік тұрақты деп атайды.

016-014 санының сегіздік бейнеленуі. Алдында 0x немесе 0X символы бар. Он алтылық цифрлардың (0, 1,..., 9, A, B, C, D, E, F) тізбегі он алтылық тұрақтылықтар деп аталады. 0x16- 22 санының он алтылық түрдегі бейнеленуі. OXFF- 255 санының он алтылық түрлендіруге бейнеленуі. Ондық тұрақтылықтар Си тілінде 0-ден 32767 бүтін ондық сандарымен шектеледі, ал ұзын бүтін сандар (long) 0-ден –2147483647-ге дейін барады.

Нақты тұрақтылар. Нақты сандарды сипаттау үшін ЭЕМ жадында жылжымалы нүкте түрінде бейнелейтін тұрақтылар қолданылады. Әрбір нақты тұрақты бүтін бөліктен және бөлшек бөліктен тұрады да, дәреже белгісін е

немесе E көрсетеді (2.1-кесте). Мысалы, 44. 3.14159 44e0 .314159E1 0.0. Си тілінде барлық бүтін тұрақтылар 0-ден 32767 аралығында жататын сандар int типімен сипатталады.

2.1-кесте. Тұрақтылар және оларға сәйкес типтер

Мәліметтер типі	**Анықталу аралықтары**
Float	3.4E- 38-ден 3.4E+38 – дейін
Double	1.7E-308-ден 1.7E+308 – дейін
long double	3.4E-4932-ден 1.1E+4932 – дейін

Нөлдік көрсеткіш. Null – көрсеткіш бірден-бір арифметикалық емес тұрақты.

Атап өту тілінің тұрақтылары. Бүтін аттары бар тұрақтыларды, атап өту арқылы енгізуге болады. Enum атап_өту_типі {аты_бар_тұрақтылар_тізбегі} бұл жерде Enum - қызметші сөз. Атап_өту_типі- оның аты. Мұны жазу міндетті емес.

Аты_бар_тұрақтылар_тізбегі - үтірмен бөлінетін идентификаторлар тізбегі. Мысалы:

```
enum {one=1, TWO, THREE, FOUR};
 enum DAY {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
                THURSDAY, FRIDAY, SATURDAY};
enum BOOLEAN {NO,YES};
```

Егер де, тізбекте ’=’ белгісі бар бірде-бір элемент болмаса, онда тұрақты 0-ден басталады, арықарай 1-ге артып отырады.

Жолдар немесе жолдық тұрақтылар. Жолдық тұрақтыларды Си тілінде “жолдық литералдар” деп атайды. Жолдық тұрақтылар екі жағынан жақшаға алынған символдар тізбегімен анықталады.

“Жол үлгісі”

Егер де, былай жазылса “\n Мәтін\ n 3-қатарға\n орналасады” онда бұл мәтін экранға үш қатар болып шығады.

2.3 Айнымалылар және ат берілген тұрақтылар

Айнымалылар дегеніміз - бағдарламаның орындалу барысында мәндерін өзгертуге болатын шамалар. Әрбір айнымалы оны бағдарламада қолдану алдында анықталуы тиіс, яғни оған жады бөлінуі керек. Айнымалыға бөлінетін жадының көлемі, осы айнымалының типіне байланысты болады.

Жазылу түрі: Типі айнымалылар_атының_тізбегі.

Бұл жерде, айнымалылардың аты дегеніміз, үтірмен бөлінген идинтификаторлар тізбегі. Си бағдарламалау тілінде келесі бүтін санды типтер анықталған:

Char – ұзындығы 8 биттен аспайтын бүтін сан;

shortint – қысқа бүтін сан;

Int – бүтін;
Long – ұзын бүтін сан;

Ал нақты типтер келесідей анықталған:

Float – бір дәлдікті нақты сан;
double – екілік дәлдікті нақты сан;
long double – максимальды дәлдікті нақты сан;

Мысалы:

```
float    x, X, cc3, pot_8;
double  e, B4;
int  a,b, stop;
```

Тілдің синтаксисіне байланысты айнымалылар анықталғаннан кейін, оларда қандай да бір мән болмайды. Си тілінде айнымалыларды сипаттаған кезде-ақ оларға қандайда бір бастапқы мән беруге болады.

Тип айнымалының_аты=бастапқы_мән;

Мұндай әдісті инициализация деп атайды. Инициализацияның меншіктеуден айырмашылығы, меншіктеу бағдарламаның орындалу барысында іске асырылады, ал инициализация айнымалыға жадыдан орын бөлінгенде орындалады.

Мысалы:

```
float  pi=3.14, cc=1.23;
int   year= 1997;
```

Си тілінде айнымалылардан басқа бекітілген аттары бар тұрақтыларда бар. Тұрақтының аты ретінде бағдарламашының өзі таңдаған, қызметші сөздермен сәйкес келмейтін идентификаторлар пайдаланылады.

Жазылуы:

```
const  тип тұрақты_аты = тұрақты_мәні;
```

Бұл жерде const – типтің квалификаторы. Ол анықталатын объектінің тұрақты мән қабылдайтыны және оны тек оқуға болатынын көрсетеді. Тип – объектінің типтерінің бірі, тұрақтының_аты- идентификатор тұрақтының_мәні - оның типіне сәйкес болуы қажет.

Мысалы:

```
const double E= 2.718282;
const long    M = 99999999;
const         F = 765;
```

Ең соңғы қатарда тұрақтының типі көрсетілмеген, сондықтан оған өзіне сәйкес int типі меншіктеледі. Бұдан басқа тұрақтыларды енгізуге препроцессорлық директива қолданылады:

 # define <тұрақты_аты> <тұрақты_мәні>

Бұл жерде директиваның соңына “нүкте үтір” қойылмайды.
 # define директивасына келесі тақырыптарда толығырақ тоқталамыз.

```
const  double   E= 2.718282;
# define  E 2.718282
```

Екеуінің айырмашылығы, бірінші өрнекте E тұрақтысы анықталған кезде оның типі міндетті түрде көрсетіледі. Екінші өрнекте E тұрақтысы анықталған кезде оның типі, өзіне меншіктелген мәнге байланысты анықталады.

```
# define  Next 'z’
```

Бұл жерде ’z’ символын сипаттау үшін Next идентификаторын енгізеді.

Бақылау сұрақтары

1. Си тілінің алфавитіне нелер жатады?
2. Си тілінің идентификаторларына мысал келтіріңіз.
3. Си тілінің қызметші сөздері.
4. Си тілінде тұрақтылар қалай сипатталады.
5. Си тілінде жадының кластары қалай сипатталады.
6. Жолдарды сипаттауға мысал келтіріңіз.
7. Атап өту тұрақтыларын атаңыз.
8. Айнымалылар және ат берілген тұрақтыларға мысал келтіріңіз.
9. Бүтін санды типтерге мысал келтіріңіз.
10. Нақты санды сипаттайтын типтерге мысал келтіріңіз.

III БӨЛІМ. СИ ТІЛІНДЕ БАҒДАРЛАМАЛАУҒА КІРІСПЕ

3.1 Бағдарламаның мәтіні және препроцессор

Си тіліндегі әр бір бағдарлама - бұл препроцессорлі директиваларды және объектілермен функцияларды сипаттаулардың және анықтаулардың тізбегі. Препроцессорлі директивалар бағдарламаның мәтінін оны компиляция жасағанға дейін басқарады. Анықтаулар, функциялар мен объектілерді енгізеді.

Объектілер - бағдарламада өңделетін мәліметтерді сипаттау үшін қажет.

Функциялар – бағдарламаның мүмкін болған барлық іс-әрекеттерін анықтайды. Си тілінде жазылған бағдарлама міндетті түрде өңдеу кезеңінің үш түрінен өтеді:

- бағдарлама мәттінің препроцессорлі өзгеріуі;
- компиляция;
- компановка;

Тек осы кезеңдерден өткеннен кейін ғана бағдарламаның орындалатын машиналық коды дайындалады.

Препроцессордың міндеті - бағдарламаның компиляция кезеңіне дейін өзгерту. Препроцессорлы өңдеудің ережесін бағдарламашы препроцессордың директиваларының көмегімен анықтайды. Әр бір препроцессорлі директива ’#’ символымен басталады. Бұл тақырыпта бізге екі директива жеткілікті # include және # define. Препроцессор бағдарлама мәтінен ’#’ символынан басталатын жолдарды іздейді. Мұндай қатарлар препроцессорге бағдарламаның мәтінін өзгертетін іс-әрекеттерді анықтайтын командалар немесе директивалар ретінде қабылданады.

 # define - бағдарлама мәтініндегі ауыстыру ережесін көрсетеді.

 # include директивасы бағдарлама мәтінінің осы жерінде қандай текстік файлды қосу керектігін анықтайды. **# include** директивасы бағдарлама мәтініне “тақырыптық файылдар” каталогындағы файлдың мәтінін қосуға арналған. Ол Си тілінің стандартты кітапханаларымен бірге орнатылады.

include <тақырыптық_файылдың_аты> бұл директива бағдарлама мәтініне көрсетілген тақырыптық файлдан тек сипаттауларды ғана қоя алады.

Бағдарламаның құрылымы. Препроцессорлы өңдеу орындалғаннан кейін, бағдарлама мәтінінде бірде-бір препроцессорлі директива қалмайды. Бағдарламаның бұдан кейінгі мәтіні сипаттаулармен анықтаулардан тұрады. Егерде негізгі объектілерді, сипаттауларды және анықтауларды қарастырмасақ, онда бағдарлама негізінен функциялар қатарынан тұрады. Осы функциялардың арасынан **main** функциясы міндетті түрде көрсетіліуі тиіс. Бұл функция бағдарламаның ең негізгі функциясы болып есептеледі және бұл функциясыз бағдарлама орындалмайды.

Бағдарлама мәтіні төменде келтірілген:

препроцессордің - директивалары.

void main ( )

```
{
  Объектілерді_анықтау;
  Орындалатын_операторлар;
}
```

Препроцессордің директивалары тек қана бағдарламаның басында жазылуы міндетті емес. Қажет болған жағдайда оларды бағдарламаның кез-келген жеріне орналастыруға болады, бірақ тақырыптық файлдарды қолдануда олар бағдарламаның басында жазылуы тиіс. Бағдарламаның әрбір функциясының алдында, сол функция орындалғандағы алынатын мәннің типі туралы мәліметтің көрсетілуі тиіс. Егерде функция ешқандай мән қайтармайтын болса онда **void** типі көрсетіледі. **void main ( )** тақырыптық жолдан кейін функцияның денесі орналасады.

Функцияның денесі - бұл фигуралы жақшаларға алынған анықтаулар, сипаттаулар және орындалатын операциялар тізбегінен тұратын блок. Анықтаулар мен сипаттаулар блокта орындалатын операциялардың алдында орналасады. Әрбір анықтаудан, сипаттаудан және операторлардан кейін ’ ; ’ символы қойылады.

Анықтаулар бағдарламада өңделетін мәліметтерді сипаттау үшін объектілерді енгізеді.

Сипаттаулар компилляторға объектілермен функциялардың аттарын және қасиеттерін анықтап береді. Бағдарламаға енгізу-шығару құралдарын сипаттау үшін компилятордың стандартты библиотекаларының ішінен **#include <stdio.h>** директивасы қолданылады. **stdio.h** тақырыптық файлының аты былай оқылады. **std-standard** (стандартты), **i-input** (енгізу), **o-output** (шығарудың), **h – head** (тақырыбы).

Экранға шығару функциясы. Ақпаратты экранға шығару үшін **printf()** функциясы жиі қолданылады. Ол мәліметтерді ішкі машинаның кодынан символдық түрге айналдырып дисплейдің экранына айналдырады. Әрбір бағдарламашының мәліметтерді форматтауға, яғни оларды экранға қалай шығаруына ықпал етуге мүмкіндігі бар. Форматтау мүмкіндігі функцияның атында f литерімен **print** сөзінен кейін көрсетілген (print formatted) printf( ) операторы төмендегіше жазылады:

printf(форматты _ жол, аргументтер _ тізімі);

Операциялар. Операция таңбалы өрнектерді есептеу және қалыптастыру үшін операциялар қолданылады. Бір операцияны жазу үшін көп жағдайларда бірнеше символдар қолданылады.

“[ ] ”,”( )”, “? :” таңбаларынан басқа, операциялардың барлық таңбаларын компилятор бөлек лексемалар түрінде қабылдайды.

Операциялардың орындалу үстемділігі

Рангі	Операциялар	Ассоциа-тивтілігі
1.	( ) [ ] -> .	→
2.	! ~ + - ++ -- & * (тип) sizeof	←
3.	* / % (мультипликативті бинарлы)	→
4.	+ - (аддитивті биналы)	→
5.	<< >> (разрядпен жылжыту)	→
6.	< <= >= > (қатынастар)	→
7.	== != (қатынастар)	→
8.	& (разрядты конъюкция (“ И”) ”және”)	→
9.	^ (разрядты (“ ИЛИ”) ”немесе”)	→
10.	| (разрядты дезъюкция (“ ИЛИ”) ”немесе”)	→
11.	&& (конъюкция (“ И”) ”және”)	→
12.	|| (дезъюкция (“ ИЛИ”) ”немесе”)	→
13.	? : (шартты операция)	←
14.	= *= /= %= += -= &= ^= |= <<= >>=	←
15.	, (үтір операциясы)	→

Сөйлем құрамына байланысты бір лексема бірнеше түрлі операцияларды орындайды, яғни операцияның бір ғана таңбалары түрлі өрнектерде қолданылады және сөйлем құрамына байланысты түрліше орындалады. Мәселен бинарлық операция & - бұл разрядты конъюкция, ал унарлық операция & - бұл адресті анықтау операциясы. Бір рангілі операциялардың үстемділігі ең жоғары. Үстемділігі бірдей операциялардың орындалу сатысы бірдей, егерде амалдарда олардың бірнешеуі қолданылатын болса, онда олар ассоциативтік ережесі бойынша солдан оңға қарай немесе оңнан солға қарай орындалады. Егерде операцияның бір таңбасы кестеде екі рет көрсетілген болса, онда оның бірінші пайда болуы (үстемділігі жоғары) унарлық операцияға сәйкес келеді де, екінші пайда болуы бинарлық операцияға сәйкес келеді.

Унарлық (бірорындық) операциялар. Бірорынды префиксті және постфиксті операцияларды бейнелеу үшін келесі таңбалар қолданылады:

& – операнданың адресін анықтау (ранг 2);

* – адресі бойынша байланысу операциясы, яғни операндта көрсеткен объектінің адресі бойынша, оның мәнін алу; операнда сілтеме болуы тиіс. (ранг 2).

\- – унарлық алу (минус) арифметикалық операнданың таңбасын өзгертеді;

\+ – унарлық қосу, унарлық алу мен симметрия үшін енгізілген;

~ – бүтінсанды аргументтің ішкі, екілік кодын разряд бойынша инвертациялау;

! – логикалық не – операнданың мәнін логикалық жоққа шығару. Скалярлық операндаларға қолданылады. Бүтінсанды

нәтиже 0 (егерде операнда нөл болмаса, яғни ақиқат) немесе 1-тең (егерде операнда нөл болса, яғни жалған);

++ – бірге арттыру (инкремент немесе автоматты түрде өсу); екі түрі бар. Префиксті операция – опреранданы қолданғанға дейін, оның мәнін бірге арттыру. Постфиксті операция – операнданың мәнін оны қолданғаннан кейін бірге арттыру.

-- – бірге кему (декремент немесе автоматты кему) префиксті операция – операнданы қолданғанға дейін оның мәнін бірге кеміту. ++ және -- операцияларының операндалары тұрақтылар немесе сандар болмауы керек. Мысалы: ++ 5 немесе 85++ бұл қате болады.

sizeof – операндасы бар объектінің типінің көлемін (байтпен) анықтау операндасы.

Бинарлық (екіорындық) операциялар келесі топтарға бөлінеді:

- аддитивтік;
- мультипликативтік;
- жылжыту;
- разрядты;
- қатынас операциялары;
- логикалық;
- меншіктеу;
- құрылымдық обьектінің құрамын (компонентін) таңдау;
- “үтір” операциясы;
- операция ретінде қолданылатын жақшалар;

Аддиктивтік операциялар:

+ - бинарлық қосу- арифметикалық операндаларды қосу;
- - бинарлық алу – арифметикалық операндаларды алу;

Мультипликативтік операциялар:

* - арифметикалық типті операндыларды көбейту;
/ - арифметикалық типті операндыларды бөлу;
% - бүтін санды операндаларды бөлгендегі қалдықты табу;
Мысалы: 13%4=1

Жылжыту операциялары (тек қана бүтін санды операндалары үшін қолданылады)**:**

<< – сол жақтағы бүтін санды операнданың, оң жақтағы бүтінсанды

операнданың мәні бойынша анықталатын разряд бойынша солға жылжу (биттік көрсеткіш бойынша);

>> – оң жақтағы бүтін санды операнданың сол жақтағы бүтін санды операнданың мәні бойынша анықталатын разряд бойынша оңға жылжу (биттік көрсеткіш бойынша).

Мысалы: 4<< 2 бұл 16-ға тең,
5>> 1 бұл 1-ге тең.

Бірінші мысалыда 4 –тің екілік коды 100-ге тең, ол солға қарай екі позиция жылжығанда оның екілік коды 10000 болады, ал ол 16 санының екілік коды.

Қатынас операциялары:

< – кіші;
> – үлкен;
<= – кіші немесе тең;
>= – үлкен немесе тең;
== – тең;
!= – тең емес.

Логикалық бинарлық операциялар:

&& – және (И) арифметикалық операндалардың немесе қатынастардың конъюнкциясы;
| | – немесе (ИЛИ) арифметикалық операндалардың немесе қатынастардың дизъюнкциясы.

Меншіктеу операциялар:

= – жай меншіктеу, өрнектің сол жағындағы операндаға, өрнектің оң жағындағы операнданың мәнін меншіктеу.
Мысалы: P=10.3-2* x;

* = – көбейткеннен кейінгі меншіктеу, сол жақтағы операндаға екі жақтағы да операндалардың мәндерінің көбейтіндісін меншіктеу. P*= 2, немесе P=P*2;

/= – бөлгеннен кейінгі меншіктеу, сол жақтағы операндаға, сол операнданы оң жақтағы операндаға бөлгендегі мәнді меншіктеу. P/= 2.2-D немесе P=P/ (2.2-D);

%= – модуль бойынша бөлгеннен кейінгі меншіктеу, сол жақтағы операндаға, сол операнданы оң жақтағы операндаға бөлгендегі қалдықты меншіктеу.
N% = 3 немесе N= N%3.

+= – қосқаннан кейінгі меншіктеу, сол жақтағы операндаға теңнің екі жағындағы операндаларды қосқандағы шығатын мәнді меншіктеу. A+=B немесе A=A+B;

- = – алғаннан кейінгі меншіктеу, сол жақтағы операндаға сол

операндадан оң жақтағы операнданы алғандағы мәнді меншіктеу.

Жақшалар:

[] квадратты және () жай жақшалар функцияларды шақырғанда және массив элементтерін бейнелегенде бинарлық операциялардың рөлін атқарады. Жай жақшалар функцияларды шақырғанда қолданылады:
Функцияның_аты (аргументтер_тізімі) бұл жерде операндалар ретінде функциялардың_аты және аргументтер_тізімі қолданылады.
Мына өрнекте: массивтің_аты [идекс] операндалары ретінде массивтің_аты және индекс қолданылады.

Шартты үшорынды операция:

Унарлық, бинарлық операциялардан басқа шартты тернарлы операция қолданылады. Онда үш операнды қолданылады. Бұл операцияны бейнелеу үшін екі символ қолданылады, ол (?) және (;) және үш операнда қолданылады.

өрнек_1 ? өрнек –2: өрнек –3

Бірінші болып өрнек –1- дің мәні есептеледі. Егер де ол ақиқат болса, яғни нөлге тең емес болса, онда өрнек –2-нің мәні есептеледі және де ол нәтижие болып шығады. Егерде өрнек-1-дің мәнін есептегенде ол нөлге тең болса, онда нәтиже ретінде өрнек-3-тің мәні алынады.

Мысалы: x<0? –x : x;

3.2 Бағдарламалаудың қарапайым құралдары

Си тілі операторының топқа бөлінуі. Біз бұл уақытқа дейін Си тілінің алфавитін және лексемалары, мәліметтердің негізгі типтерін, тұрақтылар, айнымалыларды және барлық операциялар анықталды. Арифметикалық өрнектерді құрудың ережелері қарастырылды. Бағдарламаның құрылымының жазылуы, printf( ) функциясы арқылы арифметикалық мәндерді шығару құралдарын қарастырдық. Си тілінің тек қана бір main ( ) функциясынан тұратын қарапайым бағдарламаның құрылымына оралайық:

```
Препроцессордың-директивалары
void  main ( )
{
  объектілерді_анықтау;
  орындалатын_операторлар;
}
```

Әзірге бізге екі препроцессорлі директива # include < ...> және # define жеткілікті. Анықталған объектілер ретінде негізі типтердің тұрақтылары мен айнымалыларын енгіземіз. Ал функцияның денесіндегі орындалатын операторларға жеке тоқталып өтейік. Әрбір орындалатын оператор, бағдарламаның келесі қадамында орындалғандағы іс-әрекетті анықтайды. Операторда ешқандай мән болмайды.

Орындалатын іс-әрекеттерге байланысты операторларды екі типке бөледі:

- мәліметтерді түрлендіру операторлары;
- бағдарламаның жұмысын басқаратын операторлар.

Мәліметтерді түрлендіру операторының қарапайым түрлеріне “нүкте үтірмен” бөлінген меншіктеу операторы және кез-келген өрнектер жатады:

```
I++;        /* арифметикалық өрнек - оператор*/
X*=I;       /* күрделі меншіктеу операторы*/
X=X-4*I /* жай меншіктеу операторы*/
```

Бағдарламаның жұмысын басқаратын операторларды - бағдарламаның басқару құрылымдары деп атайды. Оларға мыналар жатады:

- күрделі операторлар;
- таңдау операторы;
- қайталану операторлар;
- өту операторлары.

Күрделі операторларға күрделі операторлардың өзі және блоктар жатады. Екі жағдайда бұл фигуралы жақшаларға алынған операторлар тізбегінен тұрады. Бірақ блоктың күрделі операторлардан айырмашылығы – блоктың денесінде оның анықтамасы болады. Мәселен мына бағдарламаның үзіндісі бұл күрделі оператор:

```
{
 n++;
 summa += (float)n;
}

ал мынау блок
{
 int  n=0;
 n++;
 summa +=( float)n;
}
```

Блок функцияның денесі ретінде жиі қолданылады.

Таңдау операторлары – шартты оператор (if) және (switch) операторы жатады. Қайталану (циклдық) операторлардың Си тілінде үш түрі бар олар: қайталануға кіру шартынан тұратын (while), қайталанудан шығу шартынан тұратын (do) және параметтірлі (for) операторлары.

Өту операторы басқарудың шартсыз беріліуін орындайды: олар goto (шартсыз өту), continue (қайталанудың ағымдағы итерациясын тоқтату), break (қайталанудан шығу), return (функциядан оралу).

Шартты оператор. Шартты операторлың жазылуының қысқа түрі (3.1-сурет):

```
if ( шарт) оператор;
```

Шарт - ретінде арифметикалық өрнектер қолданылады. Оператор шарт тек қана ақиқат болған жағдайда орындалады.
Мысалы:

```
if (  x<0 && x>-10)  x=-x;
```

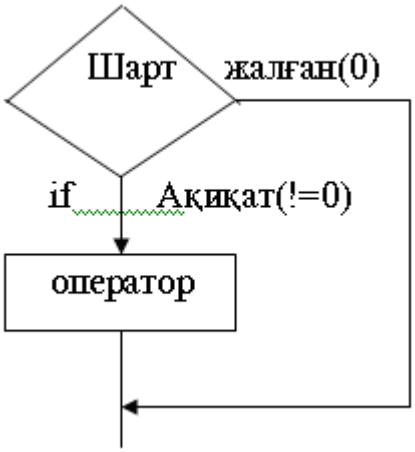
Қысқа түрінен басқа шартты операторлың толық түрі де бар (3.2-сурет):

```
if ( шарт )
оператор-1;
else
оператор-2;
```

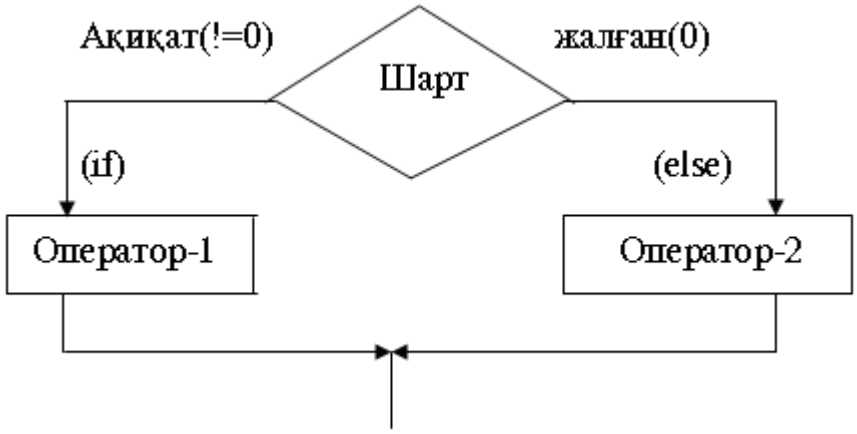
Бұл жерде шарт ақиқат болса оператор-1 орындалады, ал шарт жалған болса оператор-2 орындалады.
Мысалы:

```
if ( x>0)
  B=x;
  else
  B=-x;
```

if операторы қысқа түрде де толық түрде де бөлек оператор немесе күрделі оператор бола алады. Осы операторлың сызбасын қарастырайық.



3.1-сурет. Шартты операторлың қысқа түрі



3.2-сурет. Шартты оператордың толық түрі

Таңба және бос оператор. Таңба –бұл оператордың сол жағында орналасқан және одан “:” қос нүкте арқылы ажыратылатын идентификатор.

Мысалы: CON: X+=-8;

Шартсыз өту операторы. Шартсыз өту операторының жазылуы:

goto идентификаторы;

Идентификатор – бұл бағдарламаның таңбаларының бірі.

Си тілінің кез-келген операторының алдынан өту операторы goto арқылы бағдарламаның кез-келген жерінен тікелей сол операторға өтуге мүмкіндік беретін таңба қоюға болады.

Мәліметтерді енгізу. Мәліметтерді енгізу үшін **scanf** функциясы қолданылады. Жазылуы:

scanf (форматты_жол, аргументтер_тізімі);

scanf() функциясы пернелік тақтадан енгізілген мәліметтердің кодын “оқиды”. Бұл көрінетін символдардың коды немесе басқару кодтары болуы мүкін. **scanf()** функциясы кодтарды қабылдап, оларды ішкі форматқа түрлендіріп бағдарламаға береді. Форматты жолмен аргументтердің тізімі **scanf()** функциясы үшін міндетті түрде жазылуы тиіс. **scanf()** функциясының форматты жолын қарастырайық:

% алаңның_ені модификатор спецификатор.

