Б.С. Көбеков
А.М. Жұманазарова
Ж.Б. Қойшыбаева

С / С ++ ПРОГРАММАЛАУ ТІЛДЕРІНДЕГІ КӨРСЕТКІШТІК ТИП

Алматы 2009

ҚАЗАҚСТАН РЕСПУБЛИКАСЫ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

Қ.И. СӘТБАЕВ атындағы ҚАЗАҚ ҰЛТТЫҚ ТЕХНИКАЛЫҚ УНИВЕРСИТЕТІ

Б.С. Көбеков
А.М. Жұманазарова
Ж Б. Қойшыбаева

С / С ++ ПРОГРАММАЛАУ ТІЛДЕРІНДЕГІ КӨРСЕТКІШТІК ТИП

Университеттің Ғылыми-әдістемелік бірлестігі
оқу құралы ретінде ұсынған

Алматы 2009

ЖОК 004.4
ББК 32.973.018
К66

К 66 Б.С. Кебеков, А.М. Жұманазарова, Ж.Б. Қойшыбаева. С/С++ программалау тілдеріндегі көрсеткіштік тип: Оқу құралы – Алматы ҚазҰТУ, 2009. - 103б. Әдебиеттер тізімі – 7 атау.

ISBN 978-601-228-065-4

Оқу құралында көрсеткіштік типтің объектілерін анықтау және қолдануын бейнелейтін мәселелердің жүйелі түрде баяндалуы беріледі.

Бірінші бөлімде төмендегілер қарастырылады. статикалық және динамикалық объектілердің табиғаты, олардың программалау тәжірибесінде қолданылуы; көрсеткіштер түсінігі және оларға қолданылатын амалдар, массивтердің көрсеткіштермен байланысы, көрсеткіштер функция параметрі ретінде қалай қолданылады, функцияға көрсеткіш деген не және олардың түрлері. Бірінші бөлімнің соңында көрсеткіштер арқылы программалау мысалдары келтірілген. Екінші бөлімде көрсеткіштік тип және оның бір байланысты тізімдер, кезектер, екілік ағаштар сияқты күрделі құрылымдағы динамикалық объектілерді өңдеу үшін қолданылуы қарастырылады.

Оқу құралы 050704 – «Есептеу техникасы және жүйелердің программалық қамтамасы» және 050602 – «Ақпараттану» мамандықтарында оқитын бакалаврлар мен жоғары оқу орындарының оқытушыларына арналған.

ЖОК 004.4
ББК 32.973.018

Пікір жазғандар: *И.Т. Өтепбергенов* – техн. ғыл. докторы, профессор, ҚазКТА;
Б.Д. Хисаров – техн. ғыл. кандидаты, профессор, АЭжБИ;
Б.И. Жұмағалиев – техн. ғыл. кандидаты, профессор, ҚазҰТУ.

Қазақстан Республикасы Білім және ғылым министрлігінің 2009 жылғы жоспары бойынша басылды

ISBN 978-601-228-065-4

$\hat{E}\,\frac{2404090000}{00(05)-09}$

© Кебеков Б.С.,
Жұманазарова А.М.,
Қойшыбаева Ж.Б.
© ҚазҰТУ, 2009

Қ. И. Сәтбаев атындағы Қазақ ұлттық техникалық университетінің 75 жылдығына арналады

Кіріспе

«С/С++ программалау тіліндегі көрсеткіштік тип» оқу құралын жазуға бірнеше себептер әсер етті, солардың ішінде төмендегілерді атап өту керек:

– авторлар төменгі курстарында программалау тілдері мен техноло-гиясына байланысты әртүрлі пәндерден дәріс беруі. студенттердің динами-калық объектілерімен жұмыс істеудің тәсілдерін қиын меңгеретінін көрсетті;

– программалау бойынша ғылыми және оқу-әдістемелік сипаттағы белгілі баспалардың бүтін бір қатары «Көрсеткіштік тип» тақырыбы студенттерге қажетті түрде емес, ал жүйесіз және сәйкес әдістемелік нұсқауларсыз берілгенін көрсетеді;

– Паскаль тіліне қарағанда Си тілі де, С++ тілі де оқыту мақсатында емес, ал олардың авторлары арқылы кәсіби программистерге арнап құрылған, сондықтан Си тілін, әсіресе Си++ тілін оқыту, сондай-ақ үйрену оңай жұмыс емес;

– Осы тәрізді оқулықтар баспасының жоқтығы.

Ұсынылып отырған оқу құралында авторлар ең бірінші кезекте көрсеткіштік типтің шын болмысын түсіну негізін не құрайтынын және көрсеткіштік тип объектілерін қолдану арқылы мәліметтердің әртүрлі типтік құрылымдарын программалау механизмдерін көрсетуде студенттерге жәрдем беру мақсатын көздейді.

Авторлар оқулық мазмұнына оқу құралын пайдаланатын оқырмандар процедуралық программалаудың негізгі әдістерімен, дербес жағдайда құрылымдық программалау негізіндегі концепциялармен таныс деп ұйғарып қарайды. Оқу құралының мазмұнын табысты меңгеру үшін қажетті сілтемелер

болып объектілі-бағытталған программалаумен байланысты емес Си программалау тілінің негізгі құралдарымен ғанысу болып табылады.

«С/С++ программалау тіліндегі көрсеткіштік тип» оқу құралының мақсаты – көрсеткіштік типті қолданудың болмысы мен техникасын толығырақ түсінуге ұмтылған барлық қызығушылар мен оқытушыларға, студенттерге көмектесумен ғана шектелген емес, сондай-ақ салдар ретінде, программалауды оқыту үрдісінің өзін де неғұрлым қызықты ету және жетілдіру.

Оқу құралында программалаудың құрылымдық және объектілік парадигмаларында қолданылатын және дамытылатын көрсеткіштік типтің негізгі концепциялары мен механизмдері талқыланады және көрсетіледі

Көрсетілетін мысалдар негізінен жеткілікті қарапайым және программалаудың нақты есептеріне ұқсас. Дегенмен, мұндай мысалдардың бағалылығы сонша, оларды үйрене отырып, студенттер көлемді алғашқы мәтіндердің қысымында қалмай отырып-ақ көрсеткіштік типтің негізгі концепциялары мен механизмдерін түсіне алады.

1. КӨРСЕТКІШТІК ТИП ЖӘНЕ ПРОГРАММАЛАУ НЕГІЗДЕРІ

1.1. Статикалық және динамикалық программалық объектілер

Статикалық объектілер термині – программа орындалар алдында тікелей пайда болатын, оның барлық орындалу барысында орын алатын және мәндер өлшемі (оларды орналастыруға қажетті машиналық жад көлемі мағынасында) программа орындалуы барысында өзгермейтін объектілер. Статикалық объектілер оларға сәйкес баяндаулар арқылы пайда болады. Мысалы, **x** айнымалысының баяндалуы:

Float x[10]

мәні оң нақты сандардан тұратын (жадқа орналастырылуы мағынасында) тізбек болуы мүмкін жөндемді типтегі (массив) статикалық айнымалы программада қолдануға кіргізілетінін көрсетеді. Программада программалық объектілерге сілтеме олардың аталуын көрсету жолымен іске асырылады. Машиналық тілде объектіге сілтеме, оның компьютер жадындағы орнын көрсету жолымен іске асырылады, яғни орын әдетте, объектінің адресімен – осы объект орналасқан жад байтты ұяшығының нөмірімен беріледі. Осылайша осындай объектілерді компьютер жадына орналастыру және программада осы объектілерге сілтемелер қалыптастыру бойынша барлық жұмыстар трансляция кезеңінде орындалады, бұл статикалық объектілерді пайдаланудың ұтымды жағы болып табылады.

Дегенмен, программалауда тек статикалық объектілерді пайдалану белгілі бір қиындықтар туғызады. Әсіресе тиімді программа алу ғұрғысынан, бұл тек практикалық есептерді шешу ғана емес, сондай-ақ 20-ғасырдың аяғында кең қолданысқа енген және дамыған, ал 21-ғасырдың басында болашақта маңыздылығымен қалатын ең соңғы программалау технологиясын қолдануда да маңызды орын алады.

Өйткені кейде біз алдын ала программалық объекті мәнінің өлшемін, тіпті бұл объект бар бола ма, жоқ па екенін білмейміз.

Программаның орындалуы үрдісінде пайда болып, өңделетін осы тектес программалық объектілер динамикалық объектілер деп аталады

Қандай да бір программалық объект жадтың белгілі бір көлемін ала отырып, программа орындалуының барлық уақытына керек емес, сондықтан жоғарыда көрсетілген кемшіліктерді жою үшін заманына сай программалау тілдерінде динамикалық объектілерді қолданады. Шындығында мұндай объектілерді қолданудың өзіндік қиыншылықтары бар. Солардың бірі динамикалық объектілер программа орындалуы барысында пайда болады. Ал оларға қолданылатын әрекеттерді программа жазу барысында олардың орындалуына дейін беру керек. Бұл қиындықты жеңу үшін мәндердің арнаулы типі – *көрсеткіштік тип* қарастырылады. Мұндай типтегі айнымалы мәндерін көрсеткіштер деп атайды. Олар белгілі бір типтегі объектілердің адресі қызметін атқарады

1.2. Көрсеткіштер ұғымы

Көрсеткіштер – бұл C/C++ тілдерінің мүмкіндіктерін меңгерудің қиындықтарының бірі. Көрсеткіштердің болуы біл тілдерді жүйелік программалау үшін ыңғайлы етеді.

Көрсеткіштер көмегімен төмендегілер орындалады:

– символдар, массивтер, жолдар, құрылымдар, құрылымдар массивтері өңделеді;

– программаның орындалуы барысында жаңа айнымалыларды динамикалық құру;

– нақты параметрлердің адрестерін функцияларға беру;

– функция адрестерін функцияларға параметр ретінде беру.

Сонымен, көрсеткіш – бұл басқа айнымалының адресін қамтитын айнымалы немесе тұрақты.

Көрсеткіштердің мәндері – бұл белгісі жоқ бүтін сандар және оларды бүтін санды шамалар сияқты өңдеуге болады. Олар айнымалы қайда орналасқанын хабарлайды.

Анықтамасында **void** - мәні жоқ типі қолданылған көрсеткіштер кірмейді. Мұндай көрсеткіштер кез келген типтегі объектіге сілтенеді, бірақ оларға атын алып тастау, яғни '*' амалын қолдануға болмайды.

'*' амалының нәтижесі көрсеткіш адрестелген объект *pc өрнегі pc көрсеткіш сілтенетін **char** типіндегі айнымалыны белгілейді.

Мысалы:

*pk = *pk + *pi,

Мұнда жадтың бүтін мәндерін қосу (динамикалық объектілердің ішіндегісі) және адресі pk көрсеткішін қамтитын жад бөліктеріне қосындыны меншіктеу жүріп жатыр. Көрсеткіштердің мүмкіндіктері мен жетістіктері функциялармен, массивтермен, жолдармен, құрылымдармен жұмыс істегенде тиімдірек көрінеді.

1.2.1. Көрсеткіштердің баяндалуы және пайдаланылуы

int *pı баяндалуында әрі қарай **int** типіндегі айнымалыны адрестейтін көрсеткіш құрылады. Көрсеткіштер тек белгілі бір типтегі адресті ғана сақтауы мүмкін. Егер бір типтегі көрсеткішке басқа типтегі айнымалы адресін меншіктесе қате болады.

Мысалы: **int** *pi;
float x = 1.5;
pi = &x;

Мұнда **float** типіндегі айнымалы адресін көрсеткішке меншіктеу қате.

Көрсеткіштерді баяндағанда жұлдызша көрсеткіш атына тікелей қатысты, сондықтан бұл белгіні олардың әрқайсысының аттарының алдына қою талап етіледі. Мысалы:

int *pa, *pb, *pc;

Бұл мысалда барлық көрсеткіштер стектік жадқа орналасқан.

Программаның келесі үзіндісін қарастырамыз:

```
int ia = 15. ib = 37. ic:
int * pi;
pi = &ia; ic = *pi; ia = ib; ib = ic;
```

мұндағы ia және ib айнымалылары арасында мәндер алмасуы жүреді.

Енді келесі үзінді орындалғанда экранға қандай нәтиже шығатынын анықтаймыз:

```
char c1 = 's' , c2 = 't';
char *pc1, *pc2, *pca;
pc1 = &c1, pc2 = &c2,pca = pc1;
pc1 = pc2; pc2 = pca;
printf("%c%c\n", *pc1, *pc2),
```

Жауабы: ts.

Көрсеткішті инициалдау мысалында программа қарастырамыз.

```
#include<stdio.h>
#include<string.h>
int main()
 {
   char *ps = "Dog Rex"; // ps көрсеткішінде жолдың бірінші
                         // символының адресі сақталады
   int i;
 for (i = strlen(ps)-1; I >= 0; i--)
     printf("%c", ps[i]);
     printf("\n %s \n", ps);
     return 0;
 ]
```

Программаның орындалу нәтижесі.

```
xeR goD
Dog Rex.
```

1.2.2. '&' амалын қолдануға шектеу

'&' амалын бұрыс қолдану жағдайын қарастырамыз. Амперсанд - '&' амалы программалық объект адресін көрсетеді және осы адрес көрсеткіштің мәні болады.

```
int *pi=&48;  // байланысты болмағандықтан тұрақтыны
                  // жад ұяшығымен адрестеуге болмайды
int k=5;
 pi = &(k+15); // сандар адресін алуға болмайды
 register int i;
                                pi = &i;
// регистрлік айнымалы жадта емес,
// ал тікелей процессор регистрінде
// орналасуы тиіс болғандықтан болмайды.
```

Сонымен, көрсеткіш – бұл белгілі бір типтегі айнымалы адресін сақтайтын айнымалы немесе тұрақты. Адрестелетін айнымалы типі стандарт, саналымды, құрылымды, бірігу және **void** болуы мүмкін.

void типіне көрсеткіш кез келген типтегі мәнді адрестей алады. Көрсеткіштің өзі үшін жад өлшемі және сақталған адрес форматы компьютерден және жадтың моделінен тәуелді. **stdio.h** стандарт файлынан алынған **NULL** тұрақтысы адрестің нөлдік мәнінің көрсеткіштерін инициализациялауға арналған.

Көрсеткіш айнымалы үшін жүргізілген жад аймағының нөлдік байты адресін қамтиды. Адрестелген программалық айнымалының типі осы айнымалының мәні үшін қанша байт жүргізілегінін анықтайды. Мысалы, келесі:

```
int k;
int *pi = &k,
```

баяндауда k айнымалысы үшін жадтан 4 байттағы аймақ бөлінеді, мысалы 56534 адресінен, онда pi көрсеткішінің адресі осы бөліктің **бірінші байтының адресі** болады, яғни 56534.

Жалпы, көрсеткіштердің 3 түрін бөліп көрсетуге болады: программалық объектіге көрсеткіш, функцияға көрсеткіш және **void** көрсеткіш.

Функцияға көрсеткіш функцияны шақырған кезде басқару берілетін адресті қамтиды. Функцияға көрсеткіш функцияны жанама шақыру үшін, сондай-ақ функция атын басқа функцияға оның параметрі ретінде беру үшін пайдаланылады.

Мысалы:

int (*pf) (double, double);

декларациясы **int** қайтаратын және екі **double** типті параметрге ие **pf** атты функцияға көрсеткішті белгілейді.

Объектіге көрсеткішті баяндау мына түрге ие.

тип-аты *көрсеткіш идентификаторы;

мұндағы тип сілтемеден және биттік өрістен басқа кез келген болуы мүмкін.

Көрсеткішке көрсеткіш анықтау мүмкіндігі бар. Көп деңгейлі жанама адресация үшін көрсеткіштерді баяндау мысалын қарастырамыз.

```
int k = 123;            // k – айнымалы аты
int *pi = &k;           // pi – айнымалыға көрсеткіш
int **ppi = &pi;        // ppi – айнымалыға көрсеткішке көрсеткіш
int **pppi = &ppi;      // pppi – айнымалыға көрсеткішке
                        // көрсеткішке көрсеткіш
```

void – қа көрсеткішті адресін сақтау талап етілген объектінің белгілі бір типі анықталмаған жағдайда қолданған ыңғайлы. C++-те мұндай механизм полиморфизмді жүзеге асырады.

void – қа көрсеткішке кез келген типтегі көрсеткіш мәнін меншіктеуге болады.

void* көрсеткішінің типі тек адрестерді сақтауға ғана арналған.

Көрсеткіш тұрақты немесе айнымалы болуы, сондай-ақ тұрақтыға немесе айнымалыға сілтенуі мүмкін.

Баяндаудың бірнеше мысалдарын қарастырамыз:

```
int k,                  // бүтін айнымалы
const int ci = 1;       // бүтін тұрақты
int *pi;                // бүтін айнымалыға көрсеткіш
```

```
const int *pci;              // бүтін тұрақтыға көрсеткіш
int *const cp = &k:          // бүтін айнымалыға тұрақты-көрсеткіш
const int *const cpc = &ci; // бүтін тұрақтыға тұрақты-көрсеткіш
```

Көрсеткіш пен жұлдызша арасындағы **const** модификаторы көрсеткіштің өзіне тиісті және оның өзгеруіне тиым салады, ал жұлдызшаның сол жағындағы **const** ол сілтенетін тұрақты мән береді. Көрсеткішті инициализациялау үшін ***&*** - *адресті алу* амалы қолданылады.

Көрсеткіштерді екі баяндауын қарастырамыз және олардың арасындағы айырмашылықты табамыз:

```
const type *p1;          // type типіндегі тұрақтыға көрсеткіш
type *const p2=&k;    // type типіндегі k айнымалыға тұрақты
```

көрсеткіш p1 көрсеткішіне кез келген айнымалы сияқты type типіндегі мәліметтер қамтитын жадтың кез келген ұяшығының адресі меншіктелуі мүмкін, салдарында осы ұяшықтың ішіндегісі көрсеткіш арқылы өзгертіле алмайды.

```
*p1= (type)float_value;  // жадтың қорғалған ұяшығының
                         // мәнін өзгертуге болмайтын жағдай
```

p2 айнымалысы тұрақты көрсеткіш ретінде жарияланған Ол type типіндегі мәліметтері бар жад ұяшығы адресін қамтуы мүмкін, бірақ бұл адрес өзгертілмейді. Сол себептен тұрақты көрсеткішті инициализациялау міндетті түрде оның баяндалуымен бірге бір мезгілде болуы керек. Осындай көрсеткіш сілтенген ұяшық ішіндегісі еркін өзгертіле алады.

```
p2=&k;                  // қорғалған адресті өзгерту мүмкін емес
*p2=(type) float_value; // жад ұяшығы ішіндегісін мүмкін болатын өзгерту
```

Массив аты оның бірінші элементінің өзгермейтін адресін көрсетеді, яғни массив аты – бұл тұрақты - көрсеткіш

Көрсеткіштерді баяндауға тағы бірнеше мысалдар келтіреміз:

```
int *pi;      // int типіндегі мәліметтерге айнымалы-көрсеткіш
char *pc;     // char типіндегі мәліметтерге айнымалы-көрсеткіш
float *pf;    // float типіндегі мәліметтерге айнымалы-көрсегкіш
int arr[5];   // arr- бес мәнге ие массив аты
              // int типіндегі, массив аты – көрсеткіш –
```

```
                    // тұрақты
int *arr[10];       // arr-мәндері int типіндегі
                    // он көрсеткіштен тұратын массив;
int (*par)[10];     // int типтегі он элементтен тұратын
                    // массивке көрсеткіш;
float (*pf)()       // pf – float типіндегі мән қайтаратын
                    // параметрсіз функцияға
                    // көрсеткіш–айнымалының аты. pf– бұл
                    // функцияның машиналық кодына көрсеткіш
float *f();         // f- float типіндегі
                    // мәнге көрсеткіш қайтаратын
                    // параметрсіз функцияға
                    // тұрақты-көрсеткіштің аты
```

1.2.3. Көрсеткіштерді инициалдау туралы толығырақ мәліметтер

Көрсеткіш үйін деп аталатын компьютердің динамикалық жадымен жұмыс істегенде қолданылады. Бұл программа орындалуы барысында талаптарға сәйкес орын бөлуге болатын бос жад. Динамикалық жадтың динамикалық айнымалылар деп аталатын бөліп алынған бөліктеріне кіру мүмкіндігі тек көрсеткіштер арқылы болады. Динамикалық айнымалылардың өмір уақыты олардың құрылған мезетінен программа жұмысының соңына дейін немесе көрсеткіштерді нөлдеу арқылы жадтың анық босағанына дейін анықталады. Динамикалық жадпен жұмыс істеудің екі әдісі бар:

- **malloc(), free()** функциялары шоғырын пайдалану – Си тілі;
- **new** және **delete** операторларын қолдану – C++.

