1 Си тілінде бағдарламалау орталары және оларда жұмыс істеу

2 СИ++ бағдарламалау тілінің негізгі элементтері

2.1 Тілдің алфавиті және мәліметтер типі

Тілдің алфавитінде латын бас және кіші әріптері, цифрлар және арнайы символдар (нүкте, үтір, апостроф, қос нүкте) пайдаланылады.

Ескертпелер (комментария) /* және */ символдары арасында жазылады. Немесе // символдың артында жазылады.

Тілдегі айнымалы, функция, объект аты ретінде пайдаланылатын идентификатор ұғымы негізгі болып табылады. Идентификатор 32 символдан тұруы мүмкін. Оның аты әріптерден басталу қажет. Онда әріптер мен цифрлар пайдаланылады. Си тілі регистрге сезімтал. Мысалы: SUMM, Summ, summ айнымалылары әр – түрлі болып есептеледі.

СИ тілінде мәліметтердің келесі типтері бар.

1. **int** (бүтін) –32768 -ден 32767-ға дейінгі аралықта мәндерді қабылдайтын бүтін сандардың типі. int типті айнымалы жадыда 16 бит орын алады;
2. **short** (қысқа бүтін) – осы типтегі айнымалылар int типті айнымалылардан артық болмау керек. Осы типтегі айнымалы жадыда 16 бит орын алады;
3. **long** (ұзын бүтін) – –2147483648 –дан 2147483647-ға дейінгі аралықта мәндерді қабылдайтын бүтін тип. long типті айнымалы жадыда 32 бит орын алады;
4. **char** (символдық) – мәндері символ болатын тип;
5. **unsigned** (белгісі жоқ) – СИ тілінде кейбір (char, short, int, long) типтерді unsigned модификаторы көмегімен тек қана оң сандарды қабылдауға мүмкіндік беретін тип. int типін сипаттағанда «unsigned int a;» орнына «unsigned a;» деп жазуға болады;

Мәліметтер типі	Min	Max
255signed char	- 128	127
signed short0 unsigned char	-32768	32767
unsigned short	0	65535
signed int	-32768	32767
unsigned int	0	65535
signed long	-2147483648	2147483647
unsigned long	0	4294967295

6. **float** (нақты) – мәндері нақты сан болатын тип. Нақты сандар экспонента түрінде жазылуы мүмкін, мысалы, -1.58e+2 (ол $-1{,}58 \cdot 10^2$ тең). float типті айнымалы 32 бит орын алады. Оның мәні +3.4e-38 –ден +3.4e+38-ға дейінгі аралықта жатуы мүмкін;
7. **double** (дәлділік) – float типті айнымалыдан қарағанда жадыда екі есе орын алатын (64 бит) нақты айнымалыларды анықтайды. Олардың мәндері +1.7e-308-дан +1.7e+308-ға дейін болуы мүмкін.
8. **long double** (ұзын нақты) – типті айнымалы жадыда 10 байт орын алады. Олардың мәндері 3.4E-4932 – дан 3.4E+4932- ға дейін болуы мүмкін.
9. **boolean**

2.2 Сызықтық бағдарламалау операторлары

2.2.1 Шығару операторы

Жазылуы: printf("басқару жолы", аргумент1, аргумент2, ...);

Басқару жолында үш типті объект жазылуы мүмкін: қарапайым символдар, түрлендіру спецификациясы, басқарушы символ-константалар.

Әр- бір түрлендіру спецификациясы % белгісінен басталып, түрлендіру жасайтын символмен аяқталады. Түрлендіру символы айнымалы типіне байланысты болады:

1) d – аргумент мәні ондық бүтін сан;
2) o – аргумент мәні сегіздік бүтін сан;
3) x – аргумент мәні он алтылық бүтін сан;
4) c – аргумент мәні символ;
5) s – аргумент мәні символдар жолы;
6) e – аргумент мәні экспонента түріндегі нақты сан;
7) f – аргумент мәні жылжымалы нүктемен нақты ондық сан;
8) u – аргумент мәні белгісі жоқ бүтін сан;
9) p – аргумент мәні көрсеткіш (адрес).

Егер % белгісінен кейін символ жазылмаса, онда экранға сол белгінің өзі шығады. printf функциясы қанша аргумент бар және олардың типтері қандай екенін анықтау үшін басқару жолын пайдаланады.

Мысалы, %6f – алты позициядан тұратын өрісте жылжымалы нүктемен санды шығару; %.2f – нүктеден кейін екі цифрмен нақты санды шығару; %6.2f – нүктеден кейін екі цифрмен нақты санды алты позициядан тұратын өріске шығару.

Мысалы, a=3,687 және b=10,17 болса, онда

printf(“%7f %8f”,a,b);

командасынан кейін экранда нәтиже келесідей шығарылады:

_ _ 3.687 _ _ _10.17

(7 поз.) (8 позиция)

Мысалда көріп отырғандай бос позициялар пробелмен толтырылады.

Егер printf(“%.2f %/2f”, a, b); функциясын пайдаланса онда нәтиже келесідей болады:

3.69 10.17 ,

Егер printf(“%7.2f e”,a,b); функциясын пайдаланса онда нәтиже келесідей болады:

_ _ _ 3.68 1.010000e+01

(7 позиция)

Спецификация алдындағы «минус» таңбасы санды сол жағынан шығарады. Мысалы:

printf(“%-6d”,s);

функциясынан кейін экранда келесі жазу шығады:

336_ _ _

(6 позиция)

Ескертетін жағдай, егер өріс ұзындығы сандағы цифрлардың санынан кем болса, онда өріс қажетті өлшемге дейін ұзарады.

Жоғарыда айта кеткен басқарушы символ-константалардың ішінен көп пайдаланылатын келесілер:

1) \a – қысқа дыбыс сигналын шығару үшін;
2) \b – меңзерді бір позицияға солға жылжыту;
3) \n – жаңа жолға көшу үшін;
4) \r – меңзерді жолдың басына қою үшін;
5) \t – көлденең табуляциялау;
6) \v – тігінен табуляциялау.
7) \0 – бос литера, NULL

Мысалы, i=50 болсын, келесі функцияның жазуынан кейін

printf(“\t ЭЕМ\n%d\n”,i);

алдымен көлденең табуляция (\t) орындалады, яғни экран шетінен 8 позиция орын қалдырады. Содан соң ЭЕМ сөзі шығып, меңзер келесі жолдың басына түседі (\n), d форматы бойынша і-дің бүтін мәні шығады. Соңында меңзер қайтадан келесі жолға түседі. Сонымен экранда келесідей нәтиже шығады:

_ _ _ _ _ _ _ _ ЭЕМ

50

2.2.2 Енгізу операторы

Жазылуы: scanf(«басқару жолы», аргумент1, аргумент2,...);

Басқару жолы printf функциясы секілді қарапайым символдар, түрлендіру спецификациясы, басқарушы символ-константалар жазылуы мүмкін.

Бірақ scanf функциясында, аргумент ретінде айнымалы пайдаланылса, оның алдында & символы қойылады. & операторы айнымалының адресін алуға мүмкіндік береді. Айнымалы мәнін енгізгенде scanf функциясы оны адресі бойынша жадыға сақтайды.

Мысалы: бүтін типті і және нақты типті а айнымалы үшін мәндерді енгізу қажет болса, келесі функция жазылады:

scanf(“%d%f”,&i,&a);

scanf және printf функцияларын пайдалану үшін <stdio.h> файлын қосу қажет.

Мысалы, экранға "Сәлем, достар!" жазуын шығару программасы келесі түрде жазылады:

```
#include <stdio.h>
main()
{
printf("Сәлем, достар!");
}
```

2.2.3 Меншіктеу операторы және өрнектер

Си тілінде меншіктеу операторы «=» белгісімен белгіленеді. Жазылуы:

айнымалы = мән;

Басқа тілдерге қарағанда Си тілінде меншіктеу операторын өрнектерде пайдалануға болады. Мысалы, if ((c=a+b)<0) printf("c саны нөлден кіші').

Тағы бірнеше рет меншіктеу мүмкіндігі бар. Мысалы:

x=y=z=a*b;

Бұл жерде алдымен a*b мәні z-қа, содан соң y-қа, сондан кейін x-қа меншіктеледі.

Си тілінде константалардың мәнін өзгертпеу үшін const модификаторы пайдаланады. Мысалы: const float a=3.5;

const j=47;

Өрнектерді айнымалыларды есептеуге арналған формула ретінде пайдаланады. Олар *операндтардан* (айнымалылар, константалар және т.б.) құралады және операция белгілерімен (қосу, азайту, көбейту және басқамен) байланысады.