Бұл жерде модификатор қолданбаса да болады. Онда тек % және спицификатор қолданылады. Сандық мәліметтерге келесі спецификаторлар қолданылады:

d – ондық бүтін сандар (типі int);

u – таңбасыз ондық бүтін сандар (типі unsigned int);

f – нақты сандар үшін (типі float);

e - нақты сандар үшін (типі float);

Алаңның_ені - бүтін оң сан, ол енгізілетін мәнге неше байт символ керектігін анықтайды. Модификатор ретінде келесі символдар қолданылады:

h- short int типіндегі мәліметтерді енгізу үшін (hd);

l- long int(/d) немесе double (lf, le) типіндегі мәліметтерді енгізу үшін;

L-long double(lf,le) типіндегі мәліметтерді енгізу үшін.

scanf() функциясының printf() функциясынан айырмашылығы scanf() функциясының аргументі бағдарламаның обьектілердің адрестері қабылданады. Жалпы жағдайда бұл – айнымалылардың адрестері.
Мысалы: Scanf(“%d %f %f”,&n,&z,&x);

3.3 Қайталану операторлары

Басқа тілдерге ұқсас, Си тілінде де қайталануды (циклді) ұйымдастырудың арнайы құралдары бар. Көп бағдарламалау тілдерінде қайталану операторы екі элемнттен тұрады қайталану басынан және денесінен. Денесі, қайталану орындалатын операторлардан тұрады, ал қайталанудың басы осы операторлардан қайталанудың орындалуын ұйымдастырады. Қайталанудың басы оның денесінен бұрын орналасады. Си тілінде үш қайталану операторлар қолданылады **while, for, do.** While және for қайталанулары келесі сызба бойынша түрғызылған.

Қайталанудың басы
Қайталанудың денесі

do қайталану операторының құрылымы басқаша – онда қайталанудың денесі, осы дененің қайталанып орындалуын ұйымдастыратын құрылыммен шектелген. Сондықтан **do** қайталану операторының басы не тақырыбы туралы айту дұрыс емес.

Үш қайталану операторларда да қайталану денесі - бұл бөлек немесе фигуралы жақшаларға алынған операторлар тізбегінен тұратын күрделі оператор. **while** қайталану операторы былай жазылады (3.3-сурет):

while (шарт**)**
Қайталану_денесі

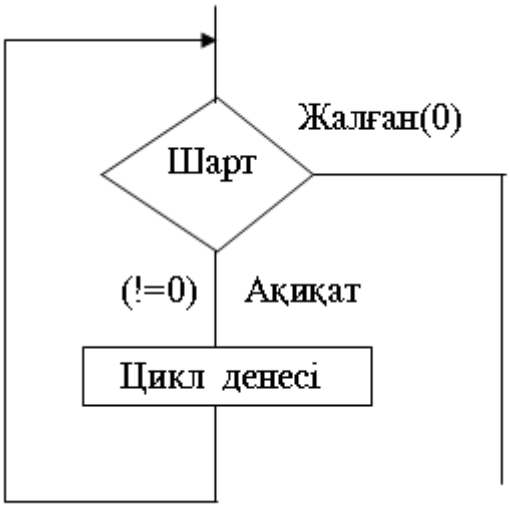
Шарт ретінде көбінесе қатынастар немесе логикалық өрнектер қолданылады. Егерде ол ақиқат болса, онда қайталану денесі осы шарт жалған болғанша қайталанып орындала береді. Бұл жерде шарттың ақиқаттылығы әрбір қайталану денесінің орындалуының алдында тексеріледі. Шарт арифметикалық өрнек болуы да мүмкін. Мұндай жағдайда қайталану шарт нөлге тең болғанша орындала береді. **do** қайталану операторының жазылуы (3.4-сурет):

do
қайталану_денесі;
while (шарт**);**

Шарт логикалық немесе арифметикалық өрнек.
do қайталану операторында қайталану денесі шартқа байланыссыз бір рет орындалады. Әрбір қайталанудың денесі орындалғаннан кейін шарт тексеріледі, егерде ол жалған болса, онда қайталанудың орындалуы тоқтатылады. **for** параметрлі қайталану операторының жазылуы (3.5-сурет):

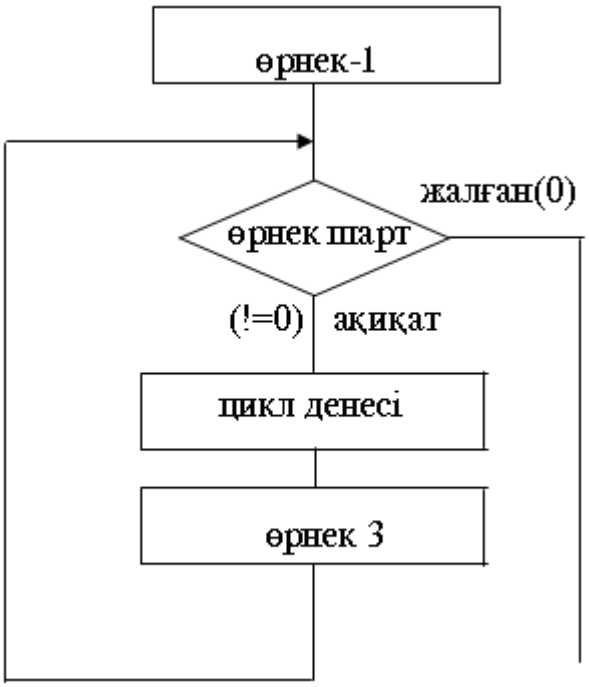
for (өрнек_1, өрнек_шарт; өрнек_3)
қайталану денесі;

Бірінші және үшінші өрнектер for операторында үтірмен бөлінген бірнеше өрнектерден тұруы мүмкін. Бірінші өрнек қайталану басталмай тұрғандағы іс-әрекетті анықтайды, яғни қайталануға бастапқы шартты анықтайды, көп жағдайда бұл меншіктеу өрнегі. Өрнек_шарт – бұл логикалық немесе арифметикалық өрнек. Ол қайталанудың операторының орындалуының жалғасуын немесе аяқталуын анықтайды. Егерде ол ақиқат болса, онда қайталану операторының денесі орындалады да өрнек_3 есептеледі. Қайталану операторларының сызбалары:



3.3-сурет. while қайталану операторының сызбасы

3.4-сурет. do қайталану операторының сызбасы

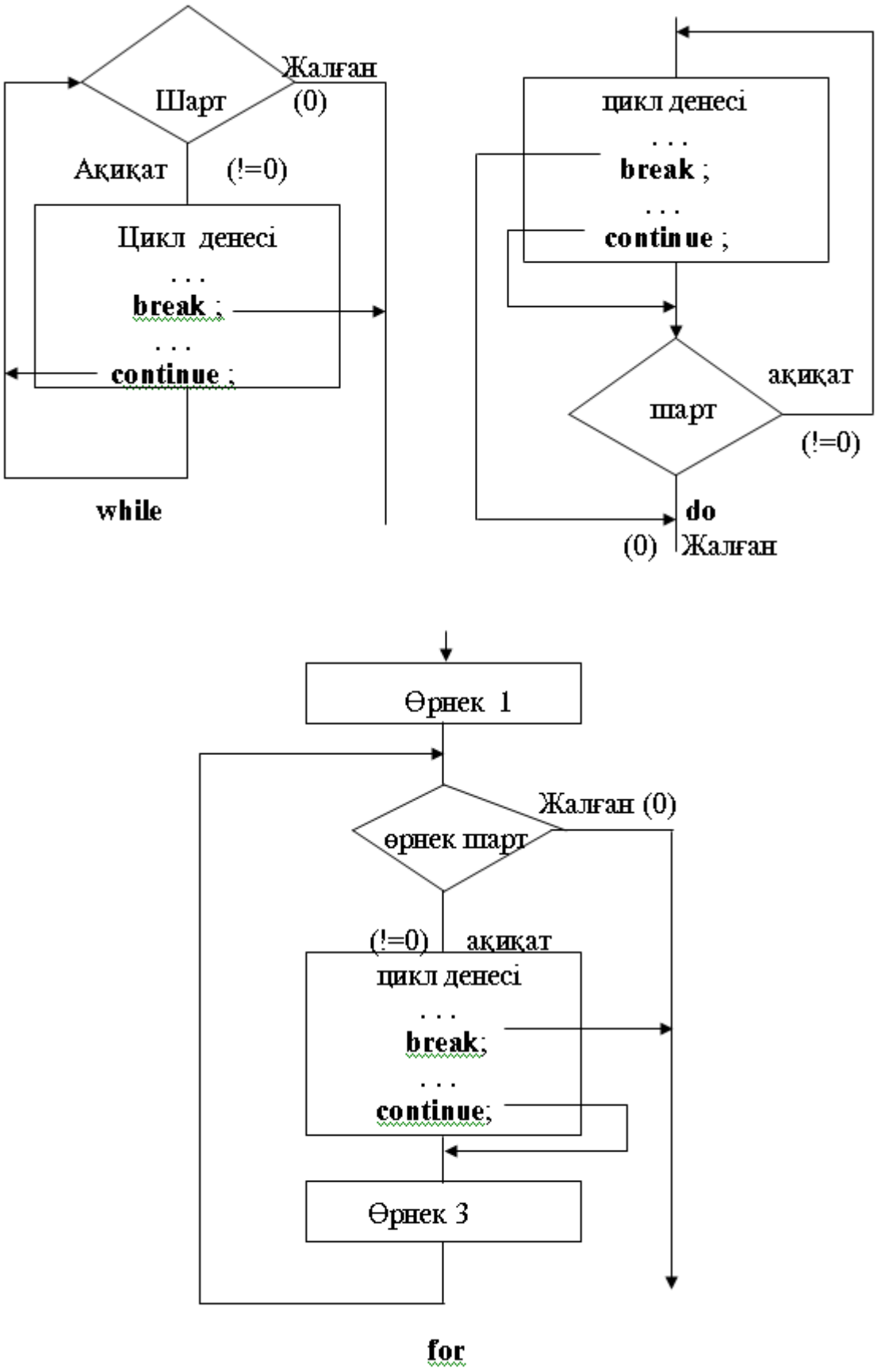


3.5-сурет. for параметрлі қайталану операторының сызбасы

break операторы. Бұл опертор ағымдағы қайталану операторының орындалуын тоқтатып, онан кейін орындалатын қайталану операторына басқаруды береді. Қайталануды тоқтату операторы көбінесе қайталанудың орындалу шартын қайталанудың басында және соңында емес, ортасында тексеру үшін қолданылады. Көбінесе бұл операторлың мынандай құрылымы қолданылады:

```
{
операторлар
if( шарт) break;
операторлар
}
```

continue операторы. Бұл оператор қайталанудың келесі итерациясына өту үшін қолданылады. **continue** операторы **break** операторына қарама-қарсы. Ол қайталану денесінің кез-келген нүктесінде ағымдағы итерацияны тоқтатып for немесе while өрнектерінде анықталған қайталануды жалғастыру шарттарын тексеруге өтеді. Шартты тексеру нәтижелеріне байланысты қайталану не тоқтатылады не қайталанудың жаңа итерациясы орындалады. **continue** операторын бір қайталанудан екінші қайталануға өткенде орындалатын операторлар тізбегі өзгергенде қолдану ыңғайлы немесе қайталанудың денесі тармақталып орналасқан болса қайталану операторларында **break** және **continue** операторларының орындалу сызбасы (3.6-сурет).



3.6-сурет. Қайталану операторларында **break** және **continue** операторларының орындалу сызбасы

3.4 Массивтер және қайталану операторларының сатылап орналасуы

Бағдарламалау тілдеріндегі “массив” ұғымының пайда болуы, математикадағы матрица ұғымына байланысты. Математикада матрицаның элементтерін индекстермен анықтайды. Матрицаның барлық элементтері не бүтін немесе нақты сан болады. Элементтердің мұндай біркелкілігі массивтің де қасиетіне кіре алады. Ол элементтердің типінен, массивтың атынан және оның көлеміне, яғни оның индекстерінің соңынан тұрады. Бұдан басқа оның анықтамасында әрбір индекстің қабылдай алатын мәндердің саны көрсетіледі.

Мысалы: int a[10]; он элементтен a[0],a[1], …, a[9] тұратын массивті анықтайды. float z[13][6]; Бірінші индексі 13 0-ден 12-ге дейін мән қабылдайтын, екінші индексі 6 0-ден 5-ке дейін мән қабылдайтын екі өлшемді массивті анықтайды.

Си тілінің синтаксисіне байланысты Си тілінде тек қана бір өлшемді массивтер анықталған, бірақ бір өлшемді массивтің элементі ретінде массивтерді қолдана беруге болады. Сондықтан екі өлшемді массив, массивтің ішіндегі массив ретінде анықталады. Осыған байланысты жоғарыда көрсетілген мысалыда - z массиві әрбір элементі float типіндегі 6 элементтен тұратын 13 элемент-массивтен тұрады. Кез-келген массив элементтерінің нөмері әрқашанда 0-ден басталады, яғни индекс 0-ден N-1 дейін өзгереді, бұл жерде N-индекс мәнінің саны.

Қайталану операторының сатылап орналасуы. Қайталану операторы денесінде кез-келген операторларды қолдануға болады, қайталану операторын сатылап да қолдануға да болады. Қайталану операторларын сатылап орналастырғанда **break** және **continue** операторларының орындалу ережелеріне мән берген жөн. Олардың әрқайсы тек қана өзі қолданылатын операторда орындалады. **break** операторы өзіне жақын орналасқан қайталану операторын орындалуын тоқтатады. **continue** операторы қайталану операторының орындалуын жалғастыру үшін шарттық тексерілуді қамтамасыз етеді.

Мысалы: $s = \sum_{j=1}^{10} \prod_{i=1}^{5} a_{ji}$ s-ті есептеу бағдарламасының негізгі бөлігі.

```
double  a[10][5];
        for (j=0,s=0.0; j<10; j++)
        {
        for ( i=0,p=1.0; i<5; i++)
        {
             if ( a[j][i]==0.0) break;
              p*=a[j][i];
        }
       if (i<5) continue;
        s+=p;
 }
```

Массивтерді инициализациалау. Массивтерді сипаттаған кезде оларды инициализациалауға болады. Яғни массивті сипаттаған кезде-ақ, оның

элементтеріне бастапқы мән меншіктеуге болады. Мағынасы бойынша инициализациалау дегеніміз - объектіні сипаттаған кезе-ақ, сол объектіге белгілі бір мәнді меншіктеу. Инициализацианы қолданғанда массивті сипаттау пішіні де өзгереді. Мысалы бір өлшемді массивтің элементтерінің санын көрсетпей-ақ, оған бастапқы мән беруге болады.

```
double d[ ] = {1.0, 2.0, 3.0, 4.0, 5.0};
```

Бұл мысалда компиляторда массивтің ұзындығын, оған меншіктелген бастапқы мәндердің санымен анықталады. Мұнда d[0]=1.0, d[1]=2.0, d[2]=3.0, d[3]=4.0, d[4]=5.0

Егерде массивті сипаттаған кезде оның көлемі көрсетілген болса, онда оған меншіктелетін бастапқы мәндердің саны, массивте көрсетілген элементтердің санынан артық болмауы керек. Егерде меншіктелген бастапқы мәндердің саны, массивтің ұзындығынан кем болса, онда бастапқы мәндер массивтің алғашқы элементеріне меншіктеледі.

```
int  M[8]={8,4,2};
```

Мына көрсетілген мысалда бастпақы мән массивтің M[0], M[1], M[2] элементтеріне меншіктеледі. Көпөлшемді массивтерді инициализациалау ережесі, элементтері массив болатын бір өлшемді массивтің анықтамасына сәйкес. Мысалы 3 қатардан және 2 бағаннан тұратын, элементтері нақты тип болатын екі өлшемді A массивінің элементтеріне бастапқы мән меншіктеуді былай жазуға болады:

```
double  A[3][2]={{10,20}, {30,40}, {50,60}};
```

Бұл мысалда

A[0][0]=10; A[0][1]=20; A[1][0]=30; A[1][1]=40; A[2][0]=50; A[2][1]=60.

Мұндай нәтижені басқаша жазып та алуға болады.
Мысалы:

```
double  A[3][2]={10,20, 30,40, 50,60};
```

Инициализацияның көмегімен массивтің тек кейбір элементтеріне мән меншіктеуге болады. Мысалы массивтің тек бірінші бағанның элементтеріне мән меншіктеу үшін былай жазуға болады.

```
double  Z[4][6]={{1},{2},{3},{4}};
```

Z[0][0]=1; Z[1][0]=2; Z[2][0]=3; Z[3][0]=4.

3.5 Функциялар

Функциялар анықтамасы. Алдыңғы тақырыптарда айтылғандай, Си тіліндегі бағдарлама – функциялар жиынтығынан тұрады. Функцияның сипаттамасында сол функцияны негізгі бағдарламада шақырғанда орындалатын іс-әрекеттердің тізбегі, функцияның аты, сол функция орындалғанда шығатын нәтиженің типі және функцияны шақырғанда нақты аргументтермен ауыстырылатын формалды параметрлердің жиынтығы көрсетіледі.

Функцияны шақырғанда орындалатын іс-әрекеттер осы фукцияның денесін анықтайды. Функцияның денесі фигуралы жақшаларға алынған күрделі оператордан немесе блоктан тұрады. Функцияның аты, оның нәтижиесінің типі, параметрлердің жиынтығы және олардың қасиеттері функцияның тақырыбында жазылады. Функцияны сипаттаудың классикалық құрылымы:

```
нәтиженің _типі
функцияның_аты (формальды_параметрдің_тізбегі)
формалды_параметрдің _анықталуы
            {
              объектілерді_анықтау
              орындалатын_операторлар
             }
```

Бұл жерде бірінші үш қатар функцияның тақырыбы, одан кейінгі төрт қатар функцияның денесі.

Функцияның аты - бұл кез-келген идентификатор. Бірақ функцияның аты қызметші сөздермен, басқа функциялардың аттарымен сәйкес келмеуі керек. Формалды параметрдің тізбегі - бұл бір-бірінен үтір арқылы бөлінген идентификаторлар жиынтығы. Формальды параметрдің анықталуы - олардың қасиеттерін сипаттайды.

Мысалы:

```
double   f  (a,b,c)
 int  a;
 float  b;
 double c;
  { /* функцияның денесі */}
```

Қазіргі кезде көбінесе Си тілінде формалды параметрдің тізбегі, олардың сипатталуымен біріктірілген функцияның сипатталуы қолданылады. Мұндай сипатталудың құрылымы:

```
нәтиженің _типі
функцияның_аты (формалды_параметрдің _сипатталуы)
            {
              объектілерді_анықтау;
              орындалатын_операторлар;
```

}

Мысалы:

```
double  f ( int  a, float  b, double  d )
{ /* функцияның денесі */}
```

Функцияның денесінің маңызды операторы болып функциядан оның шақыру нүктесіне қайтаратын оператор есептеледі:

```
return   өрнек;
немесе
      return;
```

Бұл оператордағы өрнек функцияға қайтарылатын мәнді көрсетеді. Егерде функция ешқандай мән қайтарылмайтын болса, онда екінші формасы return; қолданылады. Егерде бағдарламашы бағдарламада return операторын жазбаған болса, онда бұл жағдайда компилятор автоматты түрде функцияның денесінен кейін return операторын қояды.

Функцияны шақыру. Функцияны шақыру үшін жай жақшалар қолданылады:

функцияның_аталуы (нақты_ параметрлердің_ тізбегі)

Бұл жерде функцияның_аталуы дегеніміз - функцияның аты нақты_ параметрлердің_ тізбегі немесе аргументтер дегеніміз - бұл өрнектер тізбегі, олардың саны формалды параметрлердің санымен бірдей болуы керек. Формалды және нақты параметрлердің арасында типтері бойынша сәйкестік болуы керек. Егерде олар сәйкес болмаса, онда компилятор мүмкін болған жағдайда типтерді түрлендіру командаларын қосады.
Мысалы:

int g (int, long); сипатталуы.

Егерде бағдарламада бұл функция былай шақырылған болса
g (3.0+m, 6.4e +2);

Онда бұл жағдайда екі параметрде double типті. Сондықтан компилятор бұл екі параметрлердің біріншісін int типіне, екіншісін long типіне ауыстырады.
g((int) (3.0+m), (long) (6.4e+2))

Функцияның шақырылуы өрнек болғандықтан функциясының денесіндегі операторлар орындалғаннан кейін қандайда бір мән қайтарылады. Сол қайтарылатын мәннің типі функция сипатталғанда функцияның атының

алдында көрсетілетін типке міндетті түрде сәйкес болуы керек. Мысалы мына функция:

```
float ft ( double x, int n)
{
  if ( x<n) return  x;
  return  n;
}
```

Әрқашанда float типіндегі мән қайтарады. Return операторындағы өрнектер компилятор автоматты түрде типтерді түрлендіру командаларын қосады. Яғни бағдарламашыға көрінбейтін мынадай операторлар орындалады.

```
return (float) x;
return (float) n;
```

Функцияның шақырылуы әрқашанда өрнек болып есептеледі, бірақ бұл өрнекті бағдарлама жазған кезде оның жазылуы функция орындалғанда қайтаратын мәннің типіне байланысты болады. Егерде қайтаратын мәннің типіне void көрсетілсе онда мұндай функция ешқандай нәтиже қайтармайтын функция болып есептеледі. Мұндай функция бөлек оператор - өрнек түрінде шақырылады:

```
функцияның_аты (нақты_параметрдің_тізбегі);
```

Мысалы келесі функция void типіндегі мән қайтарады.

```
void printf (int gg, int mm, int dd)
{
  printf (“\n  год: %d”,gg);
  printf(“\t месяц: %d”,mm);
  printf(“\t  день:%d”,dd);
}
```

Егерде оны бағдарламада былай шақырса printf (2011,09,18);
Онда бұл функция орындалғанда экранға год: 2011, месяц:09 , день : 18 шығады. void типіндегі параметрі жоқ функцияны қолдануға болады.
Мысалы:

```
# include <studio.h>
void real_Time (void)
{
printf (“\ время:% s”,__ Time__”(час: мин: сек.)”);
}
онда осы функцияны шақырғанда   real _Time ();
```

онда ағымдағы уақытты көрсетеді. Время: 14:16:25 (час: мин: сек.)

3.6 Таңдау операторы switch

switch операторы бірнеше мүмкін болған жағдайдың біреуін таңдап алады. Оператор көрсеткіштен және әрқайсысы бір немесе бірнеше таңдау тұрақтылармен таңбаланған операторлар тізбегінен тұрады.
Жазылуы:

```
switch (көрсеткіш)
{    case тұрақты1: оператор_1;
     case тұрақты2: оператор_2;
       ……
    default: операторлар;
}
```

Мұндағы тұрақтылармен көрсеткіш нәтижесі бүтін немесе символдық тип болуы керек. Таңдау операторы орындалғанда, алдымен switch сөзінен кейінгі көрсеткіштің мәні табылып, сол мән кезекпен case сөзінен кейінгі тұрақтылармен салыстырылады. Одан кейін сол мәнге сәйкес тұрақтыдан кейінгі оператор орындалады. Егерде көрсеткіштің мәні бірде-бір таңбаға сәйкес келмесе, онда default қызметші сөзінен кейінгі оператор орындалады. Таңдау тұрақтыларының типі міндетті түрде көрсеткіштің типімен сәйкес келуі керек.
Мысалы:

```
# include <studio.h>
void main()
{
   int i;
 printf (“\n ондық сан енгізіңіз:”);
 scanf(“%d”, &i);
switch (i)
{
  case 1: printf (“%d-бір \n”,i);
   break;
 case 2: printf (“%d-екі \n”,i);
   break;
case 3: printf (“%d-үш \n”,i);
   break;
case 4: printf (“%d-төрт \n”,i);
   break;
case 5: printf (“%d-бес \n”,i);
   break;
case 6: printf (“%d-алты \n”,i);
```

```
  break;
case 7: printf (“%d-жеті \n”,i);
  break;
……..
default: printf( “%d- бұл сан емес\n”,i);
return;
}
}
```

Бақылау сұрақтары

1. Объект және функция дегеніміз не?
2. Си тіліндегі бағдарламалар неше өңдеу кезеңінен өтеді.
3. Препроцессордың қызметі не?
4. Си тіліндегі бағдарлама құрылымына мысал келтіріңіз.
5. Экранға шығару функциясы.
6. Си тіліндегі операцияларды атаңыз және оларға мысал келтіріңіз.
7. Шартты операторлардың орындалу тәртібі.
8. Шартсыз өту операторына мысал келтіріңіз.
9. Си тілінің қайталану операторлары.
10. Си тілінде массивтер түсінігі.
11. Си тілінің функциялары
12. Таңдау операторына мысал келтіріңіз.

IV БӨЛІМ. ПРЕПРОЦЕССОРЛІК ҚҰРЫЛҒЫ

4.1 Препроцессорды өңдеудің командалары және кезеңдері

Си тіліндегі бағдарламаның қажетті компонентерінің бірі болып препроцессор есептеледі. Препроцессордың міндеті-бағдарламаның бастапқы мәтінін, оны компиляция жасағанға дейін өңдеу. Препроцессорлік өңдеу, бірінен кейін бірі кезекпен орындалатын бірнеше кезеңдерден тұрады. Нәтижесінде мынандай кезекпен орындалады:

- барлық жүйелік тәуелді белгілер стандартты кодтарға айналдырылады;
- әрбір '\' символдардың жұбы және сөйлемнің соңы, олардың арасындағы бос орындар алынып тасталынып, бастапқы мәтіннің келесі сөйлемі осы жұп символдар орналасқан сөйлемге қосылады;
- мәтіннің әрбір сөйлемінен директивалармен препроцессордың лексемалары табылып, ал әрбір түсіндірме бос аралықтың бір символымен ауыстырылады;
- препроцессордың директивалары орындалып, макроауыстырулар жүргізіледі;
- символдық тұрақтылардағы және символдық сөйлемдердегі эскейп - тізбектер, мысалы '\n', олардың эквиваленттеріне, яғни сәйкес сандық кодтарға айналдырылады;
- аралас символдық сөйлемдер, бір сөйлемге біріктіріледі;
- әрбір препроцессорлік лексема Си тіліндегі мәтінге ауыстырылады;

Препроцессорлік лексемалар немесе препроцессордің лексемалары дегеніміз - символдық тұрақтылар, тақырыптық файлдың аты, идентификаторлар, операция таңбалары, препроцессорлық сандар, сөйлемдер және бос орыннан басқа кез-келген символдар.