Көрсеткіштерді инициализациялаудың бірнеше әдістері бар

1. Бар объектінің адресі көрсеткішіне меншіктеу

а) адресті алу амалының көмегімен

```
int k = 11,     // айнымалыны баяндау және инициализациялау;
int *p = &k;  // баяндалған p көрсеткішке k адресі жазылады
```

ә) басқа инициалданған көрсеткіш көмегімен

```
int  *q = p;
```

б) өздері тұрақты-көрсеткіш түрінде түсіндірілетін массив немесе функция көмегімен

```
int arr[10];    // массив баяндалуы; arr – мәні массивтің бірінші
                // элементінің адресі болатын тұрақты-көрсеткіш
int *p = arr;   // көрсеткішті баяндау және оған массивтің басты
                // адресін, яғни индексі 0 элемент адресін меншіктеу
void f(int k) {...}          // функцияның анықталуы
void (*pf) (int);            // функцияға көрсеткішті баяндау
pf = f;                      // функция адресін көрсеткішке меншіктеу
```

2. Бос мәнді меншіктеу:

```
int  *p = NULL;
int  *q = 0;                 // ұсынылатын әдіс
```

3. Жад аймағы адресі көрсеткішін анық түрде меншіктеу:

```
char  * pc = (char*) 0x0065FDFO;
```

мұндағы 0x0065FDFO – он алтылық тұрақты, ал (**char***) «char-ға көрсеткіш» типіне анық түрлендіру

4. Динамикалық жадтан орын бөлу және бөлінген орын адресін көр-сеткішке меншіктеу.

а) ***malloc*** функциясы көмегімен (тақырыптық **malloc.h** файлында)

```
int  *p1 =  (int *) malloc (sizeof (int));
```

мұндағы **malloc()** – параметрі бөлінген жадтың санын байтпен беретін жад бөлу функциясы, ал (**int***) конструкциясы **void*** - көрсеткіш типін келтіру үшін қолданылады. ***malloc()*** функциясы **void*** типіндегі көрсеткішті қай-таратындықтан бізге **int*** типіндегі көрсеткіш керек, онда қайтарылған көрсеткіш типін керек типке анық келтіруге тура келеді. Егер жад бөлінбесе функция нөлді қайтарады.

ә) **new** операторы (C++) көмегімен:

```
int *p2 = new int;
```

p_2 көрсеткішінің өзінің жады компиляция кезеңінде стекте бөлінеді.

```
int  *p3 =  new  int (20),
```

Мұнда жоғарыдағы әрекеттерден басқа 20 мәнді динамикалық жад ини-циалдауы жүреді.

```
int  * p4 = new int [20];
```

Мұнда **int типті** 20 шамасында бөлінген жад (20 элементтен тұратын массив) және басты адресі p_4 көрсеткішіне жазылады, ол енді массив аты сияқты түсіндіріледі. Осы аты арқылы массивтің кез келген элементіне индекстеу амалы көмегімен қатынас жасауға болады, мысалы p_4 [18]

malloc() функциясы арқылы бөлінген жадтың босауы ***free*** функциясы көмегімен (көрсеткіш идентификаторы), ал ***new*** амалымен бөлінген жад – ***delete*** операторымен орындалуы керек. Айнымалы-көрсеткіш сақталады (ол статикалық және стекте сақталады) және қайта инициалдануы мүмкін.

Жоғарыда келтірілген динамикалық айнымалылар былайша жойылады:

free(p_1);
delete p_2;
delete p_3;
delete[] p_4;

free() функциясын қолданғаннан кейін көрсеткішке **NULL мәнін** меншіктеген жақсы, егер соңынан оны программада тексерсе, жадтың босағаны туралы белгі беріледі:

```
#include <malloc.h>
int *p = NULL;
p = (int *) malloc(sizeof(int)*20);
...
*p = 34;
...
free(p);
p = NULL;
```

Егер көрсеткіш көмегімен бөлінген жадтың ішкі аймақтарына қатынас жасайтын болсақ, онда мұны жад бөлінген көрсеткіш көмегімен емес, ал қосымша көрсеткіш көмегімен жасаған дұрыс:

```
p = (int*)malloc(20);
int *q = p;
q++;
...
free(p);
```

Егер жад массив үшін ***new*** **амалымен бөлінген болса, онда оны босату үшін *delete[]*** конструкциясын қолдану қажет. Массив өлшемі мұнда көрсетілмейді. Егер квадрат жақшаларды көрсетпесек, қате болмайды, бірақ массивтің бірінші элементі ғана белгіленеді, ал қалғандары одан әрі амалдарға мүмкін болмай қалады. Мұндай ұяшықтар қоқыс деп аталады.

'*' амалы көрсеткіш мәні оған алдын баяндалған айнымалының адресін меншіктеу **немесе *malloc()*** функциясы немесе ***new*** амалы көмегімен жад бөлу арқылы анықталғаннан кейін қолдану керек.

Массив немесе жол аты массивтің немесе жолдың басына тұрақты көрсеткіш болып табылады.

Мысал ретінде буферді ***new*** амалы көмегімен **char*** типіндегі p көрсеткішке тапсырылған 12 байтты динамикалық жадта типтік емес қолдануды көрсетеміз. Осы буфердің бірінші төрт байтында бүтін 5 мәнін орнатамыз, ал қалған 8 байтта **double** типіндегі 7.8 мәнін орнатамыз. Буфер қажет болмай қалғанда оны ***delete*** амалымен өшіру керек.

```
int n = 5;
double d = 7.8;
char *p = NULL;
p = new char[12];
char *q = p;
*(int*)q = n;     //'char-ға көрсеткіш' типін
                  //  'int-ке көрсеткіш' типіне келтіру
                  // q объектісіне 5 мәнін меншіктеу
q = q+4;          // көрсеткішті 4 байтқа жылжыту
                  // 7.8 санын орын ауыстыру үшін
*(double*)q = d;  // көрсеткіш типін келтіру
                  // char типін double типіне және
                  //q объектісіне d объектісінің мәнін меншіктеу
. . .
delete[]p;
```

new амалын бір типті массив үшін буфер бөлу үшін қолдану **неғұрлым типтік болып табылады:**

```
char *pc = NULL;
pc = new char[2];
for (int i = 0; i<2; i++) *(pc+i) = i;
   . . .
delete []pc;
```

Бірнеше мысалдар арқылы үйренген материалды бекітеміз. Келесі программаны орындағанда қандай нәтиже шығатынын анықтаңыздар.

```
#include <stdio.h>
int main()
{
char c ='s';
const char lit ='r';          // тұрақтыны баяндау
const char *pcc = &lit;       // тұрақтыға көрсеткіш
char *const cpc = &c;         // тұрақты көрсеткіш
 //lit = 'u';        - lit-const болғандықтан дұрыс емес
//cpc = &lit; - тұрақты көрсеткіш болғандықтан дұрыс емес
  pcc = &c;
//*pcc ='q';  -const-қа көрсеткіш болғандықтан дұрыс емес
 *cpc ='q';
 printf("%c%c%c\n",lit, *cpc, *pcc);
 return 0;
}
```

Жауабы: rqq

Келесі үзіндідегі барлық амалдарды түсіндіріңіздер:

```
                int *pi;
          int k = 5;
          *pi = k;       // дұрыс емес, неге?
           pi = &k;
          *pi = *pi*k;
          printf("%d", *pi);
```

Жауабы: 25

Көрсеткішке басқа көрсеткіш меншіктеуге болады, егер екеуі де бір типке сілтейтін болса. бір типке келтіру үшін типтерді анық түрлендіруді пайдалану керек. Мысалы:

```
char *pc;
long l;
(long*) pc = &l;
 double *pd;
 int k;
 pd = (double*)&k;
```

Барлық типтерге көрсеткіштер меншіктеуге болатын **void** типіне көрсеткіш бұған кірмейді. Мысалы:

```
void *pv; int k;
int *pi;
pı = &k,
pv = pı; немесе pv = &k;
```

Бірақ **void** – қа көрсеткіштің өзі алдын ала келтірілмей тұрып, басқа типті көрсеткішке меншіктеле алмайды. Мысалы:

```
pı = (int*)pv;
```

Келтіру амалына дейін **void**-қа көрсеткіш өзгере алмайды. өйткені ол мәлиметтердің белгілі бір типі үшін емес жад ұяшығын қамтиды және компиляторға қанша байт қорға берілгені белгісіз. Сондықтан:

```
pv++;                  // мүмкін емес
k =*pv;                // мүмкін емес
(int*) pv++;           // мүмкін
```

Көрсеткіштермен жұмыс істегенде оқу программасының мысалы:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
  int main(void)
{  char *pc = NULL; int *pi = NULL,
 double * pd = NULL;
 char c='A';
       int k=3;
      double d=3.14,
```

```
        pc =&c; pi =&k; pd =&d;
       *pc = 'F'; (*pi)++, *pd = 1.23;
    printf ("Pointer pc=%p\n", pc);
    printf ("Pointed by pc value is %d\n", *pc);
    printf ("Pointed by pc value is %c\n", *pc);
            getch();
            return 0,
    }
```

Экранға pc көрсеткіш арқылы көрсетілген мәнді, яғни литердің цифрлік коды және литердің өзін, сондай-ақ осы көрсеткіш адресін шығарамыз, бұл үшін ***printf()*** функциясында % p форматындағы спецификацияны қолданады. Көрсеткіштерді жазу стилі.

- Көрсеткіш идентификаторлары p әрпінен басталады.
- Көрсеткіштерді анықтаудың екі әдісі бар.

int* pi, немесе **int** *pi,

Жұлдызшаны, не идентификаторға, не берілген көрсеткіш сілтейтін негізгі типті анықтайтын кілттік сөзге қысу керек. Егер бірінші стиль қолданылса, онда екі немесе одан көп көрсеткіштерді анықтау үшін екі немесе одан көп анықтау амалын жазу керек.

```
int* pi1;
int* pi2;
```

Ал бір анықтау операторын емес:

```
int* pi1, pi2;
```

Өйткені екінші идентификатор pi2 **int**-ке көрсеткіш емес **int** типіндегі айнымалыны береді.

Екінші стильде бір жолда көрсеткіштердің бірнеше идентификаторларын жазуға болады:

```
int *pi1, *pi2,
```

Сұрақтар және жаттығулар

1. Статикалық және динамикалық программалық объектілердің қызметтері және ерекшеліктері.

2. Көрсеткіштік типтің анықтамасы. Оларға қолданылатын амалдар.
3. Көрсеткіштік типтің болуы жүйелік программалауда не үшін C/C++ тілдерін ыңғайлы етеді?
4. `&` және `*` амалдары қандай мүмкіндіктер береді? `&` амалын қолдануға шектеулер.
5. Көрсеткіштің объектілермен жұмыс істеуі қалайша анықталады?
6. Көрсеткіштің түрлері.
7. **void**-қа көрсеткішті қандай жағдайда қолданған ыңғайлы?
8. Массив (сандық, жол) аты нені көрсетеді?
9. Динамикалық жадпен жұмыс істеудің қандай әдістері бар?
10. Көрсеткіштерді инициалдаудың қандай әдістері бар?
11. Әртүрлі типтегі көрсеткіштердің өрнегінде араластырған кезде типтерді анық түрлендіру талап етіле ме?
12. 0 мәні кез келген типке көрсеткішке түрлене ма?
13. Мәндерді көрсеткіш-тұрақтыларға меншіктеуге бола ма?
14. Бір ғана жад аймағына әртүрлі типтегі көрсеткіштер сілтене ме?
15. *көрсеткіш-аты құрылымын меншіктеу амалының тек оң жағында ғана емес, ал сол жағында да пайдалануға мысал:

```
char lit,
  char_*pc = new char;
*pc = `a`;  lit = *pc;
```

16. Келесі екі өрнек неге әртүрлі нәтиже береді:

```
* pc ++ = getchar();
* ++ pc = getchar();
```

17. new амалы float, long және char типіндегі айнымалыларды динамикалық орналастыру үшін қолданылатын программа жазыңыздар. Оларға бастапқы мән беріңіздер және осы мәндерді экранға шығарыңыздар. Айнымалы құрамын кез келген орын ауыстырулар жасаңыздар. Соңында delete амалының көмегімен жадтың бөлінген аймағын босатыңыздар.
18. new және delete амалдарының malloc() және free() функциялары алдында қандай жетістіктері бар?

19. Келесі үзіндіні new амалы қолданатындай етіп, қайта жасаңыздар:

```
char *pc;
pc = (char*) malloc(100);
. . .
strcpy(pc, "Бұл тексеру");
```

20. Динамикалық массивті инициалдауға бола ма?

1.3. Көрсеткіштерге қолданылатын амалдар

C/C++ тілдерінде көрсеткіштерге қолданылатын негізгі амалдар: меншіктеу; көрсеткіш сілтенетін объектінің мәнін алу (амалдың синонимдері: жанама адрестеу, атын алып тастау, сілтемелерді ашу), көрсеткіштің өзінің адресін алу; көрсеткіштің мәндерін өзгертудің унарлы, аддитивті және салыстыру амалдары.

Көрсеткіш – бұл белгілі бір типтегі мән сақталатын адресті қамтитын стектегі айнымалы. Егер көрсеткішке 1 санын қосса, ол жадтың келесі ұяшығына сілтенеді. Егер көрсеткішке 5 қосса, онда ол ағымдық адрестен бес ұяшықта тұрған ұяшыққа сілтенеді.

Мысалы. Көрсеткіш мәні 1000-ға тең болсын, егер көрсеткішке 1 қосса, онда нәтиже ол байланысты объектінің типінен тәуелді. Егер объект типі **char** болса, онда қосу нәтижесі 1001-гe тең, бірақ егер **int** типіндегі көрсеткішке 1 қосса, онда мән 1004 болады (өйткені **int** типі жадтың төрт байтын талап етеді). **float** типі үшін де осындай (1004).

```
int arr [3] = { 1, 2, 3 };
```

массивінде **int** * типті p көрсеткіші массивтің 0 индексті элементіне адрестелген болсын:

```
p = & (arr [0]); не p = arr;
p = p + 1;
```

амалынан кейін p неге сілтенеді деген сұрақ туады.

Жауап мынадай, ол массивтің 1 индексті элементіне сілтенеді. **int** типі жадтың төрт байтын талап ететіндіктен

көрсеткіш мәні де, яғни оның адресі төртке өзгереді Көрсеткіштердің бұл қасиеті өте ыңғайлы

(pd)++ амалы pd көрсеткішінің (**double*** типіндегі) көрсеткіш мәнін 8-ге арттырады Көрсеткіштен бүтін санды азайту да, бір көрсеткіштен екінші көрсеткішті азайту да осылайша анықталады. Бүтін массивтің көршілес элементтеріне сілтенетін **int*** типіндегі екі көрсеткіштің айырымы 4-ке тең.

N бүтін саны көрсеткішпен қосылғанда (алынғанда) көрсеткіш ол сілтенетін осы типтегі N айнымалы қанша байтқа ие болса, сонша байт жад ұяшығы бойынша жылжиды. Екі көрсеткішті азайту берілген типтегі қанша айнымалы жадтың көрсетілген ұяшықтары арасында орналасатынын көрсетеді Көрсеткіштер арифметикасы жадқа тізбектей орналастырылған мәліметтер құрылымына қолданғанда ғана мағынаға ие, мысалы, массивтерге, өйткені тек осы жағдайда ғана компьютер кепілді түрде бір типтегі айнымалыларды ұяшыққа бірінен кейін бірін қатар жазады

Кез келген типтегі көрсеткіштердің өздері жадта барлық уақытта 4 байтқа ие. яғни олармен теориялық жағынан жадтын 4 Гетабайтын (миллиард байт) адрестеуге болады Көрсеткіштің мәні байттардың жад блогына бірінші кіргендердің адресі болып табылады. Яғни, мысалы **int *** типіндегі көрсеткіш логикалық түрде жадтың 4 байтын бірден адресі болып табылады Демек, компьютер жадында осы жадтағы сандық мәндерді басқару үшін емес, ал қандай да бір байттың адресін сақтауға арналған көрсеткіштің әмбебап типі бар болуы тиіс. Көрсеткіштің осындай типі **void** кілттік сөзі арналады:

void *pv = **NULL**; **int** k = 5; pv = &k;

Мұнда pv көрсеткіші k бүтін айнымалысының адресін сақтайды, ал дәлірек ол k айнымалысымен жүргізілген байттар тобынан бірінші байтқа сілтенеді. Бұл көрсеткішке «*» амалын қолдану бәрін түсінетін компбтер тарапынан ең тосын қарсылық тудырады. Көрсеткіштердің бұл типі тек адрестерді сақтауға арналған, ал жадқа жанама қатысты емес. Егер **void*** типіндегі көрсеткішпен адрестелген жад бөлігін өңдеу керек болса, онда оны көрсеткіштердің сәйкес типіне анық түрлендіру керек.

void типіндегі көрсеткіштер көметімен кез келген типтегі айнымалылар адрестерін функцияға жіберуге болады, ал соңынан функция денесінде алынған қосымша ақпаратпен сәйкес көрсеткішті талап етілген типке келтіруге болады. Осылайша әмбебап жалпыланған әртүрлі типтегі мәліметтерді сәйкес өңдейтін функциялар құруға болады. Бұл жағдайда функция параметрлері тізіміндегі **void** кілттік сөзі немесе функция қайтарған тип ретіндегі мән "**ештеңе**" дегенді білдіреді.

Келесі программа көрсеткіштің өсу амалын көрсетеді:

```
#include <stdio.h>
int main(void)
{  double D[]={1.1, 2.2, 3.3, 4.4, 5.5};
   double *pd;
   pd = &D[0];              // pd = D;  болуы да мүмкін
   pd = pd+4;
   printf("%f\n", *pd);
   return 0;
}
```

Жауабы: 5.5

Арифметикалық әрекеттерді көрсеткіштермен орындағанда берілген көрсеткішпен адрестелген объектінің өлшемін ескеру керек. Біздің жағдайымызда өсім 4*8 = 32 болады. Нәтижеде pd көрсеткіші әр бірі 8 өлшемімен (өлшем типі **double**) төрт ұяшыққа жылжиды.

Инкремент көрсеткішті реті бойынша келесі массив элементіне орын ауыстырады, ал декремент – алдындағысына Іс жүзінде көрсеткіштер мәндері ***sizeof*** (тип) шамасына өзгереді Егер белгілі бір типке көрсеткіш тұрақтыға артатын немесе кемитін болса, онда оның мәні берілген типтегі объект өлшеміне көбейтілген осы тұрақтының шамасына өзгереді. Мысалы

```
int *pi = new int[5];
pi++                              // pi мәні 4 (байт)-қа артады
double  *pd = new double[5];
pd++;                             // pd мәні 8-ге артады.
```

Екі көрсеткіштің айырымы – байтпен алынған тип өлшеміне бөлінген олардың мәндерінің айырымы (массивтер үшін, мысалы үшінші және алтыншы элементке көрсеткіштің айырымы 3-ке тең).

Екі көрсеткішті қосуға болмайды.

Төмендегі амалдарды орындағаннан кейін K айнымалысы 4 мәнін қабылдайды:

```
int  arr[5],*pi,*qi,K;
pi = &arr[0]; qi = &arr[4];  K = qi-pi;
```

Көрсеткіштер типінің айырымы (мысалда бұл **int** типі) компилятордың ерекшеліктеріне тәуелді әртүрлі анықталады. **Си** тілінде көрсеткіштер айырымының типін анықтау мақсатында **stddef.h** тақырыптық файлының **ptrdiff_t** аты анықталған, оның қолданылуы көрсеткіштер айырымы типін бір мәнді беруге мүмкіндік береді.

```
#include <stdio.h>
#include <stddef.h>
int main (void)
{  int  arr[5];
   int  *pi,*qi;
   ptrdiff_t  K;
   pi = arr;
   qi = &arr[4];
   K = qi-pi;
   printf("\n K=%d", (int)K);    // K = 4
   return 0;
}
```

1.3.1. Арифметикалық амалдар және көрсеткіштер

`&` және `*` унарлы адрестік амалдары кәдімгі арифметикалық амалдарға қарағанда жоғарырақ приоритетке ие. Осы ережені көрсететін мысал қарастырамыз:

```
float x = 4.0, *pf, y;
pf = &y;          // pf көрсеткіші y айнымалысын адрестейді
*pf = 5;          // атын алып тастау амалының көметімен
                  // y -ке 5 мәні
                  //көрсеткіш арқылы айнымалыға меншіктейді
x = x+*pf+1;  //4.0+5+1 = 10.0 x мәні 10.0 болды
```

Арифметикалық өрнектерде `*` адрестік амалын қолданғанда, бөлу `/` және атын алып тастау `*` амалдар белгілерінің кездейсоқ үйлесімінен сақтануға тура келеді, өйткені `/*` комбинациясын компилятор комментарий басы ретінде қабылдайды. Мысалы, a/*b өрнегін былай ауыстыруға болады: a/(*b).

`*` және `++` немесе `--` унарлы амалдары бірдей приоритетке ие және қатар орналастырғанда оңнан солға қарай орындалады. Инкремент - `++` және декремент - `--` унарлы амалдары көрсеткіштің мәні қашан өзгеруі керек екенінен тәуелді өриекте оның қолдануына не дейін, не кейін екеніне сәйкес префиксті немесе постфиксті болады.

Келесі программа үзіндісін карастырайық:

```
int arr[4] = {0,2,4,6}, *pi, k:
pi = arr;         // pi arr[0] элементінің адресіне тең
k = *pi;          // k 0-ге тең; pi &arr[0]-ға тең
k = *pi++;        // k 0-ге тең; pi &arr[1]-ға тең
k = ++*pi;        // k 3-ке тең; pi &arr[1]-ге тең
k = *++pi;        // k 4-ке тең; pi &arr[2]-ге тең
k = (*pi)++;      // k 4-ке тең; pi &arr[2]-ге, ал arr[2] 5-ке тең болды
k = ++( *pi),     // k 6-ға тең; pi &arr[2]-ге, arr[2] мәні 1-ге артты
```

Адрестермен және көрсеткіштермен жұмыс істей отырып, `*`, `++`, `--` және `&`, амалдарының орындалуына мұқилт болу керек, өйткені олар өриектерде эртүрлі үйлесімде қолданылуы мүмкін. Мына программа осы амалдар приоритеттерінің ерекшеліктерін көрсетеді.