Кесте 1 – Арифметикалық операциялар

Арифметикалық операциялар	Орындалуы
+	Қосу
–	Азайту
*	Көбейту
/	Бөлу, бүтін бөлігін алу
%	Бөлгендегі қалдығын алу

Мысалы: егер b=5, c=2, бүтін типті айнымалылар болса, онда

a=b%c; d=b/c;

операциялардың орындалуынан кейін a айнымалысы 1 мәнін қабылдайды, d айнымалысы 2 мәнін қабылдайды.

Арифметикалық операциясына сәйкес меншіктеу операторын басқаша пайдалануға болады.

Мысалы, x=x+n; жазуын x+=n; деп жазуға болады. Сол секілді

x=x-n; x-=n;

x=x/n; x/=n;

x=x*n; x*=n;

x=x%n; x%=n;

Кесте 2 – Қатынас операциялары

Қатынас операциялары	Ұғымы
<	Кіші
<=	Кіші не тең
= =	Тең
>=	Үлкен не тең
>	Үлкен
!=	Тең емес

Кесте 3 – Логикалық операциялар

Логикалық операциялар	Ұғымы
&&	ЖӘНЕ, and
||	НЕМЕСЕ, or
!	ЕМЕС, not

Си тілінде логикалық тип жоқ болғандықтан, логикалық өрнектін нәтижесі сандық шама болады (жалған мәнде 0-ге тең, ақиқат мәнде 0-ге тең емес шама).

Си тілінде операнд мәнін бірге арттыру (++) және бірге кеміту (--) операциялары пайдаланады.

x++; ++x; операторлардың нәтижесі бір, бірақ пайдалану барысында айырмашылықтары бар. Мысалы:

```
main()
{
int x,y;
x=5; y=60;
x++;  ++y;
printf(“x=%d  y=%d\n”,x,y);
printf(“x=%d  y=%d\n”,x++,++y);
```

}

Нәтижесі:

x=5 y=61

x=5 y=62

x++ операторын пайдаланғанда x айнымалының мәні алдымен өрнекте пайдаланады, содан соң бірге артады. ++y пайдаланғанда алдымен y айнымалысы бірге артады, содан соң өрнекте пайдаланады.

Кесте 4 - Операциялардың орындалу приоритеті

++ --	* / %	+ -	!	< <= > >=	== !=	&&	||

Егер операция приоритетін өзгерту қажет болса, дөңгелек жақшаны пайдаланады. Олардың приоритеті жоғары.

Өрнектерде тағы шарт операциясы ? кеңінен пайдаланады.

y=x?a:b,

өрнегінде егер x нөлге тең болмаса, y=a, егер x нөлге тең болса y=b.

y=(a>b)?a:b;

өрнегі y айнымалысына a не b айнымалылардың ең үлкен мәнін қабылдауға мүмкіндік береді, яғни y=max(a,b).

2.3 Математикалық функциялар

Си тілінде математикалық функцияларды пайдаланар алдында math.h файлын қосу қажет:

#include <math.h>

Кесте 5- Математикалық функциялар

Математикалық жазылуы	СИ тілінде жазылуы	Нәтиже типтері
| X |	abs(X)	int
| X |	fabs(X)	float
arccos X	acos(X)	double, float
arcsin X asin(X)	double, float	
arctg X	atan(X)	double, float
cos X	cos(X)	double, float
sin X	sin(X)	double, float
tg X	tan(X)	double, float
e^X	exp(X)	double, float
ln X	log(X)	double, float

log X	log10(X)	double, float
√X	sqrt(X)	double, float
X^Y	pow(X, Y) double, float	

Аргумент және функция алдында мүмкін болатын тип көрсетілген (бағдарламалау барысында ол жазылмайды).

Мысалы: y-тің мәнін табатын бағдарламаны жаз.

y=x+0.25+sin $e^{x+0.25}+\sqrt{\frac{x+0.25}{2.5}}$

```
#include <stdio.h>
#include <math.h>
main()
{
float x,y,a;
scanf(“%f”,&x);
a=x+0.25;
y=a+sin(exp(a))+sqrt(a/2.5);
printf(“y=%f”,y );
}
```

3 Шарттық және таңдау операторы

3.1 Шарттық оператор if

Жазылуы: if (шарт) инструкция1; **else** инструкция2;

if – егер, else - әйтпесе

Егер жақшадағы шарт ақиқат болса, онда инструкция1 орындалады, ал егер жалған болса, инструкция 2 орындалады.

Мысалы: егер x мәні теріс сан болса, оның квадратын шығару, әйтпесе $\sqrt{x}$ мәнін шығару

```
#include <stdio.h>
#include <math.h>
main()
{
float x,y;
scanf("%f", & x);
if (x<0) y=pow(x,2);
else y=sqrt(x);
printf("y=%f",y );
}
```

3.2 Таңдау операторы switch

Жазылуы:
switch (айнымалы)
{**case** айнымалы мәні1, оператор 1; **break**;
. . .
case айнымалы мәні n, оператор n; **break**;
default: оператор n+1; **break**;}

switch – айырып қосқыш, жақшаның ішінде мәні бүтін сан болатын айнымалы анықталады. Оны селектор деп атайды.

Егер айнымалы мәні ешқайсысына сәйкес келмесе онда default сөзінен кейінгі операторлар орындалады. Егер default сөзі тұрмаса ешқандай оператор орындалмайды.

Таңдау жасалған соң, келесі тексерістерді жасамас үшін break операторы пайдаланылады. Ол switch операторынан шұғыл түрде шығады.

Мысал1:

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

```
char y;
scanf(“%c”,&y);
switch(y)
{
case ‘1’:
printf(“бұтақ 1\n”);
break;
case ‘2’:
case ‘3’:
printf(“бұтақ 2 немесе  3\n”);
break;
default:
printf(“бұтақ 1,2,3 жұмыс істемейді\n”);
}
}
```

Мысал 2: Енгізілген бағаға сәйкес сөзді шығару

```
#include<stdio.h>
main( )
{
int y;
scanf(“%d”,&y);
switch(y)
{
case 1:
case 2:
printf(“қанағаттанарлықсыз\n”);
break;
case 3:
printf(“қанағаттанарлық\n”);
break;
case 4:
printf(“жақсы\n”);
break;
case 5:
printf(“өте жақсы\n”);
break;
default:
printf(“ондай баға жоқ\n”);
}
}
```

4 Циклдік операторлар

4.1 Параметрлі қайталану операторы for

Жазылуы: for (өрнек 1; өрнек 2; өрнек 3) оператор;

Бірінші өрнек санауыш инициализациясы үшін. Екінші өрнек шартты тексеруге арналған, шарт жалған болғанда цикл аяқталады. Үшінші өрнек операторлар орындалған соң қадам санын қосады.

Мысалы: for(i=0; i<100; i+=2) [i]=i+1;

Әр өрнек «үтір» белгісімен ажыратылатын өрнектерден құрылуы мүмкін
Мысалы:
for(i=0, j=1; i<100; i++, j++) a[i]=b[j];

Мысал1: y=sin sin sin…….sin x n рет қайталанады.
У-ті табатын программаны жазу.

```
main()
{
float x,y;
int n;
scanf(“%f %d”, &x, &n);
y=sin(x);
for (int i=1; i<=(n-1); i++)
y=sin(y);
printf(“%f”, y);
}
```

Мысал 2 : $y=\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+.....+\frac{1}{n!}$ табатын бағдарламаны жаз.

```
main()
{
float y,f;
int n;
scanf(“%d”, &n);
for (int i=1; i<=(n-1); i++)
{
for (int j=1; j<=i; j++)
f*=j;
y+=1/f;
}
printf(“%f”, y);
}
```

4.2 Алдыңғы шартты қайталану циклы, while операторы

Жазылуы: **while** (шарт) оператор;

Жақшадағы шарт ақиқат болғанша циклдағы операторлар орындалады. Егер алғашқысында шарт жалған болса, цикл бір де бір рет орындалмайды.
Мысалы: Енгізілген санның цифрларының қосындысын табу бағдарламасы.

```
#include <stdio.h>
main( )
{
int n, m, s;
scanf(“%d”, &n);
s=0;
while (n>1)
{
m=n%10;
s+=m;
n/=10;
}
printf(“%d”, s);
}
```

4.3 Кейінгі шартты қайталану циклы, do-while операторы

Жазылуы: **do** оператор **while** (шарт)

Осы операторда цикл денесі кемінде бір рет орындалады. Шарт жалған мәнін қабылдағанда цикл аяқталады.
Мысалы:

```
#include <stdio.h>
main( )
{
int n, m, s;
scanf(“%d”, &n);
s=0;
do
m=n%10;
s+=m;
n/=10;
while (n>1);

printf(“%d”, s);
}
```

break операторы

Break операторы циклдан шұғыл түрде шығу қажет болғанда пайдаланылады.