Препроцессордың директиваларының өңдеу кезеңдеріне толығырақ тоқталайық. Ол орындалғанда келесі іс-әрекеттер жүзеге асырылады:

- идентификаторларды алдын-ала дайындалған символдық тізбегімен ауыстыру;
- көрсетілген тақырыптық файлдардағы мәтінді бағдарламаға енгізу;
- бағдарламадан мәтіннің бөлек бөліктерін алып тастау (шартты компиляция);
- макроауыстырулар, белгілерді параметрлік текстермен ауыстыру. Оны препроцессор белгілі бір параметрлерді (аргументтерді) есепке алып жасайды.

Препроцессордың директивалары. Препроцессордың жұмысын басқару үшін, яғни керекті іс-әрекеттерді жүзеге асыру үшін препроцессордың командалары, яғни директивалары қолданылады. Олардың әрқайсы бөлек жолда # символдан кейін орналасады.

Жазылуы:

директиваның_аты препроцессордың_лексемалары.

Мына ’#’ символдың алдына немесе одан кейін бос орын қоюға болады. Препроцессорлік директиваның соңы болып, мәтіндік сөйлемнің соңы есептеледі. Келесі препроцессорлік директивалар анықталған:

# define	–	препроцессорлік идентификатордың немесе макростың сипатталуы;
# include	–	тақырыптың файлдағы мәтінді бағдарламаға қосу;
# undef	–	препроцессорлік идентификатордың немесе макростың сипаттамасын жою;
# if	–	өрнек - шартты тексеру;
# ifdef	–	Идентификатордың анықтамасын тексеру;
# else	–	**# if** директивасының бір тармағы;
# endef	–	**#if** шартты директивасының соңы;
# elif	–	құрамдық директива **# else/# if;**
# line	–	келесі төменгі жолдың нөмерін ауыстыру;
# error	–	трансляция кезіндегі қатені шығару мәтінін құру;
#	–	бос директива.

Препроцессорлік директивалардан басқа үш препроцессорлік операциялар бар:

defined	–	операнданың ақиқаттылығын тексеру;
##	–	препроцессорлік лексемаларды біріктіру;
#	–	операнданы символдар тізбегіне ауыстыру.

 # define директивасының бірнеше түрлері бар. Олар әрқайсына өзіне сәйкес символдар тізбегімен ауыстырылатын препроцессорлік идентификаторларды анықтайды. Бағдарламаның келесі мәтінінде препроцессорлік идентификаторлар, алдын - ала анықталған символдар тізбегімен ауыстырылады. **#include** директивасы бағдарламаның мәтініне көрсетілген тақырыптық файлдардағы мәтіндерді қосады. **#undef** директивасы **#define** директивасында орындалған іс - әрекеттерді жояды. Яғни **#define** директивасында идентификатордың атын жоққа шығарады. **#if, #ifdef, #ifndef, #else, #endif, #elif** директивалары бағдарлама мәтінінің шартты өңделуін ұйымдастырады. Оларды қолданғанда бағдарламаның барлық мәтіні емес, тек осы директивалардың көмегімен белгіленген бөліктері ғана компиляция жасалынады.

 # line директивасы бағдарламадағы жолдардың нөмерлерін басқаруға мүмкіндік береді. Файлдың аты және қажетті жолдың бастапқы нөмері **# line** директивасында көрсетіледі.

#error директивасы, қателер туындағанда анықтамалық (диагностикалық) хабарландыру мәтінін шығарады.

- ешнәрсе орындамайды.

Жоғарыда айтылғандай идентификаторды, алдын-ала анықталған символдар тізбегімен ауыстыру үшін **#define** директивасы қолданылады. Жазылуы:

 # define идентификатор ауыстыру_ жолы.

Бұл директива өңделетін бағдарламаның кез-келген жерінде орын ала береді. Біріншіден идентификаторлы препроцессорлік идентификатор ретінде сипаттайды, екіншіден оны өзінен кейін орналасқан символдар тізбегімен ауыстырады.
Мысалы:

#define pi 3.14

Егерде **#define** директивасының басқа директивада анықталған препроцессорлік идентификатор қолданылса, онда кезекпен орналасқан ауыстырулар тізбегі орындалады.
Мысалы:

```
# include <limits.h>
# define  RANGE ((INT_MAX)-(INT_MIN)+1)
……
/*RANGE – int үшін анықталған диапазон */
……
int    RANGE_T = RANGE /8;
……
```

Бұл бағдарламада препроцессор сөйлем бойынша кезекпен мәтінді оқиды. **#include** <limits.h> директивасын тауып осы тақырыптық файлдағы INT_MAX (бүтін сандар үшін ең үлкен мән) және INT_MIN (бүтін сандар үшін ең кіші мән) тұрақтыларының мәнін қосады. Осыған байланысты бағдарлама төмендегіше түрлендіріледі:

```
# define   INT_MAX   32767
# define   INT_MIN  -32768
……
# define  RANGE ((INT_MAX)-(INT_MIN)+1)
……..
int    RANGE_T = RANGE /8;
……..
```

Бұл бағдарламада **# include** директивасы жоқ, оның орнында оған сәйкес мәтін ауыстырылды. Бағдарламаның келесі кезегінде мыналар орындалады:

```
# define  RANGE ((32767)-(-32768)+1)
……
```

```
int    RANGE_T = RANGE /8;
```

Ең соңында бағдарлама мәтіні мынадай болады:

```
……
int    RANGE_T =((32767)-(-32768)+1)/8;
……..
```

Бұл жерде барлық **#define** директивалары алынып тасталып компиляцияға ыңғайлы мәтін қалды.

#define директивасының жазылу форматына оралайық **#define** идентификатор ауыстыру – жолы. Егерде ауыстыру жолы өте ұзын болса оны келесі қатарға жазуға болады. Ол үшін бірінші қатардың соңына ’\’ символы қойылады.

Мысалы:

```
#define STRING6 “\n GAME Over!-\
     Ойын аяқталды!”
     ……..
     printf (STRING6);
```

Онда экранға - GAME Over! - Ойын аяқталды! болып шығады. Егерде бағдарламада массивтермен жұмыс істейтін болсақ, онда олардың ұзындығын препроцессордің көмегімен көрсетуге болады.

Мысалы:

```
# define k 40
void main ( )
{
  int M[k][k];
  float A [2*k+1],
  float B[k+3][k-3]
    …….
```

Бағдарламаны осылай жазғанда барлық массивтердің ұзындықтарын өзгерту үшін тек қана **#define** директивасындағы тұрақтының мәнін өзгертсек болғаны. Егерде бағдарламада бір айнымалының мәнін немесе бір мәтінді экранға қайта-қайта жиі шығару керек болса, онда ол үшін де **# define** директивасының қолданған ыңғайлы.

Мысалы:

```
# define k 50
# define PE printf (“\n  Элементтер саны k=%d”,k);
…….
          PE;
…….
```

Онда экранға былай жазылады: Элементтер саны k=50

Бағдарламада препроцессордың көмегімен бір идентификаторды бірнеше қолдануға болады.
Мысалы:

```
# define  M16
# define  M  ’C’
```

define M “C” деп жазатын болсақ онда Си бағдарламалау тілінің компиляторы қате шығарады. Мұндай қателерді болдырмау үшін **#define** директивасының көмегімен мәні анықталған идентификатордың мәнін басқа мәнге ауыстыру үшін **# undef** директивасы қолданылады.
Жазылуы:

 # undef идентификатор

Мұндай директива орындалғаннан кейін идентификатор қайтадан анықталмаған болады және оны қайта қолдануға болады.
Мысалы:

```
# define  M 16
# undef  M
# define  M  ’C’
# undef  M
# define  M “C”
```

#undef директивасын бірнеше бағдарламашылар және әртүрлі уақытта жазылатын үлкен бағдарламаларды жасағанда қолданған өте ыңғайлы. Өйткені мұндай жағдайда әртүрлі объектілерге бірдей ат берілуі мүмкін.

4.2 Тақырыптық файлдарда анықталған мәтіндерді бағдарламаға қосу

Тақырыптық файлдардағы мәтінді бағдарламаға қосу үшін **# include** директивасы қолданылады. Оның үш түрі бар:

 # include <файл_аты>
 # include “файл_аты”
 # include макростың_аты.

Бұл жердегі макростың аты дегеніміз **#define** директивасында анықталған идентификатор немесе макрос. Осы уақытқа дейін біз бағдарламада **# include** директивасының тек бірінші түрін ғана қолданып келдік. Ереже бойынша егерде директивада файл аты бұрыштық жақшалардың ішінде жазылған болса, онда ол файлды препроцессор стандартты жүйелік каталогтардан іздейді.

Егерде файлдың аты тырнақшалардың ішіне жазылған болса, онда процессор файлды ағымдағы каталогтан іздейді онан кейін стандартты жүйелік каталогтардан іздейді. Си бағдарламалау тілімен жұмыс жасаған кезден бастап-ақ, енгізу шығару құрылғыларының қажеттілігі туындайды. Ол үшін бағдарлама мәтінінің бастапқы жолында **#include <stdio.h>** директивасы жазылады. Бұл директиваны орындаған кезде, препроцессор енгізу шығару библиотекасымен байланыс орнатады. **stdio.h** файлын препроцессор стандартты жүйелік каталогтардан іздейді.

Қабылдаған келісімге байланысты **.h** суффиксі компилятордың библиотекаларымен жұмыс істегенде қолданылатын, библиотекалық функциялардан, анықтаулардан және типтер мен тұрақтыларды сипаттаулардан тұратын файлдарға қойылады. Мұндағы файлдарды Си тілінде тақырыптық файлдар деп атайды. **stdio.h** тақырыптық файлдан басқа кез-келген стандартты немесе арнайы дайындалған тақырыптық файлдар қолданылады. Стандартты тақырыптық файлдарда анықталған барлық аттар алдын – ала анықталған аттар болып табылады:

assert.h – бағдарламаны жоспарлау (диагностикалау) немесе тексеру.
ctype.h – символдарды тексеру және түрлендіру.
errno.h - қателерді тексеру.
float.h – нақты мәліметтермен жұмыс жасау.
limits.h- бүтін санды мәліметердің ең үлкен және ең кіші мәндері.
math.h – математикалық есептеулер.
signal.h – күтпеген жағдайлардағы өңдеу.
stdarg.h –сандар параметрлерінің айнымалыларын анықтау.
stddef.h - қосымша анықтаулар.
stdio.h – енгізу-шығару құралдары.
stdlib.h – жалпы қолданылатын функциялар (жадымен жұмыс жасау).
string.h –символдардың жолдарымен жұмыс жасау.
time.h – күн және уақытты анықтау.

Бұдан басқа **mem.h, alloc.h, conio.h, dos.h** файлдары қолданылады. Әрбір операциялық жүйелерде графикалық функциялармен жұмыс жасайтын тақырыптық файлдар қолданылады. Си тілінде **graphics.h** графикалық тақырыптық файл қолданылады. Бұл тілде әрбір орындалатын функцияны бөлек файл ретінде сақтап, қажет болғанда сол функцияны бағдарламада файлдан шақырып қолдануға болады.

Функция жазылған файлдың аты тырнақша ішіне жазылып, **#include** директивасымен шақырылады.
Мысалы:

#include “count.c”
#include “gauss.c”

Осы мысалда көрсетілген бағдарламаның мәтінін өте қысқа етіп, тек қана препроцессордың директиваларын қолданып жазуға болады.

4.3 Шартты компиляция. Тармақталу директивалары

Си тіліндегі шартты компиляцияны келесі директивалар қамтамасыз етеді. Олардың негізгі қызметі компиляцияны басқару емес, бағдарлама мәтінін препроцессорлік өңдеуден өткізу.

 # if бүтінсанды_тұрақты_ өрнек
 # ifdef идентификатор
 # ifndef идентификатор
 # else
 # endif
 # elif

Бірінші қатардағы үш директива шарттың орындалуын тексереді, келесі екеуі осы тексерілген шарттың орындалу аумағын анықтайды (**#elif** директивасын кейінірек қарастырамыз). Шартты компиляцияның директиваларын қолданудың жалпы құрылымы мынандай:

 # if шарт
мәтін_1
 # else
мәтін_2
 # endif

Бұл жердегі **#else** мәніндегі құрылымының жазылуы міндетті емес. Мәтін_1 компиляция жасалатын мәтінге тек **#if** директивасындағы шарт ақиқат болған жағдайда ғана қосылады. Егерде шарт жалған болса, онда **#else** директивасынан кейінгі мәтін компиляцияға беріледі. Егерде **#else** директивасы көрсетілмесе онда ешқандай іс-әрекет орындалмайды. **#if** директиваларының формаларының арсындағы айырмашылық мынада аталған директивалардың біріншісінде

#if бүтінсанды_тұрақты_өрнек

Тұрақты өрнектің мәні тексеріледі. Олар бүтін тұрақтылар және иднетификаторлар болуы мүмкін. Идентификаторлардың мәні препроцессорде анықталуы мүмкін, егерде анықталмаса олардың мәні нөл болып есептеледі. Егерде тұрақты - өрнек нөлге тең болмаса онда тексерілетін шарт ақиқат болып есептеледі.

Мысалы келесі директиваның орындалу нәтижесінде:

 # if 5+4
мәтін_1
 # endif

Мәтін_1 әрқашанда компиляцияланатын бағдарламаға қосылады. Екінші **# ifdef** идентификатор директивасында, **#ifdef** қызметші сөзінен кейін

орналасқан идентификатор **#define** директивасында бұған дейін анықталғаны тексеріледі. Егерде идентификатор анықталған болса, онда келесі мәтін компиляторда қолданылады.

Үшінші **#ifndef** идентификатор, керісінше келесі мәтін егерде иднетификатор **#define** директивасында анықталмаған немесе оның анықталуы **#undef** директивасмен жойылған жағдайда компиляцияланатын бағдарламаға қосылады. Бағдарламаның ағымдағы мәтінінде тармақталу операцияларын ұйымдастыру үшін **#elif** директивасы қолданылады.
Жазылуы:

 # elif бүтінсанды_тұрақты_өрнек

Бұл директиваның бағдарламада қолданғандағы құрылымы мынандай:

 # if шарт
if үшін_ мәтін
 # elif өрнек_1
мәтін_1
 # elif өрнек_2
мәтін_2
……
 # else жағдайы_үшін_мәтін
 # endif

Препроцессор алдымен **#if** директивасындағы шартты тексереді. Егерде шарт жалған болса, онда өрнек-1 есептеледі, егерде өрнек-1-де жалған болса, онда өрнек-2 есептеледі т.с.с. Егерде барлық өрнектер жалған болса, онда **#else** директивасынан кейінгі мәтін компиляцияланатын бағдарламаға қосылады. Егерде мәтіндегі директивалардан кейін орналасқан өрнектің біреуі ақиқат болса, онда компиляцияланатын бағдарламаға сол өрнектен кейінгі мәтін қосылады.

4.4 Препроцессор құрылымдарын макростармен ауыстыру

Макрос дегеніміз - анықтамасы бойынша бір символдар тізбегін екінші символдар тізбегімен ауыстыру. Ауыстыруларды орындау үшін соған сәйкес макростармен берілуі керек.

#define директивасының көмегімен бағдарламашы базалық немесе кез-келген типтерге өзінің таңбаларын енгізе алады.
Мысалы мына директива:

#define REAL long double

long double типіне **REAL** деген ат береді. Ары қарай бағдарламада объектілерге l**ong double** типін беру үшін **REAL** идентификаторын қолдануға болады.

Мысалы:

```
REAL  x, array[6];
```

Бұдан басқа параметрлі макроанықтаулар үлкен мүмкіндік береді. Жазылуы:

#define аты(параметрлер_тізімі) ауыстыру_жолы

Бұл жерде аты-макростың_аты (идентификатор), параметрлер_тізімі-үтірмен бөлінген идентификаторлар тізбегі. Макростың атымен параметрлер тізімін ашатын жақша арасында бос орын болмауы керек. Мұндай макросты шақыру үшін келесі құрылым қолданылады.

Макростың_аты (аргументтер_тізімі)

Тізімде аргументтер үтірмен бөлінеді. Әрбір аргумент-препроцессорлік лексема болып есептеледі.
Мысалы:

```
#define   max (a,b)   (a < b ? b : a)
```

Бұл өрнек бағдарламада екі аргументтің ішінен ең үлкен мәнін таңдап алуға мүмкіндік береді. Мұндай өрнекті макросты шақыру былай жазылады max(X,Y). Бұл макрос орындалғанда мынадай өрнекпен ауыстырылады (X<Y? Y:X). Егерде X<Y шарты ақиқат болған жағдайда макростің мәнін қайтарады, жалған болған жағдайда Х-тің мәнін қайтарады. Ауыстыру жолын ұйымдастыратын лексемалар тізбегіне екі операцияны **“#”** және **”##”** қолдануға болады. Бірінші операция параметр алдына қойылады, ал екіншісі кез-келген екі лексеманың арасына қойылады. **“#”** операцияны қолданғанда осы жолдағы параметрді ауыстыратын мәтін жақшалардың ішіне жазылуы тиіс.
Мысалы:

```
# define print(A)  printf ( #A”=%f”,A)
```

printf (A) макросын шақырғанда, А параметрінің орнына нәтижесі нақты тип болатын кез-келген өрнекті қолдануға болады.
Мысалы:

```
printf(sin(a/2))-макросты шақыру;
printf(“sin(a/2)””=%f”,sin(a/2));-макростың орындалуы.
```

Бағдарламаның үзіндісі:

```
double  a= 3.14159;
```

printf (sin(a/2));

нәтижесі

sin (a/2)=1.0

”##” операциясы ауыстыру жолының лексемаларының арасына қойылып, осы ауыстыру жолына қойылатын лексемаларды бір-біріне мағынасы бойынша қосуға немесе конкатенция жасауға арналған **”##”** операциясын түсіну үшін параметрлері бірдей келесі макростардың орындалуын қарастырайық:

define zero(a,b,c,d) a (bcd)
define one (a,b,c,d) a (b c d)
define two (a,b,c,d) a (b##c##d)

Макросты шақыру	Макростың орындалуы
zero (sin, x, +, y)	sin(bcd)
one (sin, x, +, y)	sin(x + y)
two (sin, x, +, y)	sin(x+y)

zero() макросында “bcd” тізбегі, бөлек бір иднетификатор ретінде қабылданады. b, c, d параметрлерін ауыстыру орындалмайды. One() макросының ауыстыру жолында аргументтер бір-бірімен бос орын арқылы бөлінген, ал ол нәтижеде сақталған. two() макросының ауыстыру жолында **”##”** операциясы қолданылып, аргументтері мағанасымен қосылып нәтиже беретін макрос орындалады.

4.5 Көмекші директивалар

Бұл тақырыпта қарастырылатын директивалар практикада көп қолданылады. Бағдарламада жолдарды нөмерлеу үшін:

#line тұрақты

директивасы қолданылады. Бұл директива компиляторға, мәтіннің келесі жолының, бүтін ондық тұрақтымен анықталатын нөмері бар екендігін хабарлайды. Директива бір уақытқа жолдың нөмерін ғана емес бағдарламада қолданылатын файлдың атын да өзгерте алады.

Жазылуы:

line тұрақты “файлдың_аты”
error лексемалар_тізбегі

#**error** директивасы орындалғанда, лексемалар тізбегінде анықталған сараптамалық хабарландыруды шығарады.

#**error** директивасы көбінесе шартты директивалармен бірге қолданылады.

Мысалы:

```
# define  NAME 5
……….
# if (NAME!=5)
# error NAME 5-ке тең болуы керек!
……..
```

NAME идентификаторының мәні 5-ке тең болмаған жағдайда келесі хабарландыру шығады:

Error < файлдың_аты > <жолдың_нөмері>:
Error directive: NAME 5-ке тең болуы керек!

Бақылау сұрақтары

1. Препроцессорлік өңдеу кезеңдерін атаңыз.
2. Препроцессордің директивалары.
3. Қандай препроцессорлік операцияларды білесіз, оларға мысал келтіріңіз.
4. Препроцессордің көмегімен идентификаторды қалай бірнеше рет қолданамыз.
5. Тақырыптық файлдардағы мәтінді мәтінді бағдарламаға қосу жолдары.
6. Шартсыз копилляция директивалары.
7. Тармақталу директивалары.
8. Препроцессор құрылымдарын макроспен ауыстыру жолдары.
9. Макрос дегеніміз не?
10. Си тілінің көмекші директивалары.

V БӨЛІМ. КӨРСЕТКІШТЕР, МАССИВТЕР, ЖОЛДАР

Өткен бөлімдерде Си тілінің барлық базалық (негізгі) типтері көрсетілді. Оларды анықтау және сипаттау үшін келесі қызметші сөздер қолданылды: char, short, int, long, signed,unsigned,float, double, enum, void.

Си тілінде базалық негізгі типтерден басқа, әрқайсысы жай типтердің негізінде алынатын туынды типтерді алудың үш әдісі анықталған:

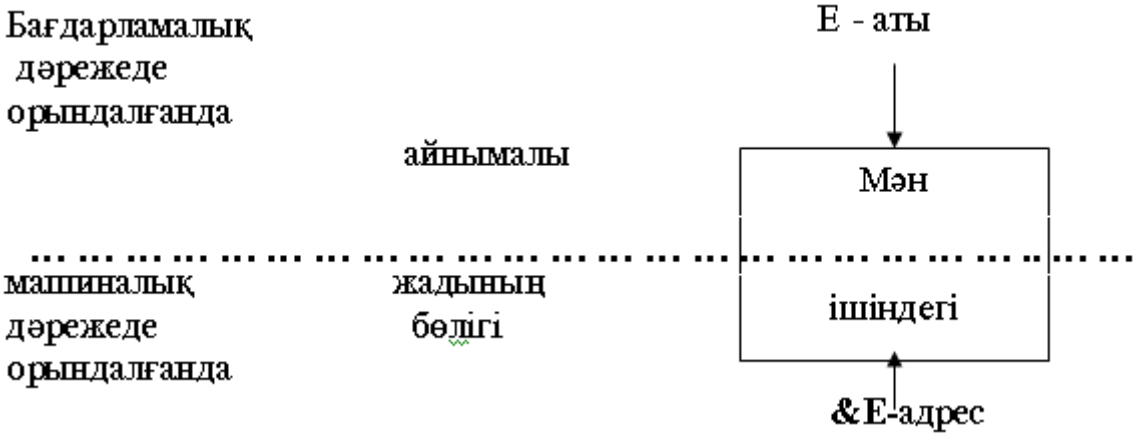
- берілген типтің элементтерінің массиві;
- берілген типтің объектісінің көрсеткіші;
- берілген типті мән қайтаратын функция.

Массивтермен және функциялармен біз бірқатар таныстық, ал бұл бөлімде көрсеткіштерге толығырақ тоқталып өтеміз.

5.1 Объектілердің көрсеткіштері

Адрестер және көрсеткіштер. Көрсеткіш ұғымын түсіну үшін айнымалы ұғымына оралайық. Бағдарламадағы әрбір айнымалы - бұл аты және мәні бар объект. Аты бойынша айнымалы шақырамыз және оның мәнін анықтаймыз. Меншіктеу операторында сол жағында орналасқан айнымалының атына, оң жағындағы өрнектің мәні сәйкестендіріледі. Машинада орындалғанда, айнымалының аты, жадыдағы өзіне бөлінген бөліктің адресіне сәйкес келеді, ал айнымалының мәні сол жадының бөлігінің мазмұнына сәйкес келеді.

Айнымалының аты мен адресінің арасындағы қатынасты келесі суреттегідей бейнелеуге болады (5.1-сурет).



5.1-сурет. Айнымалының аты мен адресінің арасындағы қатынас

Бұл суретте айнымалының аты адреспен анық байланыспаған, бірақ мысалы E=C+B меншіктеу операторында E-айнымалының аты жадының қандай-да бір бөлігіне адрес береді. Ал C+B өрнегі осы жадының бөлігіне орналасатын мәнді анықтайды.

Меншіктеу операторының сол жағында орналасқан айнымалының адресін алу бағдарламашыға мұндай жолмен алу мүмкін емес. Айнымалының адресін алу үшін Си тілінде унарлы & операциясы қолданылады. &E өрнегі машиналық дәрежеде E айнымалысына бөлінген жадының бөлігінің адресін алуға мүмкіндік береді.

& операциясы, тек жадыда орналасқан және аты бар объектілерге ғана қолданылады. Оны өрнектерге, тұрақтыларға, биттік алаңдардың құрылымдарына, регистрлік айнымалыларға немесе сыртқы объектілерге қолдануға болмайды.

Егерде бағдарламада мынадай бір бөлік қолданылған болса,

```
char  ch='G'
Int  date  1937;
float  f=2.015E-6;
```

онда ол машинаның жадында төмендегі суреттегідей болып орналасады (5.2-сурет).

Машинадағы адрестер:	1A2B	1A2C	1A2D	1A2E	1A2F	1A30	1A31	1A32
	байт	байт	Байт	байт	байт	байт	байт	Байт
Жадыдағы мәлімет:	'G'	1937		$2.015*10^{-6}$				
Аты:	**ch**	**Date**		**f**				

5.2-сурет. ЭЕМ жадындағы әр түрлі типтегі берілгендер

Суреттен айнымалылардың жадыда 16 адресі бар 1A2B байтынан бастап орналасқанын көреміз. Символдық типті айнымалылар жадыда 1-байт, бүтін типті айнымалылар 2-байт, ал нақты типті айнымалылар 4-байт орын алады. Жадыда қойылған талаптарға сәйкес:

&ch =1A2B;
&date =1A2C;
&f=1A2E.

Адрестер бүтін санды таңбасыз мән қабылдайды. Сондықтан оларды бүтін санды шамалар ретінде өңдеуге болады. & көмегімен айнымалылардың немесе бағдарламаның басқа объектерінің адрестерін анықтауға болады және анықтаумен қатар оларды түрлендіруге, сақтауға және басқа объектілерге меншіктеуге болады. Осы мақсатта Си тілінде көрсеткіш типіндегі айнымалылар енгізілген.

Қысқаша көрсеткіш: Си тіліндегі көрсеткішті мәні-типі анықталған объектінің адресі болатын айнымалы ретінде анықтауға болатын. Көрсеткіштің мәні ешқандай адреске тең болмайтын мән болуы керек немесе оны нөлдік адрес деп атайды. Оны сипаттау үшін тақырыптық файлдарда studio.h файлда анықталған арнайы NULL процессі қолданылады. Ол үшін Си тілінде ‘‘*’’ символы қолданылады.

Бұдан басқа көрсеткіш типіндегі айнымалыларды сипаттау және анықтау үшін осы көрсеткіштің сілтемеленетін объектісінің типін анықтау қажет. Сондықтан * таңбасынан басқа көрсеткіштерді сілтемеленетін объектілердің типтерін анықтайтын типтердің спецификасы қолданылады:

```
char   *z;  /*z-символдық типтегі объектінің көрсеткіші*/
int   *i,  *k;  /*i,  *k-бүтін типті объектінің көрсеткіштері*/
float  *f;  /*f-нақты типті объектінің көрсеткіші*/
```

Бұл жердегі * операциясы ол сілтеменің табу операциясының немесе адрес бойынша қатынасу операциясының таңбасы болып есептеледі. Мұндай операцияның операндасы ретінде әрқашанда көрсеткіш қолданылады. Бұл операцияның нәтижесі болып –көрсеткіш операнданың жадыдағы адресін көрсететін объект есептеледі.

