К.ТЫНЫСТАНОВ АТЫНДАГЫ ЫСЫК-КӨЛ МАМЛЕКЕТТИК УНИВЕРСИТЕТИ

Информациялык технологиялар жана программалоо кафедрасы

Эркинбаев М.А.

Си тилинде программалоонун негиздери

Каракол 2012

УДК 004
ББК 32.973-01
Э 78

ЫМУнун Окуу-методикалык бирикмеси (29.12.2012 ж. №4 токтому), Окумуштуулар кеңеши (30.12.2012 ж. №4 токтому) тарабынан басмага сунуш кылынды

Рецензент: физ.мат. илим. док., доцент Салиев А.Б.,
тех. илим. канд. доцент Искаков Р.Т.

Эркинбаев М.А.

Э 78 Си тилинде программалоонун негиздери/ К.Тыныстанов атындагы ЫМУ. -Каракол, 2012 — 138 бет.

ISBN 978-9967-454-00-2

Бул окуу куралы Си тилинде программа түзүүдө эң зарыл болгон түшүнүктөрдү жана каражаттарды камтыды. Си тилинде программа түзүүнү жаңыдан баштаган үйрөнчүктөр үчүн жөнөкөй жана түшүнүктүү тилде жазылды. Аталган окуу куралын жогорку класстын окуучуларынан баштап «Колдонмо математика жана информатика», «Эсептөө техникаларынын жана автоматташтырылган системалардын программалык жабдылыштары» адистиктеринин студентери колдонсо болот. Ошондой эле окутуучуларга да арналды.

Э 2404090000-12
978- 9967-454-00-2

УДК 004
ББК 32.973-01

Киришүү

Азыркы учурда Си тили кеңири тараган программалоо тилдеринин бири болуп эсептелет. Тилдин аталышы латын арибинин үчүнчү тамгасына туура келет. Өткөн кылымдын 70-жылдарында жаңы иштелип чыккан программалоо тилдеринин аталышын латын тамгалары менен берүү күч алган. Мисалы APL (A Programming Language) - А программалоо тили деп латын арибинин биринчи тамгасы алынган. Ал эми PDP-7 чакан компьютери үчүн UNIX аракеттер системасынын үстүнөн иштөө учурунда B программалоо тили түзүлгөн, бул латын арибинин экинчи тамгасы эле. 1972-жылы Деннис Ритчи жана Кен Томпсон тарабынан C тили түзүлгөн, бул латын арибинин үчүнчү тамгасы. Си тили жогорку деңгээлдеги программалоо тилдеринин жана ассемблер тилинин эң жакшы касиеттерине ээ болду. Жогорку деңгээлдеги программалоо тилдеринен киргизип-чыгаруунун каражаттарын, татаал түзүлүштөгү маалыматтар менен иштөө мүмкүнчүлүгүн жана башкаруучу түзүлүштөрдүн кеңири жыйындысын алса, ал эми ассемблер тилинен компьютердин эси менен иштөөчү ийкемдүү жана эффективдүү каражаттарды мурастаган.

Си тилинин биринчи китеп катары баяндалышы. 1978-жылы Брайн Керниган жана Деннис Ритчи тарабынан «C программалоо тили» деген аталыш менен жарык көргөн. Бул китепти «K&R» стандарты катары карашкан. 1982-жылы Америкалык Улуттук Стандарттар Институту Си тилинин стандартын кабыл алган, ал ANSI C деп аталган. 1989-жылы стандарттоо боюнча эл аралык группа ANSI C стандартына көп эмес өзгөртүүлөрдү киргизүү менен C89 стандартын кабыл алышкан, 1995-жылы C89 стандартына жаңы библиотекаларды кабыл алуу менен C95 стандартын жарыкка чыгарышкан. Эл аралык ISO/IEC стандарты 1999-жылы C99 стандартын кабыл алышкан. Бул стандартта C89, C95 стандарттарына бир кыйла толуктоолор кийирилген жана төмөнкү элементер менен толукталган.

- Комплексттик арифметика
- Маалыматтардын узун бүтүн типтери
- Өзгөрмөлүү узундуктагы массив
- Булдун тиби
- Англисче эмес символдордун тобу
- Калкыма чекитүү типтердин жакшыртылышы

Кийинки версиясы катары Колифорниялык фирма «Борланд» (Borland International) IBM PC компьютерлери үчүн Турбо Си компиляторун иштеп чыгышты.

1980-жылдардын башында Bell Laboratory корпорациясында Б.Страуструп тарабынан Си тили кеңейтилип иштелип чыгып «Си класстары менен» деп аталып 1983-жылы «C++» деген жаңы ат менен аталып Си тилинин объектке-багытталган версиясы болуп калды. 1990-

жылдары Microsoft компаниясы Microsoft C++, Visual C++ ошол эле жылдары Borland фирмасы Turbo C++, Borland C++ жана C++ Builder тилдерин иштеп чыгышкан. 2000-жылы Microsoft компаниясы C# (Си шарп) деген жаңы тилди иштеп чыкты.

Си тилинин мүнөздөмөлөрү.

Си тили жалпы арналыштагы тил. Си тили программа түзүүдө колдонуунун ийкемдүүлүгүн жана эффективдүүлүгүн камсыз кылат. Си тилинде программа тез иштейт. Тилдин тыкыр аныкталышы аракеттер системасынан же компьютердин тетиктеринен көз каранды эместигин тастыктайт. Си тили структуралык программалоонун операторлорунун толук жыйындысын жана амалдардын ар түрдүү тобун кармайт. Си тилинин көптөгөн амалдары, машиналык командалар менен дал келет, ошондуктан түздөн түз машиналык кодко трансляциялоо жүргүзүлөт.

Си тили функцияга жана өзгөрмөгө болгон көрсөткүчтү кармайт. Программалык объектке болгон көрсөткүч, ошол объектинин машиналык адресине дал келет. Көрсөткүчтөрдү кылдаттык менен пайдалануу, ийкемдүү аткаруучу программаны түзүү мүмкүнчүлүгүн берет. Си көрсөткүчтөрдүн арифметикасын кармагандыктан компьютердин эсинин адрестери менен иш аракеттерди түздөн түз аткаруу мүмкүнчүлүгүн берет. Си тили өзүнүн курамында алгачкы процессору (препроцессор) болгондуктан, компиляциялоо алдында тексттик файлдарды иштет.

Си тилинин жакшы жактары.

1. Эффективдүүлүгү. Си тилинде аткарылган программанын тездиги жана компактуулугу Ассемблер тилинде жазылган программага жакын.
2. Кубаттулугу. Си тили жаңы башкаруу түзүлүштөрүнүн жана маалыматтарды берүүнүн жолдорунун чоң тобуна ээ.
3. Структуралык программалоо технологиясына ээ. Си тили структуралык программалоонун операторлорунун баарын камтыйт.
4. Модулдук программаны иштеп чыгуу мүмкүнчүлүгүн кармайт.
5. Ийкемдүүлүгү. Си тилинде түзүлгөн программа башка компьютерлерге, башка аракеттер системасына жеңил которулат.
6. Тездиги жана кыскалыгы. Си тилинде түзүлгөн программа башка тилдеги программаларга караганда кыска түзүлөт да тез аткарылат.

Си тилинин кемчиликтери.

1. Программисттен жогору квалификацияны талап кылат.
2. Программистке эң чоң эркиндик берет, мисалы тиби жок программалоо. Бул мүмкүнчүлүктү туура эмес пайдалануу көптөгөн катачылыкка алып келет

1-Бөлүм. СИ ТИЛИНИН ЭЛЕМЕНТТЕРИ

1.1 Си тилинин негизги түшүнүктөрү

Си тилинин ариптери. Ариптер катары ASCII коддоо таблицасынын символдорунун тобу кирет.

Ысымдар. Си тилинде ысымдар менен өзгөрмөлөрдү, функцияларды, белгилерди белгилешет. Ысымдар төмөндөгү эрежелерди эске алуу менен түзүлөт:

•Ысымдар латын ариптеринен, цифралардан жана астынкы сызык «_» белгисинен түзүлөт.

•Библиотекалык функцияларда, системдик өзгөрмөлөрдө жана турактууларды белгилөөдө биринчи белгиси «_» астынкы сызык менен башталгандыктан, ысымдарды бул белги менен белгилөөдөн качуу керек.

•Си тилинде латын тамгаларынын баш тамгалары жана кичине тамгалары айырмалангандыктан name жана NAME ысымдары ар башкача болот.

•Ысымдардын узундуктары Си тили үчүн биринчи 8 белги алынса, Турбо Си үчүн биринчи 32 белги алынат, ал эми C++ үчүн мындай чектөө жок.

Си тилинин алгачкы процессору

Си тилинин алгачкы процессору башка программалар үчүн кирүү маалыматтарын иштете турган программа болуп эсептелет жана программаны компиляциялоого чейин керектүү файлдарды издеп таап аларды ишке киргизет. Алгачкы процессор макроаныктоолорду жардамы менен текстерди аныктайт бул #define- буйругу менен иштетилет жана файлдарды программага киргизүүнү ишке ашырат бул #include- буйругу менен жүргүзүлөт. Алгачкы процессордун #define- буйругуна токтололу. Бул буйрук турактууларды аныктоодо жана макроаныктоолорду түзүүдө колдонулат. Турактууларды аныктоонун жалпы жазылышы:

1. #define < турактуунун ысымы> < турактуунун мааниси>
2. #define < турактуунун ысымы> < турактуунун ысымы>
3. #define < турактуунун ысымы> < турактуулардан түзүлгөн туюунтма>

Мисалы:

```
#define NOL  0
#define  X    NOL
#define  BIR    1
#define  TUI    10*BIR+5
```

Макроаныктоолорду түзүүдө #define- буйругунун жалпы жазылышы:

#define < ысым > < алмаштыруучу сап >

< ысым >- макростун ысымы, чакан программаны макрос деп аташат.