Мысал:

```
while (st>0 && st<25)
{
if (st==4||st==8||st==12)
break;
}
```

if операторындағы шарт ақиқат болғанда цикл жұмысы аяқталады.

continue операторы

Бұл оператор өзінен кейін тұрған операторларды ескермей келесі итерацияға өтеді.

Мысал:

```
while (st>0 && st<25)
{
if (st==4||st==8||st==12)
continue;
}
```

Бұл жерде continue операторынан кейін тұрған операторлар орындалмайды және келесі итерацияның басына көшеді.

5 Массивтер және көрсеткіштер

5.1 Массивтер

Массив – бұл элементтері бір типті болатын мәліметтер жиыны.

Массивтерді айнымалыларды секілді сипаттайды:

```
int a[100];
float c[10][20];
```

Біріншісі бүтін типті a[0],a[1], … ,a[99] (индекстелу нөлден басталады) элементтерден тұратын бір өлшемді массив болып табылады. Екіншісі элементтері нақты тип болатын екі өлшемді массив болып табылады. Ол 10 жолдан, 20 бағаннан тұрады.

Массивті келесідей инициализациялауға болады:

```
int a[6]={10,20,30,40,50,60}; немесе int a[]={10,20,30,40,50,60};
float[2][3]={{10,20,30},{40,50,60}};
```

Мысал 1. 10 нақты саннан тұратын бірөлшемді S массиві берілген. Осы массивті кері ретпен шығару.

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

```
{float s[10];
int i;
for (i=0;i<10;i++)
  scanf(“%f”,&s[i]); /*массив элементтерін енгізу*/
for (i=9;i>=0;i--)
  printf(“%f”,s[i]); /* массив элементтерін кері ретпен шығару*/
}
```

Мысал 2. Бес бағаннан және бес жолдан тұратын бүтін типті екі өлшемді массив берілген. Массивтегі ең үлкен элементтің орналасу номерін анықтау.

```
#include <stdio.h>
main()
{int i,j,max,in,jn,a[5][5];
for (i=0;i<5;i++)
for (j=0;j<5;j++)
scanf(“%d”, &a[i][j]); /*матрицаны енгізу*/
max=a[1][1];
for (i=0;i<5;i++)
for (j=0;j<5;j++)
if (a[i][j]>max)
{in=i; jn=j;}
printf(“%d %d”, in, jn);
}
```

Мысал 3. D(6,6) массивінің басты және кері диагональдің орындарын ауыстыру.

```
#include <stdio.h>
main()
{int i,j,a,d[6][6];
for (i=0;i<6;i++)
for (j=0;j<6;j++)
scanf(“%d”, &d[i][j]);

for (i=0; i<5; i++)
{a=d[i][i];
d[i][i]=d[i][6-i];
d[i][6-i]=a;
}
for (i=0; i<6; i++)
for (j=0; j<6; j++)
printf(“%d%c”, d[i][j], (j==5)?”\n”:” “);
}
```

Матрица элементтерін жолдар бойынша шығаруда ? операциясы пайдаланылады. Егер j=5, онда курсор келесі жолға түседі (“\n”), басқа

жағдайда пробел шығарылады. Экранға символдық шама шығарылатындықтан %c спецификациясы пайдаланылады.

5.2 Көрсеткіштер және сілтемелер

Көрсеткіш – бұл кейбір мәліметтердің орналасу адресін сақтайтын айнымалы.

Көрсеткішті пайдалану:

1) Кез келген мәліметтердің орналасу жеріне нұсқау жасай алуға болады.

2) Құрылымдық мәліметтердің элементтеріне (массив, жолдар, құрылымдар, жазбалар) қол жеткізетін көрсеткіш ретінде пайдалануға болады. Көрсеткішті индекстеу арқылы массив немесе құрылым сақталынған байттар тізбегіне қол жеткізуге болады.

3) Көрсеткішті пайдалана отырып бағдарламаның орындалу барысында жаңа айнымалыларды құруға болады. Си тілі бағдарламаға қажетті көлем жадыны (байтта) сұратуға мүмкіндік береді. Осы әдісті *динамикалық үлестіру* деп атайды. Оның көмегімен жадының кез келген көлемін пайдалануға болады.

Келесі бағдарламаны қарастырайық:

```
main()
int a,*p;
p = &a;
a = 421;
printf("a  орналасуы: %p\n",&a);
printf("a  мәні: %d\n", a);
printf("p  мәні: %p\n", p);
printf("адрестелінген мән p: %d\n",*p);
```

Осы бағдарламада екі айнымалы сипатталған a және p. Айнымалы a – бүтін типті, p – бүтін айнымалыға көрсеткіш, яғни айнымалының адресін сақтайды. Көрсеткішті сипаттағанда алдында (*****) белгісін қояды. Келесі жолда a айнымалысының адресі p-ға меншіктеледі. Бұл жердегі **(&**) адрестік оператор айнымалының адресін алуға мүмкіндік береді. Содан соң 421 мәні a-ға меншіктеледі. Нәтижесінде экранға келесі жазу шығады:

```
a орналасуы: 166E
a мәні: 421
p мәні: 166E
адрестелінген мән p: 421
```

Сонымен &a – a айнымалысына сілтеме, p – көрсеткіш, *p – p көрсеткіші көрсететін адрестегі мән.

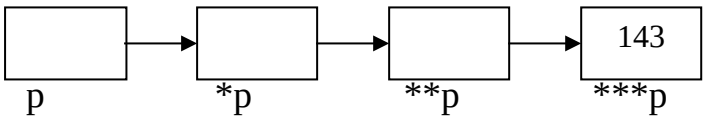
Сілтемелерге мысалдар:

1) int i=1;

int & r=i; r және i айнымалылары бір адреске сілтеме жасайды

int x=r; x=1
r=2; i=2
r++; i=3;
2) char c1=’a’;
char* p=&c1; p көрсеткіші c1 айнымалының адресіне сілтеме
char c2=*p; c2=’a’

Көрсеткіштердің келесі жазуын пайдалануға болады: ***p=143;



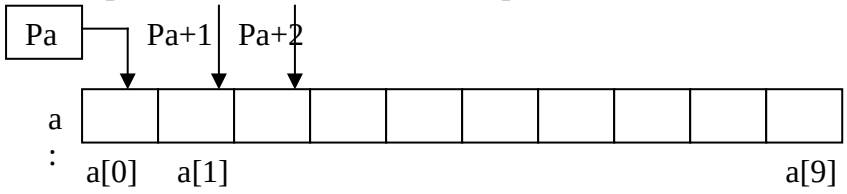
Бұл жазу көрсеткішке көрсеткіш пайдаланғанын көрсетеді.

5.3 Бірөлшемді массивтер және көрсеткіштер

Индекстеу операциясымен жасалынатын массивтің кез келген элементіне қол жеткізуді көрсеткіш арқылы жасауға болады.

Мысалы, int a[10];

он элементтен тұратын a[0], a[1], … ,a[9] массив болсын дейік және pa массивтің бірінші элементіне a[0] көрсеткіш болсын.



int *pa;

pa = &a[0]; немесе pa = a; деп жазуға болады

Онда pa+5 жазуы a[5] элементтің адресін көрсетеді, ал *(p+5) жазуы a[5] элементтің мәнін көрсетеді.

(a+i) = = &(a[i])

*(a+i) = = a[i]

Бұл жазбалар бір біріне тең. a массиві көрсеткіш ретінде сипатталмаса да *(a+i) жазу орынды болады.

Мысал 1: Массивті көрсеткіш көмегімен шығару

```
int arr[]={1,2,3,4,5,6,7,8,9,10};
for (int i=0;i<10; i++)
{
printf(“%d”, *(arr+i));
}
```

Көрсеткіш айнымалы болғандықтан pa=a немесе pa++ деп жазуға болады, бірақ a=pa; жазуы қате болады. pa++ операторы a массивінің келесі элементіне көшуді орындайды.