```
#include <stdio.h>
int main (void)
{ int i1 = 10, i2 = 20, i3 = 30; // айнымалылар стекке орналасады
                                 // i1 ең үлкен адреске ие
```

```
    int *pı = &ı2,
    printf("\nМәні i2 = %d", *&i2);  // алдымен  1-ге артқан i2 мәні
    printf("\nМәні i2 = %d",*&++i2);
    printf("\nМәні i2 = %d", *pi);   // көрсеткіш мәні 1-ге артады
    printf("\nМәні i2 = %d", *pi++);
    printf("\nМәні ı1 = %d", *pi)   // алдымен 1-ге артқан ı1 мәні
    printf("\nМәні i1 = %d", ++*pi);  // i2 алдымен pi –ге кемиді
    printf("\nМәні i2 = %d", *--pı); // i3 мәні pi-ге кемиді, соңынан
                                      // алынған i3мәні 1-ге артады
    printf("\nМәні i3 = %d", ++*--pi);
    return 0;
}
```

Программаның орындалу нәтижесі

i2 = 20 мәні
i2 = 21 мәні
i2 = 21 мәні
i2 = 21 мәні
i1 =10 мәні
i1 = 11 мәні
i2 = 21 мәні
i3 = 31 мәні

1.3.2. Көрсеткіштер және қатынастар

Көрсеткіштерге салыстыру амалдары қолданылады: `>`, `>=`, `<`, `<=`, `- =`, `!=`. Көрсеткіштерді қатынастарда қолдануға болады. Көрсеткіштерді осындай типтегі көрсеткішпен немесе шартты нөлдік адрес мәндерін белгілейтін NULL тұрақтысымен ғана салыстыруға болады.

Мысалы:

```
#include <stdio.h>
float x[] {1.0,2.0,3.0,4.0,5.0};
int main (void)
{ float *pf1, *pf2, int i;
  printf("\n Көрсеткіштер адресі:&pf1 = %p,&pf2 = %p",
&pf1, &pf2);
```

```
    printf("\n Массив элементтерi адресi: \n" );
    for(i=0; i<5; i++)
      { if(i == 3) printf("\n");
     printf("&x[%d]=%p", i, &x[i]);
      }
    printf("\n Массив элементтерi мәндерi: \n");
    for(i=0; i<5; i++)
     { if ( i == 3) printf ( "\n");
    printf ( "x[%d] = % 5.1 f",i, x[i]);
     }
    for(pf1 = &x[0], pf2 = &x[4]; pf2 >= &x[0]; pf1++, pf2--)
     {
    printf("\n pf1 = %p *pf1 = %5.1f  pf2 = %p *pf2 = %5.1f",
pf1, *pf1, pf2, *pf2);
    printf("\n pf2-pf1=%d", pf2-pf1);
     }
return 0;
}
```

Көрсеткіштер айырымының мәндерін және **printf**() функциясында адрестерін басқанда %d – түрлендіру спецификациясы қолданылған ондық белгісі бар бүтін санды шығару.

Сұрақтар және жаттығулар

1. Көрсеткіштерге қолданылатын амалдар.
2. Көрсеткіштің өзгертілетін жадтың өлшемі бүтін санмен қосқанда (алғанда) немен байланысты?
3. N бүтін сан көрсеткішпен қосылғанда (немесе одан алғанда) көрсеткіш жад ұяшықтары бойынша қанша байтқа орын ауыстырады?
4. Көрсеткіштер "арифметикасы" мәліметтердің қандай құрылымына қатысты қандай мағынаға ие?
5. Қандай программалық объект, көрсеткіштің өзіне ие: адрес, ұзындығы және мәні. Көрсеткіштің осы қасиетін түсіндіріңдер.

6. **void*** көрсеткіші қандай пайдалы қасиеттерге ие?
7. **void*** типіндегі көрсеткішпен адрестелген бөлікті өңдеу керек болса, не істеу қажет?
8. **void** кілттік сөзі функция параметрлері тізімінде, не ол қайтаратын нәтиже типі ретінде нені білдіреді?
9. Си тілінде қолданылуы көрсеткіштердің айырымы типін бір мәнді беруге мүмкіндік беретін көрсеткіштер айырымы типін конкретизациялау мақсатында не анықталған?
10. Көрсеткіштер «арифметикасында» инкремент (декремент) унарлы префикстік және постфикстік амалдарының айырмашылығы қандай?
11. Келесі жазбалар неге мүмкін:
    ++ & k; -- & k ++; мұндағы **k** – бүтін гиптеті айнымалы?
12. **%** p – ны **printf()** операторына түрлендіру спецификациясы нені көрсетеді?
13. Келесі объектілер анықталсын:

```
int a = 10, b = 20, c = 30;          // айнымалы стекке ие
int  *p = &b:                        // максималдық адрес
int  k ;
```

Баспаға не шығады:

```
k = *&b;          printf("*&b = %d \n", k);
k = *& ++ b;      printf("*& ++b = %d \n", k);
k = *p;           printf("*p = %d \n", k);
k = *p ++;        printf("*p ++ = %d \n", k);
k = *p;           printf("*p = %d \n", k),
k = ++ * p,       printf("++*p = %d \n", k);
k = *-- p;        printf("*--p = %d\n", k),
k = ++*--p;       printf("++*--p = %d", k),
```

14. Келесі фразалар арасында логикалық байланыс құру:
    1) **&p** – көрсеткіш адресі.
    2) p көрсеткіш. Көрсеткіш мәні.
    3) Объект адресі. P көрсеткішінің мәні
    4) ***p** – p көрсеткішімен адрестелген объект. Объект мәні.

Осы байланыстардың сұлбасын графикалық түрде сызу.

1.4. Массивтер мен көрсеткіштер арасындағы байланыс

Жоғарыдағы материалдардан көрсеткіштер мен массивтер арасындағы тығыз байланыс көрініп тұр, олардың элементтері компьютер жадында тізбектеліп, әр бөлімдерде орналасқан. Осыған орай адрестердің үлкендігімен массив элементтерінің және арифметикалық көрсеткіштердің спецификациясы, '++' амалымен көрсеткіштердің типтері көрсеткіштердің мәндерін 1, 2, 4 және 8 бірліктеріне өзгерте алады. Бұл '++' амалы көметімен кез келген типті массив элементерінен кезектесіп өтуге жол береді.

Жоғарыда келтірілген талдауға қарағанда, бұл талдау одан да тығыз. Өйткені, массивтің аты квадрат жақшаларсыз қолданылатын массивтің бірінші элементінің адресін көрсетеді.

Массив элементтерінің адрестері компиляция орындалғанда анықталынып, программамен жұмыс істеген кезде өзгертілмейді. Қорыта келгенде, массивтің аты көрсеткіштің тұрақтысы болып келеді (0-ге тең, индексті массив элементінің адресімен инициализацияланған).

Сонымен, массивпен екі тәсілмен жұмыс істеуге болады екен, не алғашқы амалы синтаксис индекстеуі арқылы немесе көрсеткіш синтаксисі арқылы. Екі тәсілдің біреуін таңдау программистің талғамына тікелей байланысты. Оларды тіпті бір программада қолдануға болады.

Массивпен жұмыс істетенде екі тәсілді қолдану:

```
#include<stdio.h>
#include<conio.h>
int main(void)
{ int  arr[8] = {1, 6, 3, 8, 2, 8, 1, 3};
  int  sum, i ;
  for( i = 0, sum = 0; i < 8;  i++)
  sum = sum + arr[i];
  printf("\n қосынды = %d \n", sum);
  sum = 0,
  for( i = 0; i < 8; i++)
  sum = sum + *(arr+i);
```

```
    printf("қосынды = %d", sum);
    getch();
    return 0;
}
```

Осындай екілік синтаксисі жадтың динамикалық буферлеріне де қатысты болады. Егер келесі буфер 10 бүтін мәнге (40 байт) белгіленген болса, **int** *p = (**int***) ***malloc*** (**sizeof(int)***10);

онда бөлек бүтін мәндерге көрсеткіш синтаксисіне қатынаған сияқты әрекеттесуге болады:

```
*( p + 4)              // 5 - элементтің мәніне қатынас
```

p [4] – индекстеу көмегімен «инверттелген» массив символдарының есебін көрсеткіштердің көмегімен әртүрлі тәсілдермен қарастырайық

Бірінші шешім:

```
char  str[80],  lit;
char  *pc,  *qc,
for (pc = str, qc  = &str[79]; pc < qc: pc++. qc--)
{ lit = *pc,   *pc = *qc;   *qc = lit, }
```

Циклдың басында көрсеткіштерді инициалдаймыз, яғни оларды массивтің бірінші және соңғы элементтеріне адрестейміз; циклды басқару үшін екі циклды салыстырамыз, инкремент және декремент амалдарымен түрлендіреміз. Цикл денесі амалының мағынасы көрініп тұр.

Екінші шешім.

```
char  str[80], lit, *pc, *qc;
for (pc = str, qc = &str[79]; pc < qc;) // көрсеткіштер өзгерісі
                                        // цикл денесіне көшірілді
{lit = *pc;   *pc++ = *qc;   *qc-- = lit;}
```

Үшінші шешім:

```
char str[80], lit, *pc, *qc;
 pc = str;
 qc = &str[79];
 while (pc < qc)
 {  lit = *pc;   *pc++ = *qc;   *qc-- = lit;}
```

Төртінші шешім – индекстелген көрсеткіштердің араласуы:

```
char str [80]. lit:
int i;
for( i=0; i < 40; i++)
{  lit = *(str+i);  *(str+i) = *(str+(79-i)); *(str+(79-i)) = lit;  }
```

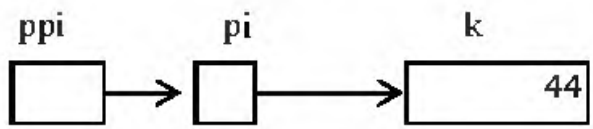
Сонымен, &str[i] және (str+i) - i позициясының басынан басталатын, массивтің бір элементінің адресін анықтайтын екі форма болып келеді.

(str+i) – бұл i массив элементінің адресі, ал *(str+i) – бұл элементтің i мәні, біздің жағдайда - бұл символ. Өйткені массивтің аты тұрақты болып келеді, массивтің басталған адресін қамтитындықтан массив атына '++', '--' амалдары қолданылмайды; массив атына мән беруге болмайды.

Екі өлшемді массивпен жұмыс істейтін көрсеткіш синтаксисін қарастыру үшін бастапқыда көрсеткішке көрсеткішті қарастырамыз, ол жай типті мәліметтер көрсеткішіне қарағанда жоғары реттелген кез келген тип.

```
int k;
int *pi = NULL;
int ** ppi = NULL;
pi = &k,
ppi = &pi;
**ppi = 44;                    // К мәні 44 саны болады
```

Осындай байланыстың графикалық түрі:



Көрсеткішке көрсеткішті анықтау үшін (екілік көрсеткіш) '*' символын, ал екілік типтен жай типке қатынау үшін '*' амалын екі реттен қолдану керек.

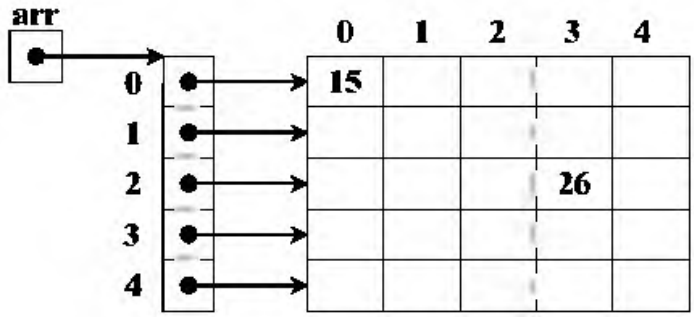
Егер ppi екілік көрсеткіш арқылы жадпен қатынассақ, онда '*' операциясы бір рет қолданылады, ол аралық pi көрсеткішіне қатынауға және оның мәнінің адресін өзгертуге жол беретіндіктен, * ppi = = pi болады.

Көрсеткіш синтаксисін екі өлшемді массивпен қалай жұмыс істейтінін қарастырамыз. Бұл жағдайда көрсеткіш синтаксисі екі өлшемді массивтің атына негізделген, өйткені екі өлшемді массив "екілік көрсеткіш – тұрақтысын" көрсетеді, '*' амалын бір рет қолдану арқылы жадтағы адрес блогына және массивтің бірінші элементіне өтуге болады. Егер (матрица аты +1) мәніне '*' амалын қолдансақ, онда жад блогының адресіне өтуге мүмкіндік аламыз, ол жерде екі өлшемді массивтің екінші жолы орналасқан.

int arr [5] [5]; екі өлшемді массив үшін *arr мәні 5 жолдан және 5 бағанадан массивтің бірінші жолын көрсетеді (нөлдік индексімен), ал *(arr + 1) мағынасы екінші жолға көрсеткіш болып келеді, және т.с.с.

* (* (arr + 2) + 3) – arr массив элементінің 3 - жолы 4 - бағанасының бүтін мағынасын береді, ал ол 26 саны. **arr мағынасы – бұл 1-жолдың 1-бағынының бүтін саны немесе 15 саны. * arr мағынасының мәні – ол бірінші жолға көрсеткіш, ал ** arr осы жолдың біріншісі

Жоғарыда айтылғандардың графикалық түрі:



Көрсеткіш мағынасы – ол байттар блогындағы бірінші байттың орны.

Төменде екі тәсілді екі өлшемді массивте қолдану мысалы көрсетілген:

```
# include <stdio.h>
int main(void)
{  int  arr[2][3] = {{1, 6, 3}, {8, 2, 8}};
  int sum  = 0, i, j;
```

```
    for ( i = 0; i < 2; i++)
    for( j = 0; j < 3; j ++)
       sum = sum + arr[i][j];
       printf("\n қосынды = %d \n", sum);
       sum = 0;
       for ( i = 0; i < 2; i ++)
       for( j = 0; j < 3; j ++)
       sum = sum + *(*(arr + i) +j);
       printf("қосынды = %d", sum);
       return 0;
 }
```

Екі өлшемді бүтін массивке динамикалық жадтан орын бөлгіміз келсе, онда көрсеткішті массивке анықтау керек. C/C++ тілдерінің көрсеткш типтері буферде динамикалық жадта жол мен бағандарға орын бөлуді қамтамасыз етеді:

```
int ( *par )[4];
par = new  int [12][4];
for( i = 0;  i < 12;  i++)
for( j = 0;  j < 4;  j++)
par[i][j] = i + j;
delete [ ] par,
```

Бұл жерде par көрсеткішінде келесі тип болады:

```
int (*)[4],
```

ол «int типтегі төрт элементті массивке көрсеткш» болып оқылады.

Бұл типтердің абстрактты бейнелеуіне мысал.

Екі өлшемді массив үшін динамикалық жадтан орын бөлудің басқа тәсілін көрсетейік n*m:

```
int  **par;
 par = (int**)malloc(sizeof( int* )* n),
```

par екілік көрсеткішін жад блогында байт (4*n) – тен адрестейміз. Блок – бұл көрсегкіштер массиві. Өйткені malloc() функциясы void * типіне көрсеткішті қайтарады, сондықтан алмасу міндетті түрде орындалады.

Келесі қадамда n жад блогында екі өлшемді массивте m бүтін мәндерді белгілейміз:

```
for( i = o;  i < n;  i++)
par[i] = (int*)malloc(sizeof(int)*m).
```

енді алынған екі өлшемді массивті инициалдаймыз:

```
for( i = 0;  i < n,  i++)
for( j = 0;  j < m;  j++)
par [i][j] = i + j;
```

мұндағы par – көрсеткішке көрсеткіш, par[i] – i - көрсеткіш массивінің элементі, матрицаның i-жолына көрсеткіш болып келеді.

Процедурада жадты босату үшін көрсеткіш массивінде көрсеткіштерді жою керек:

```
for( i = 0;  i < n;  i++)
free ( par[i]);
free ( par);
```

1.4.1. Көрсеткіштер массиві

Күрделі мәліметтермен жұмыста көрсеткіштер массиві динамикалық жадты басқару және компьютер жадын тиімді пайдалану үшін қолданылады.

Көрсеткіштер массивтерінің дәлдік өлшемдері мынадай анықтамалар арқылы енгізіледі:

*тип * массив_ аты* [өлшем] ;
*тип * массив_ аты* [ ] = *инициализатор*;
тип массив_ аты [өлшем] = *инициализатор*;

Мысалы:

```
int data[6];                    // кәдімгі массив
int *p1[6];                     // көрсеткіштер массиві
int *p2[ ] = { &data[0], &data[4], &data[2] };
```

Бұл жерде **p1** және **p2** массивтерінің әрбір элементі **int** типті объектілер-ге көрсеткіш болып келеді.

Көрсеткіштер массивін матрица жолдарын сұрыптау қызық. Сұрыптау тәсілінің саны көрсеткштер массивінің санын анықтайды. Массив элементтерінің өсу және кемуін реттейтін жай программаны қарастырайық:

```
#include<stdio.h>
int main (void)
{  float arr[ ] = {5.0,2.0,3.0,1.0,6.0,4.0};
float  *pmin[6];
float  *pmax[6],
float  *p;
int  i, j ;
for ( i = 0; i < 6; i++)
pmin [i] = pmax [i] = &arr[i],
for ( i = 0;i < 5;i ++)
for ( j = i + 1; j < 6, j ++)
    {   if ( *pmin[i] < *pmin[j] )
              {
                    p = pmin[i];  pmin[i] = pmin[j]; pmin[j] = p;
              ]
         if (*pmax[i] > *pmax[j] )
              [
                   p = pmax[i]; pmax[i] = pmax[j]; pmax[j] = p,
              ]
        ]
printf("\n Кему бойынша:\n");
for ( i = 0;i < 6;i ++)
printf("\t%5.1f",*pmin[i]);
printf("\n Өсу бойынша:\n");
for (i = 0;i < 6; i++)
printf("\t%5.1f",*pmax[i]),
return 0;
}
```

Енді көрсеткіштер массивін қолдану мен екі өлшемді массивті әртүрлі объектілерге жадтан динамикалық орын бөлуді қарастырайық. Кез келген ұзындықтағы енгізілген сандық мәндерді тәрістеп экранға шығару керек:

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

```
int  **p,              // көрсеткіштер массиві үшін
                       // бөлінген жад блогына p - көрсеткіш
int  i, j, k, n;
int  *q;               // жад блогы үшін q - көрсегкіш
printf("\n Жолдар санын енгізіңіз n = ");
scanf("%d",&n);
p = ( int **)calloc(n,sizeof(int *)),
q = ( int *)malloc(sizeof(int) *n);
for( i = 0; i < n; i++)
   {
      printf("Жол ұзындығын енгізіңіз q[%d] = ",i);
      scanf("%d",&q[i]);
       p[i] = ( int *) calloc(q[i],sizeof(int));
       for( j = 0; j < q[i]; j ++)
          {
           printf("p[%d]{%d] = ", i, j);
           scanf("%d",&k);
           p[i][j] = k;
           ]
   ]
printf("\n Өңдеу нәтижелері");
for( i = n-1; i >= 0; i--)
   {
    printf("\n Жол %d,  элементттер%d:\n",i,q[i]);
    for( j = 0; j < q[i]; j++)
    printf("\t%d", p[i][j]),
    free(p[i]);        // Жол жадын босату
   }
    free(p)            // Жадты көрсеткіштер массивінен босату
   free(q);            // int массивінің жадысын босату
   return 0;
]
```

1.4.2. Жолдар және көрсеткіштер

Жоғарыда көрсеткіштер мен сандық массивтер арасындағы байланыс қарастырылды. Жол да массив, сондықтан оның көрсеткішпен байланысына таң қаларлық ештеңе жоқ.

Жолдық тұрақты – бұл соңы нольдік символ ('\0') болатын символдық массив. Келесі түрде жолға көрсегкішті көрсетіп. көрсеткіш синтаксисін инициалдауға болады:

```
char *ps = " My string",
```

жолының соңында '\0' символы қосылады.

char * типті ps көрсеткішіне жолдың бірінші символының адресі меншіктеледі, яғни 'М'

Бұл жолды басқаша жазуға да болады:

```
char *ps;
ps = "My string";
```

Жолды анықтаудың бастапқы форматы:

```
char str[20];
```

str массиві стектен берілген анықтаманы өңдеу барысында жад алады. ps көрсеткішімен байланыс орнататын жолға анықтаманы өңдеу барысында жад бөлінбейді. Сондықтан келесі амалдар болса

```
scanf ("% s", str);   // амал дұрыс
scanf ("% s", ps);    // амал дұрыс емес,
                      // өйткені жолға орын жоқ
```

Бірінші өтеді, ал екінші қатеге ұшыратады, ол анықталмаған жад орнына енгізу. ps көрсеткішімен екінші енгізу амалын қолдану үшін, кейбір жад бөлімдерін байланыстыру керек. Бұл үшін бірнеше тәсіл бар. Біріншіден, ps өзгеріміне стекте анықталған символдық массивтің адресін беруге болады, яғни ps = str. Екіншіден, ps көрсеткішін «жиын» жад бөлігінде «өңдеуге» болады.

Мысалы,

```
ps = (char*)malloc(80),
```

меншіктеу амалы 80 байтты орын бөліп, ps көрсеткішімен байланысады. Енді екінші енгізу амалын қолдануға болады.

Келесі екі жолды қарастырайық:

```
char *ps = "My string";
char str[] = "My string";
```

Негізгі айырмашылық мынада, ps көрсеткішінің мәні өзгерімді, өйткені көрсеткіш ол - өзгерімді, ал str массивінің аты өзгертілмейді, өйткені массив аты – бұл тұрақты көрсеткіш.

Сондықтан келесі бейнелеу қате:

```
char arr[15],
arr = "My string";
```

өйткені меншіктеу амалы бірінші жолдың адресін arr объектісіне көшіреді. Мұнда қате тұралы хабар шығады, өйткені массивтің аты arr - бұл тұрақты

```
char *ps;
scanf("% s", ps),
```

үзіндісі қате, өйткені ps көрсеткіші жарияланған, бірақ инициалданбаған.

Мәселені келесі түрде шешуге болады:

```
char str[10];
char *ps = str;
```

ұзындығы он символдан тұратын жолды оқи алады, өйткені **scanf()** функция-сында массивтің бұрыниан бар адресі беріледі:

```
scanf("% s", ps);
```

Динамикалық жадта жолға орын бөлу үшін **char *** көрсеткішін сипаттау, сосын **new** амалы немесе **malloc** функциясы көмегімен орын бөлу керек:

```
char *ps = new char[N];
char *ps = (char *)malloc(sizeof(char)*N),
```

Бұл жағдайда жолдың ұзындығы программа барысында өзгертілуі мүм-кін. Динамикалық жолдарды басқа динамикалық массивтер сияқты құрғанда инициалдауға болмайды

```
char *ps = "It is a constant";
```

жолдық өзгерімді емес, ал жолдық тұрақтыға көрсеткш, ол өзгертілмейді.

char * типті көрсеткіштер көмегімен символдың массив ретінде жолға байланысуы оңай. Көрсеткіштер әртүрлі

ұзындықтағы сөздермен жұмыс істегенде тиімді. Келесі программада инициалданғаннан кейін жолдар жиынымен байланысқан **char** * типті бір өлшемді массив анықталған **printf()** функциясы және %s ауыстыру спецификациясы жолдық көрсеткішке қолдану параметрлерін пайдалануға жол береді, экранға arp[i] көрсеткішінің мәні емес, адрестелген жолдың мәні шығады.