< алмаштыруучу сап >- турактуу же туюнтма болушу мүмкүн.

Макроаныктамалар параметрлүү же параметрсиз болуп кезигиши мүмкүн, параметрсиз макроаныктамалар турактууларды аныктоо менен дал келишет.

Параметрлүү макроаныктамаларга мисалдар:

```
#define  max(x,y)    x>y?x:y
#define  min(x,y)    x>y?y:x
```

Алгачкы процессордун #include- буйругу программаны компиляциялоого чейин стандарттык киргизип-чыгаруу(stdio.h), математикалык стандарттык функцияларды(math.h), динамикалык өзгөрмөлөрдү уюштруудагы функцияларды(alloc.h), саптарды иштетүүчү функцияларды(string.h) жана экран менен клавиша түзүлүштөрү менен иштөөчү функцияларды(conio.h) ишке киргизүүчү файлдарды иштетет.

Мисалдар:

```
#include < stdio.h >
#include < math.h >
#include < alloc.h  >
#include < string.h >
#include < conio.h >
```

Си тилиндеги программанын түзүлүшү.

Си тилинде программа функциялардын тобунан турат жана программа негизги функциялардан башталат. Негизги функциянын ысымы main кыргызча которгондо башкы, негизги дегенди түшүндүрөт. Программаны аткаруу негизги функциялардан башталат. Си тилиндеги программада main ысымдуу бир гана функция болушу керек.

Си тилиндеги программанын жөнөкөй структурасы (түзүлүшү)

```
/*түшүндүрмө*/                    1-сап
#include <stdio.h>                2-сап
main (  )                         3-сап
{                                 4-сап
<ички баяндоолор>;                5-сап
<иш аракеттер>;                   6-сап
}                                 7-сап
```

1-сапта /*түшүндүрмө*/ - Си тилинде түшүндүрмөлөр программанын окулушун жана көрсөтмөлүүлүгүн жогорулатуу үчүн колдонулат. Түшүндүрмөлөр программанын каалаган жерине жазыла берет. Түшүндүрмөлөр /*жана */ белгилеринин ортосунда жазылат.

2-сапта алгачкы процессордун буйругу б.а. программа компиляция болоор алдындагы иш аракеттерди жүргүзөт.

3-сапта негизги функциянын ысымы, ысымдан кийин тегерек каша сөзсүз коюлушу керек.

4-сапта «{» ачылган фигуралык кашаа негизги функциянын тулкусунун башталышы. Паскаль тилиндеги биринчи begin менен дал келишет.

5-сапта <ички баяндоолор> программада колдонулган өзгөрмөлөрдөн типтерин, көрсөткүчтөр ж.б. баяндайт.

6-сапта <иш аракеттер> операторлор, функциялар жазылат.

7-сапта «}» жабылган фигуралык кашаа негизги функциянын аякталышын билгизет.

Мисалы:

```
/* Менин Си тилиндеги биринчи программам */   1-сап
# include <stdio.h>                            2-сап
main (  )                                      3-сап
{                                              4-сап
printf («Саламатсызбы! \n»);                   5-сап
}                                              6-сап
```

Жогорку программада 1-сабы түшүндүрмөнү, 2-сабы алгачкы процессордун директивасын, 3-сабы негизги функцияны, 4-сабы негизги функциянын тулкусунун башталышын, 5-сап чыгаруу функциясын, ал эми 6-сабы негизги функциянын аякталышын кармайт.

Бул программанын жыйынтыгы Саламатсызбы! деген сөздү экранга чыгарат жана жылгыч (курсор) жаңы саптын башына келтирилет.

1-сап: Түшүндүрмө берүүчү сап.

2-сап: Алгачкы процессордун буйруктарын берет.

Алгачкы процессор-программа компиляция болоор алдында, бир канча системдик буйруктарды аткаруучу системалык программа.

Компиляция - бул транслятордун бир түрү болгон компилятордун иштөө процесси.

Программист жазган программа ***баштапкы модуль*** деп аталат.

Машиналык буйрукка айланган программа ***объектик модуль*** деп аталат.

Кураштырылгандан кийинки программа б.а. курагыч(компановка) иштегенден кийинки пайда болгон программа .exe же .com кеңейишиндеги болуп ***аткаруучу*** же ***жүктөлүүчү модуль*** деп аталат.

Мисалы: Эки сандын суммасын чыгаруу.

```
/* эки сандын суммасы  */
# include <stdio.h>
main ( )
{
int a, b, sum;
printf( " \n Эки бүтүн санды киргиз \n" );
scanf( " %d,%d ", &a, &b );
sum=a+b;
printf( " a+b=%d  \n  ", sum);
}
```

Жыйынтык: Эки бүтүн санды киргиз

3,5

a+b=8

Математикалык стандарттык функциялар

sin(x) - синус (аргументи радиан менен);

cos(x) - косинус (аргументи радиан менен);

tan(x) - тангенс (аргументи радиан менен);

asin(x) - арксинус (жыйынтык радиан менен);

acos(x) - арккосинус (жыйынтык радиан менен);

atan(x) - арктангенс (жыйынтык радиан менен);

sinh(x) - гиперболикалык синус;

cosh(x) –гиперболикалык косинус;

tanh(x) - гиперболикалык тангенс;

log10(x) - ондук логарифм;

pow10(x) -10 санын x даражасына көтөрүү;

log(x) - натуралдык логарифм;

exp(x) – e санын x даражасына көтөрүү;

sqrt(x) – x санынан квадраттык тамыр чыгаруу;

pow(x,y) - x санын y даражасына көтөрүү;

fabs(x) – кош тактыктагы x санынын абсолюттук чондугу;

abs(x) - x бүтүн санынын абсолюттук чондугу.

Математикалык стандарттык функцияларды колдонуш үчүн программага #include <math.h> сабын жазуу керек

Мисалы:

```
#include <stdio.h>
#include <math.h>
main()
{
int x , y;
x = 2;
y = 3;
printf("%d нин  %d-даражасы барабар %d\n", x, y, int(pow(x, y)));
}
```

Жыйынтык: 2 нин 3-даражасы барабар 8

1.2 Өзгөрмөлөр жана турактуулар.

Өзгөрмө жана турактуулардын негизин чоңдук түшүнүгү түзөт. Чоңдуктар программалоо тилинде абдан көп кезиге турган термин.

Чоңдуктар – бул кандайдыр бир же сандык, же символдук, же логикалык маанилерди кармаган түшүнүк. Чоңдуктар эки түргө бөлүнөт: өзгөрмө жана турактуу.

Туракттуулар - Си тилинде туракттуулар типтерге бөлүнөт: бүтүн типтүү туракттуу(integer), символдук типтеги(char-character) туракттуу, калкыма же өзгөрүлмө чекитүү(floating point) туракттуу, саналуучу(enumeration) туракттуу жана сап(string) туракттуусу.

Бүтүн типтүү туракттуулар төмөндөгү таблица менен берилди.

Тиби	Өлчөмү	Маанилер аралыгы
integer	2 байт	-32768ден 32767ге чейин
unsigned integer	2 байт	0дөн 65535ке чейин
long integer	4 байт	-2147483648ден 2147483647ге чейин
unsigned long integer	4 байт	0дөн 4294967295ке чейин

Бүтүн типтүү туракттуулар кыскача (int) сөзү менен программанын текстинде ондук, сегиздик жана он алтылык эсептөө системасында жазылышы мүмкүн. Туракттуулар узун бүтүн туракттуулар да боло алышат. Узун бүтүн туракттуулар татаал эсептөөлөрдү жүргүзүүдө колдонулат.

Калкыма же өзгөрүлмө чекитүү(floating point) туракттуулар төмөндөгү таблица менен берилди.

Тиби	Өлчөмү	Маанилер аралыгы
float	4 байт	3.4E-38ден 3.4E+38ге чейин
double	8 байт	1.7E-308ден 1.7E+308ге чейин
long double	8 байт	1.7E-308ден 1.7E+308ге чейин

*Калкыма чекиттүү туракттуулар***:** ондук бөлчөктүн бүтүн бөлүгүнөн анын бөлчөк бөлүгүн ажыратуучу чекиттин абалын

өзгөртсөк ал калкыма чекит деп аталат. Мисалы: 24.0 саны берилсе анны калкыма чекиттүү кылып жазсак болот 2.4E+1 же 240.0E-1 ж.у.с.

Символдук типтеги туракту улар.

Бул типтеги туракту улар тырмакча(‘) белгисине алынып жазылат (‘a’, ‘2’) жана ASCII коддоо таблицасындагы баардык символдор кирет. Бул символдор шарттуу түрдө көрүнүүчү жана көрүнбөөчү болуп эки топко бөлүнөт.

Көрүнүүчү символдук туракту уларды экранга же кагазга чыгарууга болот. Көрүнбөөчү символдук туракту улар графикалык көрүнүшкө ээ болбойт. Мисалы: сапты которуу символу, үн чыгаруу символу ж.у.с.

ASCII коддоо таблицасында 0 дон 31 ге чейиннки номердеги символдор башкаруучу символдор болуп эсептелет.