Ескерту *(a+2) мен *a+2 өрнектерінің айырмашылықтары бар:

*(a+2) –a массивінің үшінші элементі;
*a+2 – массивтің бірінші элементіне 2 санын қосу.

Мысал 2. Бірөлшемді массивті қарапайым тәсілмен және көрсеткіштерді пайдаланып шығару.

```
#include <stdio.h>
int a[6]={10,20,30,40,50,60};
main ()
{int i, *p;
for (i=0; i<6; i++)
printf("%d",a[i]); /*массивті қарапайым тәсілмен шығару*/
for (p=&a[0];p<=&a[5];p++)
printf("%d",*p); /*массивті көрсеткішті пайдаланып шығару*/
for (p=&a[0],i=0; i<6; i++)
printf("%d",p[i]); /*көрсеткішті пайдаланатын тағы бір тәсіл*/
}
```

Егер p=&a[i], онда p++ операциясынан кейін p –да a[i+1] элементінің адресі сақталынады.

Мысал 3. Көрсеткіш көмегімен массив элементтерінің арифметикалық ортасын табу.

```
#include <stdio.h>
int a[]={10,20,30,40,50,60}
main()
{int i,*p;
float s;
p=a;
for (s=0,i=0; i<6; i++)
s+=*(p+i); /*массив элементтерінің қосындысы*/
s=s/6; /*массивтің арифметикалық ортасы*/
printf("%f",s);
}
```

Мысал 4. Массив элементтерін кері ретпен шығару.

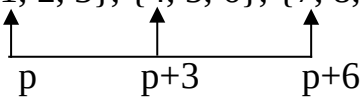
```
#include <stdio.h>
main()
{float s[10];
int *p,i;
for (i=0;i<10;i++)
scanf("%f",s[i]);
p=&s[9]; /*көрсеткіш массивтің соңғы элементінің адресін алады */
for (i=0; i<10; i++)
printf("%f",*(p-i)); /*элементтерді кері ретпен шығару*/
for (p=&a[9]; p>=&a[0]; p--) /*кері ретпен шығарудың тағы бір тәсілі*/
printf("/n%d",*p);
```

```
}
```

5.4 Екіөлшемді массивтер және көрсеткіштер

Мысалы, 3 бағаннан және 3 жолдан тұратын а массиві берілсін және р массивтің бірінші элементіне а[0] көрсеткіш болсын, онда

```
a[3][3]= {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```



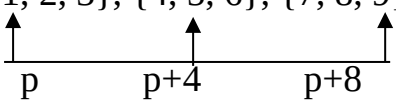
p+3 көрсеткіші а[1][0] элементінің адресіне көшеді, p+6 көрсеткіші а[2][0] элементінің адресін көрсетеді.

Мысал 1. а матрицасы берілген. Экранға басты диагональ элементтерін, бірінші жол элементтерін және бірінші баған элементтерін көрсеткіштерді пайдаланып шығару.

```
#include <stdio.h>
main ()
{
int a[3][3]={{10,20,30},
{40,50,60},
{70,80,90}};
int *pa[3]={a[0],a[1],a[2]};
/*a массивінің жолдарына pa көрсеткішін сипаттау және бастапқы мәндерін меншіктеу: pa[0]=a[0]; pa[1]=a[1]; pa[2]=a[2]*/
int p=a[0]; /*a массивінің бірінші элементіне көрсеткішті сипаттау */
int i;
for (i=0;i<9;i+=4)
printf("%d",*(p+i)); /*бас диагональ элементтерін шығару*/
for (i=0; i<3; i++)
printf("%d",*p[i]); /*бірінші жолдың элементтерін шығару*/
for (i=0; i<3; i++)
printf("%d",pa[i]); /*бірінші бағанның элементтерін шығару*/
}
```

Басты диагональ элементтері а[0][0], а[1][1], а[2][2] болғандықтан, оған сәйкес р, p+4, p+8 көрсеткіштері пайдаланады. Сондықтан қадам 4-ке көбейеді i+=4.

```
a[3][3]= {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```



5.5 Динамикалық массив

Көрсеткіштер тақырыбында динамикалық үлестірілу туралы айта кеткен болатынбыз. Си тілі бағдарламаға қажетті көлем жадыны (байтта) сұратуға мүмкіндік береді. Осы әдісті *динамикалық үлестіру* деп атайды. Оның көмегімен жадының кез келген көлемін пайдалануға болады. Мысалы: массив өлшемділігі белгісіз. Ол енгізілетін санға байланысты болсын дейік. Динамикалық массивті пайдаланып массив өлшемділігін анықтау және массивті 1-ден енгізілетін санға дейін толтыру.

Бағдарлама 1: new функциясын пайдаланып

```
main()
{
int count;
scanf(“%d”, &count);
double *a=new double[count];  // динамикалық массивке жадыны дайындау
for (int i=0; i<count; i++)
{
a[i]=i+1;
printf(“%d”, a[i]);
}
delete [] a;   // динамикалық массивке дайындалған жадыны тазалау.
}
```

Бағдарлама 2: calloc функциясын пайдаланып

```
#include <stdio.h>
#include <alloc.h>
main()
{
int i, n;
float *a;
scanf(“%d”, &n);
a=(float*)calloc(n,sizeof(float));  // динамикалық массивке жады беріледі
for (i=0; i<n; i++)
{
a[i]=i+1;
printf(“%d”, a[i]);
}
free(a);   // динамикалық массивке берілген жадыны тазалау.
}
```

Бұл жердегі sizeof – тип өлшемін анықтау операциясы;

new - жадыны үлестіру операциясы. Динамикалық жадының бөлігіне қол жеткізуге мүмкіндік береді. Операнд ретінде тип аты пайдаланылады. Бұл операция жадыда орналасқан объектінің адресін қайтарады;

delete – new операциясымен бөлінген динамикалық жадыны тазалау операциясы.

Егер массив символдық тип болса, malloc функциясы пайдаланылады.

6 Құрылымдар және көрсеткіштер

Құрылым – бұл әр – түрлі типті мәліметтер жиыны. Ол массивке ұқсас, бірақ оның элементтеріне аты бойынша қол жеткізеді. Мысалы, мектептегі оқушының мәліметтері: фамилия, аты, туған күні, сыныбы, жасы.

Жазылуы:

struct тип {элемент типі 1 элемент аты 1;
элемент типі n элемент аты n; };

Мысалы:

```
struct   date { int day;
int month;
int year;} ;
```

Жақшадан кейін осы типті айнымалылар сипатталуы мүмкін, мысалы:

struct date {…} a, b, c;

При этом выделяется соответствующая память.

Құрылымды құрылымда пайдалануға рұқсат берілген. Мысалы

```
struct  okushy { char fam [15];
aty [15];
struct date tkun;
int synyp, zhasy;};
```

Жоғарыда анықталған date типінің үш элементі бар: күн, ай, жыл. Олардың мәндері бүтін (int). Okushy құрылымында: fam [15]; aty [15]; tkun, synyp, zhasy элементтері бар. fam [15]; aty [15]; –бұл 15 символдан тұратын массив. Tkun айнымалысы date (ішкі құрылыммен) сипатталған.

Құрылымдық типті айнымалыларды келесідей сипаттайды:

struct okushy okuslar [50];

okuslar массиві okushy типті 50 элементтерден тұрады.