Сонымен, мысалдағы *z таңбасы, z символы көрсететін char (символдық айнымалы) типіндегі объектіні анықтайды. *z, *i, *f таңбалары, осы типтерге сәйкес келетін айнымалылардың құқығына ие. *z=’ ’;- операторы, z көрсеткіші анықтайтын жадының адресіне бос орын символын енгізеді. *k=*i=0; операторы адрестері k,i көрсеткіштерімен анықталатын жадының бөлігіне бүтін нөлдік мәндерді енгізеді. Көрсеткіш, осы көрсеткіштің сипаттамасында көрсетілген типке сәйкес келетін объектіге сілтемеленеді.

E=B+C меншіктелу операторында жасырын түрде берілген, сол жақтағы айнымалының адресінің жұмысын анық көрсетуге болады. Ол үшін меншіктелу операторының біреуін келесі тізбектермен ауыстыру керек:

int E,C,B,*m; /* айнымалыларды және m көрсеткішін сипаттау*/

m=&E; /*m көрсеткішіне E айнымалысының адресін меншіктеу*/

m=B+C; / C+B өрнегінің мәнін m көрсеткішінің мәніне тең адрестегі жадының бөлігіне енгізу*/

Көрсеткіштерге орындалатын операциялар. Си тілінде көрсеткіштерге орындалатын келесі негізгі операциялар анықталған:

- меншіктелу;
- көрсеткіш сілтемеленетін объектінің мәнін алу;
- көрсеткіштің өзінің адресін алу;
- көрсеткіштің мәнін өзгертетін унарлы операциялар;
- аддитивті операциялар және салыстыру операциялары.

Осы аталған операцияларға толығырақ тоқталып өтейік.

Меншіктелу операциясында, меншіктелу операторының таңбасының сол жағында көрсеткіштің аты, ал оң жағында мәні бар көрсеткіш немесе шартты нөлдік мәнді анықтайтын NULL түрақтысы орналасады.

Мысалы:

```
i=&date;  k=i;   z= NULL;
```

Бұл операторларды қарастыра отырып, * көрсеткіштің аты өрнегі арқылы көрсеткіш анықтаған адрестегі мәнді алуға болатынын ескертеміз. Алдындағы мысалда, date айнымасының мәні анықталған болатын (1937), одан кейін оның адресі i көрсеткішіне және k көрсеткішіне меншіктелді, сондықтан *k көрсеткішінің мәні бүтін 1937- ге тең. Бұл жерде date айнымалысының аты да, i, k көрсеткіштері де, date айнымалысына бөлінген жадының бөлігіне қатынас орнатуды қамтамасыз етеді. Яғни, *k=өрнек, *i=өрнек, date=өрнек операциясының барлығы ЭЕМ жадысының тек бір ғана бөлігінің мәнін өзгертуі кез-келген айнымалы сияқты, көрсеткіш типіндегі айнымалының да аты, жадыдағы өзінің адресі және мәні болады.

Көрсеткіштің мәнін экранға шығаруға немес мысалда көрсетілгендей басқа көрсеткішке меншіктеуге болады. Көрсеткіштің адресі & унарлы операциясының көмегімен анықталады. & көрсеткіштің _ аты өрнегі, жадыда көрсеткіштің қайда орналасқанын анықтайды. Жадының осы көрсетілген бөлігінің мазмұны көрсеткіштің мәні болып табылады. Көрсеткіштің атының, адресінің, мәнінің арасындағы арақатынасты келесі суреттегідей көрсетуге болады (5.3-сурет).

5.3-сурет. Көрсеткіштің аты, адресі және мәнінің арасындағы ара қатынас

‘++’ және ‘--’ унарлы операцияларының көмегімен көрсеткіш типіндегі айнымалылардың сандық (арифметикалық) мәні, осы айнымалылармен байланысты мәліметтердің типіне сәйкес әртүрлі өзгереді. Егер де көрсеткіш char типімен байланысты орындалса, онда ’++’ және ’--’ операцияларынан кейін оның сандық мәні 1-ге өзгереді. Егер де көрсеткіш int типімен байланысты болса, ++i, i++, --k, k-- операциялары көрсеткіштердің сандық мәнін 2-ге өзгертеді. float және long типімен байланысты көрсеткіштердің, ’++’, ’--’ операцияларынан кейін сандық мәні 4-ке өзгереді. Осылай көрсеткіштің мәні бірге өзгергенде, көрсеткіш осы типпен анықталатын алаңның келесі немесе алдыңғы басына өтеді.

Көрсеткіштерге аддиктивті операциялар әр-түрлі қолданылады, анықтап айтқанда оларды қолданғанда бірқатар шектеулер қойылады. Екі типті көрсеткішті айнымалыны бір-біріне қосуға болмайды бірақ көрсеткішке қандай-да бір бүтін шаманы қосуға болады. Мұндай жағдайда есептелетін мән тек қана қосылатын бүтін шаманың мәніне ғана емес көрсеткішке байланысты объектінің типіне де тәуелді болады.

Мысалы: егер де көрсеткіш long типі бүтінсанды объекті болса, онда оған бірді қосқанда, оның мәні 4-ке артады немесе көбейеді. Алу операциясының қосу операциясына қарағанда, қосу операциясын тек қана көрсеткішке және бүтін шамаға қолданылса, алу операциясын және объектісі бір типті екі көрсеткішке қолдануға болады. Оның көмегімен бірдей типті екі объектінің айырымын табуға болады және сонымен екі объектінің жадыда орналасу арақашықтығын анықтауға болады.

Арифметикалық операциялар және көрсеткіштер. ’&’ және ’*’ унарлы адресті операцияларының, арифметикалық операцияларға қарағанда орындалу деңгейі жоғары. Келесі мысалды қарастырайық:

```
float  a=4.0 ,   *u, z;
u=&z;
*u=5;
a=a+*u+1;
/* бұл жерде  a=10, u- мәні өзгермейді, z=5 */
```

’*’ адресті операциясын арифметикалық өрнектерде қолданғанда, бөлу / және ’*’ операцияларын бірге қолданғанда өте мұқият бөлу керек, себебі бұл таңбалардың біріккен әрекетін (комбинациясын) компилятор комментария ретінде қабылдайды.

Мысалы: келесі өрнекті a/* u былай жазу керек a/*(u) унарлы ’*’, ’++’ , ’--’ операцияларының орындалу дәрежесі бірдей, сондықтан оларды бірге қолданғанда олар солдан оңға қарай орындалады. Бүтін санды n-нің мәнін қандай да бір массивтің элементін адрестейтін көрсеткішке қосқанда, көрсеткіш ағымдағы элементтен n-ші позицияда түрған элементтің адресіндегі мәнді қабылдайды. Егер де массив элементтерінің ұзындығы d байтқа тең болса, онда көрсеткіштің сандық мәні (d*n) –ге өзгереді. Осы аталған ережелерді көрсету үшін келесі мысалыда қарастырайық.

```
int x[4]={0,2,4,6},  *i,y;
i=& x[0];              /* i x[0] элементтің адресіне тең */
y=*i;                  /* y 0-ге тең; i & x[0]-ге тең */
y=*i++;                /* y 0-ге тең; i & x[1]-ге тең */
y=++*i;                /* y 3-ге тең; i & x[1]-ге тең */
y=*++i;                /* y 4-ке тең; i & x[2]-ге тең */
y=(*i)++;              /* y 4-ке тең; i & x[2]-ге тең */
y=++(*i);              /* y 6-ке тең; i & x[2]-ге тең */
```

Көрсеткіштер және қатынастар. Көрсеткіштерге қатынас амалдары да қолданылады. ’>’, ’>=’, ’!=’, ’==’, ’<=’, ’<’. Бірақ көрсеткіштерді тек типтері сәйкес көрсеткіштермен салыстыруға болады, көрсеткіштерге операцияларды қолданып және соның нәтижесінде қабылдаған мәндерін экранға шығаратын мысал қарастырайық.

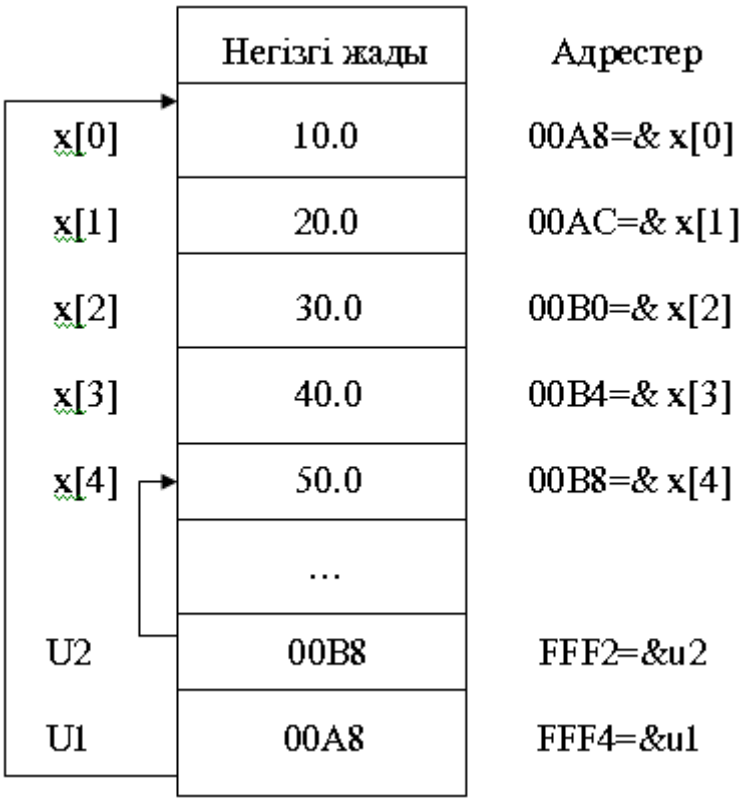
Көрсеткіштердің мәндерін **printf ()** функциясының көмегімен экранға шығару үшін **%p** спецификаторы қолданылады.

```
#include <stdio.h>
float x[ ]={10.0, 20.0, 30.0, 40.0, 50.0};
void main  ( )
{
 float *u1,  *u2;
 int  i;
 printf (“\ n Көрсеткішінің адресі: &u1=%p &u2=%p”,&u1,&u2);
 printf (“\ n массив элементінің адресі: \ n”);
 for (i=0; i<5; i++)
  {
    if (i ==3)   printf(“\ n”);
    printf  (“& x[%d]=%p”,i,&x[i]);
  }
 printf (“\ n Массив элементінің мәндері:\ n”);
 for (i=0; i<5; i++)
  {
    if ( i==3)  printf (“\ n”);
    printf  (“ x[%d]=%5.1 f ”, i, x[i]);
  }
 for (u1=& x[0], u2=& x[4]; u2>=& x[0]; u1++, u2--)
 {
 printf (“\ n u1=%p  *u1=%5.1 f    u2=%p *u2=%5.1f ”,u1, *u1,u2, *u2);
 printf (“\ n  u2-u1=%d”, u2-u1);
 }
}
```

Бағдарламаның орындалу барысында мынандай нәтижелер экранға шығады.

Көрсеткіштің адресі: &u1=FFF4 &u2=FFF2
Массив элементтерінің адрестері:
&x[0]=00A8 &x[1]=00AC &x[2]=00B0
&x[3]=00B4 &x[4]=00B8
Массив элементтерінің мәндері:
x[0]=10.0 x[1]=20.0 x[2]=30.0
x[3]=40.0 x[4]=50.0
u1=00A8 *u1 =10.0 u2 =00B8 *u2=50.0
u2-u1=4
u1=00AC *u1=20.0 u2=00B4 *u2=40.0
u2-u1=2
u1=00B0 *u1=30.0 u2=00B0 *u2=30.0
u2-u1=0
u1=00B4 *u1=40.0 u2=00AC *u2=20.0
u2-u1=-2
u1=00B8 *u1=50.0 u2=00A8 *u2=10.0
u2-u1=-4

float x[5] массивінің және көрсеткіштердің қайталану орындалғанға дейінгі жадыда орналасу сызбасы (5.4-сурет).



5.4-сурет. Массив пен көрсеткіштердің жадыда орналасу сызбасы

5.2 Көрсеткіштер мен массивтер

Анықтамасы бойынша көрсеткіш дегеніміз-объектінің немесе функцияның адресін алуға мүмкіндік беретін өрнек. Бағдарлама бөлігін қарастырайық:

```
int  x,y;
int  *p=&x;
p=&y;
```

Бұл жердегі p-объект көрсеткіш, ал &x, &y-өрнек көрсеткіштер. p-айнымалысының типі &x, &y мәндерінің типіне сәйкес. Адреспен, яғни өрнек-көрсеткішпен объект-көрсеткіштің арасындағы айырмашылық мынада. Объект-көрсеткіштің мәнін өзгертуге болады, ал өрнек-көрсеткіштің мәні өзгермейді. Сондықтан да өрнек-көрсеткіштерді тұрақты-көрсеткіштер, ал объектінің көрсеткіштерін айнымалы-көрсеткіштер немесе көрсеткіштер деп атайды.

Си тілінің синтаксисіне байланысты индексі көрсетілмеген массивтің аты да тұрақты-көрсеткіш болып табылады. Өйткені ол осы массивтің бірінші элементінің адресіне тең. Мұны әрқашанда массивтермен және көрсеткіштермен жұмыс жасағанда есте сақтау керек. Символдық массивтерді, көрсеткіштерді қолданып нәтиже беретін мысал қарастырайық. char типіндегі массивтің ұзындығы 80-ге тең болсын.

```
char z[80],s;
char  *d,  *h
(*d,h-символдық объектінің көрсеткіштері*)
for (d=z,  h=&z[79]; d<h; d++, h--)
{
s=*d;
*d=*h;
*h=s;
}
```

Қайталану басында d көрсеткішіне z массивінің 1-ші элементінің адресі меншіктеледі. h-көрсеткішіне z массивінің ең соңғы элементінің адресі меншіктеледі. Ары қарай қайталану d<h болғанша орындала береді. Әрбір қайталану итерациясынан кейін d-нің мәні 1-ге артады, ал h-тің мәні 1-ге кемиді. Бірінші итерациядан кейін z[0] және z[79] элементтерінің мәндері ауыстырылады, өйткені d- z[0]-дің адресі, h- z[79]-адресі. Екінші итерация орындалғаннан кейін d-нің мәні z[1], h-тің мәні z[78]-ге тең болады да, әрі қарай қайталану жалғаса береді.

Бұл есепті шешудің екінші жолы:

```
char   z[80], s,  *d,  *h;
for (d=z, h=&z[79]; d<h;)
{
```

```
s=*d;   *d++=*h;   *h--=s;
}
```

Бұл бағдарламада d көрсеткішінің артуы және h көрсеткішінің кемуі циклдің денесінде орналасқан. *d++ және *h-- өрнектеріндегі 1-ге көбею, кему операцияларының орындалу дәрежесі "*" унарлы адресті операциясымен бірдей. Сондықтан көрсеткіштермен массив элементтерінің мәндері емес, көрсеткіштердің өзі өзгереді.

Динамикалық жадының массивтері. Си бағдарламалау тілінің стандартына сәйкес массив дегеніміз-бұл әрқайсының қасиеттері (атрибуттары) бірдей элементтер жиыны. Жадыда массивтің бастапқы элементіне сәйкес адрестен басталып, бірінен кейін бірі қатарымен орналасады. Массивтің сипатталуы:

Тип массивтің_аты [элементтер_саны];

Массивтің_аты - массив элементтерінің орналасуына бөлінетін жадының облысына сілтеме болып орналасады.

Элементтер_саны - тілдің синтаксисіне байланысты тұрақты өрнек болуы керек. Тип – массивтің әрбір элементіне бөлінетін жадының көлемін анықтайды. Сонымен, массив элементтерінің жалпы саны және осы массивке бөлінетін жадының көлемі, осы массивті сипаттаған кезде-ақ толық анықталады. Бірақ мұндай сипатталуды қолдану барлық уақыттада қолайлы болмайды. Кейде массивке бөлінетін жадының көлемі тек нақты есепті шешу үшін ғана бөлінуі қажет және бұл есепті шешу үшін қажет жадының көлемі алдын-ала белгісіз болуы мүмкін. Мұндай көлемі айнымалы массивтерді қалыптастыруды, көрсеткіштердің және жадыны динамикалық бөлу құрылғыларының көмегімен ұйымдастыруға болады. Бұл аталған құрылғыларды Си тілінің стандартты библиотекаларындағы alloc.h және stdlib.h тақырыптық файлдарында сипатталған библиотекалық функциялардан бастаймыз. Бұл функциялар туралы мәліметтерді төмендегі кестеден көре аламыз (5.1-кесте).

5.1-кесте. Жадыны бөлетін және босататын функциялар

Функция	Қысқаша анықтама
Malloc	**void * malloc (unsigned s);** Көрсеткішті, ұзындығы s байт болатын динамикалық жады облысының бастапқы жағына қайтарады. Сәтсіз аяқталған жағдайда NULL мәнін қайтарады.
Calloc	**void *calloc (unsigned n, unsigned m);** Көрсеткішті, әрқайсы m байт болатын n элементтерінің орналасуы үшін бөлінген динамикалық жады облысының бастапқы жағына қайтарады. Сәтсіз аяқталған жағдайда NULL мәнін қайтарады.
Realloc	**void * realloc (void*bl, unsigned ns);** Алдын-ала бөлінген динамикалық жадының көлемін ns

	байтқа өзгертеді. bl-өзгеретін жадының бастапқы адресі. Егер де bl, NULL-ға тең болса (егер де жады бөлінбеген жағдайда), онда malloc функциясы сияқты орындалады.
Free	**void * free(void * bl);** Алдын-ала бөлінген, бірінші байтының адресі bl-ге тең динамикалық жадының облысын босатады.

malloc(), calloc() және realloc() функциялары массив параметлерінің мәніне сәйкес жадыны динамикалық түрде бөледі. Бұл функциялардың әрқайсысының қайтаратын мәнінің типі void*. Мұндай типті көрсеткішті кез-келген типті көрсеткішке түрлендіруге болады. free() функциясы, malloc(), calloc() немесе realloc() функцияларының көмегімен бөлінген жадыны босатады. Жадының бөлінген аймағы туралы мәлімет free() функциясына, типі void* болатын параметр-көрсеткіш арқылы беріледі.

Келтірілген бағдарлама динамикалық жадыны бөлу malloc() және босату free() функцияларының қолдану ерекшеліктерін көрсетеді. Келесі есепті шешу керек. Алдын-ала саны белгісіз, нақты сандардың жиынын енгізіп, кері ретпен шығару керек. Бағдарлама мәтіні төмендегідей болады:

```
# include <stdio.h>
# include <stdlib.h>
void main ( )
{
/* жадының бөлінетін облысы үшін қажет көрсеткіш*/
 float  *t;
 int i, n;
 printf (“\ n n=”);  /*n-элементтер саны*/
 scanf (“%d”,&n);
 t= (float *) malloc (n*sizeof (float));
 for (i=0;  i<n;  i++)  /*сандарды енгізу циклы*/
  {
   printf (“x[%d]=”,i);
   scanf (“%f”,&t[i]);
  }
/*нәтижелерді экранға шығаратын цикл*/
 for (i=n-1;  i>=0;  i--)
  {
   if (i%2==0)  printf(“\ n ”);
   printf (“\ t x[%d]=%f”,i,t[i]);
  }
 free (t);   /*жадыны босату*/
}
```

Бұл бағдарламаның орындалу нәтижесі мынандай:

```
n=4
x[0]=10
x[1]=20
x[3]=30
x[4]=40

x[3]=40.000000     x[2]=30.000000
x[1]=20.000000     x[0]=10.000000
```

Бағдарламадағы int n – float типіндегі енгізілетін сандардың көлемі, t- n енгізілетін сандар үшін бөлінетін аймақтың алғашқы адресіне сілтемеленетін көрсеткіш. t көрсеткіші float типіндегі n мәндер үшін бөлінетін аймақтың адресінің мәнін қабылдайды.

Жадының бөлінген аймақтарына рұқсат алу t[i] және t[i-1] индексті операцияларының көмегімен жүзеге асырылады. free (t) операторы t көрсеткішімен байланысқан, алдын-ала бөлінген динамикалық жадыны босатады.

Массивтің сипаттамасымен анықталған массивтердің динамикалық массивтерден айырмашылығы бар. Олардың ең негізгі айырмашылығы динамикалық массивтің атының болмауы. Бұл айырмашылықтың маңыздылығын түсіну үшін массив элементтерінің санын sizeof операциясының көмегімен анықталатын стандартты процедураны қарастырайық. sizeof операциясын шақырудың екі түрі бар:

```
sizeof (тип)
sizeof өрнек
```

Бірінші түріндегі типке кез-келген объектілердің типін қолдануға болады. sizeof операциясы орындалғанда байтпен анықталатын операнданың көлемі есептеледі. Егер де операнд өрнек болса, онда оның мәні анықталады. Ол үшін алдымен осы мәнге байланысты тип анықталады, одан кейін осы типке байланысты оның ұзындығы байтпен есептеледі.
Мысалы:

```
sizeof  (long) әдетте  4-ке тең;
sizeof  (long  double)  әдетте 10 байтқа тең;
```

Бағдарламада нақты анықталған массив элементтерінің санын есептеу үшін sizeof операциясын қолданған ыңғайлы. Ол үшін бұл операцияны екі рет массивтің нөлдік элементіне. Егер де sizeof операциясында операнд ретінде индексіз массив аты қолданылса, онда жалпы осы массвике бөлінген жадының ұзындығының мәні қайтарылады. Сонымен, келесі өрнектің мәні болып,

sizeof(массивтің_аты)/ **sizeof** (массивтің_аты [0])

массив элементтерінің саны есептеледі.

Келесі бағдарламаның бөлігінде x массивінің барлық элементтері есептеледі.

```
float  x[8];
for (i=0; sizeof [x]/ sizeof(x[0])>i;  i++)
…….x[i]……
```

Егер де sizeof операциясын, 5.1-кестеде көрсетілген функциялардың көмегімен динамикалық массив үшін бөлінген жадының бастапқы адресінің мәнін қабылдайтын көрсеткішке қолданатын болсақ, онда жадының динамикалық облысының көлемі емес, тек көрсеткіштің көлемі ғана анықталады. Осы айтылғандарды көрсету үшін бағдарламаның келесі бөлігін қарасытырайық:

```
double * pointer;
pointer = (double*) malloc (100);
…… sizeof (pointer)
```

Бұл бағдарламада, malloc() функциясының параметірінің мәніне тәуелсіз, sizeof(pointer) өрнегінің мәні әрқашанда бірдей болады. Өйткені ол әрқашанда pointer көрсеткіші үшін бөлінген жадының бөлігінің шамасына тең болады.

Көрсеткіштер массивтері және көпөлшемді массивтерді моделдеу. Алдыңғы тақырыптарда көлемдері шектелген екі өлшемді массивтерді қолдану мысалдары қарастырылды. Бірнеше айнымалы көлемдері бар көпөлшемді массивтерді тікелей анықтау немесе сипаттау әдістері Си тілінде жоқ. Сондықтан мұндай есептерді шешу үшін жадыны динамикалық түрде басқаруға арналған, библиотекалық функцияларда анықталған жүйелік құрылғыларды пайдалану қажет. Олардың мүмкіндіктерін қарастырудан бұрын, алдымен көрсеткіштер массивін енгізу керек. Көрсеткіштер массиві келесі сипаттаулар арқылы енгізіледі:

```
тип * массивтің_аты [көлемі];
тип * массивтің_аты [] =инициализатор;
тип * массивтің_аты [көлемі] =инициализатор;
```

Бұл жердегі тип-базалық типтердің бірі;

Массивтің_аты – кез-келген идентификатор;

Көлемі- трансляция кезінде есептелінетін тұрақты өрнек;

Инициализатор – фигуралы жақшаларға алынған осы типке сәйкес келетін мәндердің тізімі.

Мысалы:

```
int data [6];  /*кәдімгі қарапайым массив*/
int *pd [6]; /*көрсеткіштер массиві*/
int *pi [ ]={ &data [0], &data [4], &data [2]};
```

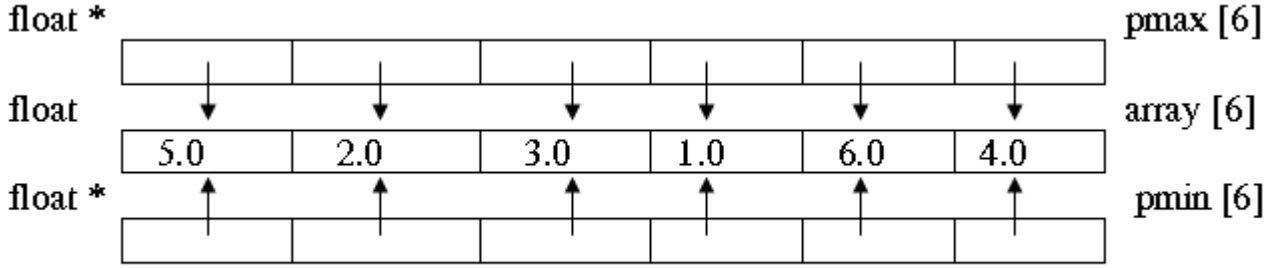
Бұл мысалдағы pd және pi массивтерінің әрбір элементі int типіндегі объектілерге көрсеткіш болады. pd[j] және pi[k] массивтерінің әрбір элементерінің мәні болып, int типіндегі объектінің адресі есептеледі. pd көрсеткіш массивінің int типіндегі барлық 6 элементі де инициализацияланбаған. pi массивтерінің үш элементі, data массивінің элементтерінің нақты адрестерімен инициализацияланған. Көрсеткіштер массивін, оның элементтерін басқа массивтің элементтерін адрестеу үшін қолданғанда, оның үлкен мүмкіндіктері пайда болады. Мұндай жағдайда әр-түрлі көлемді, күрделі объектілерді сұрыптау есептерін оңай шешуге болады. Мысал ретінде бірөлшемді массив элементтерін орын ауыстырмай-ақ бір мезетте өсу және кему ретімен шығаратын бағдарламаны қарастырайық:

```
# include <stdio.h>
# define B 6
void main ( )
{
  float array [ ]={5.0, 2.0, 3.0, 1.0, 6.0, 4.0};
  float *pmin [B];
  float *pmax [B];
  float *e;
  int   i,j;
  for (i=0; i<B;   i++)
  pmin  [i]=pmax[i]=&array[i];
  for(i=0; i<B-1; i++)
  for(j=i+1;  J<B; J++)
     {
      if (*pmin [i]<*pmin[j])
        {
         e=pmin[i];
         pmin [i]=pmin[j];
         pmin [j]=e;
        }
     if (*pmax [i]>*pmax [j])
       {
        e=pmax [i];
        pmax [i]=pmax[j];
        pmax [j]=e;
       }
     }
  printf (“\ n  кему ретімен : \ n ”);
  for (i=0; i<B; i++)
   printf (“\ t % 5.3 f”, *pmin [i]);
  printf (“\ n өсу ретімен :\ n”);
    for (i=0; i<B; i++)
```

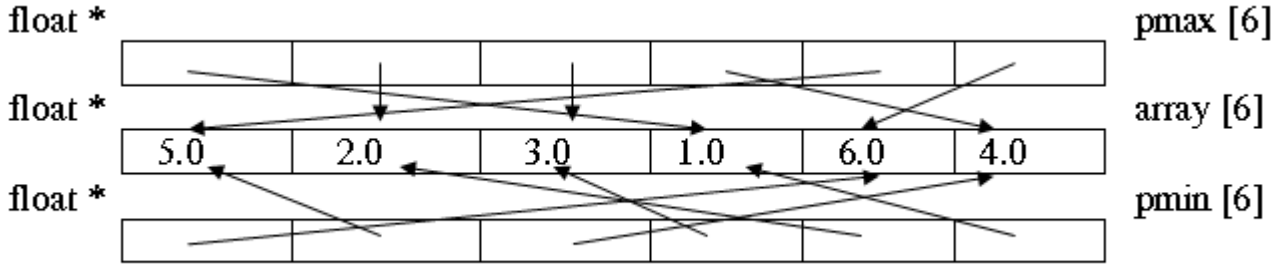
```
    printf ("\ t % 5.3 f", *pmax [i]);
  }
```

Бағдарламаның орындалу нәтижесінде:
кему ретімен:
6.000 5.000 4.000 3.000 2.000 1.000
өсу ретімен:
1.000 2.000 3.000 4.000 5.000 6.000

Бағдарламада pmin[6] көрсеткіштер массивінің элементтерінің мәндерін ауыстыру арқылы, array[6] массивінің элементтерінің мәндерінің кему ретімен орналасқан тізбегі анықталған. pmax[6] көрсеткіштер массиві де осындай есепті шешеді, бірақ онда array[6] массивінің элементтерінің мәндері өсу ретімен орналасады. Төмендегі суретте (5.5-сурет) array[6] массивінің элементтерінің, көрсеткіштер массивінің элементтерімен адрестелген бастапқы және соңғы нәтижелері көрсетілген. pmin[6] және pmax[6] массивтерінің элементтерінің мәндерін ауыстырып орналастыру үшін бағдарламада float * e көмекші көрсеткіші енгізілген.