Бул башкаруучу символдор Си программасында көрүнбөс символ катары колдонулат.

Буларды кээде башкаруучу коддор деп да аташат. Бул башкаруучу коддор \ белгисинен кийин жазылат. Мисалы:

\a үн чыгарууну берет

\b бир кадамга артка аракетин аткарат

\f жаңы бетке өтөт

\n сапты которот

\t горизанталдык таблицаны жүргүзөт

\v вертикалдык таблицаны жүргүзөт

Саналуучу(enumeration) туракту улар маңызы боюнча бүтүн типтүү туракту улар болушат. Мисалы:

enum {kүn,dүishondү,sheishenbi,sharshenbi,beishenbi,juma,ishenbi,jekshenbi};

Бул баяндоодо kүn деген туракту 0 туракту̀усун билдирет, dүishondү- деген туракту 1 туракту̀усун билдирет, sheishenbi- деген туракту 2 туракту̀усун билдирет, sharshenbi- деген туракту 3 туракту̀усун билдирет, beishenbi- деген туракту 4 туракту̀усун билдирет, juma- деген туракту 5 туракту̀усун билдирет, ishenbi- деген туракту 6 туракту̀усун билдирет, jekshenbi- деген туракту 7 туракту̀усун билдирет.

Сап туракту̀улары экиден ашык символдордун удаалаштыгынан турат жана кош тырмакчага(“) алынып жазылат

Мисалы; “Си тили”, “КПМ-31 тайпасы”.

Өзгөрмөлөр - бул программанын иштөө процесинде өзүнүн маанисин өзгөртүүчү чоңдукту айтабыз. Өзгөрмө ысымга жана мааниге ээ болот. Өзгөрмөлөрдүн ысымдары латын тамгаларынан, цифралардан жана «_» төмөнкү сызык белгисинен турат. Ысым сөзсүз түрдө латын тамгасынан башталышы керек. Өзгөрмөнүн ысымы анын мааниси жайгашкан эстеги орунду аныктайт. Өзгөрмө статикалык жана динамикалык болуп эки түргө бөлүнөт.

Статикалык өзгөрмөлөр программанын аткарылыш процессинде учурдагы эстен алган ордун туруктуу кармап турат жана программа аткарылып бүткөнчө өзүн жоготпойт. Динамикалык өзгөрмөлөр-деп программанын аткарылыш процессинде бирде пайда болуп, кайра жоюлуп туруучу өзгөрмөнү түшүнөбүз.

1.3 Маалыматтардын типтери.

Си тилинде колдонулуучу маалыматтар типтерге ээ болушат. Типтер негизги жана туунду типтер болуп эки топко бөлүнүшөт. Негизги типтерге char – символдук, int – бүтүн, float – чыныгы,

double – кош тактыктагы чыныгы сандар, void – куру типтери кирет. Бул типтердин кыскача мүнөздөмөлөрү :

1. char – тибиндеги маалымат 1 байт өлчөмдө болуп, мааниси ASCII кодоо таблицасындагы символдор эсептелет. Мисалы: ‘a’, ‘k,’ ‘z’, ‘+’, ‘*’, ‘ф’ программада тырмакчага алынып жазылат

2. int (integer) – кызматчы сөзү менен мааниси бүтүн сан болгон өзгөрмөлөр баяндалат. Бүтүн типтеги маалымат 2 байт өлчөмдө болуп маанилери -32768 ден 32767 чейинки бүтүн сандарды алат.

3. float – кызматчы сөзү менен мааниси чыныгы сан болгон өзгөрмөлөр баяндалат.

Чыныгы типтеги маалыматтар 4 байт өлчөмдө болуп, маанилери 3.4E-38ден 3.4E+38ге чейин чыныгы сандарды камтыйт.

4. double – кызматчы сөзү менен мааниси кош тактыкта чыныгы сан болгон өзгөрмөлөр баяндалат. Кош тактыктагы чыныгы сандар 8 байт өлчөмдө болуп, маанилери 1.7E-308ден 1.7E+308ге чейинки сандар эсептелет.

5. void – кызматчы сөзү менен куру маанини баяндайт жана төмөндөгү учурларда колдонулат:

•Маанисин кайтарып бербеген функциялардын тибин көрсөтүүдө.

•Мааниси түздөн түз колдонулбаган туюнтмалардын тибин баяндоодо.

Жогорку негизги типтерди өркүндөтүүгө болот, ал үчүн төмөнкү кызматчы сөздөр колдонулат: unsigned, short, long. Өркүндөтүүчү сандар типтерди баяндоочу сөздөрдөн мурун жазылат. Мисалы: unsigned char, short int, long int. Бул типтерди жана алардын өркүндөтүлүшүн төмөндөгү таблица менен берсек болот:

Тиби	Байт боюнча өлчөмү	Маанилердин аралыгы
char	1	-128ден 127ге чейин
unsigned char	1	0дөн 255ке чейин
int	2	-32768ден 32767ге чейин
unsigned int	2	0дөн 65535ке чейин
long int	4	-2147483648ден 2147483647ге чейин
unsigned long int	4	0дөн 4294967295ке чейин
float	4	3.4E-38ден 3.4E+38ге чейин
double	8	1.7E-308ден 1.7E+308ге чейин
long double	10	3.4E-4932ден 3.4 E+4932ге чейин

1.4 Типтерди өзгөртүү

Информацияларды Си тилинде иштетүүдө маалыматтардын типтерин өзгөртүү процесстери жүрөт. Типтерди өзгөртүүнүн үч түрү каралган: айкын эмес, арифметикалык жана айкын өзгөртүп түзүүлөр.

Типтерди айкын эмес өзгөртүп түзүү

Бул өзгөртүп түзүү камтылган программалар аткарылган учурда кездешет. Камтылган программада формалдуу жана анык параметрлер түшүнүктөрү бар. Айкын эмес өзгөртүп түзүүсү анык параметрлерди дал келишкен формалдуу параметрлердин типтерине өзгөртүү жүргүзүлгүн учурда аткарылат. Айкын эмес өзгөртүп түзүүдө мүмкүн болгон типтерди өзгөртүү төмөндөгү таблица менен берилет.

Өзгөртүлүүчү типтер	Түздөн түз алмаштырылуучу типтер
char	int, short int, long int
int	short int, long int, char, float, double
short int	int тибине окшош
long int	int тибине окшош
float	double, int, short int, long int
double	float, int, short int, long int

Арифметикалык өзгөртүп түзүү

Си тилинде аткарылуучу амалдардын көпчүлүгү типтерди өзгөртүп түзүшөт, себеби амалдардагы операндаларды жалпы бир типке келтирүү жүргүзүлөт. Операнда-бул арифметикалык амал белгисинин эки жагындагы орунду билгизет. Си тилинин амалдарды аткарган өзгөртүп түзүүлөр амалдардын өзгөчөлүгүнөн жана операндалардын типтеринен көз каранды болушат. Туюнтмаларды эсептөө учурунда типтердин өзгөртүп түзүүлөр жүргүзүлөт. Мындай өзгөртүп түзүүлөрдү арифметикалык деп атайбыз. Арифметикалык өзгөртүп түзүүлөр төмөнкү схема менен жүргүзүлөт.

- char, enum, short типтери int тибине, float тиби double тибине өзгөртүп түзүлөт

- эгерде операндалардын биринин тиби double болсо анда экинчи операнда дагы double тибине, ал эми биринчи операнда long double болсо, экинчи операнда дагы long double тибине өзгөртүлүп берилет.

- менчиктөө операторунда оң жагындагы эсептелинген натыйжанын тиби сол жагындагы өзгөрмөнүн тибине өзгөртүлөт.

Айкын өзгөртүп түзүү

Айкын өзгөртүп түзүүдө “келтирүү” иш аракети жүргүзүлөт. Келтирүү иш аракети-деп каалагандай эле туюнтманын алдына

тегерек кашаанын ичине тип коюууну түшүнөбүз. Туюнтманын тиби кандай гана болбосун, анын алдына тегерек кашаанын ичине коюлган типке өзгөртүлөт.

Мисалы; (float)2+3; (int)6.7-5.3 Ал эми exp, log ж.б. функциялары кош тактыкта (double) чыгарылышы керек болсо, аргумент float тибинде болсо, мисалы; ln(x) функциясын алсак x-float тибинде анда log((double)x) деп колдонсок болот.

1.5 Маалыматтарды киргизүүнүн жана чыгаруунун каражаттары

Маалыматтарды киргизип жана чыгаруунун стандарттык агымдары, түшүнүгү жөнүндө. Агым-деп буфер менен коштолгон файлды түшүнөбүз. Буфер-бул учурдагы эстин өзүнчө бир бөлүгү болуп эсептелет. Буфер аралык, ортолук, ортомчу түшүнүгүн берет. Буфердик информацияларды киргизүү-бул киргизүү түзүлүшүнөн маалыматтар буферге келип түшөт, андан колдонмо программаларга жиберилет дегенди билдирет. Көптөгөн программалар киргизилүүчү маалыматтарды атайын түзүлгөн файлдардан-агымдардан окуйт. Си программалоо тилинде төмөндөгү маалыматтарды киргизип-чыгаруунун агымдары (файлдары) колдонулат.

-stdio.h файлы маалыматтарды киргизип-чыгаруунун стандарттык каражатын,

-stdin.h файлы стандарттык киргизүүнү,

-stdout.h файлы стандарттык чыгарууну камсыз кылат.