```
#include <stdio.h>
int main (void)
[ char  *arp[] = { "One", "Two","Three","Four","Five"};
int i, n;
n = sizeof (arp)/sizeof(arp[0]);          // массив ұзындығы
for ( i = n; i > 0; i--)
printf("\n % s", arp[i]);
return 0;
}
```

Программа нәтижесі

```
Five
Four
Three
Two
One
```

C/C++ тіліндегі барлық стандартты функциялар жолды өңдеу барысында жолдық көрсеткіштермен жұмыс жасайды. Осындай көп функциялардың ішінде бізге **strtok()** функциясын қарастырған пайдалы, оның көмегімен жолдарды лексемдерге – бөлінбейтін тізбектелген литерлерге бөлуге болады.

```
#include <stdio.h>
#include <string.h>
int   main (void)
{ char str[] = "This sentence; has - 5 words.";
  char *p;
  printf ( "%s \n", str);
  printf("жолы сөзге таратылады:\n");
  p = strtok(str," ;-.");
  while( p != NULL)
```

```
    {  printf("% s\n", p);
       p =  strtok( NULL. "  ;-.");
     }
     return 0;
   }
```

strtok() функциясының параметрлері ағымдағы жолға көрсеткіш және келесі бөлгіштерден тұратын жолдық тұрақты болып келеді: пробел, үтір - нүкте, дефис және нүкте. **strtok()** функциясы ағымдағы жолдан лексеманы қайтарады, жоғарыда айтылған бөлгіштердің көмегімен басқа жолдардан бөлек тұрған жол. **p** – бұл бөлгіштен кейінгі символға негізделген көрсеткіш. содан бастап келесі лексеманы іздеу басталады. Кітапханалық **strtok()** функциясын шақырмау керек. **strtok()** функциясы егер жолда лексемалар табылмаса нөлдік көрсеткішті қайтарады.

Программа нәтижесі:

```
This sentence;has-5 words.
Сөзге таратылады:
This
sentence
has
5
words
```

Сұрақтар және жаттығулар

1. Тік жақшасыз қолданылатын массивтердің аты нені білдіреді?
2. Көрсеткіш және амалдар синтаксисін индекстеу дегеніміз не?
3. і позициясынан басталатын массивтің бір элементінің адресін анықтайтын екі форманы көрсетіңіз.
4. Екі өлшемді массивпен жұмыс жасайтын көрсеткіш синтаксисінің мағынасы неде?
5. '++' , '--' амалдары массив атына қолданылама? Меншіктеу амалының сол жағында массивтің аты қолданыла ма?
6. Нақты сандарды енгізіп, кері мәндерін шығару.

7. Өлшемі анықталған көрсеткіштер массивтерін енгізетін, барлық анықтамаларды айтып шығыңыз.
8. Келесі жазу дұрыс па:
   массив_аты == &массив_аты == & массив_аты[0]?
9. Келесі анықтамалар болсын:

```
int m[ ] = {10,20,30,40};
int i = 1, k;
```

k өзгерісі қандай мәндер қабылдайды:

```
1  k = m[i];          printf("\n m[i] = %d\n",m[i]);
2  k = *(m+i++);      printf("*(+i++) = %d\n",*(m+i++)),
3  k = *(++i+m);      printf("*(++i+m) = %d\n",*(++i+m));
4  k = i[m];          printf("i[m] = %d\n",i[m]);
5. k = *(i--+m),      printf("*(i--+m)=%d\n",*(i--+m)),
6. k = i--[m];        printf("i--[m] = %d\n",i--[m]),
7. k = *(i--+m),      printf("*(i--+m) = %d\n",*(i--+m));
8  k = --i[m];        printf("--i[m] = %d",--i[m]);
```

1.5. Функция параметрлерінің көрсеткіштері

Функцияны шақыруды эртүрлі механикалық параметрлер **мәні, көрсеткіші** және **сілтемесі** бойынша орындауға болады. Мән бойынша параметр беру сұлбасы нақты параметрінің көшірмесін білдіреді. Сондықтан да объектіні шақыру программалары нақты параметрді қолдану кезінде функция денесін өзгертуге болмайды. Бірақ шақырылған функцияның объектіні шақыру программаларының мәнін жанама түрде өзгертуге болады. Осы мүмкіншілікті көрсеткіш аппараты қамтамасыз етеді. Көрсеткіштің көмегімен функция кез келген шақырылған программалар *адресін* береді, ал бөлу көрсеткіші арқылы біз объект шақырылған прораммалардың шын көрсетілген адресті көруге рұқсат береді. Функция параметрін өзгертпей объект шақырылған программа шін өзгертуге болады, себебі көрсеткіш параметрі бір адресті көрсетеді.

Мысалы:

```
#include <stdio.h>
void  positive (int *p),
{
   *p = (*p>0)?  *p :  -*p;
]
int main (void)
{ int  k = -3;
  positive(&k);
  printf("\n k = %d", k);
  return 0;
}
```

Жауабы: k = 3.

Positive() функциясының параметрі **int*** көрсеткіштің типін көрсетеді. Осы функцияны шақыру кезінде k өзгеріс адресін нақты параметрде қолданып, **p** көрсеткішін иемденеді. *p атын өзгерткенде сақтау бөлімін p көрсеткішіне жіберуді қамтамасыз етеді. Сол себепті былай көрсетіледі:

*p = (*p > 0)? *p -*p,

p – к адрестің өзгерісі: *p - к өзгерістің құрамы: *(&k) - к өзгерістің құрамы.

Басқа мысалда негізгі программада екі объекті мәнінің өзгерісін көрсету үшін нақты формальды параметр функциясы нәтижелері функциядан шақырылған негізгі программалар нүктесіне көрсеткіш сияқты жазылуы тиіс. Функция денесінен объект шақырылған программалар көрсеткіш арқылы параметрлер адресін көрсеткіші ретінде көреміз.

```
#include <stdio.h>
int  main(void)
{ int  m, n;
  void  change( int *, int *);
  printf ("\n Енгізіңіз m =");
  scanf ("% d", &m);
  printf ("\n Енгізіңіз n =");
  scanf ("% d", &n);
  change (&m, &n );
```

```
printf("\n Жауап алмасу: m = %d n = %d", m, n),
 return 0;
}
void change ( int *p, int *q)
{ int  term;
term = *p;   *p = *q;   *q = term;
]
```

Негізгі программаларда m және n өзгеріс мәнін қолданушы пернетақтадан енгізеді change() формальды параметр функциясы int * көрсеткіш типі арқылы көрсетіледі. Функцияның қойылымы өзгеріс мәнінің орнын сол сілтеген параметрге алмастыру. change() функциясын шақырған кезде өзгеріс адресі m және n нақты параметрлер ретінде қолданылады.

1.6. Массивтер мен жолдар функция параметрлері ретінде

Кейде массивті өңдеу үшін қандай да бір функцияға жіберу керек. Оны істеу оңай, себебі біз білетіндей массивтің аты элементтің бірінші адресін көрсетеді. Іс жүзінде функцияда тек массивтің бастапқы адресі беріліп және әрі қарай онымен синтаксистік көрсеткіш арқылы жұмыс істейді. Сыртқы функция барлық мағлұматты алу үшін массив өлшемін жеке аргумент бойынша беру керек (көп өлшемді массивке қатысты).

Функция бір өлшемді массивте элементтің барлық қосындысын есептеуге арналған мысалды қарастырайық:

```
#include <stdio.h>
int Mysum (int *p, int size);
int main (void)
{ int ar[3] = {1.6.3};
 int sum;
 sum = Mysum (ar,3);
 printf ("қосынды = %d\n", sum);
return 0;
 }
```

```
int Mysum (int *p, int size)
{ int result = 0;
 for (int I = 0; i < size; i++)
 result = result + *(p+i);
 result = result + p[i];
 return result;
 }
```

Көп өлшемді массивтің функцияға берілуін қарастыру үшін оқу тапсырмасын алайық: екі өлшемді массивті функцияға беріш, одан осы функцияда бүтін элементтердің қосындысы анықталуы керек.

Мысалы:

```
#include <stdio h>
int Fun (int ar[2][3], int n , int m );
int main (void)
{ int ar[2][3] = {1,2,3,4,5,6};
  int sum,
  sum = Fun (ar. 2, 3);
  printf ("\n қосынды = %d\n", sum);
  return 0;
 }
  int Fun (int a[2][3], int n, int m)
{ int s=0,
 for (int i = 0; i<n; i++)
 for (int j = 0; j<m; j++)
 s = s + a[i][j];
 return 0;
 }
```

Функцияны анықтағанда бірінші квадрат жақшадан 2 санын алып тастауға болғанмен барлық санды алып тастауға болмайды.

Екілік көрсеткішке кез келген әдіспен нақты жаңарту

```
sum = Fun(( int **) ar, 2, 3);
```

программада қателікке ұшыратады.

Дұрыс шешім екі өлшемді массив Fun кіріс функция параметрі ретінде көрсетілуі тиіс. Ол былай көрсетіледі:

int Fun (**int** (*a)[3], **int** n, **int** m),

келесі функцияны шақырған кезде:

sum = Fun (ar, 2, 3);

Бұл жағдайда программа дұрыс және қатесіз шығады.

Біз оқу тапсырмасын қиындатайық: Fun функциясында AR екі өлшемді массивті сол өз өлшеміндегі ar массивте инициалдап, ar массивіндегі элементті екі есе етіп аламыз. Бұдан басқа AR инициалданған массивтің барлық қосындысын және Fun функциясын қайтарамыз. Осы есептің шешімі төменде қарастырылған:

```
#include <stdio.h>
int Fun (int a[2][3],int n,int m,int b[2][3], int *);
int main (void)
{ int ar[2][3] = {1,2,3,4,5,6};
  int AR[2][3],
  int ret, sum;
  ret = Fun (ar, 2, 3, AR, &sum);
  printf ("ret = %i sum = %d\n", ret, sum);
   return 0;
  }
  int Fun(int a[2][3],int n,int m,int b[2][3],int *p)
  { int s = 0, sum = 0;
    for (int i = 0; i<n; i++)
    for (int j = 0; j<m; j++)
  { s = s + a[i][j];
      b[i][j] = 2*a[i][j];
      sum = sum + b[i][j];
  }
  *p = sum;
   return s;
  }
```

Жауабы: ret = 21, sum = 42.

Параметр функция ретінде массивке бірнеше ескертулер жасайық.

Функция аты:

```
int f ( int  n, int  A [] ) . . .
int f ( int  n,  int  *p)   . . .
```

Формальды параметрдің сипат тізімі бірдей құқылы болып келеді. Бірақ бірінші жағдайда А атының рөлі көрсеткіш ретінде айқын емес.

Функция денесінде массив элементін A[i] индекстеу амалы көмегімен және ұсынылған адрес мәнін екі бөлу арқылы жасауға болады. Яғни *(p+1) синтаксистік көрсеткіш формуласын қолдануға болады.

Массив әрқашан функция болып көрсеткіш арқылы берілетін болса, онда ішкі функцияның элемент массивінің мәнін өзгертуге, қарапайым амалды индекстеуге және элемент массивін екі бөлу көрсеткішінде қолдануға болады.

Бір өлшемді массив элементінің квадрат мәнін функция арқылы сипаттау.

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

Программа нәтижесі:

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

Функция аты былай болуы мүмкін:

void quart (int n, float ar []).

pf[i] индекстелген өзгерістер орнына *(pf + i) формуласын қолдануға және функция денесінде pf[i] деп жазуға болады.

Көрсеткіш мәнін дене функциясы ішінде өзгерту ең қызық мүмкіндік болып табылады. Мысалы, цикл денесінде төмендегідей амал жазуға болады:

*** pf *= * pf++;**

Қандай жағдайда guart() функциясын төмендегі әдісте қолдануға болады:

*** pf++ *= * pf++;**

Массив аты функция денесінде тұрақты көрсеткішті қабылдамайды. Ал **main()** негізгі программада әрқашан массив аты тұрақты болғандықтан, программаның негізгі денесінде. мысалы, **printf()** функциясында z[i] мәнінің орнына *z++ формуласын қолданғанда, қате туралы хабарланады.

Нәтижесі массив аты тұрақты көрсеткіштің мәні немесе массив элементінің нөлдік адресі ретінде бола алады. Массив аты функция параметрі ретінде қолданылады, сол функция денесіндегі қарапайым көрсеткіш рөлін орындайды. Формальды параметр функциясының сипат тізімінің эквиваленттілігі мынадай формада көрсетілген:

int ar[] және **int** *ar.

Жолдың функция параметрі ретінде қолданылуы. Жолдар нақты параметр бір өлшемді массив **char**[]немесе **char*** типті көрсеткіш бола алады. Екі жағдайда да жол ішіндегі нақты параметр көмегімен функцияға массив символының басты адресі беріледі. Қарапайым массивке қарағанда параметр жолының ұзындығын көрсету міндетті емес. Эскейп – жол соңы

'\0' жолдың соңын және элемент жолын шектен шықпай таңдауға мүмкіндік береді.

Стандартты функцияларды жолды өңдеу есептерінде қарастырамыз.

Жол ұзындығын шығару функциясы.

```
int len (char s [])
[ int n;
 for (n = 0; s[n] ! = '\0'; n++) ;      // денесіз цикл
 return n;
}
```

Компилятор әрқашан массивті оның басы ретінде, ал индексті массив басының алмасуы бойынша қабылдайды. Бұл функцияның көрсеткішпен жұмыс істеу механизмін көрсетеді:

```
int len ( char * ps)
{ int n;
  for ( n = 0; *ps ++; n++) ;
  return n:
}
```

Төмендегі нұсқасы жолдағы екі өзгерістің көрсеткішін береді:

```
int len ( char * ps)
{ char  *q;
  for ( q = ps;  *q ++;  );
  return  (q - p);
 }
```

Жолды кері айналдыру функциясы

```
void   invert ( char * ps)
{ char lit;
  int  i, j, m,          // m – позиция нөмірі  '\0'  жолда
  for ( m = 0; *ps ++; m++)
```

```
    for ( i = 0, j = m-1;  i < j;  i++, j++)
    { lit = *( ps + i);
      * ( ps + i )  = * ( ps + j),
      * ( ps + j) = lit;
    }
  }
```

'\0' кері айналдыру кезінде жолдың соңында қалады.

Іздеу функциясы

```
        // s2 жолын  s1 жолда іздеу
        // стандартты функцияның ұқсас түрі  strstr()
int  index ( char * ps1, char * ps2)
{ int   i , j, d1, d2;                    // d1, d2 – жол ұзындығы
                                          // жол ұзындығын шығару
   for ( d1 = 0; * (ps1 + d1); d1++ )
   { for ( d2 = 0; * (ps2 + d2); d2 ++)
     if ( d2 > d1)
     return  -1;
    }
    for (i = 0; i <= d1– d2;  i ++)
    {
      for ( j = 0; j < d2, j++)              // салыстыру циклі
      if ( * ( ps2 + j) != * (ps1 + ( i+j)))
      break;
      if ( j == d2)   return  i;
     }
      return   -1;
    }
```

Функция позиция нөмірін қайтарып, бастапқы ps2 жолдық үзіндісіне ps1 жол бөлігі бір–біріне ұқсас болып келеді.

Жолды салыстыру функциясы (стандартты функция **strcmp()**).

```
int row ( char s1 [] , char s2[] )
{  int  i, d1, d2,                        // d1,d2 – жол ұзындығы
```

```
for (d1 = 0; * (s1 + d1 ) != '\0'; d1++)
for ( d2 = 0; * ( s2 + d2) != '\0': d2++)
if ( d1 != d2)    return  -1;
for ( I = 0; i< d1; i++)
if(  *s1++ != *s2++)   return ( i+ 1);
return 0;
]
```

Функция денесінде жолдық массив элементті екіге бөлу көрсеткіші арқылы жасалады. s1++ және s2++ операциясы арқасында бастапқы «жөндеу» көрсеткіші массив арқылы өзгертіледі Бір мезгілде функция сол жолдық массивті *(s1 + d1) және *(s2 + d2) формула көмегімен қаратамыз және s1 мен s2 мәнін өзгеріссіз шығады.

Жолды қосу функциясы

```
void conc (  char * ps1 , char *ps2)
{ int  i, d1;                          // d1 – бірінші жол ұзындығы
  for ( d1= 0; *( ps1 + d1)  != '\0';  d1++ )
  for ( i=0; *( ps2 + i) != '\0'; i++)
  *( ps1 + d1+ i)  = *( ps2 + i );
  *( ps1 + d1 + i) = '\0';
  }
```

Нәтижеде бірінші аргументтің мәні ps1, ал екінші аргумент ps2 өзгермейді.

Жол ішіндегі мәліметті көшіру функциясы

Си тілінде жолды меншіктеу амалы жоқ болғандықтан, әртүрлі мәтінді көшіру әдісін қарастырамыз:

```
void copy ( char *ps1 , char ps2)  // ps1 – түпнұсқа, ps2 – көшірме
  { int i;
    for ( i = 0;  ps2[i] != '\0'; i++)
    ps1[i] = ps2[i] ;
     ps1[i] = '\0';
    }
```

```
void copy ( char c1[], char c2[])
{ int i = 0;
  do
  {c1 [i] = c2[i];
  }
  while ( c1[i++] )  != '\0');
 }
 void copy ( char *ps1,  char *ps2)
 {                          // алмастыруды меншіктеу амалы
                            //   талапты көрсету
  int i = 0;
  while ( ( ps1[i] = ps2[i] ) != '\0')
  i ++;
 }
```

Келтірілген мысалда '\0' салыстыру міндетті емес, себебі нөл мәнін меншіктегенде әрқашан **нақты** болып есептеледі. Сондықтан:

```
void copy ( char  * ps1 ,  char  * ps2)
{ int i =0;
  while ( ps1[i] = ps2[i] )
  i++;
 }
```

Соңында ең қысқа әдіс келтіреміз:

```
void copy ( char  *ps1, char  * ps2)
[ while ( *ps1++  =  *ps2++);
 }
```

copy () функциясын қолдану мысалы:

```
#include <stdio.h>
int main ( void)
{ char str[] = "My string";
  void copy ( char [], char []);          // функция прототипі
  char s[80];
  copy (s, str);                          // шақыру функциясы
  printf ( "\n% s" , s);
  return 0;
 }
```

Функция параметрі ретінде жолды қолданып, тағы бір мысалды қарастырайық. Функция жол ішіндегі мәліметті экранға тікелей және кері тізім арқылы шығару. Бұл жағдайда жолды функцияға қалай беру керектігін және оған жолдық объектіні ары қарай өңдеп, құру қажеттілігін қарастыру керек.

Жолды функцияға берудегі көрсеткіш синтаксисін қолданамыз: **void** Print_of_stroka **(const char** * s).

Функция тақырыбының төмендегі қасиеттерін қарастырамыз.

- функция ештеңе қайтармайды, **void** нені көрсетеді;
- s - жол көрсеткішінің параметрі болып келеді. Ол функция көрсеткішін шақырғанда жолдық аргументтің бірінші байтының адресін иемденеді,
- **const** модификаторы оқу режимінде тек жолдық түпнұсқа функциясының жұмысын анықтайды.