а) массивтер реттелгенге дейін



б) массивтер реттелгеннен кейін

5.5-сурет. Көрсеткіштер массивінің реттелген түрі

Си тілінде массив элементтерінің типі және оның ұзындығы бірдей болу керек. Кейбір жағдайларда көлемі бірдей емес объектілерді өңдеу қажеттілігі пайда болады. Мысалы қатарлардағы элементтер саны әртүрлі екі өлшемді массивпен жұмыс жасау үшін оның көлемін осы массивтегі элементтер саны ең көп қатар бойынша сипаттауға тура келеді. Ал мұндай жағдай жадыны керек емес болса артық пайдалануға әкеліп соқтырады. Осындай жағдайларда көрсеткіш массивтерін және жадыны динамикалық бөлу құралдарын қолдану

жоғарыда айтылған қиындықтарға ұрынбай өтіп, жадыны тек қана қажеттілікке ғана бөліп, пайдалануға мүмкіндік береді. Осы айтылған мүмкіндіктерді көрсету үшін келесі есепті қарастырайық.

Сандар қатарын енгізіп, оларды кері ретпен шығарады. Қатарлардың саны бағдарламаның басында, ал әрбір қатардың ұзындығы (яғни әрбір қатардағы элементтер саны) әрбір тізбегінің алдында енгізіледі. Бұл есепті шешу бағдарламасы мынадай:

```
# include <stdio.h>
# include <alloc.h>
void main ( )
{
 doubl  ** line;   /* line-көрсеткіштер массивіне бөлінетін, жады
                      блогының көрсеткіші*/
  int i, j, n;
 double  dd;
 int  *m;   /*жадының блогы үшін көрсеткіш*/
 printf (“\ n Қатарлар санын енгізіңіз n=”);
 scanf (“% d”,&n);
 line=(double **) calloc (n, sizeof (double*));
 m=(int *) malloc (sizeof (int)*n);
 /* қтарлар саны бойынша цикл ұйымдастыру*/
for (i=0; i<n; i++)
    {
     printf (“ m [%d] Қатарының ұзындығын енгізіңіз=”,i);
     scanf (“% d”,&m[i]);
     line[i]=(double *) calloc (m[i], sizeof (double));
     for (j=0; j<m[i]; j++)
     {
      printf (“ line [%d] [%d] =”,i,j);
      scanf (“% le”,&dd);
      line [i][j]=dd;
     }
    }
   printf (“\ n Орындалу нәтижесі: ”);
   for (i=n-1; i>=0; i--)
   {
    printf (“\ n  %d қатар, %d элемент: \ n ”,i,m[i]);
    for (j=0; j<m[i]; j++)
    printf (“\ t %f”, line [i] [j] );
    free (line [i]); /* Қатардың жадысын босату*/
   }
  free (line); /*Жадыны көрсеткіштер массивінен босату*/
free(m); /*Жадыны int массивінен босату*/
}
```

Бағдарламаның орындалу нәтижесі:

Қатарлар санын енгізіңіз: n=5
M[0] қатардың ұзындығын енгізіңіз =4

Line [0][0]=0.0
Line [0][1]=0.1
Line [0][2]=0.2
Line [0][3]=0.3

M[1] қатардың ұзындығын енгізіңіз =3

Line [1][0]=1.0
Line [1][1]=1.1
Line [1][2]=1.2

M[2] қатардың ұзындығын енгізіңіз =1

Line [2][0]=2.0

M[3] қатардың ұзындығын енгізіңіз =4

Line [3][0]=3.0
Line [3][1]=3.1
Line [3][2]=3.2
Line [3][3]=3.3

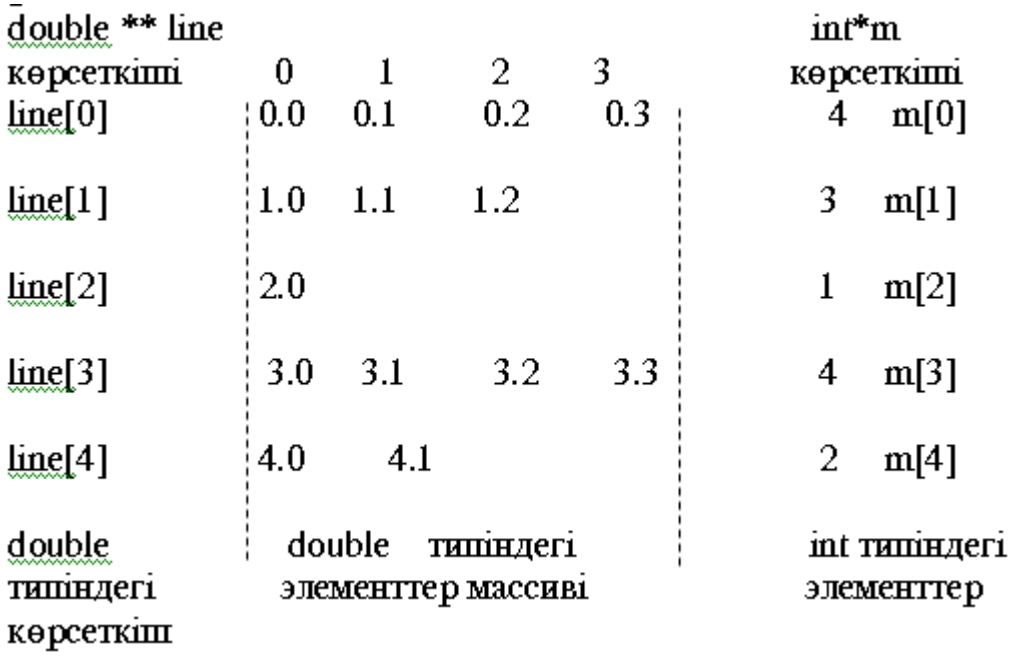
M[4] қатардың ұзындығын енгізіңіз =2

Line [4][0]=4.0
Line [4][1]=4.1

өңдеу нәтижесі:

4 қатар, 2 элемент:
4.000000 4.100000
3 қатар, 4 элемент:
3.000000 3.100000 3.200000 3.300000
2 қатар, 1 элемент:
2.000000
1 қатар, 3 элемент:
1.000000 1.100000 1.200000
0 қатар, 4 элемент:
0.000000 0.100000 0.200000 0.300000

Бағдарламаның орындалу барысында қалыптасатын барлық массивтерді көрсету үшін төмендегідей суретті (5.6-сурет) қарайық.



5.6-сурет. Әр түрлі ұзындықтағы матрицаларды енгізу сызбасы (біздің жағдайымызда жол саны n=5, әрбір жолдағы мәндер саны: 4, 3, 1, 4, 2)

Бағдарламада Line- double * типіндегі көрсеткіштер массивінің көрсеткіші. Олардың әрқайсысы line[i] double типіндегі элементтерінің динамикалық түрде бөлінетін ұзындығы m[i]-тең жадының бөлігін адрестейді. m-бүтін массивтердің көрсеткіші, әрбір элементінің мәні line[i] көрсететін массивтің ұзындығына тең.

Жадыны бөлуге арналған әртүрлі функциялардың жұмысын көрсету үшін бағдарламада calloc() және malloc() функциялары қолданылды. Олардың арасындағы айырмашылық параметрлерінің саны және мағанасына байланысты. Бағдарламадан шыққаннан кейін динамикалық түрде бөлінген жадының облыстарын босату қажет. Сол үшін бағдарламада free() функциясы бірнеше рет қолданылып отыр.

5.3 Символдық ақпарат және жолдар

Мәтіндік ақпараттармен жұмыс жасау үшін Си тілінде символдық тұрақтылар, символдық айнымалылар және жолдар қолданылады. Символдық мәліметтерді сипаттау үшін char базалық типі енгізілген. Символдық айнымалылар сипаттаудың жалпы түрі:

char айнымалылар_атауларының_тізімі;

Мысалы: char a,z;

Си тілінде символдық мәндерді енгізу және шығару үшін printf() және scanf() библиотекалық функцияларының форматты жолдарында %c түрлендіру

спецификациясы қолданылады. Мысалы соңына нүкте қойылған және сөздері бірнеше бос орындармен бөлінген сөйлем енгізіп, оны экранға шығарғанда сөйлемдегі сөздердің арасында тек бір ғана бос орын қалдырып шығару керек. Мұндай бағдарламаның мәтіні төмендегідей болады:

```
# include <stdio.h>
void main ( )
{
 char z, s;
 printf (“\ n  соңына нүкте қойылған сөйлем енгізіңіз  \ n ”);
 for ( z=s=’ ’; z!=’ . ’; s=z)
    {
     scanf   (“%c”, &z);
     if  (z==’ ’ && s==’ ’)  continue;
     printf (“%c”,z);
    }
}
```

Бағдарламада екі символдық айнымалы қолданылады: z-енгізілетін символды оқу үшін, s – алдыңғы символды сақтау үшін. Циклдың басында z және s айнымалылары “бос орын” мәнін қабылдайды. Келесі символ z айнымалысының мәні ретінде енгізіледі және z, s айнымалыларының мәндері талданады. Егер де z, s айнымалыларының мәндерінің бірі “бос орынға” тең болмаса ол экранға шығарылады. Қайталану басында z айнымалысы нүкте символымен салыстырылады, тең болмаған жағдайда s айнымалысының мәні ретінде сақталады. Ары қарай қайталану нүкте енгізілгенше қайталана береді.
Мысалы:

```
YYYYYY     yyyyyy    hhhh    ttttt.
YYYYYY yyyyyy hhhh ttttt.
```

printf() және scanf() функциясынан басқа символдарды енгізу үшін арнайы функциялар қолданылады.

getchar() - параметрсіз функция. Бұл функция клавиатурадан енгізілген символдар легін бір-бірлеп оқиды. scanf() функциясын қолданғандағыдай енгізілген мәліметтерді оқу <Enter> пернесін басқаннан кейін орындалады. Бұл енгізілген ақпараттағы қателерді <Backspace> пернесі арқылы өшіріп, дұрыстауға мүмкіндік береді. putchar (X) –X-тің символдық мәнін экранға шығарады. Бұл функцияларды келесі есепті шешу үшін қолданамыз. Соңында нүкте қойылған сөйлем енгізіп, осы сөйлемдегі символдардың санын есептейтін бағдарлама құрамыз:

```
# include  <stdio. h>
void main ( )
{
```

```
    char  z;
    int  k;
    printf (“Соңында нүктесі бар сөйлем енгізінің  \ n ”);
    for ( k=0;  ( z=getchar())!=’ . ’; )
    if (z!=’ ’)     k++;
    printf (“\ n  Символдар саны=%d”,k);
}
```

Бағдарламада for қайталану операторының басында z=getchar өрнегі жақшаға алынған, өйткені салыстыру операциясының меншіктелу операциясына қарағанда орындалу дәрежесі жоғары. Бүтін сандарды қолдануға мүмкіндік беретін тілдің синтаксисі барлық жерде символдарды да қолдануға болады. Мұндай келісім символдарды бүтінсанды шамалар сияқты реттеуге мүмкіндік береді. Мысалы ондық цифрлардың ішкі кодтары ASCII кодтар кестесінде сандық мәндері бойынша реттелген. Келесі бағдарлама 0-ден 9-ға цифрларды және олардың ішкі кодтарын оналтылық түрде экранға шығарады:

```
# include  <stdio.h>
  void main ( )
{
  char  z;
  for ( z=’0’;   z<=’9 ’;  z++ )
     {
      if (z==’0’ || z==’5’) printf (“\ n”);
      printf (“%c-%x”, z, z );
     }
  }
```

Бағдарламаның орындалу нәтижесі:

```
0-30    1-31   2-32   3-33   4-34
5-35    6-36   7-37   8-38   9-39
```

Бағдарламада символдардың мәнін шығару үшін printf() функциясында %c спецификациясы, ал олардың оналтылық ішкі кодын шығару үшін %x спецификациясы қолданылады.

Латын алфавитінің әріптерінің де ішкі кодтары осылай реттелген. Сондықтан оларды бағдарламада ретімен шығару оңай.

Мысалы:

```
 # include  <stdio. h>
   void main ( )
   {
    char  z;
    for ( z=’A’;   z<=’ Z ’; z++ )
    printf (“%c”,z );
   }
```

Бағдарламаның орындалу нәтижесі:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Жолдар және жолдық тұрақтылар. Бағдарлама жолдар және жолдық тұрақтылар жақшаға алынған символдар тізбегі ретінде қолданылады. Жолдың символдарының арасында эскейп-тізбектер де қолданылуы мүмкін.
Мысалы:

“1234567890”
“\ t Топ құрамы”
“ Қатардың басы \ n қатардың соңы”.

Бірақ жолдардың бірқатар ерекшеліктері бар. Транслятор жолға ЭЕМ жадынан бөлек орын бөледі. Жолды жадыға орналастыра отырып транслятор оның соңына ’\0’ символын яғни нөлдік байт қосады. Жолда бір ғана символ “А” болуы мүмкін, бірақ жолдағы “А” символының “А” символдық тұрақтысынан айырмашылығы жадыда оған 2 байт орын бөлінеді. Басқа тілдердегідей Си тілінде жолдарды сипаттау үшін бөлек бір тип бөлінбеген.

Жолдар - символдар массиві ретінде қолданылады, және оның типі әрқашанда char[ ] болады. Символдар массивіне яғни жолдарға қарапайым меншіктеу операторының көмегімен мән меншіктеуге болмайды. Массивке жолды меншіктеу не инициализацияның, не енгізу функциясының көмегімен жүзеге асырылады. Символдық жолдарға scanf() және printf() функцияларында %s түрлендіру спецификациясы қолданылады.
Мысалы :

```
# include  <stdio. h>
   void main ()
   {
     char  B[]=”Бағдарламалау тілі”;
     printf (“%s”,B);
   }
```

нәтижесі: Бағдарламалау тілі

Бағдарламада В-массивінің ұзындығы-19 элементке тең, яғни массивке енгізілетін жолдың ұзындығы (18-символ) және қосымша жолдың соңын көрсететін нөлдік байт. Бұл мысалдағы массивті инициализациялау үшін 19 байт орын бөлінген. Жолдық тұрақтылардың көмегімен символдық массивтерді инициализициялаудың қысқартылған түрі болып есептеледі.

Қарапайым инициализациялауды қолдануға да болады, ол үшін массив элементтерінің бастапқы мәндерін фигуралы жақшаға алынады және массив элементтерінің соңына жолдың соңын білдіретін нөлдік символды ’\0’ қоюды

ұмытпау керек. Мысалы жоғарыда көрсетілген символдық массивті инициализациялау былай жазылады:

```
char B[ ]={’c’,’e’,’3’,’a’,’m’,’,’,’’,’0’,’t’,’k’,’p’,’o’,’n’,’c’,’r’,’!’, ’\ 0’ };
```

Бөлек символдардан тұратын жолды бағдарламада қолданған кезде осы жолдың соңын білдіретін нөлдік символды ұмытпау керек. Мысал ретінде келесі есепті қарастырамыз: “Сөздері бір-бірінен бос орын арқылы ажыратылатын және соңы нүктемен бітетен сөйлем енгізіп, оның ең соңындағы сөзді экранға бөліп шығару керек”. Бұл есепте қойылған шарттарды талдай отырып келесі қате тудыруы мүмкін ерекше жағдайларды қарастырып өткен жөн:

- сөйлем соңына нүкте қойылмауы мүмкін;
- сөйлемдегі бірінші сөздің алдына бос орын қойылуы мүмкін;
- сөздер арасына бірнеше бос орындар қойылуы мүмкін;
- сөйлемнің соңын білдіретін нүктенің алдына бос орын қойылуы мүмкін;
- сөйлемде сөздердің болмауы (тек қана нүкте болуы мүмкін).

Осы аталған жағдайларды болдырмау үшін келесі келісімді қабылдаймыз.

Енгізілетін сөйлем экранның бір ғана қатарында орналасатын болсын, яғни сөйлемнің ұзындығы 80 символға тең.

Сөздердің арасына бірнеше бос орын енгізілмеу үшін, ағымдағы енгізілген символды және оның алдында енгізілген символды тексеру керек.

Сөйлемде бір де бір сөздің болмауын ескеру үшін, әрбір енгізілген сөздің ұзындығын есептеп отырамыз. Егер де сөздің ұзындығы 0-ге тең болса, онда ’.’ нүкте символын енгіземіз. Ол бос сөйлем екенін білдіреді.

Бағдарламаның мәтіні:

```
# include  <stdio.h>
   void main ( )
{
  char s, ss;      /*s-енгізілетін символ,ss-алдыңғы енгізілген символ*/
  char A[80] ;  /*сөйлемді енгізу үшін керек массив */
  int i, k;       /*k-сөздің ұзындығы*/
  printf (“Соңына нүкте қойылған сөйлем енгіз:  \ n ”);
  for (i=0, s=’ ’, k=0; i<=79; i++)
     {
       ss=s;   s=getchar ();
         if (s==’ ’)  continue;
         if (s==’.’)    break;
         if (ss==’ ’)   k=0;
         A[k]=s;  k++;
     }
/*нүкте арқылы немесе сөйлемді енгізгеннен кейін бағдарламадан шығу*/
  if (i==80 || k=0)
```

```
printf (“ Дұрыс емес сөйлем   \ n ”);
else
  {
   A[k]= ’\ 0’; /*Жолдың соңы*/
   Printf (“Соңғы сөз:%s”,A);
  }
}
```

Бағдарламада енгізілген мәліметтерді оқу і параметрлі қайталану операторында символ бойынша бір-бірлеп орындалады. Егер де бос орын енгізілсе, онда continue операторы қайталану операторының келесі итерациясына өткізеді және A[ ] массивінің бірін k элементтерінде сөйлемнің ең соңғы сөзі сақталынады. Егер де енгізілген символ бос орын немесе нүкте болмаса, онда алдыңғы символ тексеріледі. Егер де ол бос орын болса, онда k нөлге тең болады да келесі сөздің символдары енгізіледі.

Келесі операторлар орындалғанда енгізілген символ A массивінің k элементіне жазылады да k-ның мәні бірге арттырылады. Қайталанудан шығу нүкте енгізілгенде немесе 80 символ енгізілген жағдайда орындалады. Бағдарламаның орындалу нәтижесі соңына нүкте қойылған сөйлем енгіз:

Орфографиялық сөздік.

Соңғы сөз: сөздік

Осы бағдарлама көрсеткендей символдық жолдармен жұмыс жасау-бұл char типіндегі массивтермен жұмыс жасау. Тағы да бір мысал қарастырайық. Цифрлардан және бос орыннан тұратын тізбек бар. Бағдарламада жолды енгізген кезде осы цифрлар және позицияның нөмерін шығаратын бағдарлама жазайық:

```
# include  <stdio.h>
  void main ( )
  {
  char  z[ ]=”0123456789”;
  char s;
  int i,j;
  printf (“Символдар жолын енгіз: \ n ”);
  for (i=1; (s=getchar())!=’\n’; i++)
    {
       for (j=0; j<11;  j++)
          if (s==z[j])  break;
     if (j==11)
        printf (“%d нөмерлі %c символында қате \ n ”, i,s);
    }
}
```

Бағдарламада символдарды енгізу і параметрлері қайталануында “\n” символы пайда болғанша орындалады. Әрбір енгізілген символ s параметрлі қайталануында алдын-ала анықталған z массивінің элементтерімен салыстырылады. Егер де салыстырылған символдар сәйкес келсе, онда қайталанудан шыққан жағдайда j 11-ге тең емес. Егер де сәйкес емес символ енгізілсе, онда осы символдардың позициясының нөмері шығарылады. Бағдарламаның орындалу нәтижесі:

Символдар жолын енгіз: 124E-22 16
4 нөмерлі E символында қате
5 нөмерлі – символында қате

Жолдар және көрсеткіштер. Көрсеткіштермен массивтердің арасындағы байланыс бізге таныс. Келесі екі сипаттаманы қарастырамыз:

```
char A[20];   /* жолды енгізуге болатын көрсеткіш*/
char *B;      /*жолды байланыстыруға болатын көрсеткіш*/
```

А массивіне осы сипаттама өңделгеннен кейін-ақ автоматты түрде жады бөлінеді. Ал В көрсеткішіне жады бөлінбейді. Егер де ары қарай мынадай операторлар орындалатын болса,

```
scanf (“%s”,A);  /*дұрыс оператор*/
scanf (“%s”,B);  /*дұрыс емес оператор*/
```

Онда бірінші оператор дұрыс орындалады, ал екінші оператор орындалған кезде анықталмаған жады бөлігіне енгізуге болмайды деген қате шығарады. Бұл екінші енгізу операторының орындалуынан бұрын жадының белгілі бір бөлігін В көрсеткішімен байланыстыру керек. Мұны орындау үшін бірқатар мүмкіндіктер бар. Біріншіден, В айнымалысына алдын-ала анықталған символдық массивтің адресін меншіктеуге болады. Екіншіден В көрсеткішіне жадыны динамикалық түрде бөлетін құрылғылардың көмегімен бөлінген жадының бөлігін меншіктеуге болады.

Мысалы, мына оператор B=(char *)mallloc(80); 80 байтқа тең жадының бөлігін В көрсеткішімен байланыстырады. Осыдан кейін ғана жоғарыда көрсетілген Scanf (“%s”,B); енгізу операторын қолдануға болады. char* типті көрсеткіштердің көмегімен символдық массив түріндегі жолдармен жұмыс жасаған қолайлы. Сөйлемнің немесе сөздің ұзындығы әртүрлі болған жағдайда, алдын-ала көлемдері анықталған массивтерді емес, көрсеткіштерді қолданған ыңғайлы. Келесі бағдарламада, бастапқы мәндері бірқатар жолдардың тізбегімен байланысқан бірөлшемді char типіндегі массив көрсеткіші анықталады. Бағдарлама орындалғанда экранға көрсеткіштің мәні емес, ол адрестейтін жолдың мәні шығады.

```
# include  <stdio.h>
  void main ( )
```

```
{
  char * point [ ]={“thirteen”,”fourteen”,”fifteen”,”sixteen”,
                    ”seventeen”,”eighteen”,”ninteen”};
 int i,n;
n=sizof (point)/sizeof(point[0]);
for (i=0; i<n; i++)
printf (“\ n% s”, point[i] );
}
```

Орындалу нәтижесі :

```
thirteen
fourteen
fifteen
sixteen
seventeen
eighteen
ninteen
```

Бақылау сұрақтары

1. Си тілінің туынды типтерін атаңыз.
2. Объектілердің көрсеткіштері дегеніміз не?
3. Қысқаша көрсеткіштер түсінігі, мысал келтіріңіз.
4. Көрсеткіштерге орындалатын операцияларды атаңыз.
5. Көрсеткіштерге арифметикалық операцияларды қолдану жолдары.
6. Көосеткіштерге қолданылатын қтынас амалдары.
7. Массивтерді көрсеткіштер арқылы сипаттау.
8. Динамикалық жадының массивтерімен жұмыс жасау жолдары.
9. Көрсеткіштер массивтерін сипаттау жолдары.
10. Символдық айнымалыларды сипаттау.
11. Жолдық тұрақтылардың жазылуы.
12. Жолдар және көрсеткіштер түсінігі.