Бул файлдардын ысымдары std-стандарттык, in-input, io-input/output, h-heder files-башкы файл дегенди билдирет.

Маалыматтарды калыптап киргизүүнү стандарттык функциялары

scanf-функциясы.

scanf-функциясы маалыматтарды stdin стандарттык

агымынын окуп жана аларды, берилген адресттик аргументтер аркылуу, учурдагы эске адрестер боюнча жайгаштырат.

scanf-функциясынын жалпы берилиши:

scanf (“<калып>” [, & <1-ысым>, & <2-ысым>, …]);

Бул жазылышта <1-ысым>, <2-ысым>, ж.б.у.с. – адресттик аргументтер, алардын алдыларына & белгиси сөзсүз коюлушу керек.

<калып>-калыптоонун сабы, өзгөчө жазылышы “%<тамга>” менен жазылат. Мисалы; %d, %f, %c ж.б.у.с.

&-адрестерди алуу белгиси. Маанилери киргизилүүчү өзгөрмөлөр, scanf функциясы үчүн адрестер аркылуу жазылгандыктан, ар бир өзгөрмөнүн ысымынын алдыларына сөзсүз & адрестерди алуу белгиси коюлат.

Мисалы:

scanf (“%d”, &n)

scanf (“%d %f %d %c”, &m, &x, &k, &ch)

Калыптоонун белгилери.

%d-бүтүн ондук сан

%u-белгиси жок бүтүн сан

%p-көрсөткүчтүн белгиси

%n-көрсөткүчтүн белгиси

%f же %F-калкыма чекитүү ондук сан

%e же %E-калкыма чекитүү ондук сандын экспоненциалдык түрү

%c-жалгыз символ

%s-символдордун сабы же удаалаштыгы

%x-16лык эсептөө системасындагы сан

%o-8дик эсептөө системасындагы сан.

Калыптоо сабы үч типтеги символдорду кармашы мүмкүн.

1. чагылдырылбаган (көрүнбөгөн) символдор
2. чагылдырылган (көрүнгөн) символдор
3. калыптоонун белгилери

Калыптоо сабынын чагылдырылбаган символдору болуп-ачык (пробел) белгиси жадыбалдоонун символдору жана жаңы сапка өтүү белгиси эсетелет. Калыптоо сабындагы чагылдырылган символдор катары “,”-үтүр белгиси алынат, эгерде бул белги калыптоо сабында бар болсо анда маалыматтарды киргизүү сабында да сөзсүз болушу керек.

Мисалы: scanf (“%d, %d”, &n, &m) болсо n жана m өзгөрмөлөрүнүн маанилерин да үтүр менен ажыратуу керек, мейли n=5 жана m=8 болсо 5, 8

Калыптоо белгилери киргизүү талаасын өзгөртүүнү башкарат. Киргизүү талааларында төмөндөгү маалыматтар болушу мүмкүн:

-кийинки чагылдырылбаган символго чейинки баардык символдор

-көрсөтүлгөн калыптоо белгисине баш ийбеген өзгөртүүгө жатпаган биринчи символго чейинки баардык символдор. Мисалы: экилик системадагы “2” же ”3” чейинки, сегиздик системадагы “8” же ”9” цифрасына чейинки сандарды киргизүүдө.

-n сандагы символдор, n болсо киргизүү талаасынын узундугу.

scanf –функциясынын калыптоо белгилеринин жазылышынын жалпы түрү.

%[*] [N]k

k-калыптоо тамгалары мисалы: d, f, c ж.б.

*-белгиси киргизүү сабындагы маалыматты окуйт, бирок эч бир аргументке менчиктебейт.

Мисалы: scanf (“%d %*c %d”, &i, &j); функциясында киргизүү сабында 50+20 туюнтмасы болсо i=50, j=20 менчиктелип “+” символу окулуп, бирок эч бир өзгөрмөгө менчиктелбейт.

N-талаанын кеңдиги б.а. киргизүү сабындагы маалыматтардын узундугу. Эгерде scanf-функциясы чагылдырылбаган же өзгөртүүгө жатпаган символду кезиктирсе анда, аз сандагы символ окулушу мүмкүн. Мисалы: scanf (“%2d %f”, &i, &x) ал эми киргизүү сабында 56789 75 болсо i=56 биринчи эки орунда турган цифралардан түзүлгөн санды i-өзгөрмөсүнө менчиктейт, ал эми x=789,0 бул үч орундан баштап биринчи ачыкка чейинки цифралардан турган санды x-өзгөрмөсүнө калкыма чекитүү сан катары менчиктейт.

Калыптап киргизүүнүн тамга белгилерине кеңири токтололу. Бүтүн сандарды калыптап киргизүүдө %d, %i-белгилери колдонулат, мында %d ондук бүтүн сан үчүн, ал эми %i-белгиси эки түрдө %0 жана %x түрүндө кезигет. %0-түрү сегизилик, %x-түрү он алтылык бүтүн сандарды киргизүүдө колдонулат. %s-саптарды киргизүүгө арналган, киргизүүчү сап маалыматтары “ачык” (пробел) белгиси менен аякталышы керек. %c-бир гана символду киргизүүгө арналган, бул символ катары “ачык” жана чагылдырылбаган белгилер да эсептелинет. Чагылдырылбаган символду аттап кетип символду окуу үчүн 1s белгиси колдонулат.

Мисалы: scanf (“%s %c%c %*s %3d %d”, sap, &c1, &c2, &i1, &i2, &i3); мейли өзгөрмөлөр мындай баяндалсын int i1, i2, i3; char sap[20], c1, c2; Киргизилүүчү маалыматтар сабы “abc defg -123456789” болсо анда sap=”abc\0” scanf функциясында sap-өзгөрмөсү символдордун массиви катары берилгендиктен жана sap массивинин индекси жазылбагандыктан, массивдин адреси катары кабыл алынат., жана sap-ысымынын алдына &-белгиси коюлбайт. sap-өзгөрмөсүнүн мааниси катары abc деген биринчи “ачыкка” чейинки сап менчиктелет жана ал массивдин акыркы элементи катары атайын коду 0 болгон “\0” символу менчиктелет, муну нөл цифрасы менен алмаштырбоо керек, себеби нөлдүн коду “0” эмес. %n, %p-белгилери көрсөткүчтөрдү

киргизүү арналган, мында %n белгиси көрсөткүчтү кеңейтилген калыпта киргизүүдө колдонулат.

%[үлгү]-калыптоонун белгисине токтололу, квадраттык кашаанын ичинде, киргизилүүчү саптарды түзгөн символдордун тобу жазылат. Эгерде квадраттык кашаанын ичинде биринчи символ “^”-белгиси болсо киргизүү сабындагы кашаага алынбаган баардык символдор окулат. Эгерде квадраттык кашаанын ичинде саналуучу маалыматтар болсо башын жана аягын көрсөтүп коюу жетиштүү.

Мисалы:[0,1,2,3,4,5] болсо аны [0-5] деп жазса болот.

```
char sap[10];
int i, j;
float x;
scanf (“%2d %f %*d %[0-9]”, &i, &x, sap);
```

Мейли киргизүү сабында “65187 0123 56A72” маалыматы болсун. Анда %2d калыбы боюнча 65 саны i бүтүн маанилүү өзгөрмөгө ыйгарылат б.а. i=65 болот ал эми %f калкыма чекитүү калып боюнча x=187.0 мааниси окулат. Ал эми %*d калыбы боюнча 0123 сабы окулуп, *-белгиси болгондуктан 0123 сабы эч бир өзгөрмөгө ыйгарылбайт. Ошондой эле %[0-9] калыбы боюнча sap өзгөрмөсүнө ”56\0” мааниси ыйгарылат. Себеби sap өзгөрмөсү 10элементтен турса дагы %[0-9] калыбы киргизүү сабында “A” тамгасына чейинки гана цифраларды ыйгара алат.

scanf-функциясы киргизүү сабын окууну, саптын аягына жетпей токтошунун себептери.

1) Калыптоо символунда менчиктөөнү жокко чыгарган *-белгиси жолукса.

2) N символу окулса, жана N киргизүү сабынын узундугу болсо.

3) Өзгөртүүгө жатпоочу символ кезиксе б.а. көрсөтүлгөн калып белгисине дал келбеген (өзгөртүүгө жатпаган) символ болсо мисалы: сегиздик эсептөө системасы үчүн 8 же 9 цифрасы.

4) Киргизүү сабындагы кезектеги символ квадраттык кашаанын ичиндеги символдорго дал келбесе

scanf-функциясынын иш аракетинин аякташынын себептери.