```
void Print_of_stroka ( const char *s)
{ char * term, * p,
short len, i; // жолдың түпнұсқа ұзындығын белгілеу үшін іздестіру
              // «үйін» локальды жолдық объектіні сақтау үшін керек
   len = strlen (s);
   term = (char *)malloc( sizeof(char) * (len + 1));
   strcpy (term , s);                  // локальды жолдық объектіні
                                       // жолдық түпнұсқаға көшіру
                          // p көрсеткішін жолдық объектіге адрестеу
   p = term;
 // жолдық объекті ішіндегі тізімін тікелей бағанаға шығару
   while ( *p )                        // жол біткенге дейін
   printf (" % c \n ",  *p ++ );
```

p++ формуласы екі амалды құрайды: '' екіге бөлу амалын тек көрсеткішке ғана қолданады және p көрсеткішіне жіктелеген сақтау аймағы құрамын қайтарады; '++' – постфиксті инкремент амалы – '*' амалынан кейін p көрсеткіші қолданады. Көрсеткіш пен арифметиканы өсу p көрсеткіші арқылы адресті өзгертуге мүмкіндік береді. Онда объект қысқа

санды типті байт үшін көрсеткішпен байланысады (бұл жағдайда **char**). Мұнда ол р көрсеткіш бірлікке ығыстырылады, яғни жолдық объектінің келесі оң жақ литеріне бекітіледі

1.7. Функцияға көрсеткіштер

Жоғарыда қарастырылғандардан көрсеткіш рөлі C/C++ тілдерінде өте зор екенін көрсетеді. Функцияға көрсеткіш келесі түрде анықталады:

Нәтиже типі (*көрсеткіш аты) (функция параметрлері),

Мұнда функция аты көрсетілмеген, ал функцияны шақыру функцияға көрсеткіш көмегімен жанама түрде өтеді.

Функцияға көрсеткішті анықтағаннан кейін оған функция адрестерін меншіктеуге болады, бірақ тек сигнатуралары, яғни параметрлерінің типі және тізімі, сондай-ақ функцияға көрсеткіш анықтағандағы сияқты мәндермен дәл беттесетін қайтарылған мән типі барын ғана болады.

Мысалы:

```
int f1( int, double);
double f2( double, int),
int ( *pf) (int, double);
        pf = f1;   // дұрыс
        pf = f2;   // қате, сигнатуралары әртүрлі
```

Көрініп тұрғандай, функция аты – бұл оның машиналық кодының басына тұрақты көрсеткіш.

Функцияны көрсеткіш бойынша шақыру әдеттегідей функция көмегімен жүзеге асырылады:

```
pf ( int_value, double_ value)
```

өйткені біз анықтаған функцияға көрсеткіш және функция аты бір ғана типке ие.

Функцияларға көрсеткіш бірдей кодпен әртүрлі функциялар шақыруға мүмкіндік береді. Мысалы:

```
#include < stdlib.h >
#include < conio.h >
        // функциялар прототиптері
 int   f1(int, double),
 int   f2(int, double);

  int main(void)
{
      int  ( *pf )( int, double) = NULL,
      int  k,
            pf = f1;
                k  = pf (2, 4.1);
            pf = f2,
                k  = pf (2, 4.1);
        getch();
              return 0;
}
          // f1 және f2 анықталуы

      int   f1(int a, double b)   { return  a * (int)b;}
      int   f2(int a, double b)   { return a + (int)b,}
```

1.7.1. Функцияның көрсеткіштер массивтері

Функцияға көрсеткіштер массивтері басқа объектілер массивтерінен анықталу форматтарымен ерекшеленеді

Тип (*массив аты [өлшемі])(параметрлер_спецификациясы);

Мысалы:

```
int ( *par[4] )( char);
```

мұндағы par – әрқайсысына программа мәтінінде анықталған функция адрестерін меншіктеуге болатын төрт функцияға көрсеткіштен құратын массив. Функцияға көрсеткіш анықтағанда инициалдайтын өрнек ретінде көрсеткішті анықтағандағы типтетідей функция адресі қолданылуы керек Массив тіл синтаксисіне сәйкес көрсеткіштер және функциялармен қатар туынды тш болып табылады. Функциялар

массивін құруға болмайды, бірақ функцияларға көрсеткіштер массивін анықтауға болады. Бұл үшін барлық мүмкін болатын өңдеу тарамдары бір типтегі функциялар (бірдей типтегі қайтарылған мән және параметрлердің бірдей спецификациясы) түрінде беріледі. Соңынан әрбір элементіне белгілі бір өңдеу функциясының адресі меншіктелетін көрсеткіштер массиві анықталады. Әрі қарай негізінде осы не басқа өңдеу функциясы таңдалуы тиіс шарт қалыптастырылады. Соңынан әрбір шартқа функцияға көрсеткіштер массиві элементіне қатынас орындалатын индекстің анық мәні сәйкес қойылады.

массив_аты [индекс] (нақтылы_параметрлер_тізімі);

ал сәйкес өңдеу функциясын шақыру келесі түрге ие:

(*массив_аты [индекс] (нақтылы_параметрлер_тізімі)),

Функцияларға көрсеткіштер массивін қолданудың баяндалған сұлбасын меню, дәлірек айтқанда, қолданушы меню көмегімен басқаратын программаларды ұйымдастыру үшін қолдану ыңғайлы.

Осындай әдістің ең жалпы жүзеге асыру сұлбасын келесі файлдарды өңдеу программасы көрсетеді:

```
#include < stdio.h >
#include < conio.h >
#include < stdlib.h >                    // exit() функциясы үшін
            //файлдарды өңдеу үшін функцияларды анықтау
void  sozd_f (char *name)
  {printf("\n Файл құру бойынша әрекет - %S",name);}
 void  del_f (char *name)
   {printf("\n,файлды жою бойынша әрекет - %S",name);}
 void  read_f(char *name)
   {printf("\n файлды оқу бойынша әрекет - %S",name);}
 void  mod_f(char *name)
   {printf("\n файлды өзгерту бойынша әрекет - %S",name);}
 void  close_f(char *name)
   {printf("\n файлды жабу бойынша әрекет - %S",name);
                          exit(0);
        }
```

```
typedef void (*MENU)(char*); //  MENU – бұл void(char*) типіндегі
                             //  функцияға көрсеткіштер типі
   //меню функцияларының адрестері кестесін инициалдау
  MENU actions[5] = {sozd_f, del_f, read_f, mod_f, close_f},
int main(void)
{
    int n;                              // n – меню пунктінің нөмірі
    char file_name[30];
            printf("\n 1 – Файл құру"),
            printf("\n 2 – Файлды жою");
            printf("\n 3 – Файлды оқу");
            printf("\n 4 – Файлды өзгерту");
            printf("\n 5 – Программадан шығу");
   while(1)
   {
     while(1)  //цикл дұрыс нөмірді енгізгенше орындалады.
       {
            printf("\n Меню пунктінің нөмірін енгізіңіз: ");
            scanf("%d",&n);
                if (n>=1 && n<=5) break;
printf("\n Меню пункті нөмірінде қате. Таңдауды қайталау!");
       }
            if (n! = 5)
               { printf("\n Өңдеу үшін файл атын енгіз: ");
                 scanf("% S", file_name);
               }

  //Осы функцияға көрсеткіш бойынша функцияны шақыру
               ( *actions [n-1] ) ( file_name );
   }
                 getch();      return 0;
}
```

1.7.2. Функцияға көрсеткіштер параметрлер ретінде

Функцияға көрсеткіштер функциялардың өздері өңдеу объектілері қызметін атқаруы тиіс боғанда С/С++ тілдерінің таптырмас құралы болып есептеледі. Функцияға көрсеткіштер формалдық параметрлер сиякты алдын ала анықталмаған басқа функцияны өңдеу әдісінің осы және басқа әдісін жүзеге асыратын функциялар құруға мүмкіндік береді. Функцияға көрсеткіштер қолдану ерекшеліктерін көрсету үшін функция параметрі ретінде f(x) функциясы түбірлерін есептеуді [A;B] локальдау аралығында EPS дәлдігімен қарастырамыз. Сандық әдісті келесі функция тақырыбы түрінде белгілейміз:

float root(функцияға_көрсеткіш, **float** A, **float** B, **float** EPS);

Тип атын – "функцияға көрсеткіш" енгіземіз:

typedef float (*pFunc)(**float**);

Енді pFunc типіндегі көрсеткіші бар берілген функция түбірін есептеу үшін функцияны анықтауға болады. Функция прототипі былай болады:

float root (pFunc F, **float** A, **float** B, **float** EPS);

root() функциясы көметімен $y = x^2 - 1$ функциясының түбірлерін [0,2] кесіндісінде анықтау үшін тестілік программаны қарастырамыз:

```
#include < stdio.h >
#include < stdlib.h >
#include < conio.h > // exit() функциясы үшін
                  //функцияға көрсеткіш типінің атын анықтау:
                   typedef float (*pFunc)(float);
            //түбірлерді есептеу үшін функция анықтау.
            float  root ( pFunc F, float A, float B, float EPS)
              {      float  x, y, c,  Fx, Fy, Fc;
                            X = A;   y = B,
          Fx= (*F)(x);          // функция мәні сол жақ шекарада
          Fy= (*F)(y);          // функция мәні оң жақ шекарада
            if (Fx* Fy> 0.0)
                  printf("\n Локальдау кесіндісі дұрыс емес");
                  exit( 1); // программаның апатпен аяқталуы
                }
```

```
        do
          {
            C = (y-x) / 2,
            Fc= (*F)(c);    //функция мәні кесінді ортасында
                  if ( Fc* Fx > 0)
                        { x = c;  Fx = Fc; }
                        else
                              { y = c; Fy = Fc;}
          }
        while ( Fc != 0 && y-x > EPS);
              return c;
    }

#include < math.h >
          // тестілік  y = x*x-1 – функциясын анықтау:
                float f (float x)
                { return x*x-1; }
      int main (void)
      {
        float root (pFunc, float, float, float);
        float result;
        result = root (f, 0.0, 2.0, 1e-5);
        printf("\n Тестілік функцияның түбірі = %f", result);
                getch();
                return 0;
      }
```

Программаның орындалу нәтижесі:

Тестілік функцияның түбірі = 1

1.7.3. Функцияға көрсеткіш функция қайтаратын мән ретінде

Меню ұйымдастырғанда вариантар саны мен сәйкес әрекеттер саны орындалу нүктесінде емес, ал «аралық» функцилда анықталған жағдайларда осы аралық функциядан

орындалуы тиіс белгілі бір функцияның адресін қайтару ыңғайлы. Бұл адресті функцияға көрсеткіш мәні түрінде қайтаруға болады.

Менюді ұйымдастырудың осындай ерекшеліктерін көрсететін программаны карастырамыз. Программада бірінші екі функция f1() және f2() мына түріндегі прототиппен:

```
int f(void):
```

берілген үш функция анықталған, яғни параметрлердің бос тізімі, ал қайтарылған мән **int** типінде

```
int ( *menu(void))(void):
```

прототипімен берілген үшінші функция menu() өз кезегінде параметрлердің бос тізімін және **int** типіндегі мән қайтаратын функцияға көрсеткіш мәнін қайтарады.

menu() функциясы қолданушыға менюдің екі бөлігін таңдауға мүмкіндік береді.

Меню бөліктеріне жоғарыда анықталған функциялар сәйкес келеді, олардың біріне көрсеткіш қайтарылған мән болып табылады. Бөлік нөмірін қате таңдағанда қайтарылған мән **NULL** болады.

Негізгі программада f1(), f2() функцияларының адрестері мәнін қабылдайтын pFunc көрсеткіші анықтаяған. Циклда menu() функциясына катынас орындалады және егер нәтиже **NULL**-ге тең болса, онда программа «соңын» баспаға шығарады және орындалу аяқталады. Кері жағдайда

```
int k = (*pFunc)(),
```

шақыруы адресі pFunc көрсеткішінің мәні болатын f1() немесе f2() функцияларының орындалуын қамтамасыз етеді.

```
#include < stdio.h >
#include < conio.h >
    int f1(void)
    { printf("\n Бірінші әрекет:");     return 1; }
    int f2 ( void)
    { printf("\n Екінші әрекет: ");   return 2;}
    int ( *menu (void)) (void)
     [    int n;                  // n – меню пунктінің нөмірі
                          // функцияға көрсеткіштер массиві:
```

```
int (*menu_items[] ) () = { f1, f2},
printf("\n Меню бөлігін таңдаңыз (1 немесе 2): ");
 scanf("% d", &n),
   if ( n < 3  &&  n > 0)
       return  menu_items[ n-1];
   else
       return  NULL,
 }
int   main(void),
{
    int ( *pFunc) (void);     // функцияға көрсеткіш
    int k;
    while (1)
     {  pFunc = menu ();            // менюге қатынас
       if ( pFunc = = NULL)
            {   printf("\n Соңы");
                    return;
             }
              // таңдалған функцияны шақыру
                k = ( *pFunc) ();
                   printf("\t k = % d", k);
        }
             getch();
             return 0;
}
```

Программаның орындалу нәтижесі

```
Меню бөлігін таңдаңыз (1 немесе 2):2
Екінші әрекет: k = 2
Меню бөлігін таңдаңыз (1 или 2): 1
Бірінші әрекет: k = 1
Меню бөлігін таңдаңыз (1 или 2):5
Соңы!
```

Көрсеткіштер массивін анықтауды **typedef** сипаттар тізімі енгізілген қосымша аттар көмегімен қамтамасыз етуге болады. Мысалы:

```
int ( *menu_items[] ) () = { f1, f2};
```

Былай көрсетуге болады:

```
typedef int ( *Menu_action) (void);
Menu_action  menu_items [] = {f1, f2};
```

Мұнда **typedef** Menu_action атының белгіленуін «**int** типіндегі мән қайтаратын бос параметрлері бар функцияға көрсеткіш» типі ретінде енгізеді.

1.8. Көрсеткіштермен программалаудың пайдалы мысалдары

Алдыңғы бөлімдерде көрсеткіштік типті, осы типтегі объектілерді анықтау синтаксисін, оларға қолданатын амалдарды жеткілікті үйрендік. Енді бірнеше маңызды мысалдар ретінде үш есеп қарастырамыз.

- Бекітілген үлеспен динамикалық жадтың бөлінуі (жадтың жаңа үлесін жиі бөліп алумен байланысты шығындарды азайту үшін).
- Кез келген өлшемдегі сандық екі өлшемді массивтердің динамикалық құрылуы (компиляция кезінде белгісіз).
- Массивті сұрыптау (жалпыланған программалау мысалы). Бірінші есептен бастаймыз

Массивтерді үйренгенде массив өлшемі туралы ақпарат жоқ болса, программа орындалуы кезінде жад бөлуге тура келеді. Бірақ мұндай тиімді жүйе де кейде жарамайды, егер элементтер саны туралы ақпарат бірден емес. ақырындап үлеспен түссе, оларды кәдімгі массивтердегідей шағын гүрде сақтау керек. Сондықтан жадты бекітілген үлкен үлестермен бөлеміз және түскен мәліметтерді бөлінген үлкен буферге тікелей орналастырамыз. Бұл буфердің әрбір толуында жад буферін қайта белгілеу керек, бұл жиі болады, алдыңғы буферден қандай бір бекітілген өлшемінен үлкен болады. Жадтың динамикалық бөлінуінің түйіршектелген әдісі туралы айтуға болады.

Қорытындысында программа тиімді бола түседі, өйткені артық жұмыстан құтылуға мүмкіндік туады. Міне сәйкес мысал:

```
#include < stdio.h >
#include < malloc.h >
#include < conio.h >
                    // глобалдық айнымалылар
int BigGrain = 12,
int CurrLimit = 0;
int CurrSize = 0;
bool FirstTime = true;
bool PlaceInBuffer (int **, int);
void FreeBuffer (int *);
   int main (void)
   {       int *pBuf = NULL;
    char answer;
    int value;
    bool ret,
    for ( , ; )
{ printf("Жаңа бүтін мәтіндер енгізуді қалайсыз ба?");
while (scanf ("%c", & answer), (answer! = 'n' && answer! = 'y')),
if ( answer == 'n') break;
printf ("\n Келесі мәндерді енгізіңіздер:");
scanf ("%d", &value);
ret = PlaceInBuffer (&pBuf, value);
if (ret == false) break;
 }
        FreeBuffer (pBuf);
        getch ();
        return 0,
   }
            bool PlaceInBuffer (int **p, int val)
{             static int *pCur = NULL;
if (FirstTime)
   {       *p = (int *)malloc(sizeof(int)*BigGrain);
      if (p == NULL)
```

```
            return false;
    CurrLimit = BigGrain:
    pCur = *p;
    FirstTime = false;
     }

if (CurrSize *size of(int) == CurrLimit)     // буфер толған
  (          CurrLimit += BigGrain;
        realloc((void*)(*p),sizeof(int)*CurrLimit);
        if (*p== NULL)
        return  false;  // жад бөліне алмайды
        pCur = *p + CurrSize;
  }
       *pCur ++ = val;
        CurrSiz + = l;
          return  true
}
   void FreeBuffer(int *q)
   {if (q! = NULL) free (q);}
```

Мұнда біз **realloc()** функциясын қолдандық, бөлінген жадтың автоматты босауымен бекітілген көрсеткіш бойынша жадтың қайталана бөлінуі, сондықтан ***free()*** функциясын шақыру қажеттілігі жоқ

Енді екінші – кез келген өлшемдегі бүтін екі өлшемді массивтің динамикалық құрылуы туралы практикалық есепке көшеміз. Егер тек бірінші өлшем бойынша белгісіз болса (бұл айнымалының мәні болуы мүмкін), ал екіншісі бойынша өлшем компиляция кезінде, онда бәрі жай ғана:

int (*A)[100] = **new int**[N][100],

Осылайша бүтін екі өлшемді N*100 массив динамикалық құрылады, элементтері кәдімгі A[i][j] өрнегімен адрестеледі. мұндағы i және j – массив элементінің индекстері. А массивінің аты **int(*)[100]** типіне ие.

Егер екі өлшем де компиляция кезінде белгісіз болса, онда екі көрсеткіш жиынын пайдалану талап етіледі. Біреуі **int**** типінде, ал екіншісі **int*** типіндегі. Шешімі:

```
#include < stdio.h >
#include < malloc.h >
#include < conio.h >
int main (void)
{     int i, j, nRow, nCol;
      int **A = NULL,
      int *p = NULL;
      printf ("Жолдар мен бағаналар санын енгізіңіз:");
      scanf ("%d % d", &nRow, &nCol);
            // матрица және оның барлық элементтері
       //  жолдарына көрсеткіштер үшін буфер
      A = (int **)malloc(sizeof(int *) * nRow);
      p = (int *)malloc(sizeof(int) * (nRow * nCol)),
            // жолдарға көрсеткіштерді инициалдау
      for (i = 0; i < nRow: i++) *(A + i) = p+i*nCol:
             // матрицаны инициализациялау
      for (i = 0; i < nRow; i++)
      for (j = 0; j < nCol; j++)        A[i][j] = i+j;
      for (i = 0; i < nRow; i++)
      for (j = 0; j < nCol; j++)
      printf("A[%d][%d] = %d\n", i, j, A[i][j]);  // жадтың босатылуы
       if (p! = NULL)        free(p);
       if (A! = NULL) free (A),
       getch();
       return 0;
}
```

Сандық массивтерді сұрыптау туралы ең белгілі және маңызды есепті қарастырамыз (өсу және кему бойынша). Сұрыптау алгоритмінен «көпіршік» (ағылшынша – bubble sort) әдісін таңдаймыз, ол сұрыптауды N-нан аспайтын өтулермен жасайды (N – массив элементтерінің саны).

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <conio.h>
#define N 10 // функция прототипі
void BubbleSort(void*,int,int,int((*)(void*,void*)));
int MyCompare (const void*, const void*);
void MySwap (void*, void*, int);
int main (void)
{  int A[N],
     for (int i = 0; i < N; i++) A[i] = rand();
     BubbleSort ((void*) A, N, sizeof(int), MyCompare);
        getch();
        return 0;
}
void BubbleSort(void*p, int ArrSize, int ElemSize,
int CompFun (void*, void*))
{  for(int i = 0; i < ArrSize; i++)
   for(int j = 1, j < ArrSize – i; j++)
   { void *p1 = ((char*)p + (j-1) * ElemSize),
   void *p2 = ((char*)p + j * ElemSize);
       if(CompFun(p1,p2)>0)
  MySwap(p1, p2, ElemSize);
   }
}

int MyCompare(const void* pElem1, const void* pElem2)
{  return (int)(*(int*) pElem1 - *(int*) pElem2), }
void MySwap (void *p1, void *p2, int Size)
{  void *p = malloc (sizeof (char) * Size);
        memcpy (p, p1, Size);
        memcpy (p1, p2, Size),
        memcpy (p2, p, Size);
          free (p);
}
```

Берілген программа «жалпыланған программалаумен», яғни жеткілікті әмбебап шеберлікте жобаланған. Бұл **void*** типіндегі көрсеткіштерді қолданумен қамтамасыз етіледі, яғни қайталанып қолданылатын программалық компоненттерді құруға мүмкіндік береді.