Мысал 1:

```
#include < stdio.h >
struct comp  {   int zhad;
 int ozu;
char model [20]; };
```

/* zhad, ozu, model элементтерінен тұратын comp типті құрылымды сипаттау */

struct comp DK= {80, 1024, “Pentium 4”}

```
/* comp типті DK айнымалысын сипаттау */
main ( )
{
printf (“ Дербес компьютер % s\n\n “, DK.model);
printf ( “қатты диск сиымдылығы - %  d  Гбайт \n”, DK.zhad);
printf ( “жедел жады - % d  Мбайт \n”, DK. ozu);
}
```

Мысал 2: құрылымда құрылымды пайдалану

```
# include < stdio.h >
struct date { int day;
int month;
int year;   };
struct person { char fam [20];
char im [20];
char ot [20];
struct date f1;};
main ( )
{
struct person ind1;
printf ( “ Адамның фамилиясын, атын, әкесінің атын, туған күнін, \n айын және туған жылын енгіз \n”);
scanf (“ % S  % S  % S  %d  %d %d”, &ind1.fam, &ind1.im, &ind1.ot,
& ind1.f1.day, &ind1.f1.month, &ind1.f1.year );
printf (“ Фамилиясы, аты,әкесінің аты: % S % S % S  \n”, ind1.fam, ind1.im, ind1.ot);
printf (“ туған жылы - % d  \n”, ind1.f1.year);
printf (“ туған айы - % d  \n”, ind1.f1.month);
printf (“ туған күні - % d  \n”, ind1.f1.day);
}
```

Мысал 3: құрылымдар массиві

```
#include < stdio.h >
struct computer { int mem, sp;
char model [20];
} pibm [10];
main ( )
{ int i, j, k, priz;
for ( i=0; i<10; i++)
{
printf (“ ЭЕМ моделі - ”);
scanf (“%S”, &pibm [i].model );
printf ( “жедел жады көлемі -”);
scanf (“%d”, &pibm[i].mem);
printf (“винчестер көлемі - ”);
```

```
scanf ( “%d , &pibm[i].sp ”);

for (j=0; j<10, j++);
{
printf (“ дербес компьютер %s\n ”, pibm[j].model);
printf (“жедел жады көлемі - % d Гб \n ”, pibm[j].mem);
printf (“винчестер көлемі - % d Мб \n ”, pibm[j].sp);
}
```

Мысал 4: Құрылымға көрсеткішті пайдалану

```
main()
{
struct person
{
char fam[80];
char name[80];
char patronymic[80];
} man;
person *per=&man;
printf(“Фамилияны енгізіңіз”);
fgets(per->fam,80, stdin);
printf(“Атын енгізіңіз”);
fgets(per->name,80, stdin);
printf(“Әкесінің атын енгізіңіз”);
fgets(per->patronymic,80, stdin);
printf(“%s %s %s \n”, per->fam, per->name, per-> patronymic);
}
```

Нүкте – тура таңдаудың операциясы. Көрсеткішті пайдаланғанда нүктенің орнына -> белгісі пайдаланады. Ол жанама таңдауды білдіреді.

7 Функциялар және ішкі программалар

7.1 Функциялар

Басқа жоғары деңгейдегі бағдарламалау тілдеріне қарағанда Си тілінде ішкі программалар функция мен процедураға бөлінбейді. Бағдарлама тек функциядан құрылады. Функция – бұл нақты есепті шешуге арналған операторлардың және өрнектердің жиыны. Функция арасындағы байланыс аргументтер, нәтиже мәндері және сыртқы айнымалылар арқылы жасалады.

Мысалы:

```
float srzn(int a, int b)
{
int y;
y=(a+b)/2;
```

return(y);
}

Функция нәтиже ретінде бір мәнді бере алады. Ол үшін қайтару операторы пайдаланылады:

return (өрнек);

Бұл жердегі өрнек айнымалы болуы мүмкін.

Мысал 1: санның абсолютті шамасын функция көмегімен есептеу.

```
#include <stdio.h>
#include "abc.cpp"
main()
{
int a=10, b=0, c=-20;
int d,e,f;
d=abc(a);
e=abc(b);
f=abc(c);
printf("%d %d %d", d, e, f);
}
```

Функцияның өзі

```
abc(x)
int x;
{int y;
y=(x<0)?-x:x;
return(y);
}
```

Көрсетілген бағдарламада функция типі сипатталмаған. Ол тек қана функция нәтижесі бүтін типті болғанда мүмкін. Басқа жағдайда типті сипаттау қажет.

Және де осы бағдарламада функция бөлек файлда abc.cpp сақталған және #include процедурасымен қосылған. Тырнақша (“ “) белгісімен қоршалған файл ағымды каталогтан ізделінеді (бағдарлама мен функция ағымды каталогта сақталу керек), ал < > жақшаға алынған файл кітапхана катологында ізделінеді.

Функцияны бағдарламаның ішінде де сипаттауға болады.

Мысал 2: f=$\sqrt{x}+y/z$ формуласын функция көмегімен есептеу.

```
#include <stdio.h>
double vv(x,y,z)
double x, y, z;
{
double f;
f=sqrt(x)+y/z;
return(f);
}
main()
{
```

```
double x=5.5, f, y=10, z=20.5;
f=vv(x,y,z);
printf(“%f “, f);
}
```

Егер функция main() функциясынан кейін сипатталса оның прототипін көрсету қажет.

Мысалы: алдыңғы есеп

```
#include <stdio.h>
double vv(double x, double y, double z);
main()
{.......}
double vv(x,y,z)
double x, y, z;
{......}
```

Си тілінде сонымен қатар, аргументі жоқ функцияны және ешқандай нәтиже қайтармайтын функцияны пайдалануға болады. Нәтиже қайтармайтын функцияға void типін пайдалану қажет. Ол қайтарылатын мәннің болмауын білдіреді.

Мысал 3: a және b айнымалылардың орнын ауыстыру.

```
#include <stdio.h>
#include “izm.cpp”
main()
{ int a,b;
scanf(“%d %d”, &a, &b);
izm(&a,&b);
printf(“%d %d”, a, b);
}
```

izm.cpp файлында сақталған функция

```
void izm(a, b);
int *a, *b;
{ int c;
c=*a;
*a=*b;
*b=c;
}
```

Егер функция аргументі ретінде массив аты пайдаланса, онда оған массивтің басталу адресі беріледі. Элементтердің өздері көшірілмейді. Функция массив элементтерін индекстеу арқылы басынан жылжыта алады.

Мысал 4: S массивінің элементтерін орындарымен ауыстыру: біріншісін екіншісімен, үшіншісін төртіншісімен және т.с.

```
#include <stdio.h>
void reverse(s)
```

```
int s[];
{
int a,i;
for (i=1; i<5; i+=2)
{a=s[i]; s[i]=s[i+1]; s[i+1]=a;}
}
main()
{int i,j,s[6];
for (i=0; i<6; i++)
scanf(“%d”,&s[i]);
reverse(s); /* reverse функциясын шақыру*/
for (i=0; i<6; i++)
printf(“%d”,s[i]);
}
```

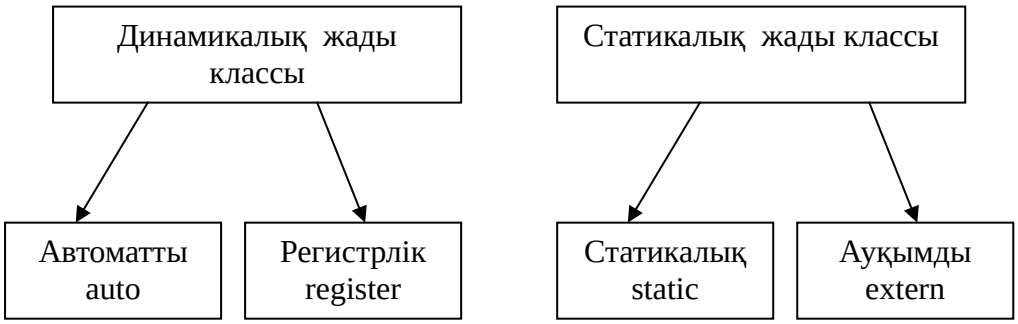
Бірөлшемді массивті int s[]; деп сипаттауға болады, ал екі өлшемді массивті сипаттағанда екінші жақшада баған саны жазылуы қажет, мысалы: a[][3].

Мысал 5. a(5,5) массивінің барлық элементтерін екіге арттыру.

```
#include <stdio.h>
void mas(a)
int a[][5]; /* a массивін сипаттау */
{int i,j; /* i,j айнымалыларын сипаттау*/
for (i=0; i<5; i++)
for (j=0; j<5; j++)
a[i][j] = 2*a[i][j]; /*массив элементтерін екі есе арттыру*/
}
main()
{int a[5][5];
int i,j;
for (i=0;i<5;i++)
for (j=0; j<5; j++)
scanf(“%d”,a[i][j]); /*массивті енгізу*/
mas(a); /* mas функциясын шақыру*/
for (i=0; i<5; i++)
for (j=0; j<5; j++)
printf(“%d”, a[i][j]); /*нәтижені экранға шығару*/
}
```

7.2 **Жады класстары**

Жады классы объектінің не айнымалының орналасуын анықтайды. Жады класстарының екі түрі болады:



auto – айнымалы локальді (не автоматты үлестірілетін) жадыда орналасқанын көрсетеді. Бағдарлама бөлігінен не функция қайтарылғанда үлестірілген локальді жадының облысы тазаланады және ондағы айнымалылар жойылады. Бұл спецификатор көп пайдаланылмайды, себебі функция денесінде не операторлар бөлігінде сипатталған барлық айнымалылар автоматты түрде локальді жадыда орналасады.

register – айнымалы жиі пайдаланылатынын көрсетеді. Мүмкіндік болса осындай айнымалылардың мәндері процессордың ішкі регистрлеріне орналасады. Егер регистрлер бос болмаса, транслятор бұл айнымалыларды локальді жадыға орналастырады.

static – тек қана функция аттарына қолданылады. Статикалық айнымалылар автоматты айнымалыларға ұқсас, бірақ функция өз жұмысын бітіргенде автоматты айнымалылар секілді статикалық айнымалылар жойылмайды. Функцияның бір шақыруынан келесі шақыруына дейін компилятор олардың мәндерін сақтайды. Статикалық айнымалы тек файлдағы функциялармен пайдаланыла алады.

extern – айнымалыны бағдарламаның барлық модульдерінде пайдалануға болатынын көрсетеді. Айнымалыны басқа файлда анықтау үшін extern спецификаторын пайдаланады.