1) Калыптоо белгилерине кезектеги символдун тиби туура келбесе

2) Калыптоо белгиси түгөнгөн учурда

3) Киргизүү сабындагы кийинки символ EOF символу менен аяктаса

scanf-функциясынын иштөө мисалдары:

```
# include <stdio.h>
main ( )
{int a, b;
char ch;
scanf (“%d%d”, &a,&b); /* сандарды киргизүүдө алардын арасын каалагандай сандагы “ачык” белгиси менен ажыратып киргизсе болот */
scanf (“%d %d”, &a,&b); /* киргизүүдө эки санды ортосун каалагандай сандагы “ачык” белгиси менен бөлсө болот же киргизүү баскычы<enter> менен киргизсе болот */
scanf (“%d, %d”, &a,&b);  /* киргизүүдө эки санды арасын үтүр менен ажыратып, үтүр менен сандын ортосуна “ачык” белгисин койсо да койбосо болот*/
scanf (“%d,%d”, &a, &b); /* киргизүүдө эки сан үтүр менен гана ажыратылат*/
scanf (“%d%c”, &a,&ch);  /* киргизүүдө сандан кийин “ачык”белгиси жок эле символ жазылыш керек*/
scanf (“%d  %c”, &a, &ch);/* киргизүүдө сандан кийин “ачык”белгиси жана символ жазылышы керек*/
  }
```

Саптарды б.а. символдордун удаалаштыгын киргизүү үчүн gets-функциясы колдонулат жалпы берилиши gets(s).

s-киргизилүүчү сап же символдордун массиви. gets-функциясы стандарттык stdin кирүү агымынан сапты окуйт. Киргизилүүчү саптын баардык символдорун жана “ачык” белгисин кошо окуп аларды S-массивине жайгаштырат. Сапты окуу, киргизүү <Enter> баскычын басканга чейин окуйт. Бул баскычты баскандан кийин s сабынын акыркы символу катары “\0” нөлдүк символду кошот. gets-функциясынан scanf функциясынын айырмасы бул функция сапты “ачык” же жадыбалдоо же саптын аягын билдирүүчү белгилер кезиккенге чейин эле окуйт. Демек scanf функциясы менен саптагы “ачык” белгини окуй албайбыз, ал эми gets-функциясы менен аны окуй алабыз. getchar-функциясы, бул функциянын баяндалышы. stdio.h файлында кармалган жана маалыматтарды киргизүү сабынан бир гана символду окуйт. getchar-функциясы, stdin-стандарттык кирүү агымынан <Enter> клавишасын баскандан кийин бир гана символду окуп, анны экранга чагылдырат.

getch жана getche-функциялары conio.h файлында жайгашкан жана киргизилүүчү символду түздөн-түз консолдон (бир тараптуу түзүлүш) б.а. клавиатурадан алат. Экөөнүн айырмасы: getch функциясы киргизилген символду экранга чыгарбайт, ал эми getche-функциясы киргизилген символду экранга чыгарат. getch функциясы кандайдыр бир клавишаны басканга чейин программанын иштөөсүн токтотуп туруш үчүн колдонулат. Жогорудагы үч функция тең параметрге ээ болбойт жана бул функциялар char типтүү функциялар болуп эсептелет. Жогорку функциялардын маанилери символдук өзгөрмөлөргө менчиктеле алат.

Мисал-1:

```
char ch;
```

ch=getch ( );

putch (ch)

1-сапта ch өзгүрмөсүнүн тиби символдук тип катары баяндалган.

2-сапта getch ( ) функциясынын мааниси ch символдук өзгөрмөсүнө менчиктелип жатат.

3-сапта ch өзгүрмөсүнүн мааниси чыгарылат.

Мисал-2: программанын иштөөсүнүн башында экранды тазалап, программанын иштөөсүнүн жыйынтыгын, кандайдыр бир каалагандай клавишаны басканга чейин кармап турган программа түзөлү.

```
# include <stdio.h>
# include <conio.h>
main( )
{ …
clrscr( );
…
puts(“ишти аяктоо үчүн каалагандай баскычты басыңыз”);
getch( );
}
```

Программада gets жана getchar-функцияларын колдонгондон кийин киргизүү маалыматтарын, агым катары тазалоо керек. Программанын иштөөсүнүн башында киргизүү агымы автоматтык түрдө тазаланат. Бирок буферлештирилген киргизүүдө каалагандай стандарттык киргизүү функцияларын колдонгон учурда киргизүү агымын тазалоо сунуш кылынат. Киргизүү агымын б.а. stdin-файлын тазалоо үчүн fflush-стандарттык функциясы колдонулат.

Мисалы:

```
scanf (“%d”, &n);
fflush (stdin);
```

scanf (“%d”, &i);
fflush (stdin);
gets (sap);

Программанын экинчи сабындагы stdin агымын тазалоочу fflush-функциясы туура эмес киргизүү жүргүзүлгөн учурда колдонулат. 4-сабында fflush-функциясы scanf-функциясы аткарылгандан кийин буферде “саптын аягы” символун жоюу үчүн жана gets-функциясы туура иштеши үчүн колдонулду. scanf жана getchar функциялары маалыматтарды киргизгенден кийин, киргизүү агымында <Enter> баскычынын кодун калтырып коюшат, ошондуктан gets-функциясынын иштөөсүнүн алдында, дайыма fflush-функциясын сөзсүз колдонуу керек.

Калыптап чыгаруунун стандарттык функциясы

Стандарттык чыгаруу функциясынын жалпы жазылышы.

printf (“<калып>”, [<1-туюнтма>, <2-туюнтма>, …]);

<1-туюнтма>, <2-туюнтма>, … маанилери чыгарыла турган туюнтмалар. Туюнтма катары: туракту, өзгөрмө, функцияга кайрылуу алынышы мүмкүн калыптоо сабы чыгаруучу символдорду, башкаруучу символдорду жана калыптоо белгилерин кармайт. Калыптап чыгаруу функциясын мисал менен түшүндүрөлү.

Мисалы: printf (“сумма=%d\n”, sum);

Бул жазууда “сумма =” экранга чыгарыла турган сап туракту усу %d-калыптын белгиси тактап алганда бүтүн сандарды чыгарууда колдонулат. \n-башкаруучу символ, бул символ башкарууну же иш аракетти кийинки же жаңы сапка өткөрөт. sum-мааниси чыгарыла турган өзгөрмө.

Маалыматтарды калыптап киргизүүдөгү калыптын баардык белгилери чыгаруу функциясында да толугу менен колдонулат. printf-функциясы үчүн калыптын белгилеринин жалпы жазылышы: %[<тегиздөө>][<орун>] [<тактык>] <тип>

<тегиздөө>-чыгарылуучу маани экранда оң же сол жагынан тегизделе тургандыгын билгизет. Эгерде “-”белгиси коюлса

чыгарылуучу маани сол жактан тегизделет, ал эми эч кандай белги коюлбаса чыгарылуучу маани оң жагынан тегизделет.

<орун>-чыгарылуучу мааниге берилген жалпы орундардын саны. Эгер орундардын саны чыгарылуучу маанилерден ашып калса анда ашык орундар “ачык” орун менен толукталат, ал эми орундардын саны чыгарылуучу маанилерден кем болуп калса, анда чыгарылуучу маани ошол бойдон чыгарылат.

<тактык>-калкыма чекитүү float double типтери үчүн ондук чекиттен кийин канча цифра чыгарылышы керек экендигин билдирет. Ал эми сапты (символдордун удаалаштыгын) чыгара турган болсо саптын башынан баштап канча символ чыгарыла тургандыгын билгизет. Эгерде калкыма чекитүү сандар үчүн <тактык> берилбесе анда ондук чекиттен кийин 6 цифра чыгарыла тургандыгын, ал эми <тактык> 0гө барабар болсо калкыма чекитүү сандын бүтүн гана бөлүгү чыгарылат дегенди билдирет.

Мисалдарды карайлы;

№	Маанилер	Калып белгилери	Чыгарылгандар
1	1959	%10d	------1959
2	1959	%-10d	1959------
3	5.95634178	%10f	5.926341
4	5.92634178	%10.0f	5
5	5.92634178	%10.3f	5.926
6	Информатик	%10s	Информатик
7	Информатик	%10.7s	Информа
8	Информатик	%.4s	Инфо

printf-чыгаруу функциясынын мисалдарын карайлы:

1-Мисал:

```
int a, b;
char ch;
a=4;
b=7;
```

ch=’+’;

printf (“a=%d, b=%d\n ch=%c”, a, b, ch);

бул программанын бөлүгү аткарылганда экранга a=4, b=7; ch=+; маалыматтары чыгарылат, \n башкаруу символу өзүнөн кийинки маалыматтарды жаңы сапка которот .

printf (“%25c”, ‘_’) чыгаруу сабына 25 “ачык” символун чыгарат.

printf (“Анар”) Анар деген ысымды чыгарат

printf (“Анар”, “Бакыт”) жалгыз Анар деген ысымды чыгарат

printf (“Анар” “Бакыт”) АнарБакыт деген ысымдарды чыгарат

2-Мисал:

int n=1, m=2;

float x=3, y=4;

printf (“\n n=%d m=%d”, n,m); /*жаңы сапка n=1 m=2 чыгарат */

printf (“\n x=%f y=%f”, x,y); /*жаңы сапка x=3.000000 y=4.000000*/

printf (“\n n=%d”, n,m); /*n=1 чыгарат m дин маанисин чыгарбайт себеби калыптоо белгиси жок */

printf (“\n n=%d, m=%d, z=%d”, n, m); /*n=1, m=2, z=0 чыгарып берет */

puts-функциясы сапты чыгарууну ишке ашырат бул функция stdio.h файлында жайгашкан.

Мисалы:

char s[20];

…

puts (s);

Бул функция s сабы түзүлгөндөн кийин ошол сапты экранга чыгарып андан кийин иш аракетти жаңы сапка өткөрөт.

putchar-функциясы stdio.h файлында жайгашкан жалпы жазылышы putchar(ch) ch-жалгыз символду кармаган өзгөрмө, бул функцияны колдонуп чыгарууда экранда жылгыч (курсор) кайсы жерде болсо ошол жерге жалгыз символду чыгарат

Текшерүүчү суроолор

1. Си тилинин негизги элементтери.
2. Си тилинин алгачкы процессору.
3. Си тилининдеги ысымдар
4. Турактуулардын типтери.
5. Өзгөрмөлөр жана анын типтери.
6. # include-буйругу.
7. #define- буйругунун мааниси.
8. Си тилинин киргизүү функциясы.
9. Си тилинин чыгаруу функциясы.
10. Сапты киргизүү жана чыгаруу функциялары.