Нәтижесінде біз абсолютті әмбебап функциялар BubbleSort () және MySwap () аламыз, сондықтан олар әр түрлі типтегі массивтерді сұрыптау үшін қайта жасалмағаны жарамды. Сұрыпталатын массив мәліметтері анық типін «сезінуші» жалғыз функция MyCompare(), ол массивтің екі элементін шамасы бойынша салыстырады. Дәл осы функцияны мәліметтердің белгілі бір типіне бейімдестіру керек. BubbleSort () функциясының іске асырылу жан-жақтылығы массив элементтерін салыстыру функциясына көрсеткіш қолданумен де қамтамасыз етіледі. Сұрыптауды кему бойынша жүзеге асыру үшін MyCompare () функциясының қайтарылатын мәнінің белгісін өзгерту жеткілікті:

```
int MyCompare (void * pElem1, void * pElem2)
{ return – (int)(*(int*) pElem1 - *(int*) pElem2); }
```

2. КҮРДЕЛІ ҚҰРЫЛЫМНЫҢ КӨРСЕТКІШТІК ТИПІ ЖӘНЕ ДИНАМИКАЛЫҚ ОБЪЕКТІЛЕРІ

2.1. Қажетті мәліметтер

Оқу құралының бірінші бөлімінде біз сіздермен көрсеткіштік типті анықтау және оны қолдануын нетіздейтін сұрақтарды толығымен қарастырып, осы типтегі объектілерді программалау нетіздерін үйрендік. Белгілі болғандай, өңделетін алгоритм шешімі тапсырмаларының қарапайымдылығы мен сенімділігін қамтамасыз ету үшін шешу үрдісінде қолданылатын мәліметтердің сәйкес құрылымдарын таңдаудың үлкен мағынасы бар.

C/C++ тілдер жинағы әмбебаптығы және тиімділігі сонша, оның қорындағы мәліметтердің әр түрлі тілде анық түрде көрінбейтін құрылымдарын қолданысқа енгізуге және пайдалануға болады. Бұл қатынаста әсіресе, көрсеткіштік тип массивтермен және мәліметтердің комбинацияланған типтерімен үйлесімде көп мүмкіндіктер береді.

Оқу құралының екінші бөлімінде біз тапсырмалардың шешу алгоритмі қарапайым түрде құрылатын объектілерді көрсететін кез келген табиғаттың динамикалық құрылымдарын жасау жолдарын қарастырамыз.

Программалау тәжірибесінде кеңінен қолданылатын бұл динамикалық құрылымдар түрлеріне стектер, кезектер, әр түрлі түрдегі тізімді құрылымдар, ағаштар, т.б. жатады.

Динамикалық объектіде кем дегенде екі өрістен тұратын құрылымы болады, біріншісі – атрибуттардан құрылады, екіншісі – басқа динамикалық объект осы объектпен байланысты мәнін адрестеуге көрсеткіш болады. Динамикалық объект бір атпен әр түрлі мәліметтер жинағын құрайды және ***struct*** кілтті сөзімен құрылымдық тип болып анықталады. Енді мұндай динамикалық объектілерден стектер, кезектер, тізімдер, кестелер және т.б. күрделі динамикалық объектілерді құрастыруға болады.

Біз құрылымдық типті толығымен қарастырмаймыз, тек қана күрделі динамикалық объектілерді өңдеу тәсілдері және оны түсінуге қажетті сұрақтарды қарастырамыз.

Сонымен, *tagName* құрылымдық типін анықтауға мысал келтірейік:

```
struct tagName
  {     int k;
        char str [6];
        double d;
        int ar[2][3];
  },
```

С++-теті бұл типтің айнымалылары келесі түрде жарияланады:

```
tagName st1, st2;
```

ал Си тілінде міндетті түрде ***struct*** кілттік сөзін қайталау керек

```
struct tagName st1, st2;
```

typedef кілттік сөзін қолдану Си тіліндегі:

```
typedef struct tagName
```

құрылымдық тип түріндегі айнымалыларды жариялауды жеңілдетеді.

```
{           int k;
            char str [6];
            double d;
            int ar[2][3],
} STRUCTNAME;
```

Осыдан кейін құрылымдық айнымалыларды мына түрде анықтауға болады:

```
STRUCTNAME st1, st2
```

Құрылымдық айнымалыларды «мәніне қарай» нақты параметрлер түрінде функцияға жіберуге, яғни көшіруге болады. Басқа әдісі - tagName көрсеткіштік типін анықтау және оны st1 айнымалысымен инициалдау:

tagName *pst = & st1,

«Бағыт» амалы көрсеткіш арқылы құрылымдық айнымалы өрісіне қатынауды қамтамасыз етеді, мысалы: pst → k

«Нүкте» амалына да осындай өрнек қолдануға болады (* pst) . k

Бұл мысалда құрылымның құрамын «анықтайтын» pst көрсеткіштің үстінде '*' атын алып тастау амалын көреміз, ал '.' амалының приоритеті '*' амалына қарағанда жоғары болғандықтан, өрнекте дөңгелек жақшалардың болуы міндетті.

Мәліметтердің құрылымдық типі көрсеткішіне қарағанда басқа тип көрсеткіштерін қолданып, құрылымдық айнымалылар өрісіне «жақындауға» болады. **char** * көрсеткіш типін анықтайық, сонда инкремент амалы, оның мәні (яғни жад адресі) 1-ге жоғарылуын білдіреді. Осы көрсеткіш көметімен жадтың кез келген аралығын ретімен «жүгіртуге», ал сәйкестік проблемасын келтірілген мәліметтер типі көметімен шешуге болады:

char *p = (**char***) &st1;

Құрылымдық айнымалы көрсеткішін сыртқы функцияға жібергенде, біз осы функцияға көшірмесін емес, сол айнымалының өзін өзгертуге мүмкіндік береміз. Құрылым денесін анықтау үшін:

struct tagName

анықталатын құрылымның көрсеткіш типі бар өрістерді қолдануы мүмкін

```
{       ...
    tagName *p;
        },
```

2.2. Бірбайланысты тізімдер

Бірбайланысты тізім – жалпы белгілеудің тиімді динамикалық құрылымы. Бірбайланысты тізімдерді құру үшін C/C++ тілдерінде көрсеткіштер мен құрылымдар туралы бұрын алған білім қолданылады.

Бірбайланысты тізімдер, массивтер сияқты, ондағы мәндер жинағын сақтауға арналған Массивтерге қарағанда. бірбайланысты тізімдер ақпараттық элементтер гуралы ешқандай білімді қажет етпейді. Яғни олар өзінің табиғаты бойынша динамикалық болып табылады.

Бірбайланысты тізімдер әрқайсысында осы байланыстың ақпараттық мағынасы және келесі байланыс көрсеткіші сақталатын *«байланыстан»* құрылады (ағылшанша *nodes*). Шындығында, барлық түйіндер тізбегін өтуге болатын және содан басталатын ең бірінші түйін болуы керек. Осылайша, ондағы ақпараттық мәнді алу үшін кез келген түйінге жақындауға болады. Соңғы түйін өзінің «көрсеткіштік» өрісінің нөлдік мәнімен мінезделеді.

Бірбайланысты тізім құру – кейбір «пайдалы» мәнімен, мысалы, оның ақпараттық өрісіндегі тізімнің түйіндер саны туралы ақпаратпен, және (NULL) нөлдік «көрсеткіштік» мәнімен ең бірінші түйін құруды білдіреді.

Шынымен де, компьютер жадын босату арқылы түйіннің құрылған бірбайланысты тізімін өшіруге мүмкіндік болу керек

Енді бірбайланысты тізімді құрастыру бойынша қазірше өңделметен барлық міндетті жұмыстарды жүзеге асыратын программа мысалын келтірейік.

Тізім түйіні айнымалы құрылымдық типті көрсетеді.

```
struct Node
{  int val;              // ақпараттық мән
   Node  *pNode;  // келесі түйінге көрсеткіш
};
```

Тізімге жаңа түйіндерді енгізуді – Add() функциясы, тізім құрамын шығаруды – Print() функциясы, ал тізімді өшіруді – Remove () функциясы орындайды.

```
#include < stdio.h >
#include < conio.h >
#include < malloc.h >
        struct Node
          {    int val;
```

```
                Node * pNode;
            };
            // функция прототипі
        void Add (Node  **ppStart, int newVal);
        void Print (Node  * pStart);
        void Remove (Node  **ppStart);
            int main (void)
            {       int ret,
                Node  *pN = NULL,
                      Add (&pN, 5);
                      Add ( &pN, 6);
                      Add ( &pN, 7);
                      Print ( pN);
                      Remove (& pN);
                        getch ( );
                        return 0;
            )

void  Add (Node  **ppStart. int newVal)
 { Node  *p = *ppStart;
   if (p= = NULL) // тізімнің бірінші элементін құрамыз
     { *ppStart = (Node*)malloc (sizeof(Node)).
       ( *ppStart) → val = newVal;
       (*ppStart) → pNode = NULL,
        return;
     )
       while ( p != NULL)
           { if (p →  pNode = = NULL)  break;
             p = p → pNode;
           }
         p → pNode = (Node*)malloc (sizeof(Node));
        (p → pNode) → val =  newVal;
        (p → pNode) → pNode  = NULL;
 }
```

```
void Print ( Node  *pStart)
    {Node  *p = pStart;
     while (p != NULL)
     { printf ("Элемент = %d \n", p → val );
       p = p → pNode;
     )
    prinrf ("\n Тізім толығымен шықты! \n");
   }
void  Remove (Node  **ppStart)
  {
     Node *p;
     p = *ppStart;
     if ( *ppStart ==  NULL)          // тізім бос
     return;
while ( p→ pNode != NULL) // тізімнің соңғы элементі емес
{if(p →pNode)→pNode==NULL)   // соңғы екі элементі
                              // p көрсеткішімен адрестеледі
          {free (p → pNode);
           p → pNode = NULL:
             p = *ppStart;
             continue;
          )
             else
             p = p → pNode;
    }

  // соңғы элементті және көрсеткішті жоямыз
  //  тізім басына    NULL меншіктейміз
             free (* ppStart);
            *ppStart = NULL,
   )
```

Берілген программада Add() және Remove() функциясында «тізім басына көрсеткішті» жіберу, яғни функция стекте оның көшірмесін емес, көрсеткіштің өзін өзгерте алатындай мақсатпен жанама түрде болады.

Мысалы, осылайша Remove (& pN) функциясын шақырғаннан кейін тізім бос екенін көрсететін осы көрсеткіштің нөлдік мәнін алуға болады.

Жадпен динамикалық жұмыс үшін **malloc**() және **free**() функция жұптарының орнына осындай табыспен C++ тілінің **new/delete** амалдар жұбын қолдануға болады.

2.3. Жолдардың бірбайланысты тізім ретінде көрсетілуі

Әдеттегіден басқа «келесі элемент» ұғымы алдыңғы элементке қарағанда осы элементтің орналасуымен тығыз байланысты жолдың векторлық көрсетілімі деп аталатын тізім түріндегі (бірбайланысты, екі бағытты және шеңберлік) жол көрсетілімі бар. Элементтерді бір тізімге тізу байланыс тәсілдері көрсеткіш механизмдері арқылы жүзеге асады

Жолдармен жұмыс жасау техникасын көрсету үшін тізім түріндегі берілген келесі, литерлерден тұратын, құрамында нүктелері жоқ және нүктемен аяқталатын берілген бос емес сөзге «А» символының кіру сандарын санау мысалды қарастырайық.

Тізіммен жұмыс жасау техникасын жай көрсетуді қажет еткендіктен бізге тиімді бағдарламаны алу мақсат емес. Бұл үшін келесі сұлбаны қолданамыз:

1 Шығу сөзін енгізу және оны бірбайланысты тізім түрінде көрсету.

2 Берілген символдың сөзге кіру санын анықтау.

3. Нәтижелерді шығаруды енгізу.

Бағдарламада динамикалық объектімен жұмыс жасауға тура келеді, сондықтан, қолдануға келесі құрылымдық типті енгіземіз

```
struct Node
{    char lit;
     Node *next;
};
```

Тізімге бүтін тәрізді сілтеме үшін, ал нақтырақ айтатын болсақ, оның бірінші түйініне қолдануға сәйкес көрсеткішті енгіземіз

```
Node  *first;
```

Құрылып қойған жетектің соңғысына қосылу үшін жетектің құрылу үрдісіне қосымша көрсеткіш қажет. Оны жазба көметімен қолдануға енгіземіз.

```
Node *p;
```

p көрсеткішін қосып алу амалы көметімен жетек тізбегі бойынша өтуді ұйымдастыру үшін қолданамыз:

```
p = p → next;
```

Осындай динамикалық құрылымдарды ұйымдастырумен байланысты тағы бір ескерту. Тізімнің бастапқысынан кейінгі жетек туралы айтылып отыр: керек пе ол немесе онсыз да жұмыс істей алуы мүмкін бе.

Бастапқыдан кейінгі жетексіз программаның мәтінін келтіреміз:

```
#include < stdio.h >
#include < conio.h >
#include < malloc.h >
#define sum 'A'
  struct  Node
  {
        char  lit;
         Node *next,
  };

int main (void)
{  Node *first, *p; // first – тізімнің бірінші түйініне көрсеткіш
                    //  p  – жұмыстық көрсеткіш
char ch;
int  к;             // к – жолдағы ізделінді символдар саны
                    // бірінші түйіннің құрылуы
printf (" \n Символды енгізіңіз:  ");
 scanf ("%c", &ch);
```

```
        first = (Node*) malloc (sizeof (Node));
        first → lit = ch;
        first → next = NULL;
          // басқа түйіндердің құрылуын циклге дайындау
                p = first;
         // жолдар литерлерін бірінен соң бірі өңдеу циклі
    while (ch!='.')
      {  printf ("\n Сол символды енгізіңіз: ");
         scanf ("% c", &ch);
          p = p → next = (Node*) malloc (sizeof (Node));
          p → lit = ch;
          p → next = NULL,
      }
```

/* Бірбайланысты тізім түріндегі шығарылған сөз. Енді тізімде бар жетектерді шығарамыз */

```
                p = first;
                putchar('\n');
                 while (p)
                { printf("% c". p → lit);
                     p = p → next;
                }

                 // sum символының сөзге кіру санын есептеу
                  к = 0;  p = first;
                   while (p)
                   { if (p → lit = = sum) k++;
                      p = p→ next;
                    )
                     // нәтижені шығару

    printf("\n Символ % с жолға кіреді - %d рет", sum, k);
                getch( );
                return 0;
}
```

Енді бастапқыдан кейінгі тізімге бірінші жетекке сілтеме көрсететін *next* өрісіндегі «нөлдік» жетекті қосамыз, ал lit өрісін

жолдағы кейбір қосымша ақпараттарды сақтау үшін қолданамыз:

```
#include <stdio.h>
#include<conio.h>
#include<malloc.h>
#define sym 'A'
    struct Node
      {   char lit;
            Node * next;
      };
int main (void)
{
      Node  *first, *p,
          char ch;
          int к;
        // бастапқыдан кейінгі жетектің құрылуы
     first = (Node*)malloc(sizeof(Node)),
          p = first;
            p → next = NULL;
     printf("\n  Жолдың бірінші әрпін енгізіңіз:");
     scanf("%c", &ch);
// етер сөз бос емес болса, символды циклдік өңдеу
    while (ch !='.')
    {
          p = p → next = (Node*)malloc(sizeof(Node));
            p → lit = ch,
            p → next = NULL,
        printf ("\n Жолдың жаңағы әрпін енгізіңіз:");
        scanf("%c, &ch"),
    )                              // тізім мазмұнын шығару
```

/*Назар аударыңыз, бірінші нұсқадағы есеп шешімінде соңғы түйін нүкте түрінде құрылуы тиіс, ал екіншісінде ол жоқ болады */

```
putchar ('\n');
p = first → next; // p көрсеткішін бірінші түйінге адрестейміз
      while ( p)
```

```
{   printf ("% c", p → lit),
    p = p -> next:
} // берілген символға кіретін сөздер санын есептеу
k = 0;        p = first → next;
while (p)
      { if ( p → lit == sym) k++;
        p = p → next;
      }            // нәтижені шығару
printf("\n Символ % c  жолға кіреді % d раз", sym, к);
            getch ( ),
           return 0;
}
```

2.4. Жол-тізбек амалын жүзеге асыру

Бірбайланысты тізім түрінде көрсетілген жағдайдағы жолдардың жүзеге асырылуының негізгі амалдарын қарастырамыз. Әрі қарай барлық жерде кез келген жолдың (сондай-ақ бос жол) бас жетегі бар болады деп есептейміз (бастапқы жетектің next өрісіндегі бос жолында **NULL** бар).

Жолдағы берілген элементті іздеу

Осы амалды орындаудың алгоритмін функция жазбасы түрінде береміз, ол көрсетілген жолға берілген элементтің бірінші кіруін және тізімнің сәйкес жетегіне сілтемені (адресті) қайтарады. Мұндай функцияның жазбасы мына түрге ие

```
Node *search (char ch, Node *beg)
                { Node  *q = beg;
// q – функцияның локальді объектісі сияқты құрылымға көрсеткіш,
// оны тізімнің басына көрсеткіштің мәні түрінде адрестейміз,
// яғни  first көрсеткішінің мағынасында
                Node  *result = NULL;
                q = q → next;
                while (q)          // тізім әлі бос емес
                   { if (q → lit == ch)
```

```
                { result = q;
                    break;
                }
            q = q → next;
            }
        return  result;
    )
```

Берілген элементтің жолдан өшірілуі

Мұнда алдымен өшіруге қатысты жолдың элементін қалай беру керек екендігін шешіп алған жөн. Ең тиімді тәсіл болып – осы элемент жүретін сол жетектің көрсеткіші көметімен алып тастау элементін беру болып табылады. Біз бұл көрсеткішті іздеуден search () функциясы көмегімен табамыз. Тізімнен түйінді алып тастау механизмі қарапайым: көрсеткіштің мағынасын өзгертсе болғаны, яғни search () функциясының көмегімен табылатын **former** көрсеткіші. Тізімнен алып тасталынған элементтің жазба функциясы келесі түрге ие:

```
void  dispose (Node  *former )
{ if (former → next != NULL)  //   тексеру сілтеліне ме
                              // тізімнің соңғы түйініне көрсеткіш
{     Node *q;
      q = former → next;    // оның өшірілуі үшін q көрсеткішін
                            //  өшірілетін түйінге адрестейміз
      former → next = q → next,
      free(q);   // жадты босатамыз
)   )
```

Берілген элементті тізімге енгізу

Тізімге жаңа түйінді енгізу – көрсеткіштердің көмегімен бір тізбекке бөлек жетектерді біріктіруге нетізделеді. Жаңа жетектің механизмі төмендегідей:

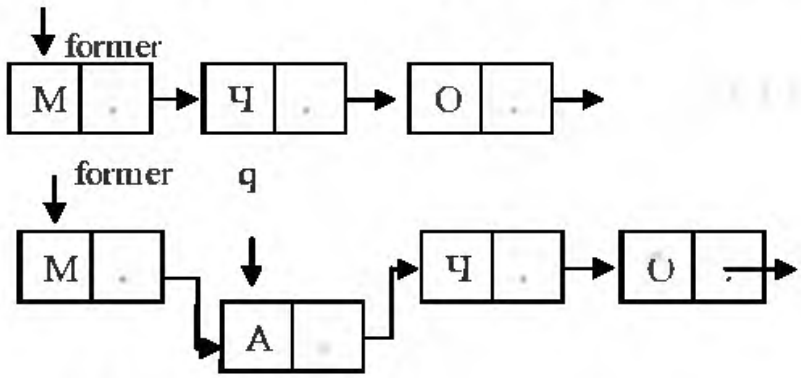
1. Қойылатын жетекті көрсетілетіндей жаңа динамикалық объектіні құру.

2. Осы объектінің *lit* өрісіне берілген символды енгізу.

3. Осы объектінің *next* өрісін сол жетектің *next* өрісі мағынасымен адрестейміз.

4. Қойылған жетек соңынан жүретін сол жетектің *next* өрісін қойылған жетекпен адрестейміз.

Сұлбалық тізім фрагментінде бұл келесі түрдегідей болады:



Осы түрде берілген элементтің сипаттамалық функциясын жолға қою келесі түрде болады:

```
void  insert (char ch, Node *former)
{
        Node *q = (Node *)malloc (sizeof (Node));
        q → lit = ch;
        q → next = former → next;
        former → next = q;
}
```

Қойылатын түйінді адрестейтін former көрсеткішін search () функциясы арқылы табылатынын ескертеміз. Оқу мысалында ішкі айнымалы позицияға түйінді қою функциясын қосуды және тізіммен жеке түйінді өшіруді қиындатқымыз келмеді. Бұл функцияларды жеке қарастырдық. Оқу құралының мысалын өз беттеріңізбен қарастырыңыздар, ал сосын негізгі программаға жетіспейтін функцияны қосыңыздар.