Мысал 5. Функция түрінде өрнекті есептеу: $f=a*x^2+b*x+c;$

```
#include <stdio.h>
int a=5, b=7, c=10,x; /* a,b,c,x бүтін типті сыртқы айнымалыларды сипаттау*/
/*функция*/
kv()
{int f;
f=a*x*x+b*x+c; /* f  айнымалысын есептеу*/
return (f); /* f мәнін бағдарламада қайтарады */
}
main ()
{ int f;
scanf (“%d”, &x); /* x айнымалысының мәнін енгізу*/
f=kv(); /*функцияны шақыру*/
printf (“%d”,f); /* f  айнымалысының нәтижесін шығару*/
```

```
}
```

Негізгі бағдарлама және функция әр түрлі файлда орналасқан мысалды қарастырайық:

```
#include “kv.cpp” /* kv.cpp файлындағы  kv функциясын қосу */
#include <stdio.h>
int a=5, b=7, c=10,x,f; /* a,b,c,x бүтін типті сыртқы айнымалыларды сипаттау */
main ()
{
scanf (“%d”, &x); /* x айнымалысының мәнін енгізу */
f=kv(); /* функцияны шақыру */
printf (“%d”,f); /* f  айнымалысының нәтижесін шығару */
}
/*функция*/
kv()
{extern int a,b,c,x,f;
f=a*x*x+b*x+c;
return (f);
}
```

8 Препроцессорлық құралдар

8.1 # include, # define, # undef директивалары

Си трансляторының препроцессор деп аталатын ендірілген құралы бар. Ол бағдарламаны компиляцияға дейін қарастырады (осыдан препроцессор термині шыққан) және бағдарламадағы барлық символдық аббревиатураларды сәйкес директиваларға ауыстырады, көрсетілген файлдарды қосады. Препроцессор үшін бағдарлама жолдары # символынан басталады.

Файлдарды қосу үшін # include директивасын пайдаланған болатынбыз.

Препроцессордың басқа директиваларын қарастырайық:

define <идентификатор> < ауыстыру>

Бұл директива бағдарламадағы идентификатор орнына ауыстыру мәтінін қояды. Егер директива келесі түрде болса:

define идентификатор (идентификатор ,…идентификатор)

онда ол аргументтері бар макроауыстырудың анықтамасы.

Мысал 1:

```
# define FOOR TWO*TWO
# define PX  printf(“x тең %d\n”,x)
#define FNT  “x тең % d\n”
# include<stdio.h>
# define TWO 2
```

```
main ( )
{    int x=TWO;
      PX;
      x=FOOR;
      printf(FNT,X);
}
```

Бағдарлама орындалған соң экранға келесі жазу шығады:

X тең 2

X тең 4

Мысал 2. Аргументтері бар # define директивасының мысалын қарастырайық. Ондай директивалармен өте мұқият жұмыс істеу керек.

```
# include<stdio.h>
# define KV(X) X*X
# define PR(X) printf("X тең %d\n",X)
main ( )
{    int x=4;
int z;
z=KV(x);
PR(z);
PR(x);
PR(KV(x+2));
pr(100/KV(2));
}
```

Бағдарламадағы KV(x) x*x ауыстырылады.

Бағдарлама жұмысының нәтижесі:

z тең 16

x тең 4

KV (x+2) тең 14

100/KV (2) тең 100

Бірінші екі жол түсінікті, ал KV (x+2) 36-ға, 100/KV (2) 50-ге тең болу керек, бірақ нәтижелері мүлдем басқа. Оның себебі X*X орнына X+2*X+2 және 100/2*2 қойылады. Егер KV (x+2) дұрыс шығарылсын десеңіз, формуланы келесідей жазу қажет:

```
# define  KV(X)  (X)*(X)
```

Онда KV (X+2) орнына (X+2)*(X+2) қойылады.

#undef ең соңғы анықталған идентификаторды жояды. Жазылу түрі:

#undef идентификатор

Мұндағы идентификатор – #define директивасымен анықталған идентификатор:

#undef ESCAPE

Бұл процедураның орындалуынан кейін ESCAPE идентификаторы анықталмайды.

8.2 #if, #ifdef, #ifndef, #else, #endif директивалары

Осы бөлімде қарастырылатын директивалар шартты компиляцияны орындауға мүмкіндік береді. Шартты компиляция – бұл кейбір шарттарды қанағаттандыратын бағдарлама бөліктерін компиляциялау. Шартты компиляция көмегімен әр түрлі жүйелерде идентификаторларға әр түрлі мәндерді бере аласыз, қажетті файлдарды қоса аласыз. Осылайша бағдарламаның орындалу барысында қажетсіз код жадыда орын алмайды.

#if директивасы if операторына ұқсас:

мысалы:

```
#if  sys==”IBM”
#include “ibm.h”
#endif
```

Мысалда көрсетілгендей ол #endif директивасымен аяқталады.

#ifdef директивасы идентификатордың анықталғанын тексереді, #ifndef директивасы идентификатордың анықталмағанын тексереді, яғни идентификатор #define –мен анықталды ма, жоқ па, соны тексереді.

Мысалы:

```
#ifndef  SIZE
#define SIZE 128
#endif
```

Егер SIZE анықталмаған болса, 128 мәнін қабылдайды.

Көрсетілген директивалар #else директивасымен бірге пайдаланылуы мүмкін. Ол else операторына ұқсас.

Мысалы:

```
#ifdef MAVIS
#include “horse.h”
#define STABLE 5
#else
#include “cow.h”
#define STABLE 15
#endif
```

Егер MAVIS идентификаторы анықталған болса, #else директивасына дейінгі барлық директивалар орындалады. Басқа жағдайда #else және #endif арасында тұрған барлық директивалар орындалады.

9 C++ тілінде файлдармен жұмыс

9.1 Файлға тізбектей қол жеткізу

Файлға тізбектей қол жеткізгенде ақпараттың алмастырылуы енгізу – шығару жүйесімен берілетін арнай буфер арқылы жасалады. Си тілінің компиляциясы енгізу –шығаруды тізбектей келетін файлдар ағымы ретінде қарастырады. Әр ағым файлмен байланыстырылады. Файл мен ағым

арасындағы байланыс оның ашылуы кезінде жасалады. Файлдың ашылуы fopen функциясымен жасалады. Бұл функция файлға көрсеткіш қайтарады.

Файлға көрсеткішті келесідей сипаттайды:

FILE * lst;

Мұндағы FILE- тип аты, ол stdio.h стандартты анықтамада сипатталған;

lst- файлға көрсеткіш (логикалық аты).

fopen функциясы бағдарламада келесідей шақырылады:

lst= fopen (файлдың физикалық аты, файлды пайдалану түрі);

Файлдың физикалық аты - сақталынған орнын көрсетеді, мысалы "D:zni.f”- D: дискісіндегі zni.f файлы үшін.

Файлды пайдалану түрі “w” (егер файлға мәліметтерді жазу керек болса), “r” (файлдан мәліметтерді оқу қажет болса) және “a” (файлдағы мәліметтерге тағы мәліметтерді қосу қажет болса) болуы мүмкін. Егер ондай файл болмаса, ол жасалады.