Маалыматтарды киргизип жана чыгаруу каражаттарына көнүгүүлөр

1. 30+20 суммасын киргизүүчү функциясын жаз.
2. int a; float x; char ch өзгөрмөлөрүн калыптап киргизүүчү функцияны жаз.
3. Телефон 5-26-05 түрүндөгү киргизүүчү функцияны жаз.
4. 19091959M сабынан a=19, в=09, c=1959 жана ch=’M’ түрүндөгү киргизүүчү функцияны жаз.
5. 30,07,2003 түрүндөгү киргизүүчү функцияны жаз.
6. Нуртегин деген ысымды чыгаруучу функцияны жаз.
7. N=2012 түрүндөгү чыгаруучу функцияны жаз.
8. X=5.000000 y=8.000000 түрүндөгү чыгаруучу функцияны жаз.
9. N=3, x=9.0 жана c=’?’ болсо

   n=3

   x=9.000000

   c=? түрүндөгү чыгаруучу функцияны жаз.
10. 3.141593 деген сандын 3.14 түрүндөгү чыгаруучу функцияны жаз.

2-Бөлүм. СИ ТИЛИНИН АМАЛДАРЫ ЖАНА ОПЕРАТОРЛОРУ

2.1 Си тилинин амалдары

Арифметикалык амалдар

Си тилинде каалагандай эле туюнтма амалдардын белгиси менен байланышкан операндалардан турат.

Операнда-бул амал белгисинин оң жана сол жагындагы орундар.

Амал белгиси - бул компиляторго же арифметикалык же логикалык иш аракетерди жүргүзүүнү кабарлайт. Си тилинде операндалардын санына жараша унардык жана бинардык амалдар бар. Унардык арифметикалык амалга минус (-) терс сандарды билгизе турган белги, ал эми унардык логикалык амалга тануу (not) амалы кирет.

Бинардык арифметикалык амалдарга кошу (+), кемитүү (-), көбөйтүү (*) жана бөлүү амалдары кирет. Бөлүүнүн эки түрү каралган бөлүү (/) жана бүтүн сандарды бөлүүдө, калдыгын табуучу бөлүү амалы (%). Бөлүү амалын жүргүзүүдө, төмөндөгүдөй эреже иштейт. Эгерде бөлүнүүчү да бөлүүчү да бүтүн типте болушса натыйжа бүтүн типте болот б.а бөлчөк бөлүгү эске алынбайт. Арифметикалык амалдарды аткарууга мисалдар:

Туюнтмалар	Жыйынтыктар
-7	-7
1+3*4-2	11
1+3/4-2	-1
1+3%4-2	2
1+3,0%4-2	-0,25

Бөлүү амалында операндалар бүтүн жана анык сандар болгон учурун карайлы.

Мисал-1;

```
init n,m;
float r;
n=9;
m=4;
r=n/m;
```

Бул программада r=2.00, себеби n жана m операндалары бүтүн типтүү, адегенде бүтүн сандарды бөлүү аткарылат жыйнтыгы 2 болот 0.25 болгон бөлүгү эске алынбайт, 2 бүтүн саны r калкыма чекиттүү өзгөрмөнүн тибине өзгөртүлүп 2.00 деп чыгарылат.

Мисал-2;

```
float x,y,z;
x=9;
y=4;
z=x/y;
```

Бул мисалда z тин маниси 2.25 болот, себеби бөлүү амалындагы эки операндасы тең калкыма чекиттүү өзгөрмөлөр.

Мисал-3; Бул мисалда типтерди келтирүү амалын бөлүүдө колдонулушун карайлы. Мейли int n, float x жана n=7 болсун

1) x=n/2; жыйынтык: x=3.000000

2) x=(float)n/2; жыйнтык: x=3.500000 себеби n бүтүн тиби, калкыма чекиттүү типке өзгөртүлүп андан кийин бөлүнүп жатат.

3) x=(float)(n/2); жыйнтык: x=3.000000 1-учурга окшош адегенде бүтүн сандардын бөлүүсү аткарылып андан кийин калкыма чекиттүү типке келтирилет.

Чоңойтуучу жана кичирейтүүчү амалдар.

Си тилинде чоңойтуучу амал катары “++” белгиси ал эми кичирейтүүчү амал катары “--“ белгиси алынат. Чоңойтуучу жана

кичирейтүүчү белги өзгөрмөнүн ысымынын алдына же өзгөрмөнүн ысымынан кийин коюлат ++x жана x++ деген жазуу x=x+1;операторуна эквивалентүү б.а x тин маанисин 1 ге чоңойт дегенди билгизет, ал эми --x же x-- жазуусу x тин маанисин 1 ге кичирейт дегенди билдирет. Чоңойтуу жана кичирейтүү белгилеринин өзгөрмөнүн ысымынан мурун же кийин жазылышы өзүнчө мааниге ээ. Эгерде чоңойтуучу (кичирейтүүчү) белги өзгөрмөнүн ысымынан мурун келсе мисалы: ++a же --b, анда өзгөрмөнүн мааниси 1ге чоңойуп же 1ге кемип(азайып) туюнтманы эсептөөгө катышат, ал эми чоңойтуучу же кичирейтүүчү белги өзгөрмөнүн ысымынан кийин келсе мисалы a++ же b--анда өзгөрмөнүн маанилери бирге чоңойтулбай же бирге кемитилбей (азайтылбай) (өзгөрмөнүн эски мааниси боюнча) туюнтманы эсептөөгө катышып туюнтма эсептелип бүткөндөн кийин гана өзгөрмөнүн мааниси 1 ге чоңойтулат же 1ге кичирейтилет.

Мисал;

```
int n,m,r;
n=3;
m=3;
r=(n++)+(--m)
```

жыйнтыгы r=5, n=4,m=2 болот.

Менчиктөө амалы

Менчиктөө амалынан жалпы берилиши.

<өзгөрмөнүн ысымы>=<туюнтма> бул амал аткарылганда <туюнтманын> тиби өзгөрмөнүн тибине өзгөртүлүп түзүлөт.

Мисал-1;

```
int x,y,z;
x=7;
y=x*2+9;
z=y/x;
```

Бул мисалдагы менчиктөө операторлорун бир сапка менчиктөө амалы катары жазса болот.

z=(y=(x=7)*2+9)/4; Менчиктөө амалын аткаруу ондон солду көздөй жүрөт.

Мисал-2;

x=y=z=10*a биринчи 10*a мааниси эсептелип z ке менчиктелет андан кийин y ке менчиктелет, аягында x өзгөрмөсүнө менчиктелет, демек иш аракеттер оңдон солду көздөй жүрөт.

Менчиктөө амалынын да өзгөчө түрү кезигет

<өзгөрмөнүн ысымы> <амал белгиси>= <туюнтма> бул жазуу

<өзгөрмөнүн ысымы> = < өзгөрмөнүн ысымы > <амал белгиси> <туюнтма> деген жазууга тең күчтө.

Мисалы x=x+5 жазуусун x+ =5 жазсак болот x+ =5 амалы x=x+5 амалына караганда тез иштейт.

Мисал-3;

int a,b	Жыйынтыктар
a=b=5;	a =5, b=5
a+=1;	a=6
b- =3;	b=2
a*=b	a=12, b=2
a/=++b	a=4, b=3
a %=b--	a=1, b=2

Логикалык жана катыш амалдары

Си тилинде *6 катыш* жана *3 логикалык амал* бар.

“<”-кичине

“<=”-кичине же барабар

“>”-чоң

“>=”-чоң же барабар

“= =’-барабар

“!=”-барабар эмес

Логикалык амалдар

&& -“жана” амалы

| | -“же” амалы

! -“эмес” же тануу амалы

Катыш амалдары эки туюнтманын маанилерин салыштырууда колдонулат. Салыштырууда жыйынтыгы же чын же жалган гана болот. Си тилинде чынды 1 ал эми жалганды 0 катары алышат.

Логикалык амалдардын чындык таблицасы

x	y	x&&y	x| |y	! x	! y
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Разряддык амалдар

Разряддык амалдар бүтүн сандардын экилик системадагы маанилери менен амалдарды жүргүзүгө арналган. Бул амалдар ар бир бит информациялар менен иш жүргүзүүнү камсыз кылат. Мындай иш аракеттер сырткы түзүлүштөр (принтер ,модем ж.б.) байланышкан программаларды түзүүдө колдонулат. Разряддык амалдар int, char типтүү өзгөрмөлөргө гана колдонулат, ал эми float, double жана void типтерине колдонууга болбойт. Разряддык амалдарга: «~»- разряд боюнча тануу ; «<<»- солго жылуу; «>>»- оңго жылуу ; «&»- разряд боюнча «жана» ; «^»- эки модулу боюнча кошуу амалы; «|» разряд боюнча «же» амалдары кирет.