VI БӨЛІМ. ФУНКЦИЯЛАР

6.1 Функцияның параметрлерінде көрсеткіштерді қолдану

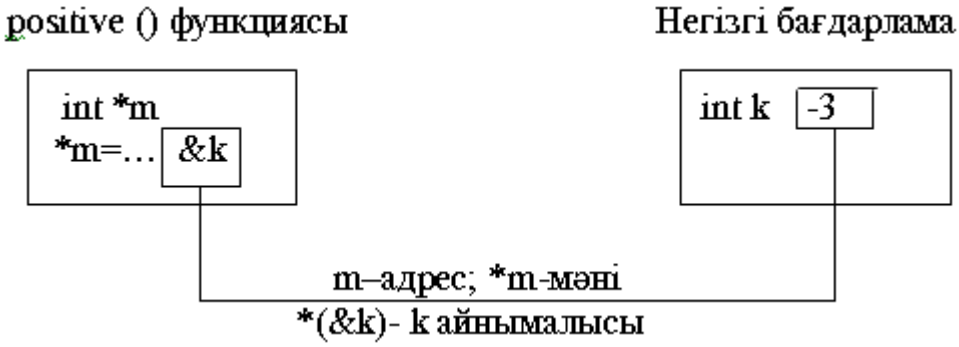
Алдағы тақырыптарда негізгі бағдарламада функцияны шақырғанда осы функцияны сипаттаған кезде анықталған формалды параметрлерге нақты параметрлердің берілу механизмі қарастырылады. Негізгі бағдарламада шақырылып тұрған функцияның нақты параметрі ретінде қолданылып тұрған объект осы функцияның денесінде орналасқан операторлардың әсерімен өзгертілмейді. Бірақ функцияны шақырып тұрған бағдарламаның объектілерінің мәндерін осы шақырылған функцияда орындалатын іс-әрекеттердің көмегімен өзгертуге мүмкіндік беретін құралдар бар. Олар көрсеткіштер. Көрсеткіштің көмегімен шақырылатын бағдарламаның кез-келген объектілерінің адресін осы шақырылатын функцияға беруге болады. осылайша, параметрдің өзін өзгертпей-ақ функцияны шақырып тұрған бағдарламаның объектісін өзгертуге болады. Келесі бағдарламаны қарастырамыз:

```
# include  <stdio. h>
  void positive (int * m)
{
  *m=*m > ? *m: -*m;
}
void main ()
{
  int k=-3;
  positive (&k);
  printf (“\nk=%d”, k );
}
```

Бағдарламаның орындалу нәтижесі : k=3

int* типіндегі көрсеткіш-бұл positive() функциясының параметрі main() негізгі бағдарламасынан бұл функцияны шақырғанда нақты параметр ретінде int типіндегі &k айнымалысының адресі қолданылады. Функцияда аргументтің мәні, яғни &k айнымалының адресі, int *m көрсеткішіне бөлінген жадының облысына жазылады. *m операциясы m көрсеткіші сілтемеленетін жадының бөлігімен жұмыс жасуға мүмкіндік береді. Осылайша мына өрнекте *m = *m> ? *m: -*m барлық іс-әрекеттер адресі &k нақты параметр ретінде қолданылып тұрған негізгі бағдарламаның айнымалысының int k мәндеріне орындалады.

positive() функциясымен негізгі бағдарламаның арасындағы байланыс келесі суретте көрсетілген (6.1-сурет). positive() функциясын шақырған кезде ол абсолюттік мәнді, адресі оның параметрі ретінде қолданылып тұрған айнымалыға меншіктеледі.



6.1-сурет. Көрсеткіш-параметрін «баптау» сызбасы

Ішкі бағдарламалар Си тілінде жоқ, бірақ егер де функцияның шақырылуы оператор-өрнек түрінде қолданылса, онда ішкі бағдарламаны шақыру операторына бара-бар команда аламыз.
Мысалы:

```
void print (int  gg, int mm,  int dd)
{
  printf (“\ n  жыл: %d ”, gg );
  printf (“\ t  ай: %d ”,mm );
  printf (“\ t  күн: %d ”,dd  );
}
```

бағдарламада былай шақырсақ print(2011,10,12)
год:2011, месяц:10, день:12

Бұл функция басқа бағдарламалау тілдеріндегі ішкі бағдарламаға өте ұқсайды. Си тілінде функцияларды ішкі бағдарламалар ретінде қолданудың тағы бір қиындығы - ол жоғарыда анықталған функцияның параметрлерінің мәндерінің ғана берілуі, яғни айнымалылардың адрестерінің емес мәндерінің берілуі. Басқа сөзбен айтқанда функцияның орындалу нәтижесінде оның нақты параметрлерінің мәндерін өзгертуге болмайды. Мұндай есептерді шешудің мынандай жолы бар. Ол үшін функцияның анықтамасында, оның параметрлері көрсеткіштер ретінде сипатталуы керек. Мұндай жағдайда осы көрсеткіштердің көмегімен функцияның денесінен, параметр-көрсеткіштермен адрестелетін функцияны шақыратын бағдарламаның объектілерімен жұмыс істеуге мүмкіндік алуға болады. Келесі бағдарламаны қарастырайық:

```
# include  <stdio.h>
   void main ()
{
  float x,y;
  void aa (float *, float *);
  printf (“\ n   x-тің мәнін енгіз:  ”);
  scanf (“%f”,&x) ;
  printf (“\ n y-тің мәнін енгіз:  ” );
```

```
  scanf (“%f”,&y);
  aa (&x, &y);
  printf (“\ n  нәтиже: x=%f y=%f ”, x,y );
}
void  aa(float * b, float * c)
{
  float  e;
  e=*b;
  *b=*c;
  *c=e;
}
```

Негізгі бағдарламада x,y айнымалылары сипатталған, олардың мәндері пернелік тақтадан енгізілген. aa() функциясының формалді параметрлері ретінде float* типіндегі көрсеткіштер қолданған. Функцияның міндеті көрсеткіш параметрлер көрсететін айнымалылардың мәндерінің орнын ауыстыру. aa() функциясын шақырғанда x,y айнымалыларының адрестері нақты параметрлер ретінде қолданылады. Сондықтан бағдарламаның орындалу нәтижесі:

```
x-тің мәнін енгіз: 33.3
y-тің мәнін енгіз: 66.6
нәтиже: x=66.600000   y=33.300000
```

Үшбұрыштың ауданын және периметрін есептейтін функция құрамыз. Бұл функцияның да формалді параметрлері ретінде float* типіндегі көрсеткіштерді қолданамыз:

```
# include <math.h>
# include <stdio.h>
void main ()
{
  float x,y,z,pp,ss;
  int  tt(float, float, float, float*, float*);
  printf (“\ n   x=  ”);
  scanf (“%f”,&x) ;
  printf (“\ t y=” );
  scanf (“%f”,&y);
  printf (“\ t z=” );
  scanf (“%f”,&z);
  if (tt(x,y,z,&pp,&ss)==1)
    {
      printf (“Периметрі =%f”,pp);
      printf (“,Ауданы =%f”,ss);
    }
```

```
  else
  printf ("\ n   Енгізілген мәнде қате");
}
int  tt(float a, float b, float c, float *perim, float *area);
{
  float   e;
  *perim=*area=0.0;
if (a+b<=c || a+c<=b || b +c<=a)
   return  0;
  *perim=a+b+c;
  e=*perim/2;
  *area=sqrt(e*(e-a)*(e-b)*(e-c));
return  1;
}
```

Бағдарламаның орындалу нәтижесі:

```
Енгізіңіз x=3
          y=4
          z=5
Периметрі =12.000000  , Ауданы =6.000000
```

6.2 Функцияның параметрлері ретінде массивтер мен жолдарды қолдану

Егер де функцияның параметрі ретінде массивтің сипаттамасы қолданылса, онда функцияға осы массивтің алғашқы элементінің адресі беріледі.

Мысалы:

```
float zz(int n,  float a[ ],  float b[ ])….
```

Бұл функцияны былай да жазуға болады:

```
float  zz(int n,  float *a, float *b)….
```

Бірінші функцияның денесінде массивтер-параметрлер элементтерін шақыру, a[i] және b[i] индекстелген элементтерінің көмегімен жүзеге асырылады. Бірақ массив элементтеріне, оларды сәйкес келетін адрестердің мәндерімен ауыстырып, яғни *(a+i) және *(b+i) өрнегін қолдануға болады.

Массив функцияға әрқашанда көрсеткіш ретінде берілетін болғандықтан, функцияда шақырылатын бағдарламада анықталған нақты параметр орнында қолданып тұрған массив элементтерінің мәндерін өзгертуге болады. Бұл индестелген массивті қолданғанда және массив элементтерін көрсеткіштерге ауыстырғанда орындалуы мүмкін. Осы мүмкіндіктерді көрсету үшін бір өлшемді массив элементтерінің квадратын шығаратын функцияны қарастырамыз:

```
# include <stdio.h>
void  quart(int  n,  float * x)
{
  int  i;
  for  (i=0;  i<n;  i++)
  *(x+i)*=*(x+i);
}
void main ()
{
  float  z[ ]={1.0, 2.0, 3.0, 4.0};
  int  j;
  quart (4, z);
  for (j=0; j<4; j++)
  printf (“\ n z[%d]=%f  ”, j, z[j]);
}
```

Бағдарламаның орындалу нәтижесі:

```
z[0]=1.000000
z[1]=4.000000
z[2]=9.000000
z[3]=16.000000
```

Функцияның параметрлері ретінде бір типті массивтерді және көрсеткіштерді қолданудың бірдей (тең) екендігін көрсету үшін, мына функцияның тақырыбын былай жазуға да болады.

```
void quart (int  n,  float x[ ])
```

Функцияның денесінде массив – параметрдің аты бар ауыстыру өрнегі қолданылған, яғни x[i] индекстелген айнымалысының орнына *(x+i) өрнегі қолданылады.

Жолдарды функциялардың параметрлері ретінде қолдану. Жолдарды функциялардың нақты параметрлері ретінде қолданғанда, олар не char[] типіндегі бірөлшемді массивтер, не char* типіндегі көрсеткіштер ретінде сипатталады. Екі жағдайларда да параметрлердің көмегімен функцияға осы жол меншіктелген символдық массивтің бастапқы адресі беріледі.

Жолдарды параметр ретінде қолданғанда, олардың жай массивтерден айырмашылығы оларды сипаттаған кезде массивтің ұзындығы көрсетілмейді. Әр бір жолдың соңына қойылатын ’\0’ символы, осы жолдың ұзындығын анықтауға мүмкіндік береді.

Төменде қарастырылатын функциялардың көбінің стандартты функциялардың библиотекаларында бара-бар функциялар (аналогтары) бар. Олар string.h және stdlib.h тақырыптық файлдарында сипатталған. Жолдың

ұзындығын есептейтін функция. Стандартты функцияларда бұл функцияға strlen() функциясы сәйкес келеді.

```
int  len (e)
char  e[ ];
{
 int  m;
 for (m=0;  e[m]!= '\0'; m++)
 return m;
}
```

Бұл функцияның тақырыбын былай да сипаттауға болады.

```
int  len (char  e[ ])
```

Бұл мысалыда функцияның тақырыбында, денесінде де массивтермен көрсеткіштердің арасындағы байланыс көрсетілмеген. Бірақ компилятор әрқашанда массивті, оның бастапқы адресіне сілтеме ретінде, ал индексті массвтің басынан оған дейінгі арақашықтық ретінде қабылдайды. Осы функцияның төмендегі нұсқаларында көрсеткіштермен жұмыс жасау құрылымдары қолданылған.

```
int  len (char *s)
{
 int  m;
 for (m=0;  *s++!='\0'; m++)
 return  m;
}
```

Формалді s көрсеткіш-параметріне функцияда нақты параметрдің мәні жазылатын жадының облысы бөлінеді. s-тұрақты болмағандықтан оның мәні өзгереді, сондықтанда мысалда s++ өрнегі қолданылады.

Жолдың аргументтерін кері ретпен шығаратын функция. Функцияның тақырыбы тілдің стандартына сәйкес келеді.

```
void invert (char e[ ])
{
 char  s;  int i, j, m;  /*m-e жолындағы '\0' символының  позициясының
                          нөмрі*/
  for (m=0; e[m]!= '\0'; m++)
  for (i=0, j=m-1; i<j; i++,  j--)
    {
     s=e[i];
     e[i]=e[j];
     e[j]=s;
```

```
    }
  }
```

Функцияның тақырыбында void қызметші сөзі қолданған. Бұл функцияның ешқандай мән қайтармайтынын көрсетеді. Бұл функцияны қолданатын мысалды қарастырайық:

```
# include <stdio.h>
void main ()
{
 char   ct[ ]=”0123456789”;
 void   invert (char[ ]);
 invert (ct);
 printf (“\ n %s”, ct);
}
```

нәтижесі:

9876543210

Жолдың ішінен сол жолда бар басқа жолды табу функциясы. Стандартты библиотекада бұл функцияға ұқсас **strstr()** функциясы бар. /* CT1 жолында C2 жолын табу*/

```
int  index (char * CT1,  char * CT2)
{
 int i, j, m1, m2;
 /* m1 және m2 жолдардың ұзындығы*/
 for (m1=0;  CT1[m1]!= ’\0’; m1++);
 for (m2=0;  CT2[m2]!= ’\0’; m2++);
 if (m2>m1)   return  -1;
 for (i=0;  i<=m1-m2; i++)
  {
    for (j=0;  j<m2; j++)
      if (CT2[j]!=CT1[i+j])
        break;
      if (j==m2)
        return  i;
  }
 return  -1;
}
```

Бұл функция CT1 жолының CT2 жолымен толық сәйкес келетін позизияның нөмрін қайтарады. Егер де CT2 жолы CT1 жолында жоқ болса, онда -1-ге тең мән қайтарады.

index функциясының қолданылуы:

```
# include <stdio.h>
void main ( )
{
  char C1[ ]=”Іазақ сөздігі”;
  int  index (char [ ], char[ ]);
  char C2[ ]=” сөз”;
  printf (“\ n %s үшін индекс =%d”,C2, index (C1,C2));
}
```

нәтиже сөз үшін индекс=6

Жолдарды салыстыру функциясы стандартты библиотекаларда бұған ұқсас **strncmp( )** функциясы бар.

Жолдарды салыстыруға тағы бір мысал қарастырайық:

```
int row (char C1[ ], char C2[])
{
  int  i, m1, m2;
  for (m1=0; *(C1+m1)= ’\0’; m1++);
  for (m2=0; *(C2+m2)= ’\0’; m2++);
  if (m1!=m2) return  -1;
  for (i=0; i<m1; i++)
  if (*C1++ !=* C2++)    return (i+1);
 return  0;
}
```

Функцияның денесінде жолдық массивтің элементтерін шақыру көрсеткіштердің көмегімен жүзеге асырылған. Егер де C1, C2 жолдарының аргументтерінің ұзындығы әртүрлі болса, онда функция -1 тең мән қайтарады, егер де барлық символдары сәйкес келсе онда 0-ге тең мән қайтарады. Егер де жолдардың ұзындықтары бірдей, бірақ символдары сәйкес келмесе, онда функция бірінші сәйкес келмеген символдарының реттік нөмрін қайтарады.

Жолдарды бір-біріне қосу функциясы. Келесі функция бірінші жолдық - аргументке, екінші жолдық-аргументті қосады.

```
void  conc(char *C1,  char *C2)
{
   int  i, m;
  /*m-бірінші жолдың ұзындығы*/
  for (m=0; *(C1+m)!= ’\0’; m++);
  for (i=0; *(C2+i)= ’\0’; i++);
  *(C1+m+i)=*(C2+i);
  *(C1+m+i)= ’\0’;
```

```
}
```

Нәтижесінде бірінші С1 аргументінің мәні қайтарылады. Екінші С2 аргументі өзгермейді. Жолдарды қосу функциясы стандартты библиотекаларда **strncat( )** функциясы бар.

Жолдан жолды бөлу функциясы.

```
void  substr (char *C1, char *C2, int n, int k)
/*C1-ағымдағы жол*/
/*C2 – бөліп алатын жол*/
/* n-бөліп алатын жолдың бастапқы позициясы*/
/* k-бөліп алатын жолдың ұзындығы*/
{
  int i,m; /*m-Берілген қатардың ұзындығы*/
  for (m=0; C1[m]!= ’\0’; m++);
  if (n<0 || n>m || k<0 || k>m-n )
    {
     C2[0]= ’\0’;
     return;
    }
 for  (i=n; i<k+n; i++)
    C2 [i-n]=C1[i-1];
  C2 [i-n]= ’\0’;
 return;
}
```

Функцияның орындалу нәтижесінде С1[ ] жолынан n нөмрінен басталатын символдан бастап ұзындығы k-ға тең символдардан тұратын С2[ ] жолы бөлінеді. Мәндер дұрыс енгізілмеген жағдайда, бос жол қайтарылады.

Жолды көшіру функциясы. Си тілінде жолдарды меншіктеу операторы болмағандықтан бір жолды екінші жолға көшіретін функцияны қолданған ыңғайлы. С2 жолын С1 жолына көшіру функциясы.

```
void  copy (char *C1, char *C2)
/*C2-түпнұсқа ,  C1-көшірме*/
{
 int i;
 for (i=0;  C2[i]!= ’\0’; i++)
 C1[i]=C2[i];
 C1[i]= ’\0’;
}
```

copy () функциясын қолдану мысалы:

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

```
void main ()
{
 char X[ ]=”GAME OVER!”;
 void  copy (char[ ], char[ ]);
 char  B[100];
 copy (B,X);
 printf (“%s\ n”, B);
}
```

Жолдарды көшіру функциясының екінші жолы:

```
void  copy (char C1[ ],  char  C2[ ])
{
 int  i=0;
 do
   {
    C1[i]=C2[i];
   }
 while (C1[i++]!=’\0’);
}
```

Бұл функцияда меншіктеу операциясы қайталану операторының жалғасуын ұйымдастыратын өрнек-шартта орналасқан. Операциялардың орындалу дәрежесіне байланысты меншіктеу өрнегі жақшаларға алынған.

```
void  copy (char* C1,  char * C2)
{
 int  i=0;
 while (C1[i]=C2[i]!= ’\0’)
 i++;
}
```

Бақылау сұрақтары

1. Функцияның параметрлеріне көрсеткіштерді қолдану жолдары.
2. pasitive() функциясына мысал келтіріңіз.
3. Функцияның параметрлері ретінде жолдарды қолдану.
4. Жолдың ұзындығын анықтау функциясына мысал келтіріңіз.
5. Жолдың ішінен сол жолда бар басқа жолды табу функциясына мысал келтіріңіз.
6. Жолдарды салыстыру функциясына мысал келтіріңіз.
7. Жолдарды бір-біріне қосу функциясы.
8. Жолдан жолды бөлу функциясы.
9. Жолды көшіру функциясы.

VII БӨЛІМ. СИ БАҒДАРЛАМАЛАУ ОРТАСЫНДА ЕНГІЗУ ЖӘНЕ ШЫҒАРУ ФУНКЦИЯЛАРЫН ҚОЛДАНУ

Си тілінің стандартында енгізу-шығару операторлары жоқ. Енгізу-шығару операцияларының барлығы Си тілінің библиотекаларында орналасқан функциялардың көмегімен жүзеге асырылады. Си тілінің библиотекасында енгізу-шығарудың үш деңгейі бар:

- легтік енгізу-шығару;
- төменгі деңгейлі енгізу-шығару;
- консолдарға және порттарға арналған енгізу-шығару.

Бұлардың ең соңғы деңгейі дисплеймен және енгізу-шығару порттарымен жұмыс жасайтын болғандықтан жүйеге өте тәуелді, сондықтан оларды қолданған жағдайда мұқият болуы керек.

7.1 Легтік енгізу-шығару

Легтік енгізу-шығару деңгейінде мәліметтермен алмасу байт бойынша жүзеге асырылады. Дискіден енгізілген, яғни файлдан оқылған жағдайда мәліметтер операциялық жүйенің буферіне жазылады да сонан соң байт бойынша немесе белгілі бір бөліктермен қолданушының бағдарламасына беріледі. Мәліметтерді файлға жазған кезде, алдымен олар буферге жазылады, буфер толғаннан кейін файлға жазылады да, мәліметтердің келесі бөлігі қайтадан буферге жазылады, осылай ары қарай мәліметтер біткенше жалғаса береді. Операциялық жүйенің буфері негізгі жадының бөлігі ретінде жүзеге асырылады. Сондықтан да енгізу-шығару буферімен орындалатын бағдарламаның арасындағы мәліметтер алмасу өте жылдам орындалады.

Си тілінің енгізу-шығару библиотекаларының функциялары файлдармен жұмыс жасағанда мәліметтер алмасу легтермен жүзеге асырылады және әртүрлі көлемдегі, форматтағы мәліметтерді өңдеуге мүмкіндік береді. Сонымен *лег* дегеніміз - бұл файл. Легтермен жұмыс жасағанда келесі іс-әрекеттерді орындауға болады:

- легтерді ашу және жабу (көрсеткіштерді нақты файлдармен легтермен байланыстыру);
- символдарды, жолдарды, форматталған мәліметтерді енгізу және шығару;
- легтік енгізу-шығарудың қателерін талдау;
- буфердің көлемін және буферленуін басқару;
- легтің ағымдағы позициясына көрсеткіш қою.

7.2 Легтерді ашу және жабу

Легтермен жұмыс жасау үшін алдымен оны инициализациялау керек, яғни оны ашу керек. Мұндай жағдайда, лек орындалатын бағдарламада алдын ала анықталған FILE типінің құрылымымен байланысады. FILE құрылымдық типі stidio.h тақырыптық файлында анықталған. FILE типінің құрылымында лектермен жұмыс жасауға мүмкіндік беретін компоненттері бар. Атап айтқанда, буферге сілтемеленетін көрсеткіш, легтің ағымдағы орнына сілтемеленетін көрсеткіш және басқа да мәліметтер.

Легті ашқан кезде бағдарламаға FILE құрылымдық типінің көрсеткіші болып есептелетін, осы легке сілтемеленетін көрсеткіштің мәні қайтарылды. Легке сілтемеленетін көрсеткіш мысалы fp, төмендегідей түрде сипатталады.

```
# include <stdio.h>
 FILE  *fp;
```

Легке сілтемеленетін көрсеткіш осы легті ашу функциясының орындалу нәтижиесінде мән қабылдайды.

```
fp= fopen (файлдың_аты, ашу_режимі)
```

Файлдың_аты және ашу_режимі параметрлері осы легпен байланысқан файлдың аты бар, символдық массивке сілтемеленетін көрсеткіш болып есептеледі. Бұл параметрлер файлды ашу функциясында жол түрінде жазылуы да мүмкін.
Мысалы:

```
fp= fopen (“t.txt”,”r”);
```

Бұл мысалыдағы t.txt-легпен байланысқан, файлдың аты;
r- файлдармен жұмыс жасағандағы режимдердің бірінің берілуі;

Легпен байланысқан файлды стандартты түрде ашу келесі алты режимдердің бірімен жүзеге асырылады:

- “w” – жазу үшін жаңа текстік файл ашу. Егер де файл бар болса, онда алдыңғы файл жойылып, файл қайтадан құрылады.
- “r” – текстік файл тек оқу үшін ашылады. Ол үшін бұл файл дискіде міндетті түрде болуы керек.
- “a” – текстік файл, оған ақпараттың жаңа легін қосу үшін ашылады. Егер де файл дискіде жоқ болса, онда жаңадан құрылады. Оның “w” режимінен айырмашылығы егер де файл дискіде бар болса, онда ол файлды жоймай ақпаратты файлдың соңына жазылады.
- “w+” – жаңа текстік файл жазу үшін және бірнеше рет дұрыстау үшін ашылады. Егер де файл бар болса, онда файлдың ішіндегі мәліметтер жазу және оқу файлдың кез-келген жерінен жүргізуге болады. Файлдың соңына да ақпарат жазуға, яғни файлдың көлемінің ұлғаюы мүмкін.
- “r+” – дискіде бар текстік файл оқу және жазу үшін ашылады. Жазу және оқу файлдың кез-келген жерінен жүргізіледі, бірақ файлдың соңына жазуға болмайды, яғни файлдың көлемі өзгереді.
- “a+” – текстік файл ашылады, егер де жоқ болса жаңадан құрылады және оның ішіндегі ақпаратты өзгертуге мүмкіндік береді, яғни файлдың кез-келген жерінен оқуға болады. “w+” режимінен

айырмашылығы дискіде бар файлдың мазмұнын жоймайды. “r+” режимінен айырмашылығы файлдың соңына жазуға болады.

Легті текстік не екілік бинарлы режимде ашуға болады. Егер де лег өзгеру үшін ашылса, яғни режим параметрінде “+” символы бар болса, онда бұл легке жазуға да, және одан оқуға да болады. Бірақ режимдерді ауыстыру (яғни жазу режимінен оқу режиміне өту және керісінше) легтің көрсеткішін қажет позицияға қойғаннан кейін жүзеге асырылады. Легті ашқан кезде келесі қателер туындауы мүмкін: осы легпен байланысты файл дискіде жоқ (оқу режимі үшін), дискіде бос орын жоқ, дискіге жазуға рұқсат жоқ және т.б. fopen() функциясы орындалғанда динамикалық жады бөлінеді. Егер де жады жеткіліксіз болса, онда “Not enough memory” (жады жеткіліксіз) деген қате шығарады. Мұндай жағдайларда легке сілтемеленетін көрсеткіш NULL мәнін қабылдайды. Файлды ашқанда қолданылатын операторлар тізбегін қарастырамыз.

```
if ((fp=fopen (“t.txt”,”w”))==NULL)
{
  perror (“t.txt файлын ашуда қате\ n”);
  exit (0);
}
```

NULL-stdio.h файлда анықталған, нөлдік көрсеткіш.

Легті ашқанда пайда болуы мүмкін қатені дисплейдің экранына шығару үшін perror() стандартты функциясы қолданылған. Файл ашылғаннан кейін онымен жұмыс жасуға болады, яғни оған ақпарат жазуға және оқуға болады. Жұмыс жасап болғаннан кейін дискідегі ашық болған файлдарды жабу керек. Ол үшін келесі библиотекалық функция қолданылады:

int fclose (легке_сілтемеленетін _ көрсеткіш);

Ашылған файлда, қайтадан жұмыс жасу үшін, мысалы мазмұнын өзгерту үшін осы файлды міндетті түрде **fclose()** функциясымен жабу керек.

7.3 Стандартты файлдар және олармен жұмыс жасауға арналған функциялар

Бағдарлама орындала бастағанда автоматты түрде бес лег ашылады. Олардың ішінен ең негізгілері болып:

- стандартты енгізу легі (**stdin**);
- стандартты шығару легі (**stdout**);
- стандартты қателерді шығару легі (**stderr**);

болып табылады.

Бұл легтер басқа іс-әрекеттер орындамаған жағдайда, **stdin** стандартты енгізу легіне пернелік тақта сәйкестендіріледі, ал **stdout** және **stderr** легтеріне

дисплейдің экраны сәйкес келеді. Стандартты лектердің көмегімен мәліметтерді енгізу-шығару үшін Си тілінде келесі функциялар анықталған:

- **getchar()/putchar()** - символды енгізу-шығару;
- **gets()/puts()** - жолды енгізу-шығару;
- **scanf()/printf()** - мәліметтерді форматтау режимінде енгізу-шығару;

Символдарды енгізу-шығару. Символдарды енгізу-шығару үшін библиотекалық **getchar()** және **putchar()** функциялары қолданылады. **getchar()** функциясы бір символды енгізуді орындайды. Оны қолданғанда шақырған функцияға енгізілген бір символдың мәні қайтарылады. **putchar()** функциясы стандартты легке бір символ шығарады және олда шақырған функцияға шығарылған символдың мәнін қайтарады.

getchar() функциясы символды int типіндегі мән ретінде енгізіледі. Бұл **getchar()** функциясының көмегімен файлдағы ақпаратты оқығанда, осы файлдың соңына жеткендегі операцияның дұрыс орындалуын қамтамасыз етеді. Мұндай жағдайда операциялық жүйе келесі символды оқу әрекетінде функцияға **EOF** (end of file) (файлдың соңы) мәнін қайтарады.