Тізім құру программасын қарастырамыз:

```
#include < stdio.h >
#include < conio.h >
 struct List
        {       int dat,
                List  *next;
        };
  int main (void)
   {     int i
        List *p, *heap = new List,              // тізім басы
        for ( p = heap, i = 0; i < 10, i++)
          { p → dat = i; // тізімнің келесі элементін енгіземіз
            p = p → next = new List;
                }
        for ( p = heap, i = 0; i < 10; i++)
        {
            printf ("%4d", p → dat );
            p = p → next;
        }
                getch ( ):
                return 0;
   }
```

Мұнда біз екі көрсеткішті: heap – тізімнің басын көрсету үшін, p – тізіммен жүру үшін және жай айнымалыны цикл есептшоты сияқты сипаттадық. Тізім массивке қарағанда, жад бойынша бір түйінге таратылған, яғни осы тізімге сақталған адрестерді қолдану арқылы біз тізім бойынша жылжитын болғандықтан, бұл ешқандай мағына бермейді

```
p = p → next;
```

Егер күрделі құрылымды өңдейтін сұрақтар қызығушылық тудырса, онда сізге кітаптармен танысуды ұсынамыз [6,7]

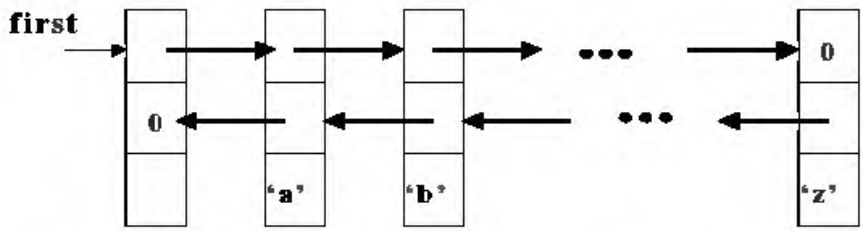
2.5. Екі бағыттық тізімдер

Тапсырмалар қатарын шешу кезінде бір бағытық тізімді қолдану кейбір қиындықтарды туғызады Біріншіден, бұл құрылымдарды өңдеу алгоритмінің қиындауына апаратын тек

бір жаққа ғана араластыру мүмкіндігі болып табылады. Бұл қиыншылықты алдыңғы жетекті адрестейтін жетекке тағы бір өрісті қосу жолымен жеңуге болады. Бұл жағдайда жетектің құрылымы келесі түрге ие:

```
struct Node
  {   char lit;          // ақпараттық элемент
      Node *pred;        // алдыңғы жетекке көрсеткіш
      Node *next,        // келесі жетекке көрсеткіш
  };
```

Осы жетектен тұратын келесі сұлба түрінде көрсетуге болатын динамикалық құрылымды екі бағытты тізім деп атайды (алдыңғы жетек тізімі):



Егер *next* жолының сапасы ретінде соңғы жетектің басты жетек адресін алатын болсақ, ал басты жетектің *pred* жол мәнінің сапасы ретінде соңғы адресті алатын болсақ, онда екі бағытты **сақиналы** тізім алынады.

Тізімдерде жолдардағы сияқты негізгі амалдар анықталған:

- Тізімнен элементті іздеу;
- Тізімде көрсетілген орынға берілген элементті қою;
- Тізімнен берілген элементті жою.

Басты жетекпен бірге екі бағытты сақиналы тізім қолданылатын осы амалдардың іске асуын қарастырайық.

Элементті қою

Элементті қою әрекетін insert() атымен функция жазбасы түрінде береміз, ол екі параметрден тұрады: оның біріншісі

қойылмалы элемент, ал екіншісі көрсеткіш former (алдыңғы), жаңа элементті қоюға болатын. мәні тізім жетегінің адресі болып табылады.

```
void  insert (char ch, Node  *former)
{
        Node  *q = (Node*) malloc (sizeof (Node));
        q  -> lit = ch,
        q → next = former → next;
        former → next = q;
         q → pred =  q → next → pred;
         q → next → pred = q;
}
```

insert() функциясының екінші нақты элементі сапасында тізімнің басында жаңа элементті қою үшін тізімнің бастапқы жетегіне көрсеткішті беру керек, яғни осы тізімге *first* көрсеткішін беру керек.

Элементті жою

Тізімнен элементті жою үшін next жолында және жетектің pred жолындағы адресте өзгерту жеткілікті. Тізіммен әр түрлі бағытта қозғалу мүмкіндігі осы звеноның өзіне көрсеткіш арқылы шығарылатын жетекті беруге мүмкіндік береді Шындығында, жетекті **free()** функциясының көмегімен жою керек.

```
void dispose (Node *pDel)
    {
  //алдыңғы шығарылатын жетекке next жолын меншіктейміз
  //  next  жолының мәні
      pDel → pred → next = pDel → next;
// келесі жетекке pred  жолын  pred жолының адресін меншіктейміз
//  жойылатын жетекке
      pDel → next → pred = pDel → pred;
      free (pDel);
}
```

Екі бағытты сақиналық тізімде элементті іздеу

Тізімде берілген элементті іздеу процедурасы тізбек жолдың ұқсастық процедурасынан айырмашылығы жоқ. Егер екі бағытты тізім сақиналы болса, онда ондай тізімде формальді түрде соңғы элементі жоқ, яғни соңғы элемент тізімнің бастапқы немесе тізімнің бірінші жетегі болатын «келесі» элементке сілтеме береді. Біздің қолданған әдістің арқасында, яғни бастапқы жетекпен, берілген элементті іздеу функциясы екі формальді параметрді қолданады: бастапқы жетекке көрсеткіш және шығарылатын элементтің мәні.

Функцияның нәтижесі қатар бойынша құрамында берілген элементі бар, бірінші жетекке көрсеткіш болып табылады.

Мұндай функцияны келесі түрде беруге болады:

```
Node *Search(char ch, Node *beg)
   { Node *q = beq; // локальді көрсеткішті көрсеткішпен адрестейміз
                    // тізімнің бастапқы элементіне
     Node *result = NULL;
     q = q → next;  // енді q – тізімнің бірінші элементінің адресі бар
     while (q)
           {if (q → lit == ch)
               {       result = q,
     break;  // жақсы нәтиже болған жағдайда циклден шығамыз
               }
              q = q -> next;
           }
                return result;
}
```

Барлық жетектерді бірдей өңдеу бастапқы жетекті қолдануға (тізімде жетектің бірінші және соңғыларын қоса санда) енгізу көмегімен жүзеге асатынына мән берейік. Барлық жоғарыда келтірілген функциялардың жазбасы тізімдегі бастапқы жетекті қолданады.

Енді қажетті тапсырманы шешу кезіндегі тізімдердің мысалын қарастырамыз.

Мысал. Соңы нүкте болып табылатын литер жолы берілген. Жолдан барлық сандарды жою қажет, әрбір **t** әрпінің алдындағы және нәтижеде жолды баспаға шығару керек.

Есепті шешудің келесі әдісі беріледі. Алдымен жолдың сыртқы мәтіндік файлда берілген литерлердің элементтері бар екі бағытты сақиналы тізім құрамыз, содан кейін есептің талаптарына сай осы құрылымның элементтерін өңдейміз, яғни **t** әрпімен әрбір алдыңдағы жетекте бар сандарды алып тастаймыз. Элементтердің өңделуін бірінші жетектен соңғы жетекке бірінен соң бірі таңдалуы кезінде жүзеге асыруға болады

```
#include < stdio.h >
#include < conio.h >
#include < malloc.h >
      struct Node
        { char lit,
          Node  *next,  *pred;
        };
                // функция прототипі
      void In_ List (char, Node *),    // тізімге жетектің қосылуы
      void  From_ List (Node *),       // тізімнен жетекті алып тастау
          int main (void)
            {     char ch;
                  Node *p,
       Node *ring;    // ring тізімдегі бастапқы жетекке көрсеткіш
       // екі бағытты сақиналы тізімнің құрылуы
      // тізімді құрылымның бастапқы жетегінің құрылуы
                p = (Node *)malloc(sizeof (Node)),
                p → next = p;
                p → pred = p,
                p → lit ='a';
                ring = p,
                printf (" \n Жолдың бірінші литерін енгізіңіз: "),
                scanf ("%c", &ch);
```

```
// Жолда литердің бірінен соң бірі оқылуы және олардың ring
тізіміне қосылуы
                while ( ch != '.')
                  {     In_ List (ch, ring → pred):
                        printf ("\n Келесі литерді енгізіңіз: ");
                        scanf("%c", &ch");
                   }
                // бірінші жетекке p көрсеткішін орнату
                   p = ring → next;
                // тізім элементтерінің бірінен соң бірінің өңделуі
                   while (p != ring)
                   {    // 't' әрпі бар тізімде келесі жетекті іздеу
                     while (p → lit ! = 't' && p ! = ring)
                           p = p → next;
if (p ! = ring)        // алдыңғы жетектерді санмен жою
     { while (p → pred → lit > = '0'&& p → pred → lit < = '9')
            From_List (p → pred);
          p = p → next;
     }
                   }
                        // нәтижені баспаға шығару
                   p = ring → next;
                   putchar ('\n');
                   while (p ! = ring)
                              {
                                   printf("%c", p → lit);
                                     p = p → next;
                                      putchar('\n');
                              }
                         return o;
           }
                    // InList() функциясын анықтау
              void  InList ( char ch, Node * former)
               {    Node * q = ( Node *) malloc ( sizeof ( Node)),
                         q -> lit = ch,
                           q -> next = former -> next;
```

```
                former -> next = q;
                    q - > pred = q -> next -> pred;
                      q -> next -> pred = q;
}
   // FromList() функциясының анықталуы
     void  FromList ( Node * pDel)
     {      pDel -> pred -> next = pDel -> next;
           pDel -> next -> pred = pDel -> pred;
                 free( pDel);
     }
```

Осы жерде құрылған алгоритм lit жолындағы бастапқы жетекте литерлер-әрш бар. Шындығында, мынандай жағдай, мысалы келесі литерлер тізбегі: 123t ...s45; яғни кез келген литер-цифр санымен басталатын және аяқталатын. Егер бұл жағдайда бастапқы жетекте литер-әріптер жоқ болса, онда алгоритм дұрыс нәтиже бермейді. Бұл талдаудың себебін оқырманға қалдырамыз.

2.6. Кезектер мен стектер

Программалауда *кезектер* деп аталатын мәліметтердің құрылымы бар. Мәліметтер құрылымы бар. Бұл мәліметтер құрылымы анықталған тапсырыстар заңдылығы мен оларға қызмет көрсету тәртібінде есептерді модельдеуге арналған. Кезектер уақыт ағыны мен кезек ұзындығы бар динамнкалық объект болып табылады, оның құрамындағы элементтер жиыны өзгеріп отырады.

Кезектерге екі амал қолданылады: элементті енгізу (қызмет көрсетуге тапсырыс) және кезектен элементті таңдау. Бұл жағдайда элемент кезектен алынып тасталады. Кезекте екі позиция болады: оның басы (бұл позициядан элемент кезектен алынады) және соңы (кезекке енгізілетін элемент позицияға орналастырылады).

Қызмет көрсетудің екі шарты болады:

FIFO – (first in-first out, яғни бірінші кезекке – бірінші кезектен);

LIFO – (last in-first out, яғни соңғы кезекке – соңғы кезектен).

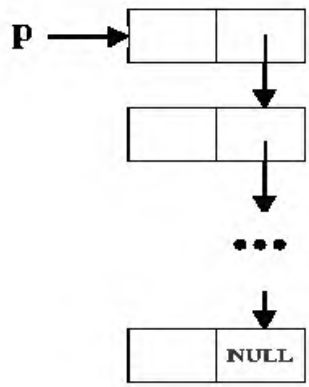
Программалауда көбінесе екінші тәртіп қолданылады, ал мұндай кезектер *стек* деп аталады. Стектер есептер шығару барысында өте қолайлы және ең көп қолданылатын мәліметтер құрылымы болып табылады.

Стек төбесі – стекке орналасқан ең соңғы элементтің позициясы. Элементті тек стек төбесінен ғана таңдауға болады, бұл кезде элемент стектен алынып тасталады, ал оның төбесінде соңғы элементтің алдында енгізілген элемент орналасады.

C++ тілінің құрылымы стектерге амалдар қолдануды тез іске асыратын құрылым буындарының динамнкалық тізбегі болып табылады. Бағдарламада стекті бір бүтін объект деп қарастыратын көрсеткіштің шамасы тек стек төбесін ғана анықтайды. Әдеттегі тізбек жағдайында стек буынының әрқайсысы тізбектің келесі буынына сілтеме жасайды. Стек түбі – стекке енгізілген ең алғашқы элемент **NULL** сілтеме шамасына тең. Стек типі элементін келесі түрде береміз:

```
struct Stack
 {
        int val;
        Stack * next;
 };
```

Сұлбалық түрде p стегін мына түрде көрсетеміз:



Егер стек бос болса, онда **NULL** сілтемесінің p көрсеткішінің мәні болады. Стекті қолданбас бұрын. оны бірінші босатып алу керек.

p = **NULL**;

Стекті іске асыру үшін негізгі амалды қарастыру қажет.

Стекке элементті енгізу

Есепті шығару барысында әр түрлі бірнеше стек қолданылады, мысалы мынадай прграмманың объектілері:

Stack *st1, *st2;

Сондықтан. стекке енгізу функциясының екі параметрі болуы қажет: оның бір параметрі керекті стекті береді, ол келесі оған енгізілетін мәнді береді

Стек элементін таңдау

Бұл амалды орындау барысында стектің элемент төбесі белгілі бір айнымалы ретінде мәнге ие болады, бұл элемент көрсетілген буында стекке тиым салу қажет.

Стекті тексеру үшін программада алдын ала стектің дұрыстығын бақылау қажет. Стектен элементті таңдау амалын іске асыратын функцияны сипаттаймыз.

```
int FromStack (Stack *p)
        {
            Stack  *q,
                int k;
              // төбеден элементті таңдау
            k  = p->val;
              // ескі төбеге сілтемені есте сақтау
             q  = p,
           // қолданылған жетекті жою
                  p =  p  -> next;
              free(q):
                return k;
        }
```

Стекпен жұмысты жай массив ретінде көрсетеміз:

```
const int  N = 10;
            int  stack[N];
            int top;                          // стек төбесінің индексі
                 // Стекті толтыру функциясы:
          void  push( )
           {
                 int k;
                 for (top = -1; top  <  N-1;)
                  {
                     printf("Стектің келесі мәнін енгізіңіз – ");
                        scanf("%d", &k),
                              Stack [++ top] = k;
                  }
           }
```

push() функциясы орындалғаннан кейін top айнымалысының шамасы N-1 болады. Top айнымалысы массивтің соңғы элементіне сілтеме жасайды. бұл **LIFO** тәртібіне сәйкес болады.

Стектен элементті алу функциясы

```
int pop ( )  { return Stack [ top--]; }
```

Мысалы:

```
for ( i = 0; i<N, i++)
  printf ("% d \n", pop ());
```

push() әдісі top индексін қолдана отырып, әрқашан стектің ең жоғарғы мәнін анықтайтын бір бүтін сан енгізеді. pop() әдісі стектен санды алып тастап, top индексінің мәнін бірге қысқартуға мүмкіндік береді. Осылайша стекте мәліметтерді – «бірінші келіп – соңынан шықты» сақтау принципі іске асырылады

2.7. Кестелер

Ақпараттық технологияларда көрсетілетін қызметтің ең көп тараған түрі ақпараттық–анықтамалық қызмет, онда мәліметтер, жаңа мәліметтерді қабылдау және сақтаулы мәлметтерді беру. Сақтаулы мәліметтер жазбаша түрде беріледі. Ақпараттық-анықтамалық жүйенің амалы – сұралған ақпаратты беру және іздеу. Осындай қиындықты болдырмау үшін деректер құрылымы, яғни кестелер әр жазылымда арнайы атпен сақталады. Керекті жазбаны қарау үшін тек оның атын көрсету керек, ал деректер құрылымы аты көрсетілген құжатты тез іздестіру нұсқауын өзі қамтамасыз етеді

Сонымен, кесте дегеніміз (жалпы айтқанда реттелмеген) аты көрсетілген бірнеше жазбалар жиынтығы. Жазбалар аты ерікті түрде ғаңдалуы жеткілікті. Бірақ та, жазбаны атымен іздеуге бұйрық бергенде, екі жазбаның салыстырмалы мүмкіндігін беру керек, яғни олардың бірі «кіші», ал екіншісі «үлкен». Сондықтан, жазбалар әр түрлі атпен сақталады. Жазбалар аты кестеде көп жағдайда оның кілті ретінде қолданылады.

Кілт ретінде көбінесе оң бүтін сандар қолданылады. Сондай-ақ, бірдей ұзындықтағы литер жолдарын қолданған ыңғайлы, өйткені бұл жолдар бір жағынан жазбаларға ыңғайлы аттар беруге мүмкіндік береді, ал екінші жағынан олардың үстінен салыстыру амалы анықталған. Осылайша, әрбір жазба кестеге еніп отырады және олар өзінің кілтін, сонымен қатар өзіне қатысты ақпаратты алып жүреді (жазу мәтіні).

Кесте үстінен ақпарат қатары ретінде келесі амалдар анықталған:

– Берілген кілтпен ақпаратты жазба кестесінен іздеу;

– Берілген кілтпен жазба кестеге енгізу (егер кестеде осы кілт атымен ақпарат болса, онда оны жаңа жазбаға өзгерту деген сөз);

– Берілген кілтпен ақпаратты жазба кестесіне өзгерту.

Кестені құрудың әр түрлі жолдары бар, олардың әрқайсысының өзінің артықшылықтары мен кемшіліктері бар. қай жолды таңдасаңыз да негізгі кестеге қарау керек.

2.7.1. Кестелердің берілу тәсілдері

Кестені көрсетудің ең оңай жолы бір бағытты тізім әдісі қолданылады, бұл әдісті алдыңғы жолда қарастырдық. Әрбір жегек гізбек, ол тізімді құрайды және жазба кілтін келесі жегекке өту жолын бағыттайды. Кесте көрсету әдісінің өзіндік ерекшелігі бар, бірақ онда ақпаратты іздеу кезінде кемшіліктер бар. Шынымен, ақпаратты іздеу кезінде жетектің барлық тізімін өзіңізге керекті ақпарат кілтін тапқанша тексеру керек.

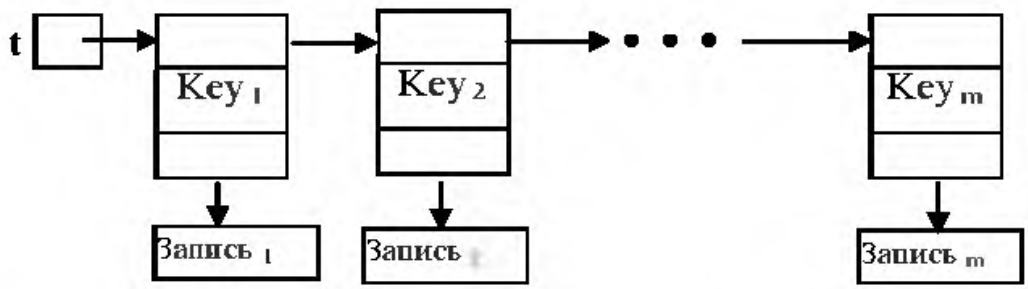
Қарастырылып отырған тәсілдің келесі кемшілігі мынада, егер осы кестеде берілген кілтпен ақпарат болмаса, онда осы фактіні орнату үшін бүкіл тізімді теру керек, сол кезде әрбір тізім басына жаңа ақпараттар енгізіледі, сондықтан ақпараттар тізімде дұрыс тіркеледі.

Кестелердегі жай тізім ретіндегі екінші кемшілікті жою үшін, оларды белгілі бір реттегі тізім регінде сақтауға болады Мысалы, олардың кілттерінің өсуі бойынша. Онда кестеге жаңа ақпаратты енгізу қиындап кегеді. Белгілі кестеге шығындарды сүйемелдеу үрдісі керек, өйткені кестеге жаңа ақпаратты енгізу ұзақ уақытты алатын болады Бірақ бұл шығындар өзін ақтайды, егер ақпаратты енгізу салыстырмалы, ал жоқ ақпаратты іздеу кәдімгі түрде болады.

2.7.2. Кестеде бинарлық іздеу

Кестеде ақпаратты іздеу процедурасын жылдамдату үшін келесі қабылдауды қолдануға болады. Ең алдымен ақпаратты кілтсіз сақтауға болатынын айтып кеткен жөн, ал кілтпен сақтағанда мәтінге бағытталуын ескеру керек. Бұл жағдайда кестені келесі сұлба регінде көрсетуге болады (key, дегеніміз – кілт, ал і – жазба)

t



Мұнда барлық элементтер бірдей мағынаны береді, ал жазба мәтіндері әр түрлі ұзындықты және құрылымды алуы мүмкін. Тізімнің әрбір элементі жазба болып табылады, олар екі жолды құрайды: олардың бірінде жазба кілті сақталады, ал екіншісінде мәтін жазбасына сілтенетін бағыт сақталады.

Өйткені, барлық элементтер бір типті, сондықтан оларды тізім ретінде емес, бір өлшемді массив ретінде біріктіруге болады (вектор). Осыған қатысты жазбаны енгізейік:

```
struct  Record
  {                          // кілт ретінде қолданамыз
   int  key;                 // бүтін сандар
   char *text;               // мәтінді жазбаға бағыттағыш
    Record   T[N];           // N элементтен тұратын массив
  };
```

Массив компоненттері өздерінде сақталған кілттердің өсуі бойынша реттелген, яғни **T[i]** key өзгеруі бойынша деп атаймыз. Бұл жағдай берілген кілт бойынша ақпаратты іздеуге мүмкіндік береді, мұндай іздеу дихотомикалық іздеу деп аталады. Бұл тәсілдің идеясы осы кілт тұрған ақпараттарды іздеуге бағытталып, T массивінің элементтерін табуға негізделген, егер осы элементтің индексі табылса, онда изделініп отырған жазба T[i] text айнымалысының мәні болып табылады.

K кілті берілген массивтің компоненттерін іздеу кезінде алдымен іздеу аймағы барлық векторлар болып табылады. Индексі **i** = N/2 болып келетін орта компонентін аламыз. Егер **k**

= T[i].key болса, онда іздеу ақталынады және T[i].text айнымалысы ізделінегін жазба болып табылады (T[i].text айнымалысының мағынасы символды массив көрсеткіші. Мысалы, келесі әдіспен шығаруға болатын: **printf**("%s", T[i].text)). Егер k != T[i].key болса, онда іздеу үрдісі жалғасады. k < T[i].key кезінде керекті компонентті массивтің сол жақ жартысынан іздеу керек, ал қарсы жағдайда – оң жақ. Демек, әрбір қадамда іздеу аймағы екі есе кішірейеді. Ағымдағы іздеу аймағының ортаңғы компоненті берілген кілттен тұрған жағдайда немесе іздеу зонасы бос болған жағдайда, яғни кестеде бұл кілттің жазбасының жоқтығын білдіреді, осы кезде үрдіс аяқталады.