Мисалдар: Мейли a =00001111 жана b=10001000

~a=11110000

~b=01110111

a<<1=00011110

a>>1=00000111

a&b=00001000

a^b=10000111

a|b=10001111

Разряддык амалдардын экилик, ондук системалардагы аткарылыштарын карайлы

Ондук түрү	Экилик түрү	Ондук системага которулушу
5	00000101	$0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+0*2+1*2^0=5$
7	00000111	$0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2+1*2^0=7$
7<<5	11100000	$1*2^7+1*2^6+1*2^5+0*2^4+0*2^3+1*2^2+0*2+0*2^0=224$
7>>5	00000000	0
7&5	00000101	$0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+0*2+1*2^0=5$
7|5	00000111	$0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2+1*2^0=7$
7^5	00000010	$0*2^7+0*2^6+0*2^5+0*2^4+0*2^3+1*2^2+1*2+1*2^0=2$

Разряддык амалдардын Си тилинде программанын бөлүгү катары жазылышы жана жыйынтыгын таблица түрүндө берилиши.

Программадагы жазылышы	Экилик түрү
int a=-2;	11111110
int rez=~a;	00000001
int a=10;	00001010
int b=12;	00001100
int x=a&b;	00001000
int y=a^b;	00000110
int z=a|b	00001110

Си тилиндеги амалдардын таблицасы аткаруу тартиби менен.

Амалдын белгиси	Амалдын аталышы
( )	Функцияга кайрылуу
[ ]	Массивтин элементин бөлүп алуу
.	Жазылыштын элементин бөлүп алуу
!	Логикалык тануу
~	Разряд боюнча тануу
-	Белгини өзгөртүү
+ +	Бирге чоңойтуу
- -	Бирге кичирейтүү
&	Адрести көрсөтүү
*	Адрес боюнча кайрылуу
(tip)	Типти өзгөртүү
Sizeof ( )	Байттагы өлчөмдү аныктоо
*	Көбөйтүү
/	Бөлүү
%	Бүтүн бөлүүдө калдыкты аныктоо
+	Кошу
-	Кемитүү
<<	Солду көздөй жылуу
>>	Оңду көздөй жылуу
<	Кичине
<=	Кичине же барабар
>	Чоң
>=	Чоң же барабар
= =	Барабар
!=	Барабар эмес
&	Разряд боюнча ”жана” амалы
^	Эки модулу боюнча кошуу амалы
|	Разряд боюнча “же” амалы
&&	Логикалык «жана»
||	Логикалык «же»
?:	Шарттуу амалы
=	Менчиктөө амалы

2.2 Си тилинин операторлору

Си тилинде операторлор жөнөкөй, курама жана куру болуп бөлүнүшөт. Бир сап бир гана оператордон турса анда аны жөнөкөй оператор дейбиз. Жөнөкөй оператор «;» белгиси менен аяктайт.

Курама операторлор- деп фигуралык кашаага алынган жөнөкөй операторлордун тобун айтабыз. Кандайдыр бир сап жалгыз «;» белгисин гана кармаса аны куру оператор деп эсептейбиз.

2.3 Сызыктуу алгоритмдерди программалоо.

Сызыктуу алгоритмдерди программалоодо негизинен киргизүү жана чыгаруу функциялары, каалагандай эле сандагы менчиктөө оператору колдонулат мындан сырткары ар кандай эсептөөлөрдү жүргүзүүдө стандарттык математикалык функциялар пайдаланылат.

Менчиктөө оператору.

Менчиктөө оператору Си тилинде сызыктуу процесстерди программалоодо колдонулат. Бул оператордун жалпы жазылышы.

<өзгөрмөнүн ысымы>= <туюнтма>;

Менчиктөө операторунда оң жактагы <туюнтма> эсептелинип жыйынтыгы сол жактагы өзгөрмөгө менчиктелет.

Мисалы: x=2; y=(x=5)*2+7; z=z/ ++y; b=b-3;

Менчиктөө операторунун колдонулушуна *мисалдар.*

1-Мисал.

```
# include <stdio.h>
main ( )
{ int a, b, x, y;
printf (“\n эки бүтүн санды киргиз\n”);
scanf (“%d,%d”,&a,&b );
```

```
x=(a++)+(--b);
y=((a=10)+4)/(++b);
printf (“\nx=%d\ny=%d”,x,y);
}
```

Жыйынтык: эки бүтүн санды киргиз
1,2
x=3
y=7

2-Мисал. $y=ax^3+bx^2+c$ a,b,c жана x тин бүтүн маанилери үчүн y тин маанисин эсепте.

```
# include <stdio.h>
# include <math.h>
main ( )
{ int a,b,c,x;
float y;
printf (“ a , b жана c өзгөрмөлөрүнүн бүтүн маанисин
киргиз\n”);
scanf (“%d,%d,%d”,&a,&b,&c);
printf (“ x тин бүтүн маанисин бер \n“);
scanf (“%d”,&x);
y=a*pow (x,3)+ b*pow (x,2)+c;
printf (“\n y=%f”,y);
}
```

Жыйынтык: a , b жана c өзгөрмөлөрүнүн бүтүн маанисин киргиз
1,2,3
x тин бүтүн маанисин бер
4
y=99.000000

Жогорку программалардагы clrscr() функциясы маалыматтарды киргизип-чыгаруучу экранды тазалап туруу үчүн колдонулган жана бул функциянын баяндалышы conio.h-файлында жаткандыктан программада # include <conio.h> сабы жазылган.

2.4 Тармактануучу алгоритмдерди программалоо.

Шартуу жана шартсыз өтүү операторлору тармактануучу алгоритмдерди программалоодо колдонулат. Си тилинде шартуу операторун жалпы жазылышы.

if (<туюнтма>) <оператор 1> else <оператор 2>;

бул жерде (<туюнтма>)-логикалык туюнтма.

Шартуу операторун аткарылышы. Адегенде <туюнтма> эсептелинет. Эгерде ал чын болсо <оператор 1> аткарылат, ал эми <туюнтма> жалган болсо <оператор 2> аткарылат. <оператор 1> жана <оператор 2> куру, жөнөкөй же курама операторлор болушу мүмкүн.

Шартуу операторун экинчи түрү.

if (<туюнтма>) <оператор>;-бул толук эмес түрү деп аталат. <туюнтма> чын болсо <оператор> аткарылат, ал эми туюнтма жалган болсо шартуу операторун иштеши токтотулат.

Мисал. a жана b сандарынын кичинесин тапкыла?

```
# include <stdio.h>
# include <conio.h>
main ( )
{int a,b,c;
clrscr();
printf (“эки санды киргиз\n”);
scanf (“%d,%d”, &a, &b);
if (a<b) c=a; else c=b;
printf (“\n эки сандын кичинеси =%d\n”, c);
```

}

Жыйынтык: эки санды киргиз

1,2

эки сандын кичинеси =1

Шартуу оператордун кабатталган түрү if (<туюнтма 1>) if (<туюнтма 2>)<оператор 1> else<оператор 2>; бул кабатталган түрдөгү биринчи else ички шартуу операторго тиешелүү деп эсептелинет, б.а. төмөндөгү жазуу менен теңдеш:

if (<туюнтма 1>)

{ if (<туюнтма 2>) <оператор 1>

else <оператор 2>}

Шартуу оператордун толук кабатталган түрү төмөндөгүдөй жазылат:

if (<туюнтма 1>)

{ if (<туюнтма 2>) <оператор 1>}

else <оператор 2>

Бул кабатталган түрүндө сырткы шартуу оператор толук түрү менен ал эми ички шартуу оператор толук эмес түрү менен берилген.

Мисалы: үч бурчтукту үч жагы менен аныктагыла.

```
# include <stdio.h>
# include <conio.h>
main ( )
{float  a,b,c;
clrscr();
printf (“ үч бурчтуктун үч жагын бер\n”);
scanf  (“%f ,%f,%f”, &a, &b, &c);
if (a>+c||b>a+c||c>a+b)
printf (“ мындай үч бурчтук жок\n”);
else if (a= =b&&a= =c)
printf (“ бул тең жактуу үч бурчтук\n”);
```

```
else if (a= =b||a= =c||b= =c)
printf (“ бул тең капталдуу үч бурчтук\n”);
else printf (“\n жактары ар түрдүү үч бурчтук\n”);
}
```

Жыйынтык: үч бурчтуктун үч жагын бер
1,2,3
жактары ар түрдүү үч бурчтук
үч бурчтуктун үч жагын бер
2,2,3
бул тең капталдуу үч бурчтук
үч бурчтуктун үч жагын бер
3,3,3
бул тең жактуу үч бурчтук
үч бурчтуктун үч жагын бер
1,2,5
мындай үч бурчтук жок

Программанын текстин кыскартып жазуу үчүн шартуу оператордогу <туюнтманы> жазууда менчиктөө амалы көп колдонулат. Менчиктөө амалы тегерек кашаага алынып жазылат. *Мисалы*; (a=2+3) бул туюнтма 5 деген маанини алат. Төмөндө программанын бир бөлүгүн карайлы:

```
if ((ch=getch())= =’q’) puts (“\n программа иштеп бүттү”);
else puts (“\n программа ишин улантууда”)
```

Клавиатурадан басылган символ ch өзгөрмөсүнө менчиктелип андан кийин ‘q’ символу менен салыштырылат.

Си тилинде тегерек кашаанын ичинде көптөгөн туюнтманы колдонсо болот, ал туюнтмалар үтүр белгиси менен ажыратылат. Тегерек кашаанын ичиндеги туюнтмалар солдон оңду көздөй аткарылат.