Файлға жазу үшін fprintf, fputs, файлдан оқу үшін fscanf, fgets кітапханалық функциялар пайдаланылады. Файлмен жұмысты аяқтағанда ол жабылу тиіс, мысалы:

fclose (lst)

lst- файлға көрсеткіш;

Мысал 1: Енгізілген мәтінді файлға жазу

```
#include <stdio.h>
main()
{ char s[50];
FILE *fl;
fl=open (“fayl.txt”, “w”);
scanf(“%s”, &s);
fprintf(fl, “%s”, s);
fclose(fl);
getchar();
}
```

Мысал 2: Файлдан жолдарды оқып экранға шығару

```
#include <stdio.h>
main()
{ char s[50];
FILE *fl;
fl=open (“fayl.txt”, “r”);
while(!feof(fl))
{
fscanf (fl, “%s”, &s);
printf(“%s”, s);
}
fclose(fl);
getchar();
```

}

9.2 Файлдың кез келген жеріне қол жеткізу

Файлдың кез келген жеріне қол жеткізу үшін fseek функциясы пайдаланылады. Оның жазылу түрі:

fseek (stream, offset, origin);

stream – файлға көрсеткіш;

origin – бағыт көрсеткіші.

Функция файлдың ішкі көрсеткішін оның жаңа орнына қояды. Ол offset қадамы бойынша және origin бағыт көрсеткіші бойынша есептелінеді.

Көрсетілген stream ағымындағы келесі операция ауысу жасалған позициядан басталады. Аргумент origin келесі мәндерді қабылдайды:

0 – файл басы;

1 – файл көрсеткішінің ағымды позициясы;

2 – файлдың соңы.

Көрсеткішті жылжыту үшін жазбадағы символдар санын білу қажет. Сондықтан n – жазба номері болсын, m – жазбадағы символдар саны.

Келесі жағдайларды қарастырайық:

origin = 0. Онда n жазбасына көшу үшін offset = (n-1)*m.

origin = 1. Келесі жолға түсу үшін offset=0. Бірінші жазбадан үшінші жазбаға көшу үшін offset = m. Кері бағытта жылжуға болады, онда offset теріс сан болу керек. Бесінші жазбадан үшіншіге көшу үшін offset = -3*m.

origin = 2. Егер төменнен жоғарыға көрсеткішті жылжыту керек болса, offset = 2*m.

Файлдың кез келген жеріне қол жеткізгенде файлдың басынан және соңынан шығып кетуге болады, сондықтан файлдың өлшемін білу керек. Ол үшін келесі үш оператор пайдаланылады:

handle =open (“файл аты”, O_CREATE);

l = file length (handle);

close (handle);

handle айнымалысының типі int, ал 1 айнымалысының типі offset айнымалысының типіне сәйкес long. Опреаторларды жазбас бұрын келесі файлдарды қосу қажет.

include < fcnt.h >

include < io.h >

Мысалы: енгізілген номерге сәйкес жазбаны шығару

```
# include <stdio.h>
# include <fcntl.h>
# include <io.h>
main ( )
{char fio[10], gr [5];
float st;
long offset,l;
```

```
int p=1, handle, n;
FILE *lf;
clrscr ( );
lf=fopen(“student.dat”,”w”);
while(p)
{
printf(“студент туралы мәліметтерді енгіз”);
printf(“\n Аты –жөніненгіз”);
scanf(“%s”,&fio);
printf(“\n топ номерін енгіз”);
scanf(“%s”,&gr);
printf(“\n стипендияны енгіз”);
scanf(“%s”,&st);
fprintf(lf,“%15s%6s%6.2f ”,fio,gr,st);
printf(“\n Бітті ме 0-ия, 1-жоқ”);
scanf(“%s”,&p);
}
fclose (lf);
handle = open(“student.dat”,O_CREAT);
l=filelength(handle);
close(handle);
n=1;
lf=fopen(“student.dat”,”r ”);
while(n)
{
printf (“жазба номерін енгіз”);
scanf (“%d”,&n);
offset=(n-1)*28;
if (offset>=1 | | n==0) continue;
fseek (lf,offset,0);
fscanf (lf,”%s %s %f ”,&fio,&gr,&st);
printf (“%15s%6s%6.2f\n”,fio,gr,st);
}
fclose(lf);
}
```

10 Символдық жолдармен жұмыс

СИ тілінде символдар жолын енгізуге және шығаруға арналған екі ыңғайлы puts және gets функциялары бар. Оларды пайдалану мысалын қарастырайық:

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

```
char q[40]; /*символдар жолын сипаттау*/
puts(“Символдар жолын енгізіңіз ”);
gets(q); /*символдар жолын енгізу */
puts(q); /* символдар жолын шығару */
}
```

Программаның жұмысы нәтижесінде экранда алдымен келесі мәтін пайда болады:

Символдар жолын енгізіңіз

Содан кейін кез келген символдық жолды енгізу қажет. Енгізілген жолдың әрбір символы gets операторының көмегімен q массивінің элементтеріне меншіктеледі. Символдарды puts операторы шығарады.

10.1Символдық жолдар және олардың жадыда орналасуы

Символдық жол – бұл char типті массив болып табылады. Жол бос литерамен, яғни (‘\0’) символымен аяқталады. Бұл дегеніміз ешқандай символ тұрмағанын білдіреді. Компилятор бұл символды автоматты түрде жолдың соңғы символынан кейін қояды. Мысалы «Сәлем!» символдық жолы жадыда келесідей орын алады:

С	ә	л	е	м	!	\ 0

Символдық жол үшін жадыны дайындағанда бір байтты жолдың аяқталу белгісіне жіберу керектігін ұмытпау қажет.

Символдық жолдарды инициализациялау тәсілдері

1 тәсіл. char stroka [ ] = “Сәлем!”;

2 тәсіл. char stroka [ ] = {‘С’, ‘ә’, ‘л’, ‘е’, ‘м’, ‘!’, ‘\0’};

Осындай инициализацияда міндетті түрде ‘\0’ - бос литераны қосу қажет.

3 тәсіл. char *str; str = ” Сәлем!”; немесе char *str = “Сәлем!”;

Бұл тәсілде көрсеткіш пайдаланылады. Есть и третий способ инициализации с помощью указателей. При этом, как и в двух выше описанных случаях, не нужно заранее выделять память для строки. Компилятор по внешнему виду строки определит необходимое количество байтов, в том числе один байт на символ конца строки. Например:

Это же самое можно записать так:

Иной путь выделения памяти заключается в явном ее задании. Во внешнем описании мы могли бы указать;

char m1[13] = “одна_строка”;

В этом случае все неиспользованные элементы автоматически инициализируются нуль–символом, как показано на рис.2.

Рис.2.

2. Ввод и вывод символьных строк

В лабораторной работе №1 уже упоминалась функция gets ( ), считывающая строки. Она получает строку от стандартного устройства ввода, которым является клавиатура. Поскольку строка не имеет заранее заданной длины, функция gets ( ) должна знать, когда ей прекратить работу. Функция читает символы до тех пор, пока ей не встретится символ новой строки (‘\n’), который создается при нажатии клавиши Enter. Функция берет все символы до (но не включая) символа новой строки, присоединяет к ним нуль–символ (‘\0’) и передает строку вызывающей программе. Если все идет хорошо, функция gets ( ) возвращает считанную строку. Если что-то неправильно или встретился символ EOF, то она возвращает символ NULL или нулевой адрес.

Рассмотрим пример считывания и вывода строки.

```
#include < stdio.h >
#include < string.h > /* Подключение файла string.h для работы со строками*/
main ( )
{ char name [ 20 ]; /*Выделение памяти*/
gets ( name ); /*Размещение введенной информации в строку name*/
printf ( “%s”, name); /*Вывод строки*/
}.
```

В приведенном примере для вывода строки использована функция printf ( ). Для вывода строк чаще всего используют функцию puts ( ). Разница между функциями состоит в том, что

printf ( ) не выводит автоматически каждую строку текста с новой строки. Если это потребуется, то нужно соответствующее указание. Так,

```
printf (“%s\n”, string);
```

дает то же самое, что и

```
puts ( string ) ;
```

Первый оператор требует ввода большого числа символов и большего времени при выполнении на компьютере. Рассмотрим пример.

```
#include < stdio.h >
#include < string.h >
main ( )
{ static char str1[ ] = “массив инициализирован”;
char *str2 = “Указатель инициализирован”;
puts ( str1 );
puts ( str2 );
puts ( &str1[4] );
```

```
puts ( str2+4);
}.
```

В результате работы получаем:

массив инициализирован
указатель инициализирован
ив инициализирован
атель инициализирован

Обратите внимание на два последних оператора. Указатель &str1[4] ссылается на пятый элемент массива str1. Этот элемент содержит символ ‘и’, и функция puts ( ) использует его в качестве начальной точки. Аналогично str2+4 ссылается на ячейку памяти, содержащую ‘а’, и с которой начинается вывод строки.