EOF тұрақтысы stdio.h тақырыптық файлында анықталған және әртүрлі операциялық жүйелерде 0 немесе –1 мәнін қабылдайды. Сондықтан **getchar()** функциясының енгізілетін легтен тек символды ғана емес, бүтін мәнді де оқуға мүмкіндігі бар. Қате енгізілген жағдайда да **getchar()** функциясы **EOF** мәнін қайтарады. **getchar()** және **putchar()** функциялары **stdio.h** тақырыптық файлында анықталған макростар болып табылады. **gets()** және **puts()** функциялары да макростар ретінде орындалады.

Пернелік тақтадан мәтін терілген кезде символдардың кодтары операциялық жүйенің ішкі буферіне жазылады және бір уақытта көру үшін дисплейдің экранына шығарылады. Пернелік тақтадан терілген символдарды өшіруге басқа символдарды теруге болады. Символдарды ішкі буферден бағдарламаға ауыстыру “Enter” пернесін басқаннан кейін орындалады. Бұл жағдайда “Enter” пернесінің коды да ішкі буферге енгізіледі.

Мысалы, егер де ’A’ пернесін және “Enter” пернесін басқан кезде, ішкі буферге “A” символының және “Enter” пернесінің кодтары жазылады. getchar () функциясы аралық мәндерді талдау жасағанда бағдарламаның жұмысын уақытша тоқтату үшін қолданатын бағдарламаның үзіндісін қарастырайық.

```
#include<stdio.h>
int main()
  {
    ptintf(“a”);
    getchar();
    printf(“b”);
    getchar();
    printf(“c”);
    return 0;
  }
```

Бұл бағдарлама орындалғанда алдымен экранға **a** символы шығарылып, бағдарламаның жұмысы келесі кез-келген символды енгізгенше тоқтатылады. Егерде мысалы кез-келген символдық пернені және “Enter” пернесін басатын болсақ, онда келесі жолға **bc** символдары шығарылады да бағдарлама өзінің жұмысын аяқтайды.

Бағдарламадағы бірінші орналасқан **getchar()** функциясы ішкі буферден енгізілген символдың кодын оқиды, ал одан кейін орналасқан **printf()** функциясы экранға **b** символын шығарады. Бағдарламаның жұмысы тоқтатылмайды, өйткені **getchar()** функциясы ішкі буферден “Enter” пернесінің кодын оқиды. Өйткені екінші функциясын шақырғанда ішкі буфер бос болған жоқ. Бұл бағдарлама дұрыс орындалады егерде бағдарламаның жұмыс істеуі тоқталады егер тек қана “Enter” пернесі басылған кезде **putchar()** функциясы параметрі ретінде берілген символды стандартты шығару құрылғысына шақырды. Келесі мысалдарды келтіреміз:

```
    int c;
1:  c=getchar();
2:  putchar(c);
3:  putchar('A');
4:  putchar('\007');
5:  putchar('\t');
```

Бағдарламаның екінші жолында, алдындағы **getchar()** функциясының көмегімен енгізілген символ экранға шығарылады. Үшінші жолда ’A’ символы экранға шығарылады. Төртінші жолда 007 кодымен (дыбыстық сигнал) берілген басқару символы шығарылады. Ең соңғы жолда көрінбейтін, курсорды табуляцияның келесі позициясына ауыстыратын басқару табуляция символы шығарылады.

getchar() және **putchar()** функцияларының қолданылуын келесі мысалда келтіреміз:

```
#include<stdio.h>
int main()
{
   int c;
   while((c=getchar())!=EOF)
        putchar(c);
   return 0;
}
```

Бұл бағдарлама **getc()** және **putc()** функцияларының көмегімен былай жазылады:

```
#include<stdio.h>
 int main()
{
   int c;
   while ((c=getc(stdin))!=EOF)
     putc(c,stdout);
   return 0;
}
```

Бұл бағдарламалардың жұмысын тоқтату үшін ctrl+C пернелерін бірге басу керек.

Ең көп қолданылатын енгізу – шығару операцияларының бірі болып символдар жолын енгізу - шығару операциясы есептеледі. Си тілінің библиотекасына стандартты енгізу - шығару легтерінің көмегімен мәліметтермен алмасу үшін жолдарды енгізу – шығару функциясы **gets()** және **puts()** енгізілген. Екі функция да бір аргумент бар – ол осы символ массивіне сілтемеленетін s көрсеткіші. Егерде жол дұрыс оқылса, онда **gets()** функциясы осы жол енгізілген s массивінің адресінің мәнін қайтарады. Егерде қате болса онда NULL мәнін қайтарады. **puts()** функциясы дұрыс орындалған жағдайда әрқашанда ’\n’ символы болатын ең соңғы енгізілген символдың мәнін қайтарады. Егерде қате болса **EOF** мәні қайтарылады. Осы функцияларды қолданудың қарапайым мысалын қарастырамыз:

```
#include<stdio.h>
char str1[ ]=”Қызметтердің аты-жөнін енгіз:”;
int main()
{
  char name[80];
  puts(str1);
  gets(name);
  return 0;
}
```

Әрқашанда Си тілінде кез–келген символдар жолы нөлдік символмен аяқталу керек ’\0’. Мысалда str1 массивінің ең соңғы элементіне нөлдік-символ трансляция кезінде автоматты түрде жазылады.

puts() функциясы үшін жолдың соңында нөлдік–символдың болуы міндетті. Кері жағдайда, яғни символдық массивтің соңында нөлдік–символ болмаса онда бағдарлама орындалуы тоқтатылады.

gets() функциясы өзінің жұмысын ’\n’ символы енгізілгенде тоқтатады. Ол пернелік тақтадан ЭЕМ-ге “Enter” пернесі басылғанда автоматты түрде енгізіледі және ’\n’ символының өзі енгізілген жолға жазылмайды. Оның орнына жолдың соңына нөлдік-символ енгізіледі. Сондықтан **gets()** функциясы символдар тізбегінің енгізілуін ғана емес, осы жолдың дұрыс енгізілуін қамтамасыз етеді. Бұл жерде мәліметтердің пернелік тақтадан енгізілуінің

мынадай ерекшелігіне мән беру керек. **gets()** фунциясы пернелік тақтадан енгізілген ақпараттың өңделуін тек “Enter” пернесі басылғаннан кейін орындайды. Сондықтан да ол керек ақпарат енгізіліп “Enter” пернесі басылғанша күтеді. Тек осы іс – әрекеттерден кейін ғана мәліметтерді бағдарламаға енгізу орындалады.

7.4 Дискідегі файлдармен жұмыс жасау

Дискідегі файлдармен жұмыс жасау үшін Си тілінде келесі функциялар қолданылады:

fgetc(), getc() - файлдан символды оқу;
fputc(), putc() - файлға символды жазу;
fprintf() - файлға форматталған түрде жазу;
fscanf() - файлдан форматталған түрде оқу;
fgets() - файлдан жолды оқу;
fputs() - файлға жолды жазу;

Екілік режимде файлдармен ақпарат алмасу. Екілік режимде ақпарат алмасу **getc()** және **putc()** функцияларының көмегімен жүзеге асырылады. Олар бағдарламада былай қолданылады:

```
c=getc(fp);
putc(c,fp);
```

Бұл жерде fp-легке немесе файлға сілтенетін көрсеткіш;
c-файлдан келесі символды қабылдауға және файлға символдың мәнін жазуға орнатылған int типіндегі айнымалы.

getc() және **putc()** функцияларын қолдану мысалы ретінде пернелік тақтадан мәліметтерді файлға енгізетін және файлдан мәліметтерді экранға шығаратын бағдарламаларды қарастырамыз.

Енгізу бағдарламасы символдарды пернелік тақтадан оқып, файлға жазады. Мәліметтерді енгізудің соңының белгісі ретінде ’#’ символын қолданамыз. Файлдың атын қолданушы енгізеді. Егер де символдар тізбегін енгізгенде “Enter” пернесі болған болса, файлға жолдарды бөлу үшін қолданылатын басқару кодтары жазылады. Бұл кодтардың мәндері бағдарламада CR және LF идентификаторымен сипатталған.
Символдарды файлға жазу бағдарламасы:

```
#include<stdio.h>
int  main()
{
    FILE *fp;
    char c;
    const char CR=’\015’;
  const char LF=’\012’;
    char fname[20];
```

```
        puts(“Файлдың атын енгіз:\n”);
        gets(fname);
        if ((fp=fopen(fname,”w”))==NULL)
            {
              perror(fname);
              return 1;
            }
          while((c=getchar())!=’#’)
            {
             if  (c==’\n’)
               {
                   putc(CR,fp);
                   putc(LF,fp);
               }
             else putc(c,fp);
        }
            fclose(fp);
            return 0;
    }
```

Келесі бағдарлама файлдан символдар легін оқып, оны дисплейдің экранына шығарады.

```
#include<stdio.h>
int main()
{
  FILE *fp;
  char c;
  char fname[20];
 puts(“Файлдың атын енгіз:\n”);
 gets(fname);
 if ((fp=fopen(fname,”r”))==NULL)
    {
   perror(fname);
   return 1;
  }
 while((c=getc(fp))!=EOF)
           putchar(c);
 fclose(fp);
 return 0;
}
```

Файлдағы символдарды кадрлармен шығару үшін бұл бағдарламаны былай жазуға болады:

```
#include<stdio.h>
int main()
{
  FILE *fp;
  char c;
  const char LF='\012';
  int MAX=10;
  int n;
 char fname[10];
 puts("Файлдың атын енгіз:");
 gets(fname);
 if ((fp=fopen(fname,"r"))==NULL)
{
    perror(fname);
    return 1;
}
while(1);
{
  n=1;
  while(n<MAX)
  {
    c=getc(fp);
    if (c==EOF)
    {
  fclose(fp);
  return 0;
     }
     if (c==LF) n++;
  putchar(c);
  }
    getchar();
  }
}
```

Бұл бағдарлама орындалғанда экранға файлдан символдардың белгілі бір бөлігі шыққаннан кейін кез-келген пернені басқанша бағдарламаның орындалуы тоқтатылады.

Файлдармен жұмыс жасағанда жолдармен алмасу. Текстік файлдармен жұмыс жасағанда **fgets()** және **fputs()** функцияларын қолданған тиімді. Бұл функциялардың **stdio.h** тақырыптық файлдағы сипаттамасы төмендегідей:

```
int fpurs(const char *s, FILE *stream);
char *fgets(char *s,int n, FILE *stream);
```

fputs() функциясы ’\0’ символымен шектелген және легке сілтемеленетін stream көрсеткішімен анықталған жолды файлға жазады және оң бүтін сан қайтарады. Қате болған жағдайда **EOF** мәні қайтарылады. **fgets** функциясы stream көрсеткішімен анықталған файлдан (n-1) үлкен емес символдар тізбегін оқып, s көрсеткіші сілтемеленетін жолға жазады.

Функция жұмысын (n-1)-ге тең символдар тізбегін оқып біткеннен кейін немесе жолдың соңын білдіретін ’\n’ символын оқығанда тоқтатады. Қосымша, әрбір жолдың соңына, осы жолдың біткенін білдіретін ’\0’ нөлдік символ жазылады. Қате болған жағдайда NULL мәні қайтарылады. Осы функциялардың мүмкіндіктерін көрсету үшін бір файлдың мазмұнын екінші файлға көшіретін бағдарламаны қарастырамыз.

Си тіліндегі кез-келген бағдарлама орындала бастағанда ол операцияның жүйеден екі аргументтің (argc және argv) мәні туралы мәлімет алады. (argument count, argument vector). Бірінші аргумент, **argv** көрсеткіштерінен массив түрінде берілетін жолдың ұзындығын анықтайды.

Келісілген шартқа байланысты argv[0] әрқашанда осы орындалатын бағдарлама жазылған файлдың атына сілтемеленетін көрсеткіш, массив қалған элементтері argv[1] ,…, argv[argvc-1] командалық жолда бағдарламаның атынан соң бос орын арқылы жазылатын параметрлерге сілтенетін көрсеткіштер. Келесі бағдарламаны қарастырамыз, бұл бағдарлама орындалу үшін командалық жолда былай жазылу керек:

> copyfile.exe 1-файлдың аты 2-файлдың аты

Бағдарламаның мәтіні:

```
#include<stdio.h>
 main(int argc, char *argv[ ])
{
  char cc[256];
   FILE *f1,*f2;
   if (argc!=3)
     {
   printf(“\n Прогамманы орындау формасы:”);
       printf(“\n copyfile.exe 1-файл.аты        2-файл. аты”);
  return 1;
    }
 if ((f1=fopen(argv[1],”r”))==NULL)
  {
    perror(argv[1]);
    return 1;
}
if ((f2=fopen(argv[2],”w”))==NULL)
{
   perror(argv[2]);
```

```
    return 1;
}
while(fgets(cc, 256, f1)!=NULL)
    fputs(cc,f2);
    fclose(f1);
    fclose(f2);
    return 0;
}
```

Бағдарлама орындалғанда егерде командалық жолда екі аргумент болмаса, яғни 1-ші және 2-ші файлдың аттары, онда argc аргументтің мәні 3-ке тең болмайды да бағдарлама (бағдарламаны орындау форматы: copyfile.exe 1-ф.аты2-ф.аты) деген хабарландыру шығарады.

Форматталған режимде файлдармен мәлімет алмасу. Форматталған мәліметтерді файлдан экранға шығару үшін және файлдарға форматталған мәліметтерді жазу үшін **fprintf()** және **fscanf()** функциялары қолданылады. Бұл функциялар stdio.h тақырыптық файылында төмендегідей анықталған:

int fprintf(легке_сілтемелетін_көрсеткіш, форматталған_жол,
айнымалылар_тізімі);
int scanf(легке_сілтемелетін_көрсеткіш, форматталған_жол,
айнымалылардың_адрестерінің_тізімі);

Бұл функциялардының **printf()** және **scanf()** функцияларынан айырмашылығы, **fprintf()** және **fscanf()** функцияларының бірінші параметрі ретінде мәлімет алмасатын легке сілтемеленетін көрсеткіш қолданылады. Мысал ретінде 1-ден 10-ға дейінгі сандарды және олардың квадраттарын int.dat файылына жазатын бағдарламаны қарастырамыз:

```
#include<stdio.h>
 int main()
{
  FILE *fp;
  int n;
     if ((fp=fopen("int.dat","w"))==NULL)
  {
    perror("int.dat");
    return 1;
       }
     for (n=1; n<11; n++)
   fprintf(fp, "%d  %d\n", n, n*n);
     fclose(fp);
     return 0;
  }
```

Бұл бағдарлама орындалғанда 1-ден 10-ға дейінгі сандар және олардың квадраттары int.dat файлына жазылады. Енді осы жазылған мәліметтерді int.dat файлынан **fscanf()** функциясының көмегімен оқып, экранға шығарамыз:

```
#include<stdio.h>
 int main()
{
  FILE *fp;
  int n, nn, i;
  if ((fp=fopen("int.dat","r"))==NULL)
    {
  perror("int.dat");
      return 1;
    }
   for (i=1; i<11; i++)
    {
  fscanf(fp, "%d %d, &n, &nn);
  fprintf(" %d  %d \n", n, nn);
    }
          fclose(fp);
  return 0;
}
```

Легтердегі позициялау. Файлдармен ақпарат алмасу үшін байттардың легтері түрінде ұйымдастырылған символдық, жолдық және форматталған мәліметтер қарастырылды. Бірақ бұл құрылғылардың барлығы файлға мәліметтерді жазуды және файлдардан ақпаратты оқуды тек кезекпен ғана орындауға мүмкіндік берді.

Ақпаратты оқу немесе жазу әрқашанда легтің ағымдағы орнынан (позициясынан) басталды. Легті оқығандағы немесе жазғандағы алғашқы позиция осы легті ашқан кездегі ашу режимдеріне байланысты немесе соңғы байттарына сәйкес болады. Лег "r" немесе "w" режимдерінің көмегімен ашылса, ағымдағы позицияның көрсеткіші легтің алғашқы байтына орналасады. Егерде "a" режимінің көмегімен ашылса онда файлдың соңына орналасады.

Әрбір енгізу - шығару операциялары орындалғанда ағымдағы позициясының көрсеткіші оқылған немесе жазылған байттардың соңына сәйкес келесі жаңа позицияға жылжиды. Алғашқы позицияның көрсеткішін, легтегі қажет байтқа жылжытатын позициялау құрылғыларын қарастырамыз. Си тілінің библиотекасында легтің ағымдағы позициясының көрсеткішін легтегі қажет байтқа жылжытатын **fseek()** функциясы анықталған.
Бұл функцияның анықталуы:

```
int fseek(легке_сілтенетін_көрсеткіш, жылжу_арақашықтығы,
  жылжудың_бастапқы_позициясы);
```

Жылжу арақаштығы - айнымалымен немесе **long** типіндегі өрнекпен беріледі және теріс сан болуы да мүмкін, яғни файлдың ішінде алға және кері қарай жылжуға болады, жылжудың бастапқы позициясы **stdio.h** тақырыптық файлында орналасқан, алдын-ала анықталған келесі тұрақтылардың бірімен беріледі:

SEEK_SET(0 мәнін қабылдайды) - файлдың басы;
SEEK_CUR(1 мәнін қабылдайды)- ағымдағы позиция;
SEEK_END(2 мәнін қабылдайды)- файлдын соңы;

long типіндегі мәліметтердің бірқатар ерекшеліктерін ескеру керек. Бұл тип int типіндегі мәліметтерге қарағанда жадыдан көп орын қажет ететін, бүтін санды тұрақтылар және айнымалылар үшін анықталған. Әдетте **long** типіндегі айнымалыларға 4 байт орын бөлінеді және осы байтпен оның мәндерінің арақашықтығы анықталады. **long** типіндегі мәліметтердің сипатталуы:

long A, p, z[16];

long типіндегі тұрақты ондық цифрлардың тізбегі ретінде жазылады және оның соңына **L** немесе **l** қосылады.
Мысалы:
0l 0L 10L 688L 33l

Егерде легте немесе файылда курсордың жылжуы орындалса, онда **fseek()** функциясы 0 мәнін қайтарады, орындалмаған жағдайда 0-ге тең емес мән қайтарады. **fseek()** функциясының қолданылу мысалдарын қарастырамыз. Кез - келген позициядан файлдың бірінші позициясына жылжу.

fseek(fp, 0L, SEEK_SET);

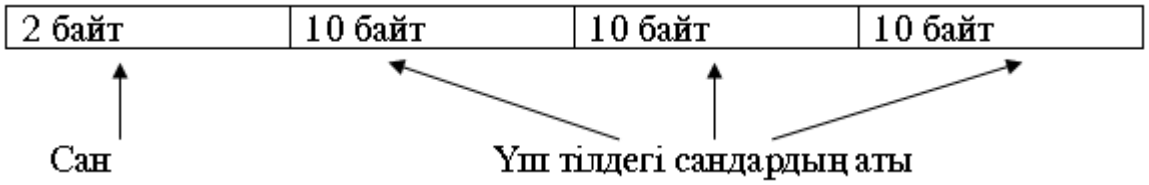
кез –келген позициядан файлдың соңына жылжу.

fseek(fp, 0L, SEEK_END);

Екі мысалдада жылжу арақашықтығы **long** типіндегі **0L** - нөлдік ондық тұрақтысы арқылы берілген. Осы айтылғандардың барлығын мысалда қарастыру үшін, 1-ден 9-ға дейін сандардың қазақ, орыс, ағылшын тілдеріндегі сөздігі жазылатын файлды қарастырамыз. Бұл сөздік үшін екі бағдарлама қажет: біріншісі сөздікті файлға енгізу, екіншісі сөздіктен қажет сөзді іздеу үшін сөздіктің мәліметтер қоры vac.dat файлында ұйымдастырылады. Енгізу функциясы 1-ден 9-ға дейінгі цифрлардың атын сұрайды. Цифрдың аты үш тілде де бос орын арқылы бір жолға жазылады. Бұл жол “Enter” пернесі басылғаннан кейін buf[] символдар массивіне оқылады бағдарламаның мәтіні.

```
#include<stdio.h>
int main ()
{
 int i,k;
 int n;
 char buf[30];
 char  *c;
 int *pti;
 FILE *fp;
 if ((fp=fopen(“vac.dat”,”w”))==NULL)
 {
   perror(“vac.dat”);
   return 1;
  }
 for (i=1; i<=9; i++);
 {
  for (k=0; k<30; k++);
    buf[k]=’ ’;
    n=i;
   pti=&n;
   printf(“\n Цифрдың атауын енгіз%d:\n”,i);
   scanf(“%s %s %s,buf, buf +10, buf +20”);
   c=(char *)pti;
  for (k=0; k<2; k++)
   putc(*c++, fp);
  c=buf;
 for (k=0; k<30; k++)
  putc(*c++, fp);
 }
fclose(fp);
return 0;
}
```

Енгізілген мәліметтер vac.dat файлына байт бойынша жазылады. vac.dat файлында құрылымы төмендегідей жазбалар пайда болады (7.1-сурет).



7.1-сурет. Жазбалардың байт түрінде орналасуы

Келесі бағдарлама алдымен санның қай тілдегі атауын шығару керектігін, одан кейін санның өзінің енгізілуін сұрайды. Тілді енгізу үшін латын әріптерінің бірінің қазақ тілінде “к” немесе “К” , орыс тілінде “r” немесе “R”

ағылшын тілінде “e” немесе “E” әрпі енгізіледі. Бағдарламаның жұмысын тоқтату үшін 0 санын енгізу керек:

```
#include<stdio.h>
int main()
{
  int i,t;
  char  *c;
  long pos;
  char buf[30];
   char ln;
   FILE *fp;
   int lang;
   if ((fp=fopen(“vac.dat”,”r”))==NULL)
   {
     perror(“vac.dat”);
     return 1;
   }
  lang=-1;
  while(lang==-1)
  {
   puts(“\n Қай тілде шығарасыз(k,r,e):”);
   scanf(“%c”,&ln);
   if((ln==”k”) | | (ln==”K”)) lang=0;
    else
      if((ln==”r”) | | (ln==”R”)) lang=1;
       else
         if((ln==”e”) | | (ln==”E”)) lang=2;
   }
  while(1)
 {
  c=buf;
  puts(“цифр енгіз(0-бағдарламаның соңы):”);
  scanf(“%d”,&t);
  if (t==0)
 {
  fclose(fp);
  return 0;
 }
pos=(t-1)*32+2+lang*10;
fseek(fp, pos, SEEK_SET);
for(i=0; i<=9; i++)
  *c++=getc(fp);
c++;
```

```
*c='\0';
printf("%d->%s\n", t, buf);
}
fclose(fp);
return 0;
}
```

Цифрдың файлдағы енгізілген тілге байланысты аударма жолының орнын (позициясын) анықтау үшін келесі айнымалылар және өрнектер қолданылған:
t-1 - ұзындығы 32 байтқа тең санның аудармасы бар жолды таңдайды;
2-цифр алаңының ұзындығы;
long*10-аударма жолындағы қажет цифр мәніне дейінгі арақашықтықты;

Қарастырылған **fseek()** функциясынан басқа Си тілінің библиотекасында легтің ағымдағы позициясының көрсеткішімен жұмыс жасау үшін келесі функциялар анықталған.

long ftell(FILE*)-легтің ағымдағы позицияның көрсеткішінің мәнін анықтау үшін;

void rewind(FILE*)-легтегі позицияның көрсеткішін легтің алғашқы позициясына қояды.

7.5 Төменгі деңгейдегі енгізу – шығару

Файлдармен жұмыс жасағанда енгізу-шығару әдетте енгізу-шығарудың стандартты операцияларын орындау үшін қолданылады. Мұндағы легтермен ақпарат алмасу үшін қолданылған функциялар бағдарламаның әртүрлі операциялық жүйеде дұрыс жұмыс жасауын қамтамасыз етеді.

Төменгі деңгейлі енгізу-шығару функциялары операциялық жүйенің енгізу - шығару құралдарын тікелей қолдануға мүмкіндік береді. Мұндай жағдайда буферлену және мәліметтерді форматтау орындалмайды.

Файлды төменгі деңгейде ашқанда, онымен файлдың көрсеткіші емес, оның дескрипторы байланысады (handle). Дескриптор бүтін мәнге ие, ол операциялық жүйенің ішкі кестелерінде файлдың ашылғаны туралы ақпаратты көрсетеді. Файлдың дескрипторы файлдармен келесі операцияларды орындағанда қолданылады. Си тілінің библиотекасында төменгі деңгейлі келесі негізгі енгізу – шығару функциялары анықталған:

- **open()/close()** - файлды ашу/жабу;
- **creat()** - файлды құру;
- **read()/write()** - мәліметтерді оқу және жазу;
- **sopen()** - файлды бір мезетте бірнеше процестермен жұмыс жасау үшін ашу;
- **eof()** - файлдың соңына жеткенін тексеру;
- **lseek()** - файлдың ағымдағы позициясын өзгерту;
- **tell()** - файлдың ағымдағы позициясының мәнін анықтау;

Төменгі деңгейлі функцияларды қолданғанда бағдарламаға осы функциялар анықталған тақырыптық файылдар қосылу керек. Бірақ бұл файлдар әр түрлі

операциялық жүйелерде әртүрлі болады. Сондықтан төменгі деңгейлі енгізу - шығару функциялары қолданылатын бағдарлама жазғанда немесе бағдарламаны басқа операциялық жүйеге ауыстырғанда осы операциялық жүйелердегі библиотекалық функцияларды қолдану керек.

Файлды ашу және жабу.

Төменгі деңгейде енгізу - шығару операцияларын қолданудан бұрын, алдымен файылды құру немесе ашу керек. Ол үшін келесі функциялардың бірі қолданылады: **open()**, **sopen()** немесе **creat()**.

sopen() функциясы бірнеше бағдарламаның бір мезетте бір файлмен жұмыс жасауына мүмкүндік беру үшін қолданылады. Мұндай жағдайда файл тек оқу үшін ашылады. Файылды ашқанда бағдарламаға мәні бүтінсанды болатын файлдың дескрипторы қайтарылады.

open() функциясының қолданылу форматы :

fd=**open**(файлдың_аты, флагтар, қолдану_құқығы);

Бағдарламада fd файлының дескрипторы int типіндегі анықталуы керек.