Іздеу амалын жүзеге асыру үшін берілген кілт бойынша жазбаның дихотомикалық іздеуі – DXTM_search() функциясын қолдануға енгіземіз. Функция берілген K кілті немесе сәтсіз іздеу кезінде нөл болатын массив компонентінің индексін қайтарады.

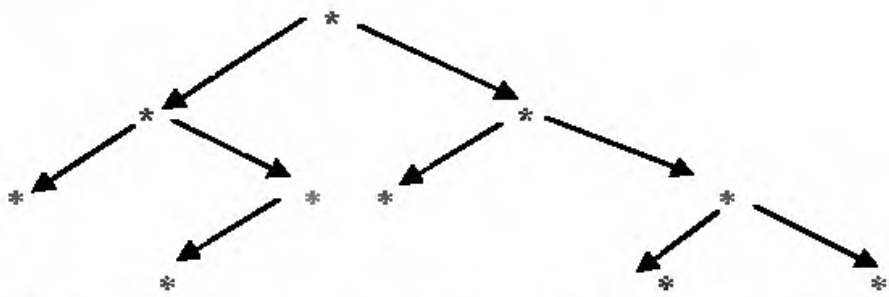
```
        int DXTM_search( int k)
{   int left, right;  // іздеудің сол жақ және оң жақ шекаралары
        int i, yes;                    // yes = 1 – жазба табылды
        left = 1; right = N;
            yes = 0;          // k – кілтті жазбалар массивте жоқ
          do
            {   i = ( left + right) / 2;
                  if ( k = T[i].key)
                        { yes = 1; break; }
                    else if ( k < T[i].key)  right = i – 1;
                            else  left = i + 1;
            }
          while ( left <= right);
                if ( yes)  return i;
                    else  return 0;
        }
```

2.7.3. Екілік ағаш

Жоғарыда келтірілген кесте көрінісінің әдісін мынадай жағдайда тиімді пайдаланамыз. кесте салыстырмалы аз түрде өзгергенде. онда көбіне іздеу жүзеге асады, себебі іздеу жылдамдығы массив компонентінің реттілігі есебінен қамтамасыз етіледі.

Кестенің үш амалын бірдей тиімді жүзеге асыра алатын, бұл тиімділік дихотомикалық іздеу кезіндегі тиімділікке өте жақын, екілік ағаш түріндегі кесте көрінісін қарастырамыз.

Екілік ағашта сұлбалық түрде анықтауға болады: бағдаршамен қосылған төбелер жиыны бар. Әрбір төбеден солға-төмен немесе оңға-төмен бағытталған екі жебеден артық жебе шықпайды. Ағаш түбірі деп аталатын ешқандай жебе кірмейтін бір ғана төбе болады. Қалған әрбір төбеге бір жебе кіреді.



Кестенің екілік ағаш түріндегі көрінісі кезінде жазба мәтіні бөлек сақталынады деп есептейміз. Кесте элементіне сәйкес келетін әрбір жегек (ағаш төбесі) жазба кілті сәйкес келетін 4 өрістен тұратын, көрсеткіштері солға-төмен, оңға-төмен төбелері және жазба көрсегкіштері мәтін болатын жазба болып табылады.

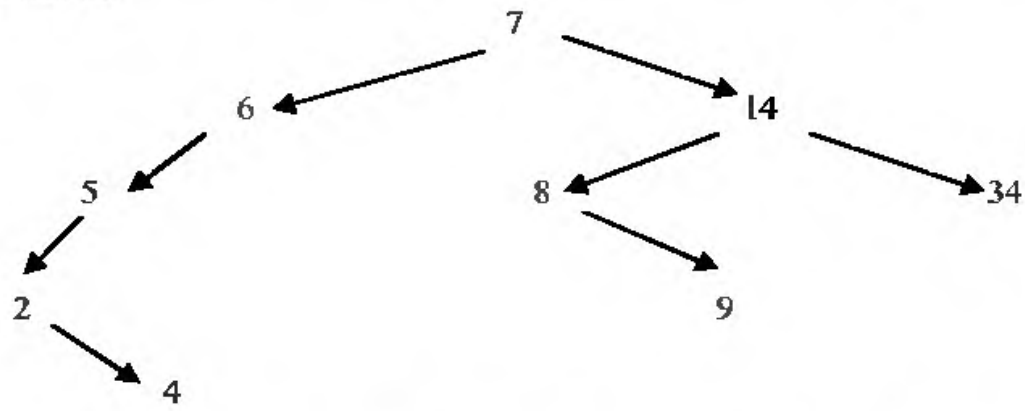
Кестеге жазбаны енгізу кезіндегі екілік ағашты тұрғызу принципі төмендегідей. 7, 6, 14, 5, 2, 8, 4, 9, 34 кілті бар жазба тізбегі бар болсын. K_1 = 7 кілті бар бірінші жазбаны ағаш түбірі деп аламыз, сонымен қатар көрші төбелердің көрсеткіш мәндері NULL-ға тең.

NULL	7	K_1	NULL

Егер келесі жазбаның K_2 кілті K_1 кілтінен кіші болса, онда бұл жазбаны сол жақ төбеге сәйкес қоямыз. Біздің жағдайымызда $K_2 = 6 < K_1 = 7$, сондықтан келесі қалыптасатын ағаштың төбесін түбір үшін сол жақ етеміз және тағы да солай қайталаймыз. Егер келесі жазбаның кілті алдыңғыдан үлкен болса, онда бұл жазба үшін сәйкесті түрде оң жақ төбені қоямыз және тағы да солай қайталаймыз.

Сонымен, егер де К кілтімен кезектегі жазба келіп түссе, онда – бірнеше рет ағаштың түбірінен бастап – К кілтін кезектегі төбе кілтімен салыстыра отырып, К кілтімен берілген жаңа төбені қосуға болатын кезекті төбені тапқанынша одан солға немесе оңға қарай жүреміз.

Біздің жазба тізбектері үшін екілік ағаш келесі түрге ие болады:



Енді қажетті екілік ағаш кестесін көрсетуді енгізуді қарастырайық. Ағаштың әрбір элементі жазба болып табылады: жазба кілті, ағаштың сол және оң жақ төбелеріне көрсеткіштер, яғни ағаштың осындай элементтеріне, сондай-ақ жазба мәтініне көрсеткіш. Екілік ағашты қолдануға келесі сипаттамалар көмегімен енгіземіз:

```
struct   Node
{       int  key;
        Node * left ,  * right,
         char  * text,
 };
        Node  * tree;
```

Ағашта жазбаны іздеу

Берілген кілтпен ағаштың төбесін іздеуді жүзеге асыратын функцияның сипаттамасын берейік: функция табылған төбеге көрсеткішті қайтарады (егер іздеу сәтті болса), кері жағдайда **NULL**-ді қайтарады. Функцияның парамегрлері болып іздеу жүргізіліп отырған ағаш және іздеу кілті табылады.

```
Node * Search_in_tree ( int k, const Node * t)
 {  Node *p = t;   // p көрсегкішін бұтақтың түпкі төбесіне апару
   int   yes = 0;        //  yes – іздеу сәттілігінің флагы
    if ( p)              //   ағаш бос емес
     do
       {  if ( p -> key == k)                    // төбе табылды
          { yes = 1. break; }
          else
            if ( k < p -> key)  p = p -> left;     // сол жақ төбеге өтеміз
              else  p =  p -> right;               // оң жақ төбеге өтеміз
 }
   while ( p);
   if ( yes )   return p;
                 else  return   NULL;      // К кілтті төбе жоқ
    }
```

Іздеу үрдісінің ұзақтылығы (осы мақсатқа таңдап алынатын төбе саны) оның ағаш құрылымына байланысты болады. Бұтақ «баланстalған» кезде ғана барынша тиімді қолданылады, яғни оның барлық төбелерінің соңғы төбесінен басқасы – оң және сол жақ қабылдағыштары бар. Бұл кезде іздеу үрдісі дихотомия кезіндегідей тез орындалады.

Жазбаны ағашқа қосу

Кестеде қосылатын жазбадағыдай кілтпен жазылған жазба жоқ болған кезді қарастырайық. Кестеге жаңа жазба енгізу үшін алдымен ағаштан қосылатын жазбаның жаңа төбесін «іліп қоюға болатын» (қосуға болатын) төбені табуымыз керек. Бұл төбе іздеуді жалғастыратын ағаштың анықталатын тармақ көрсеткішінің мағынасы **NULL** алынған кезде ғана табылады. Бұл мақсат үшін жоғарыда сипатталынған Search_in_tree() функциясын іздеу кейінгі ағаш төбесі көрсеткіші бекітілетіндей етіп модельдейміз.

```
Node * Search ( int k, Node * t)
        {   Node  * p, result;
                p = t,
                  if ( p)                    // ағаш бос емес
                    do
                      {  result = p;
                          if ( k < p -> key)  p = p -> left;
                            else   p = p -> right;
                      }
                    while ( p); return   result;
        }
```

Мұндай кестеге жазбаны қосу үрдісі функциясын анықтайтын екілік ағаш түрінде сипатталған программа мынадай қарапайым түрде жазылады:

```
void  Insert_to_tree ( int k, Node *t, char *s)
  {  Node * p, * result,
     int len;
        result = Search (k,t);  // result – s жазбасымен жаңа төбенің
                                // t бұтағының төбесіне көрсеткіш
                                //  ағашқа жаңа төбенің қалыптасуы
                  p = new Node,
                        p -> key = k;
```

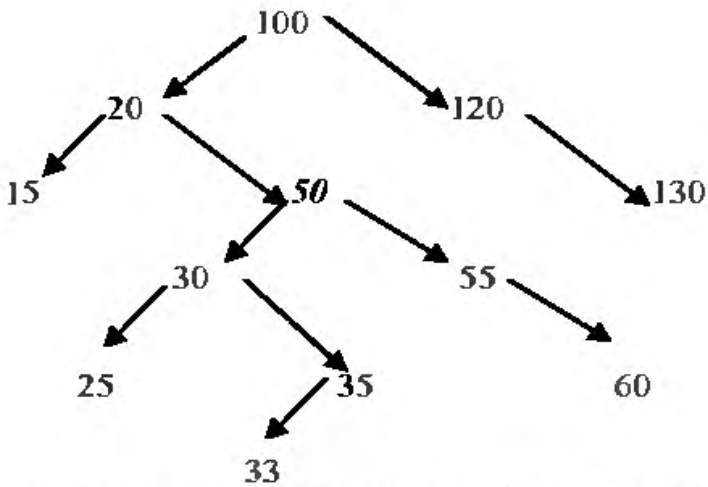
```
        p -> left = NULL;   p -> right = NULL;
                                  // жазба мәтінін кестеге енгізу
                        len = strlen (s);
                    p -> text = new char [ len + 1];
                    strcpy ( p -> text, s);
    if ( t == NULL)    // егер ағаш бос болса, онда құрылған
                       // жетекті ағаш төбесі деп аламыз
                t = p;
                        else          // ағашқа жаңа төбені қосу
                if ( k < result -> key)   result -> left = p;
                                  else   result -> right = p;
}
```

Кестеге жаңа жазбаны қосу үшін ең алдымен ағаштан жаңа төбені «ілуге», яғни қосуға болатын қосылатын жазбаға сәйкес төбені табу керек. Бұл төбе іздеуді жалғастырылуы керек ағаштың тамағына анықтайтын көрсегкіштің келесі мәні ретінде белгілі бір мән пайда болған кезде ғана табылады. Осы мақсатта іздеу біткендегі өңдеуден кейінгі көрсеткіш сол ағаш төбесіне тіркелуі үшін жоғарыда сипатталған функцияларды модификациялаймыз.

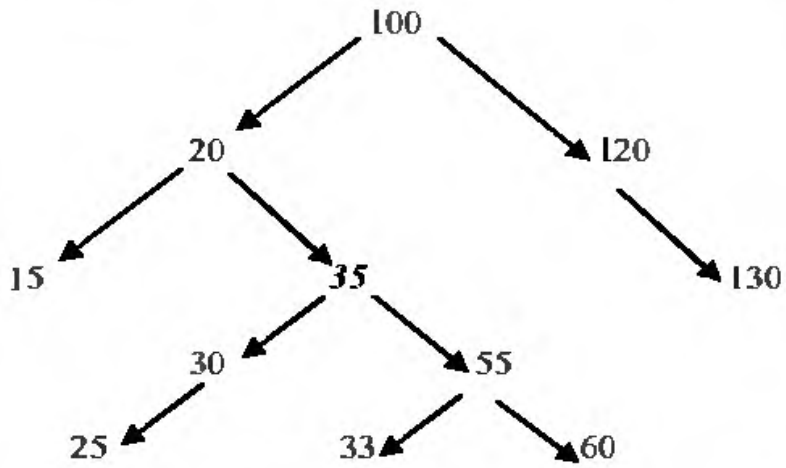
Ағаштан жазбаны өшіру

Енді екілік берілген кілті бар жетек ағаштан өшіру процедурасын сипаттауға көшейік. Егер төбесі соңғы болса, немесе одан бір ғана тармақ шығатын болса – ол үшін сәйкес сілтемені (адресті) алдыңғы төбеден түзетсе ғана болғаны Негізгі қиындық өшірілетін шыңға бір бағыт кіріп, екі бағыт шығатындықтын, екі тармақ шығатын төбені өшіру болып табылады. Бұл кезде осы сәйкес жетек ауыса алатын, өшірілген орынға қоюға болатын сәйкес ағаш жегекті іздеуіміз керек Бұндай жегек әрқашанда болады: бұл сол ағаш астының ең оң жағындағы элементі болуы мүмкін (бұл жетекті алу үшін сол тармақтағы келесі төбесіне көшуіміз керек, кейін оң жақ тармақтағы келесі төбеге өтуіміз керек, бұл сілтеме NULL-ға тең болмайынша қайталанады) немесе оң тармақтың ең сол

жағындағы элемент болуы мүмкін (бұл жегекті алу үшін оң тармақтағы келесі төбеге көшіп. сол жақ тармақтағы келесі төбесіне өтуіміз керек, бұл сілтеме **NULL**-ға тең болмайынша қайталанады). Мұндай сәйкес жетектердің тек бір ғана тармағы болады. Екілік ағаштан екі бағыт шығатын. К = 50 кілті бар төбені алып тастау сұлбасын қарастырайық:



К = 50 кілті бар жетек өшірілгенге дейінгі ағаш көрінісі.



К = 50 кілтімен төбені өшіргеннен кейінгі ағаштың түрі.

Екі мәнді ағаш ұяшықтарымен берілген кілттер процедурасы үш түрлі жағдайда болады:

1. кілті бар ұяшықтар ағашта жоқ;
2. кілті бар ұяшықтар біреуден артық емес бұтағы бар;
3. кілті бар ұяшықтың екі бұтағы бар.

Бұл мысалда кілті бар төбені алып тастау функциясын қарастырамыз, алгоритмнің авторы Н. Вирт.

```
Node *p;                    // қосымша айнымалы – төбеге көрсегкіш
void Delete ( Node * pointer);    // рекурсивті функцияның прототипі
void  Delete_from_tree ( int k, Node * t)
{                    // t бұтағынан k кілті бар төбені өшіру
    if ( t == NULL)  // алып тастау процедурасының бірінші жағдайы
        puts ( " \n берілген кілтпен төбе табылған жоқ");
          else                  // берілген кілтпен төбені іздеу
            if ( k < t -> key)  Delete_from_tree ( k, t -> left),
              else if ( k > t -> key)  Delete_from_tree ( k, t -> right);
                  else
                    {       // төбе табылды, оны өшіру керек
          p = t, // алып тастау процедурасының екінші жағдайы
                    if ( p -> right == NULL)      t = p -> left;
                      else if ( p -> left == NULL)    t = p -> right;
                        else  // алып тастаудың үшінші жағдайы
                            Delete ( p -> left );
                    }
      }
//рекурсивті функция алып тастау процедурасының 3-жағдайында шақырылады
          void  Delete ( Node * pointer)
          {     if ( pointer -> right == NULL)
                  {   p -> key = pointer -> key;
                      p -> text = pointer -> text;
                        p = pointer ;
                          pointer = pointer -> left;
                  }
                    else  Delete ( pointer -> right);
          }
```

Delete() – қосымша рекурсивті функция *P элементін өшіру оң төбешік-тен сол жақтағы ағаш астына дейін «түседі», key.

text жолдарындағы *p мәнін ауыстырып, сәйкесінше key, text жолдарындағы *pointer төбесіне ауысады. Pointer төбесіне бағытталған көрсеткішті өшіруге болады. free() стандартты функциясын қолданып, өшірілген төбенің орнын тазалауға болады.

Іздеу үрдісінің ұзақтылығы (осы мақсатқа таңдап алынатын төбе саны) оның ағаш құрылымына байланысты болады. Ағаш «балансталған» кезде ғана барынша тиімді қолданылады, яғни оның барлық төбелерінің соңғы төбесінен басқасы – оң және сол жақ қабылдағыштары бар. Бұл кезде іздеу үрдісі дихитомия кезіндегідей тез орындалады.

ӘДЕБИЕТТЕР ТІЗІМІ

1. *Подбельский В.В., Фомин С.С.* Программирование на языке Си: Учеб. пособие. – 2-е доп. изд. М.: Финансы и статистика, 2002.

2. *Подбельский В.В.* Язык С ++: Учеб. пособие. – 3–е изд. Дораб. М.: Финансы и статистика, 1998.

3. *Культин Н.Б.* С/ С++ в задачах и примерах. – СПб.: БХВ – Пегербург, 2004.

4. *Мартынов Н.Н.* Программирование для Windows на С/ С++. Том.1 М.: ООО "Бином – Пресс", 2004.

5. *Шиманович Е.А.* С/ С++ в примерах и задачах. – Мн.: Новое знание, 2004.

6. *Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д.* Структуры данных и алгоритмы. Изд. "Вильямс", М.: 2000.

7. *Седжвик Р.* Фундаментальные алгоритмы на С++. Части 1-4. М.: Изд. "ДиаСофт", 2001.

МАЗМҰНЫ

Кіріспе..... 3
1. КӨРСЕТКІШТІК ТИП ЖӘНЕ ПРОГРАММАЛАУ НЕГІЗДЕРІ..... 5
1.1. Статикалық және динамикалық программалық объектілер 5
1.2 Көрсегкіштер ұғымы 6
1.2.1. Көрсегкіштердің баяндалуы мен пайдаланылуы 7
1.2.2. '&' амалын қолдануға шектеу..... 9
1.2.3. Көрсегкіштерді инициалдау туралы толығырық мәліметтер 12
1.3. Көрсегкіштерге қолданылатын амалдар..... 20
1.3.1. Арифмегикалық амалдар және көрсегкіштер 23
1.3.2. Көрсегкіштер және қатынастар 25
1.4 Массивтер мен көрсеткіштер арасындағы байланыс..... 28
1.4.1. Көрсегкіштер массиві 33
1.4.2. Жолдар және көрсеткіштер 36
1.5 Функция парамегрлерінің көрсеткіштері 40
1.6. Массивтер мен жолдар функция параметрлері регінде 42
1.7 Функцияға көрсеткіштер 52
1.7.1. Функцияның көрсегкіштер массивтері 53
1.7.2. Функцияға көрсегкіштер параметрлер регінде..... 56
1.7.3. Функцияға көрсеткішфункция қайтаратын мән регінде... 57
1.8 Көрсегкіштермен программалаудың пайдалы мысалдары 60
2. КҮРДЕЛІ ҚҰРЫЛЫМНЫҢ КӨРСЕТКІШТІКТІК ТИПІ ЖӘНЕ ДИНАМИКАЛЫҚ ОБЪЕКТІЛЕРІ 66
2.1 Қажетті мәліметтер 66
2.2 Бірбайланысты тізімдер 68
2.3 Жолдардың бірбайланысты тізім регінде көрсетілуі 72
2.4 Жол-тізбек амалын жүзеге асыру..... 76
2.5 Екі бағыттық тізімдер..... 79
2.6 Кезектер мен стектер 85
2.7 Кестелер 89
2.7.1. Кестелердің берілу тәсілдері 90
2.7.2. Кестедегі бинарлық іздеу 90
2.7.3. Екілік ағаш 93
Әдебиеттер тізімі 101

Оқулық басылым

Көбеков Болат Сальмұханұлы
Жұманазарова Асия Мұхамаддинқызы
Қойшыбаева Жанар Бидахметқызы

С / С ++ ПРОГРАММАЛАУ ТІЛДЕРІНДЕГІ КӨРСЕТКІШТІК ТИП

Оқу құралы

РБ бастығы *З.А. Ғұбайдуллина*
Редакторы *П.А. Абдрахманова*
Компьютерде беттеген *Л.Т. Қасжанова*

Басуға қол қойылды 30.10.2009 ж.
Таралымы 500 дана Пішімі 60х84 1/16 №1 баспаханалық қағаз.
Көлемі 6,4 Бағасы келісімді. Тапсырыс № 599.

Қ. И. Сәтбаев атындағы Қазақ ұлттық техникалық
университетінің басылымы
ҚазҰТУ-дың Ғылыми-техникалық баспа орталығы
Алматы қ., Ладыгин, 32