1.3. Функции над строками

В языке Си существует много функций работы над строками. Рассмотрим некоторые из них.

1.3.1. Функция strlen (str) вычисляет длину строки. Тип результата unsigned (беззнаковый) .

1.3.2. Функция strcat (str1, str2) приписывает строку str2 к строке str1, и это объединение становится новой первой строкой. Функция strncat (str1, str2, kol) приписывает kol символов строки str2 к строке str1. Например,

```
main ( )
{ static char str1[ ] = “СТРОКА”;
char *str2 = “_СИМВОЛОВ”;
strcat ( str1, str2 );
puts ( str1);
strncat (str1, str2, 3);
puts ( str1);
}.
```

В результате работы программы получим:

СТРОКА_СИМВОЛОВ
СТРОКА_СИМ

1.3.3.Функция strcpy (str1, str2) копирует строку str2 в строку str1.

Пример.

```
main ( )
{static char str1[ ]=”СТРОКА”;
char *str2=”СИМВОЛОВ”;
strcpy (str2,str1);
puts (str2);
strncpy (st1,st2,3);
puts (st1);
}.
```

В результате работы программы получим

СТРОКА
СИМОКА

При работе strcpy информация из str1 скопировалась в str2. Строка, куда копируется информация, должна быть не меньше той строки, из которой информация копируется. Функция strncpy скопировала 3 первых символа из строки st2 в строку st1. Но поскольку признак конца строки стоит после буквы А, будет получен приведенный выше результат. Если требуется закончить строку на трех скопированных символах, то необходимо после третьего символа (по счету он будет вторым , т.к. отчёт начинается с нуля) поставить признак конца строки. Для этого в программу перед последним puts требуется ввести оператор st1[3]='\0' ;

1.3.4. Функция strchr (str,c) находит в строке str первое вхождение символа, определяемое параметром С. Функция strrchr (str,c) находит в строке str, последнее вхождение символа, определяемое параметром С.

Пример,

```
main ( )
{static char st1[ ] = ”СТРОКА”;
char  *ptr, c =’р’
ptr =strchr (st1,c);
if ( ptr )
printf(“%d”,ptr-st1);
else
printf (“нет символа”);
}.
```

В результате работы программы будет выведено число 2. Символ ‘р’ стоит на втором месте, т.к. отсчёт начинается с нуля.

Переменной С присвоено значение символа, который должен быть найден в строке. Указатель ptr после выполнения функции strchr получает либо значение адреса, где содержится искомый символ, либо его значением становится NULL, если символ не найден. Поскольку st1 является адресом первого символа строки, то ptr-st1 покажет местоположение символа в строке. Функция strrchr работает аналогично функции strchr.

1.3.5.Функция strpbrk(str1,str2) находит в строке str1 первое появление любого из множества символов, входящих в строку str2.

Пример,

```
main ( )
{ static char st1[ ]=”СТРОКА”;
char *ptr;
char *st2 = ”рок”;
ptr  = strpbrk (st1,st2);
if (ptr)
printf (“%d”,ptr-st1);
else
printf (“нет символов”);
}.
```

Результатом работы программы будет число 2.Эта программа отличается от предыдущей лишь тем, что поиск ведётся, исходя из множества символов инициализированных в нашей программе с помощью указателя st2.

1.3.6. Функция strset (str,c) заменяет все символы строки str на значение, определяемое параметром С.

Функция strnset(str,c,kol) заменяет kol символов строки str на значение, определяемое параметром С.

Пример,

```
main ( )
{ static char st1[ ]=”СТРОКА”;
{ static char st2[ ]=”СИМВОЛ”;
char letter = ’x’;
strset (str1, letter);
puts (str1);
strnset (str2, letter,3);
puts (str2);
}.
```

В результате работы программы будет получено:

```
XXXXXX
XXXВОЛ
```

1.3.7.Функция strcspn (st1,st2) определяет местоположение символа строки st1, который первым совпал с одним из символов строки st2.

Пример,

```
main ( )
{ static char str1[ ]=”СИМВОЛ”;
{ static char str2[ ]=”ВОЛ”;
int n;
n=strcspn (str1,str2);
printf (“%d”,n);
}.
```

В результате работы программы будет получено значение переменной n равное трём. Это произошло потому, что первым совпавшим символом в строках str1 и str2 является символ ‘В’ , а он стоит на третьем месте (отсчёт начинается с нуля).

Функция strspn (st1,st2) определяет местоположение символа строки st1 , который первым не совпал с одним из символов строки st2.

Пример,

```
main ( )
{ char *str1=”1234567890”;
  char *str2=”123458”;
int n;
n=strspn (str1,str2);
printf (“%d”,n);
}.
```

В результате работы программы будет получено значение переменной n, равное пяти. Такой результат получен потому, что первым не совпавшим символом в строках str1 и str2 является символ ‘6’, а он стоит на пятом месте (отсчёт от нуля).

1.3.8. Функция strrev (str) меняет порядок следования символов на противоположный.

Приведём примеры с использованием расмотренных выше функции.

Пример 1.Подсчитать количество символов ‘a’ в строке.

```
main ( )
{char *str;
int n,k,i;
gets (str); /*ввод  строки */
n = strlen (str); /* определение количества символов в строке*/
for (k=0, i=0; i<n; i++) /*подсчёт количества символов ‘a’*/
if (str[i]==’a’) k=k+1;
printf (“%d”,k); /*вывод количества символов*/
}.
```

Пример 2.Решить задачу из примера 8, применив указатель.

```
main ( )
{char *str;
gets (str);
while (*str!=’\0’); /*проверка на наличие конца строки*/
if (*str==’a’) k=k+1; /*подсчёт количества символов ‘a’*/
str++;  /*смещение указателя на очередной символ*/
puts(str);
}.
```

Пример 3. В примере демонстрируется вывод разных частей строки.

```
main ( )
{static char st [ ] = “До встречи”;
char *ptr;
ptr = st;
puts ( str ); /* вывод строки*/
puts ( ++ptr ); /* смещение указателя на первый символ строки и вывод строки*/
st [ 7 ] = ‘ \0 ’; /* признак конца строки помещается на место седьмого символа строки */
puts ( st ); /* вывод получившейся строки */
puts ( ++ptr );  /* смещение указателя с первого символа нп второй и вывод строки*/
}.
```

В результате работы программы будет выведена следующая информация:

До встречи
о встречи
до_встр
_встр

Пример 4.

```
main ( )
{static char st [ ] = “ СТРОКА ”;
char *ptr;
ptr = st + strlen ( st ); /* установка указателя на конец строки*/
while ( -- ptr >-=st ) /* смещение указателя с первого символа на второй и вывод строки*/
puts ( ptr );
}.
```

Результат работы программы :

А
КА
ОКА
РОКА
ТРОКА
СТРОКА

Пример 5. Определить первый пробел в строке и вывести строку с этой позиции и до конца. Если нет ни одного пробела, то выдать об этом сообщение.

```
main ( )
{ char *st;
gets ( st ); /* ввод строки*/
while ( st! = ‘_’&&*st! = ‘\0’)
st++;  /* останавливается на первом пробеле или нуль-символе*/
if ( st == ‘\0’)
printf ( “нет пробела”);
else
puts ( st );
}.
```

Литература

1. Подбельский В.В. Язык Cu ++: Учебное пособие. - М.: Финансы и статистика,1995, - 560 с.
2. Страуструп Б. Язык программирования Сг ++. - М.: Радио и связь, 1991. - 352 стр.
3. Собоцинский В.В. Практический курс Turbo Cu ++. Основы объктно-ориентированного программирования. - М.: Свет, 1993. - 236 с.
4. Романов В.Ю. Программирование на языке Cu ++. Практический подход. - М.: Компьтер, 1993. - 160 с.

5. Уинер Р. Язык турбо Cu . - М.: Мир, 1991. - 384 с.
6. Юлин В.А., Булатова И.Р. Приглашение к Cu. - Мн.: Высш. Шк., 1990,- 224 с.
7. Котлинская Г.П., Галиновский О.И. Программирование на языке Cu. -Мн.: Высш. Шк., 1991. - 156 с.