Файлдың_аты - символдар массивіне сілтенетін көрсеткіш, екінші параметр. Флагтар - файлды ашу режимдерін анықтайды. Ол **fcntl.h** тақырыптық файлында анықталған бір немесе бірнеше тұрақтылардан құралатын өрнек болып табылады. Файлды ашу режимдерін анықтайтын тұрақтылар:

O_APPEND – файлға ақпарат қосу үшін ашу;
O_BINARY – бинарлық екілік режимде файлды ашу;
O_CREAT – жаңа файл құру және ашу;
O_EXCL – егерде O_CREAT флагымен бірге қолданылса дискіде файл бар болса, онда қате шығарады. Бұл флаг дискіде бар файлды білместен жойып жіберуден сақтайды;
O_RDONLY – файлды тек оқу үшін ашу;
O_RDWR – файлды оқу және жазу үшін ашу;
O_TEXT – файлды тексттік режимде ашу;
O_TRUNC – дискіде бар файлды ашу және оның мазмұнын жойып басқа информация жазуға дайындау;

Үшінші параметр - қолдану құқығы тек қана файлды **O_CREAT** тұрақтысымен ашқанда қолданылады, яғни жаңа файлды құрғанда. MS DOS және Windows операциялық жүйелерінде қолдану құқығы параметрін қолданғанда келесі алдын-ала анықталған тұрақтылар пайдаланылады:

S_IWRITE – файлға жазуға рұқсат беру;
S_IREAD – файлға оқуға рұқсат беру;
S_ IREAD|S_IWRITE – файлға жазуға және оқуға рұқсат беру;

Бұл тұрақтылар **sys** каталогында орналасқан **stat.h** тақырыптық файлда анықталған. Әдетте бағдарламада оның қолдануында келесі директива қолданылады:

#include<sys\stat.h>

Егерде қолдану құқығы параметрі көрсетілмесе, онда файлды тек оқуға рұқсат беріледі.

Бірнеше мысалдар келтіреміз:

1. fd=open(“t.txt”, O_RDONLY); - файлды тек оқу үшін ашу;
2. fd=open(“new.txt”, O_WRONLY|O_CREAT|O_TRUNC,0600); - дискіде бар файлға жаңа мәлімет жазу үшін;
3. fd=open(“t.txt”,O_WRONLY|O_APENNED|O_CREAT|,0600); - дискіде бар файлға мәлімет қосу үшін;
4. fd=open(“t.txt”, O_ROWR) - файлды оқу және жазу үшін ашу;
5. if ((fd=open(“tmpfile”, O_WRONLY|O_CREAT|O_EXCL,0666)==-1) puts(“tmpfile дискіде бар \n”); - мәлімет жазу үшін жаңа файл құру;

Файлды құру үшін келесі бағдарламаны қарастырамыз:

```
#include<stdio.h>
#include<stdlib.h>
#include<io.h>
#include<errno.h>
#include<fcntl.h>
#include<sys\stat.h>
void main()
{
 int fd;
 if((fd=open(“tmpfile”,
    O_ROWR|O_CREAT|O_EXCL,S_IREAD|S_IWRITE))<0)
  {
   if (errno==EEXIST)
     fprintf(stderr,”файл tempfile дискіде бар\n”);
   exit(errno);
  }
}
```

Құрылатын файл қолданылған флагтарға сәйкес тек оқу және жазу үшін ашылады, файлды ашқанда пайда болатын қателерді анықтау үшін, **errno.h** тақырыптық файылында анықталған **errno** айнымалысы қолданылады. Стандартты библиотеканың функциясы орындалғанда облысына қателердің кодтары жазылады **errno.h** тақырыптық файлында алдын-ала анықталған **EEXIST** тұрақтысы **open()** функциясында көрсетілген файлдың дискіде бар екенін анықтайды.

Бұл мысалда бағдарлама орындалғанда пайда болатын қателерді файлға жазу үшін **fprintf()** функциясы қолданылған. Функцияда осы қателер туралы хабарландыруларды шығару үшін, алдын-ала анықталған файлдың **stderr** дескрипторы қолданылған. **open()** функциясынан басқа файлды ашу үшін **creat()** функциясы қолданылады. **creat()** функциясы жаңа файл құрып, оны ақпарат жазу үшін ашады.

Легтерді қолданғандағыдай әрбір бағдарлама орындала бастағанда автоматты түрде стандартты енгізудің, стандартты шығарудың және қателер туралы хабарландыруларды стандартты шығарудың файлдары ашылады. Бұл файлдың дескрипторларының мәні 0,1 және 2-ге тең. Оларды төменгі деңгейде стандартты файлдармен ақпарат алмасу үшін қолданылады. Әрбір операциялық жүйеде бір мезгілде бағдарламада ашуға болатын файлдардың соңына шектеулер қойылған. Әдетте олардың саны 20-дан 40-қа дейін барады. Көп файлдар өңделетін бағдарламаны қолданғанда қажет емес файлдарды уақытында жауып отыру керек. Төменгі деңгейде файлды жабу үшін **close()** функциясы қолданылады. Функцияның сипатталуы:

int close (файлдың_дескрипторы);

close() функциясы дұрыс орындалғанда 0 мәнін қайтарады. Қате болғанда –1 мәнін қайтарады. Бағдарлама жұмысын аяқтағанда барлық ашылған файлдар автоматты түрде жазылады.

Мәліметтерді оқу және жазу

Төменгі деңгейде мәліметтерді енгізу-шығару үшін **read()** және **write()** функциялары қолданылады.
Бұл функциялардың сипатталуы:

int read(**int** fd, **char** *buffer, **unsigned int** count);
int write(**int** fd, **char** *buffer, **unsigned int** count);

Функцияда оқылған және жазылған байттардың санына тең бүтінсанды мән қайтарады. **read()** функциясы buffer көрсеткішімен анықталған, буферге fd дескрипторының көмегімен ашылған файлдан үшінші count параметрімен анықталатын байттар санын оқиды. Файлдың соңына жеткенде **read()** функциясы 0 мәнін қайтарады. Файлдан мәлімет оқығанда қате пайда болса **read()** функциясы –1 мәнін қайтарады. Ақпарат оқу операциясы легтердегі енгізу-шығарудағыдай файлдың ағымдағы позициясынан басталады. Оқу операциясы аяқталғаннан кейін ағымдағы позициясы бірінші оқылмаған символдан басталады.

write() функциясы buffer көрсеткішімен анықталған буферден fd дескрипторының көмегімен ашылған файлға үшінші count параметрімен саны анықталған байттар тізбегін жазады. Мәліметтерді жазу ағымдағы позициядан басталады. Егерде операция орындалғанда қателер туындаса онда **write()** функциясы –1 мәнін қайтарады. **errno** негізгі айнымалысы **errno.h** тақырыптық

файлында алдын-ала анықталған келесі тұрақтылармен анықталатын мәндердің бірін қабылдайды:

EACCES - файл жазудан қорғалған (яғни тек оқуға болады);
ENOSPC - сыртқы құрылғыда бос аумақ жоқ;
EBADF - файлдың дескрипторы қолданылмайды;

Келесі мысалда символдар тізбегін стандартты енгізу құрылғысынан стандартты шығару құрылғысына көшіреді:

```
#include<io.h>
int main()
{
  char c[2];
  while ((read(0, c, 1))>0)
  write(1, c, 1);
  return 0;
}
```

BUFSIZE тұрақтысы (яғни легтік енгізу-шығару үшін бөлінетін буфердің көлемі) io.h тақырыптық файлында анықталған. Оның мәні MS-DOS операциялық жүйесі үшін 512 байтқа тең. Бұл бағдарламаның атын copyf деп сақтаймыз. Бұл бағдарламаны командалық жолда орындау үшін былай жазылу керек:

```
copy f1.dat  f2.dat
```

Егерде бағдарлама орындалғанда ешқандай қате болмаса онда файл мазмұны басқа файлға көшіріліп, дисплейдің экранына ешқандай хабарландыру шықпайды.

Бақылау сұрақтары

1. Енгізу-шығарудың негізгі неше деңгейі бар.
2. Легтерді енгізу-шығару.
3. Легтерді ашу және жабу.
4. Легпен байланысқан файлды стандартты ашу режимі.
5. Символдарды енгізу-шығару функциялары.
6. Дискідегі файлдармен жұмыс жасау функциялары.
7. Файлдармен жұмыс жасағанда жолдармен алмасу.
8. Форматталған режимде файлдармен мәлімет алмасу.
9. Легтерді позициялау.
10. Төменгі деңгейдегі енгізу-шығару функциялары.

VIII БӨЛІМ. ҚҰРЫЛЫМДЫҚ ТИПТЕР ЖӘНЕ ҚҰРЫЛЫМДАР. ТУЫНДЫ ТИПТЕР

8.1 Құрылымдық және туынды типтер түсінігі

Си тілінің базалық типтерінен туынды типтер қалыптастыруға болады. Көрсеткіштер, массивтер, функциялар, құрылымдар және бірлестіктер мәліметтердің туынды типтері болып есептеледі.

Құрылымдық тип-оның көмегімен анықталатын құрылымдардың ішкі құрылымын сипаттайды.

Құрылым - бұл тұтас бір типке біріктірілген аты бар мәліметтер элементтерінің жиыны.

Бір типті элементтерден тұратын массивтен құрылымның бөліктерінің типтері және аттары әр түрлі бола береді. Мысалы қоймадағы заттарды сипаттайтын, төмендегі компютерден тұратын құрылымға енгізуге болады :

заттың аты (char*);
заттың бағасы(long);
сату үшін қосылатын баға (float);
заттың көлемі (int);
зат алып келінген күн (char[9]);

Мағынасына сәйкес компоненттер Си тілінде анықталған кез-келген мәліметтердің типі болуы мүмкін.

Қоймадағы заттар көп болғандықтан, осы заттар туралы мәліметтерді сақтайтын құрылымдарды сипаттау үшін туынды тип енгізіледі және бұл типті - құрылымдық тип деп атайды.

Біз қарастырып отырған мысал үшін оны былай енгізуге болады:

```
Struct tovary
{
  char name;        /*зат аты*/
  long price;       /*бағасы*/
  float percent;    /*қосымша баға*/
  int vol;          /*заттың көлемі*/
  char date[9];   /*заттың келген күні*/
};
```

Бұл жердегі **struct** - құрылымдық типтің спецификаторы (қызметші сөз), **tovary** - құрылымдық типтің аты.

Фигуралы жақшаларда осы **tovary** типіндегі әрбір объектілерге енгізілетін элементтердің сипаттамалары орналасады. Құрылымдық типті сипаттау форматы төмендегідей:

struct құрылымдық_типтің_атауы {элементтердің_анықталуы};

struct – құрылымдық типтің спецификаторы,

құрылымдық_типтің_аты кез-келген идентификатор.

Элементтердің_аты - әрқайсысы енгізілетін құрылымдық типтің құрылымдық элементтеріне сәйкес келетін бір немесе бірнеше объектілерінің сипаттамаларының жиынтығы.

Құрылымдық типтің сипатталуы нүкте үтірмен аяқталады.

struct құрылымдық_типтің_атауы құрылымы, кез-келген типтің спецификаторының қызметін атқарады.

Struct tovary құрылымының көмегімен кез-келген анықталған құрылымды немесе осындай типтегі құрылымға сілтенетін көрсеткішті сипаттауға болады.

```
struct tovary food;
/*food атауымен құрылымның анықталуы*/
Struct tovary *point_to;
/* құрылымға point_to көрсеткішінің анықталуы*/
```

Құрылымдық типті typedef қызметші сөзінің көмегімен де енгізуге болады.

typedef struct{Элементтердің анықталуы}

Құрылымдық типтің аталуы.

Мысалы:

```
typedef struct
{
double real;
double imag;
}
complex;
```

Қарастырылып отырған анықтама құрылымдық тип енгізіп, оған complex деген атау меншіктейді. Осы атаудың көмегімен құрылымдарды енгізуге болады.

```
complex sigma, alfa;
/*екі құрылым анықталған */
```

#define директивасының көмегімен типтердің атын енгізуге болады. Мысалы кітаптар туралы мәліметтерді төмендегідей енгізуге болады.

```
#define BOOK struct\
{ char author[20]; /*Автор*/\
  char title[80]; /*Атауы*/\
  int year; /*шығарылған жылы*/\
}
```

Бұл мысалдағы BOOK-препроцессорлік идентификатор, одан кейін бірнеше жолға ауыстыру жолы орналасқан. Жолдық тұрақтыны келесі қатарға өткізу үшін “\” жалғастыру символы қолданылған.

Бағдарламада препроцессорлік деңгейде енгізілген BOOK атының көмегімен құрылымдық объектілерді немесе көрсеткіштерді сипаттауға болады.

BOOK ss,*pi;

Препроцессорлық ауыстырулар орындалғаннан кейін және түсіндірмелер алынып тасталынғаннан кейін жоғарыдағы мысал былай орындалады:

```
struct
{
  char  author[20];
  char  title [80];
 int  year;
}
 ss,*pi;
```

8.2 Құрылымдарды сипаттау

Құрылымдық типтің элементтерін сипаттау сәйкес типтердегі мәліметтерді сипаттауға ұқсас. Бірақ олардың арасында айырмашылық бар. Өйткені құрылымдық типті анықтағанда оның құрылымдарына (компонентеріне) жады бөлінбейді және оларды инициализициялауға болмайды. Басқаша айтқанда құрылымдық тип объект болып есептелмейді.

Жоғарыда қарастырылған мысалда **tovary** құрылымдық типінде, заттың аты, **name** деген аты бар **char*** типіндегі көрсеткішпен байланысты. Заттың бағасының мәні болып **price** деген аты бар **long** типіндегі элементтің мәні есептеледі және т.б. Бірақ құрылымдық типтің анықтамасында енгізілген элементтер мән қабылдау үшін, алдымен осы типтегі ең болмағанда бір құрылым сипатталуы керек. Мысалы келесі сипаттамда екі құрылым анықталған:

Struct tovary coat, tea;

Сонымен, егер де құрылымдық тип анықталған болса және оның аты білгілі болса, онда құрылымдарды сипаттау форматы төмендегідей болады:

Struct құрылымдық_типтің_аты құрылымдар _тізімі

Бұл жердегі құрылымдар-тізімі кез-келген идентификаторлар тізбегі. Бұдан басқа Си тілінде құрылымдарды сипаттау үшін екі сызба енгізілген. Біріншіден құрылымдарды бірден осы құрылымдық типті анықтағанда сипаттауға болады:

Struct құрылымдық_типтің_аты.
{элементтердің анықталуы}
құрылымдардың_тізімі;

Мысалы:

```
struct student
{char name [15];     /*аты*/
char surname [20];  /*тегі*/
int year;              /*курс*/
} student_1, student_2, student_3;
```

Бұл мысалыда student құрылымдық типінің объектісі болып есептелетін үш құрылым анықталған. Бұл үш құрылымның әр қайсысына осы құрылымдық типте анықталған барлық элементтер кіреді. Бұл құрылымдық типтің көмегімен бұдан басқа да бірнеше құрылымдар сипаттауға болады.
Мысалы:

```
struct student otl, sred;
```

Құрылымдарды сипаттауды келесі түрінде құрылымның атауы енгізілмейді. Мұндай сипаттау құрылымдарды сипаттаудың қысқа түрі болып есеп теледі және тек бір мақсатты есептерді шешу үшін қолданылады:

```
struct
{элементтердің анықталуы}
құрылымдардың_тізімі;
```

Мысалы ретінде дербес компьютердің құрылымдарын сипаттайтын құрылым қарастырамыз.

```
- процессордың типі (char [10]);
жұмыс істеу жиілігі Мгц (int);
негізгі жадының көлемі Мб (int);
қатты дененің көлемі Гб (int);
Мұндай типтегі құрылымды сипаттау

Struct {
char processor [10];
  int frequency;
  int memory;
  int disk;
} IBM_486, PENT IV, Compaq;
```

Мысалыда үш құрылым анықталады. Әр бір құрылымда дербес ЭЕМ құрылымдарын сипаттауға қолданылатын элементтер енгізілген.

8.3 Құрылымдарға жады бөлу

Құрылымдық типтерді сипаттған кезде оларға жады бөлінбейді. Жады құрылымдарды анықтаған кезді оның барлық элементтерінің мәліметтері сиятындай етіп бөлінеді.

Tovary құрылымдық типінің сипатталуы.

Struct tovary {char*name; long price; float percent; int vol; char date[9] };

Бұл суретте tovary құрылымдық типімен анықталған объектіге яғни құрылымдық бөлінген жадының сызбасы көрсетілген. Бірінші суретте құрылымның барлық элементтері бірінен кейін бірі қатар орналасқан және элементтерінің арасында ешқандай бос аралық жоқ (8.1-сурет).

Элементтің аты	Name	Price	Percent	Vol	Date
Типтері	Char*	Long	Float	Int	Char[9]
Байт	← 4 →	← 4 →	← 4 →	← 2 →	← 9 →

8.1-сурет. Элементтері бірінен кейін бірі қатар орналасқан жағдайы

Бірақ құрылымның элементтернің былай орналасуына Си тілінің стандартты ешқандай кепіл бере алмайды. Өйкені жадының адрестік кеңістігінің бөліктерінің шекаралары бойынша мәліметтерді реттегенде олардың арасында қолданылмаған бос аралықтар пайда болады.

Мұндай талаптар жүйелік аппараттың мүмкіндіктеріне компилятордың жұмысының режимдеріне байланысты болады.

Төмендегі суретте (8.2-сурет) құрылымның float percent; int vol элементтерінің арасында бос аралық бар.

Элементтің аты	Name	Price	Percent	Бос орын	Vol	Date
Типтері	Char*	Long	Float		Int	Char[9]
Байт	← 4 →	← 4 →	← 4 →		← 2 →	← 9 →

8.2-сурет. Элементтері бос орын арқылы бірінен кейін бірі қатар орналасқан жағдайы

Мұндай бос аралықтар құрылымның ең соңғы элементінен кейінде болуы мүмкін. Элементтердің арасындағы бос аралықтардың санына байланысты құрылымға бөлінетін жадының жалпы көлемі өзгеруі мүмкін. Құрылымға бөлінетін жадының көлемін келесі операциялардың көмегімен анықтауға болады:

Sizeof(құрылымның_ аты)
Sizeof(құрылымдық_типтің_аты)

Жоғарыда қарастырылған мысалда құрылымға бөлінетін жадының көлемін анықтау үшін былай жазылады.

Sizeof (struct tovary)
Sizeof (tea)

8.4 Құрылымдарды инициализациялау және меншіктеу

Құрылымдарды инициализациялау массивтерді инициализациялауға ұқсайды. Құрылымдар анықтамасында құрылымның_атынан және = белгісінен кейін фигуралы жақшалардың ішіне элементтерінің алғашқы мәндерінің тізімі орналасады.

Struct tovary coat={“Күртеше”,7.5,220, “12.10.2011”};

Құрылымдарды массивтермен салыстыра отырып меншіктеу операциясымен байланысты құрылымдардың бір ерекшелігін ескеру керек. Индекссіз массив элементтерін меншіктеу операциясында қолдануға болмайды, ал құрылымдарды бір-біріне меншіктеуге Си тілінің стандартында рұқсат бар

```
float z[5],x[5]={1.0,2.0,3.0,4.0,5.0};
z=x /*қате*/
```

Егерде жоғарыда қарастырылған tovary құрылымдық типімен мысалы coat және tea деген екі құрылым анықталған болса оларды бір біріне меншіктеуге болады:

```
Tea=Coat;
```

Құрылымдардың элементтерімен жұмыс жасау үшін төмендегідей құрылымдар қолданылады.

құрылымның_аты, элементтің _аты

Бұл жерде нүктенің алдында құрылымдық типтің аты емес, осы типпен анықталған құрылымның аты көрсетіледі. Біз қарастырып отырған мысалыда:

coat.name - char* типіндегі “ Күртеше” деген жолға сілтенетін көрсеткіш.
coat.price - 40000-ға тең long типіндегі айнымалы.
coat.percent - мәні 7,5-ке тең float типіндегі айнымалы.
coat.vol - мәні 220-ға тең int типіндегі айнымалы.
coat.date - char[9] типіндегі массив.

Құрылымның элементтеріне инициализацияның көмегімен мән енгізіп, оның мәндерін экранға шығаратын бағдарлама:

```
#include <stdio.h>
void main()
{ struct tovary
{
        char *name;
        long price;
        float percent;
        int vol;
        char date[9];
          };
```

```
struct tovary coat ={“ Күртеше”,40000,7.5,220,”12.01.97”};
printf(“\n Қоймадағы зат”);
printf(“\n Заттың аты:%s”, coat.name);
printf(“\n Бағасы:%ld”, coat.price);
printf(“\n Қосымша баға %4.1f”, coat.percent);
printf(“\n Зат көлемі :%d дана”, coat.vol);
printf(“\n Алынған уақыты:%s”, coat.date);
}
```

Құрылымдардың элементтері сәйкес типтердің объектілеріне берілген барлық құқықтарға ие. Оларды өрнектерге қолдануға және пернелік тақтадан енгізуге болады. Мысалы бағдарламаның келесі үзіндісін қарастырамыз:

```
printf(“\n Қосымша бағаны енгіз”);
scanf(“%f”,&coat.percent);
printf(“\n Зат бағасы:%ld теңге”, (long)
        (coat.price*(1.0+coat.percent/100)));
```

Бұл мысалда **scanf()** функциясының нақты параметр ретінде **coat** құрылымның **percent** элементінің адресі қолданылады.

Бақылау сұрақтары

1. Құрылымдық типтер түсінігі.
2. Туынды типтер түсінігі.
3. Құрылымдарды сипаттау.
4. Құрылымдарға жады бөлу жолдары.
5. Құрылымдарды инициализациялау.
6. Құрылымдарды меншіктеу.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

1. Шилд Г. Программирование на Borland C++ для профессионалов –Мн.: ООО «Попурри», 1998. – 800с.
2. Керниган Б., Ритчи Д. Язык программирования Си. /Пер. с англ. –М.: Финансы и статистика , 1992. -272с.
3. Болски М.И. Язык программирования Си. /Пер. с англ. Справочник. –М.: Радио и связь, 1988. -96с.
4. Керниган Б., Ритчи Д., Фьюер А. Язык программирования Си. Задачи по языку Си /Пер. с англ. –М.: Финансы и статистика , 1985. -279с.
5. Хэнкок Л., Кригер М. Введение в программирование на языке Си /Пер. с англ. –М.: Радио и связь, 1986. -192с.
6. Подбельский В.В., Фомин С.С. Программирование на языке Си. –М.: Финансы и статистика , 2000. -600с.
7. Березин Б.И., Березин С.Б. Начальный курс С и С++. –М.: Диалог – МИФИ, 1996.- 288с.
8. Культин Н. С/С++ в задачах и примерах.Санкт-Петербург. «БХВ-Петербург», 2005.
9. Болски М.И. Язык программирования Си Справочник.-М.: Радио и связь, 1988.
10. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. Задачи по программированию М.: Наука 1988.
11. Уинер Р. Язык Turbo Си: Пер. с англ. – М.: Мир, 1991. – 384с.
12. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. - М.: Мир, 1989. – 360с.
13. Зелковиц М., Шоу А, Гэннон Дж. Принципы разработки программного обеспечения: Пер. с англ. - М.: Мир, 1982. – 386с.
14. Фокс Дж. Программное обеспечение и его разработка: Пер. с англ. - М.: Мир, 1985. – 368с.
15. Язык компьютера / Под ред.и с предисл. В.М. Курочкина. Пер. с англ. - М.: Мир, 1989. – 240с.

МАЗМҰНЫ

КІРІСПЕ 3

І БӨЛІМ. АҚПАРАТ ТҮСІНІГІ МЕН АҚПАРАТТЫҢ ӨҢДЕЛУ ЖОЛДАРЫ 4

1.1 Ақпарат түсінігі мен қасиеттері 4

1.2 Ақпаратты өңдеу 4

Бақылау сұрақтары 8

ІІ БӨЛІМ. СИ БАҒДАРЛАМАЛАУ ТІЛІНІҢ НЕГЗГІ ТҮСІНІКТЕРІ 9

2.1 Си тілінің алфавиті, идентификаторлары, қызметші сөздері 9

2.2 Тұрақтылар және жолдар 10

2.3 Айнымалылар және ат берілген тұрақтылар 12

Бақылау сұрақтары 14

ІІІ БӨЛІМ. СИ ТІЛІНДЕ БАҒДАРЛАМАЛАУҒА КІРІСПЕ 15

3.1 Бағдарламаның мәтіні және препроцессор 15

3.2 Бағдарламалаудың қарапайым құралдары 20

3.3 Қайталану операторлары 24

3.4 Массивтер және қайталану операторларының сатылап орналасуы 28

3.5 Функциялар 30

3.6 Таңдау операторы switch 33

Бақылау сұрақтары 34

ІV БӨЛІМ. ПРЕПРОЦЕССОРЛІК ҚҰРЫЛҒЫ 35

4.1 Препроцессорды өңдеудің командалары және кезеңдері 35

4.2 Тақырыптық файлдарда анықталған мәтіндерді бағдарламаға қосу 39

4.3 Шартты компиляция. Тармақталу директивалары 41

4.4 Препроцессор құрылымдарын макростармен ауыстыру 42

4.5 Көмекші директивалар 44

Бақылау сұрақтары 45

V БӨЛІМ. КӨРСЕТКІШТЕР, МАССИВТЕР, ЖОЛДАР 45

5.1 Объектілердің көрсеткіштері 45

5.2 Көрсеткіштер мен массивтер 52

5.3 Символдық ақпарат және жолдар 61

Бақылау сұрақтары 68

VI БӨЛІМ. ФУНКЦИЯЛАР 69
……...
6.1 Функцияның параметрлерінде көрсеткіштерді қолдану …….. 69
6.2 Функцияның параметрлері ретінде массивтер мен жолдарды қолдану ……..…… 71
Бақылау сұрақтары 78

VII БӨЛІМ. СИ БАҒДАРЛАМАЛАУ ОРТАСЫНДА ЕНГІЗУ ЖӘНЕ ШЫҒАРУ ФУНКЦИЯЛАРЫН ҚОЛДАНУ …….. 79
7.1 Легтік енгізу-шығару ……... 79
7.2 Легтерді ашу және жабу ……. 79
7.3 Стандартты файлдар және олармен жұмыс жасауға арналған функциялар ……. 81
7.4 Дискідегі файлдармен жұмыс жасау …….. 85
7.5 Төменгі деңгейдегі енгізу – шығару ……... 94
Бақылау сұрақтары 98

VIII БӨЛІМ. ҚҰРЫЛЫМДЫҚ ТИПТЕР ЖӘНЕ ҚҰРЫЛЫМДАР. ТУЫНДЫ ТИПТЕР …… 99
8.1 Құрылымдық және туынды типтер түсінігі ……. 99
8.2 Құрылымдарды сипаттау ……. 101
8.3 Құрылымдарға жады бөлу ……... 102
8.4 Құрылымдарды инициализациялау және меншіктеу ……... 103
Бақылау сұрақтары 105

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ……..…… 106

Тұрымбетов Т.Ә.

СИ ТІЛІНДЕ БАҒДАРЛАМАЛАУ

Оқу құралы

Пішімі 60х84 1/12
Көлемі 111 бет 9,25 шартты баспа табағы

Таралымы 20 дана.
Ш.Есенов атындағы КМТжИУ
Редакциялық - баспа бөлімінде басылды.
Ақтау қаласы, 32 ш/а.