Мисалы: (simvol=ch, ch=getch()) тегерек кашаанын ичинде simvol жана ch өзгөрмөлөрү char тибиндеги өзгөрмөлөр. Адегенде

simvol ch өзгөрмөсүнүн маанисин менчиктейт, андан кийин клавиатурадан киргизилген символду ch өзгөрмөсү алат. Төмөндөгү программанын бөлүгүн карайлы:

ch=’x’;

if ((simvol=ch, ch=’z’)= =’x’) puts (“бул ‘x’ символу \n”);

else puts (“бул ‘z’ символу \n”);

Бул программанын жыйынтыгы ‘z’ тамгасы эсептелет.

Си тилинде шартуу амал каралган анын белгилениши “?:”

Жалпы жазылышы <шарт> ? <туюнтма 1> : <туюнтма 2> шартуу амалдын аткарылышында адегенде <шарт> текшерилет эгерде <шарт> чын болсо <туюнтма 1> аткарылат, ал эми <шарт> жалган болсо <туюнтма 2> аткарылат.

Мисалы: эки сандын кичинесин тандоо маселесин карайлы.

```
# include <stdio.h>
# include <conio.h>
main ( )
{int a,b,c;
clrscr();
printf (“эки бүтүн санды киргизгиле”);
scanf (“%d %d”, &a, &b);
c=(a<b)?a:b;
printf (“берилген эки сандын кичинеси c=%d\n”, c);
}
```

Жыйынтык: эки бүтүн санды киргизгиле

7,9

берилген эки сандын кичинеси c=7

2.5 Кайталануучу алгоритмдерди программалоо

Кайталануучу алгоритмдерди Си тилинде программалоо үчүн үч түрдөгү цикл оператору каралган. Аларга кайталануунун саны берилген цикл, шарттан кийинки цикл жана шартка чейинки цикл

операторлору кирет. Кайталануунун саны берилген цикл операторунда кайталанууну уюштурган өзгөрмөнү параметр деп аташат. Ошондуктан бул цикл операторун параметрлүү цикл деп да коюшат.

1) Параметрлүү цикл операторунун жалпы берилиши.

for (n1;n2;n3) <оператор>;

n1-параметрдин баштапкы мааниси.

n2-кайталанууну уюштуруучу шарт б. а. циклден чыгууну же улантууну текшерүүчү шарт.

n3-параметрдин өзгөрүү кадамы.

<оператор> - циклдин тулкусу деп аталат б. а. циклдин кайталануучу бөлүгү. Циклдин тулкусу жөнөкөй же курама болушу мүмкүн. Циклдин тулкусу курама болсо б. а. бирден көп операторлорду же функцияларды кармаса анда циклдин тулкусу фигуралык кашаага алынып жазылат. Көпчүлүк учурда циклдин параметрин i -тамгасы менен белгилешет.

1) Параметирдин мааниси өсүү тартибинде,же кемүү тартибинде болушу мүмкүн.

Мисалы. 10дон 1ге чейин бүтүн сандардын көбөйтүндүсүн тап?

```
# include <stdio.h>
# include <conio.h>
main ( )
{int i,p;
clrscr();
p=1;
for (i=10; i>=1;i--) p*=i;
printf (“p=%d”,p);
}
```

Жыйынтык: p=24320

2) Циклдин параметиринин өзгөрүш кадамы ар кандай болушу мүмкүн.

for (i=2;i<90;i+=7)<оператор>;

Мисалы. 5ке так бөлүнгөн эки орундуу сандардын суммасын тапкыла.

```
#include <stdio.h>
# include <conio.h>
main ()
{int i,s ;
clrscr();
s=0;
for (i=10 ; i<=99; i+=5)s+=i;
printf (“s=%d”,s);
}
```

Жыйынтык: s=945

3) Циклдин параметири катары символдорду колдонууга болот.

for (char ch=’a’; ch<=’z’; ch++)<оператор>;

4) n3-өзгөрүү катары параметирди кармаган туюнтманы колдонсо болот.

for(x=1;y<=50;y=3*x++ +5)printf (“x=%d y=%d”,x,y);

Бул мисалда параметирдин өзгөрүү кадамы катары y=3*x++ +5 туюнтмасындагы x++ чоңойтуучу амал колдонулду.

5) Цикл операторунун жалпы жазылышындагы n3-параметирдин өзгөрүү кадамы жазбай койсо болот.

Мисалы;

```
int i,n;
n=3;
for(i=5; n<=100;)n*=i;
```

Бул *мисалда* параметирдин өзгөрүү кадамы циклдин тулкусуна т.а n*=i: операторунда өзгөрөт. Жогорудагы цикл операторун башкача түрүн жазса да болот.

int i,n;

n=3;

for (i=5; n<=100; n*=i);

Бул жазууда циклдин тулкусу куру оператор менен жазылып калды.

6) Цикл операторунда n1 параметири бир гана жолу аткарылгандыктан, дайыма эле өзгөрмөнүн ысымын кармай бербейт.

Мисалы:

for (printf (“ сан киргиз\n”); n<=5;scanf (“%d”,&n);

бул мисалда n дин маанисин киргизүүдө 5тен чоң сан болмоюнча цикл аткарыла берет.

7) үтүр (,) - белгиси параметирлүү цикл операторунда, параметирлердин ордуна, бир канча туюнтмаларды жазууда ажыратуу белгиси катары колдонулат.

Мисалы:

for (i=0,j=50; i<j ; i++,j--);

бул мисалда циклдин өзгөрүү параметири катары кош i, j параметирлери колдонулган.

Шарттан кийинки кайталануучу оператор

Шарттан кийинки кайталануучу операторду жалпы түрүнүн берилиши *while(<туюнтма>)<циклдин тулкусу>;* Бул операторду аткарылышы адегенде туюнтманын мааниси эсептелинет, эгерде ал чын болсо <циклдин тулкусу> аткарылат ал эми жалган болсо иш аракет цикл операторунан кийинки операторго берилет. <циклдин тулкусу> жөнөкөй, курама же куру операторлор болушу мүмкүн. Чексиз кайталануу болбошу үчүн <туюнтманын> маанисин өзгөртүп туруучу иш аракеттер <циклдин тулкусунда> болушу зарыл.

Мисалы: эки сандын эң чоң жалпы бөлүүчүсүн эсептөөчү программаны карайлы.

```
#include <stdio.h>
# include <conio.h>
      main ()
      { int x,y;
      clrscr();
      printf (“Эки бүтүн санды киргиз\n”);
      scanf(”%d,%d”,&x,&y);
      while (x!=y)
      { if  (x>y)x-=y;
      else y-=x;
        }
      printf (“Эки сандын эң чоң жалпы бөлүүчүсү=%d”,x);
      }
```

Жыйынтык: Эки бүтүн санды киргиз

9,12

Эки сандын эң чоң жалпы бөлүүчүсү=3

Шартка чейинки кайталануучу оператор

Жалпы жазылышты: *do<циклдин тулкусу> while (<туюнтма>);* Бул оператордун аткарылышы шарттан кийнки кайталануучу операторго эле окшош, бир айырмачылыгы <циклдин тулкусу> сөзсүз бир жолу аткарылат.

1-Мисал. Берилген аралыкта, x санынын жатаарын аныктоочу праграмма.

```
      #include <stdio.h>
       # include <conio.h>
       main()
       { int a,b,x;
       clrscr();
       printf (“a жана b аралыгын бергиле\n”);
       scanf(“%d,%d”,&a,&b);
```

```
do{printf (“x санын киргиз \n”);
scanf(“%d”,&x);
if(x<a|| x>b)printf(“x берилген аралыкта  жатпайт \n”);
}while(x<a||x>b);
printf(“x саны берилген аралыкта  жатат \n”);
printf(“%d<%d<%d”,a,x,b);
}
```

Жыйынтык: a жана b аралыгын бергиле

```
  5,9
x санын киргиз
7
x саны берилген аралыкта  жатат
5<7<9
```

2-Мисал. Сандарды баскычтар аркылуу киргизип жана алардын суммасын эсептөөчү программа.

```
#include <stdio.h>
# include <conio.h>
 main()
 {
 int sum,n;
 clrscr();
 printf (“бүтүн сан киргизүүнү башта \n ”);
 printf(“бүтүн сандарды киргизүүнү аяктоо үчүн 0дү басыңыз
\n”)
  sum=0;
  do{ printf (“бүтүн сан киргиз \n ”);
  scanf(“%d”,&n);
  sum+=n;
  }while (n!=0);
```

```
  printf(“sum=%d”,sum);
 }
```

Жыйынтык: бүтүн сан киргизүүнү башта

бүтүн сандарды киргизүүнү аяктоо үчүн 0дү басыңыз
бүтүн сан киргиз
1
бүтүн сан киргиз
3
бүтүн сан киргиз
5
бүтүн сан киргиз
7
бүтүн сан киргиз
9
бүтүн сан киргиз
0
sum=25

2.6 Тандоочу, токтоочу жана улантуучу операторлор

Тандоочу оператор.

Көптөгөн маселелерде кандайдыр бир туюнтманын маанилерине карата бир канчалаган багыттар боюнча иш аракеттер аткарылат. Ошондуктан мындай маселелерди программалоодо Си тилинде өзүнчө оператор колдонулат. Бул операторду тандоо оператору деп аташат.

Тандоо операторунун жалпы түрү.

```
switch <туюнтма>
{case n1:m1; break;
case n2:m2; break;
  . . .
case nk:mk; break;
```

default: m; break

}

Тандоо оператору эки бөлүктөн турат:-операторду*н башы* жана *тулкусу*.

башы switch <туюнтма>

тулкусу {case n1:m1; break;

case n2:m2; break;

. . .

case nk:mk; break;

default: m; break

}

Тандоо операторунун толук жана толук эмес болуп эки түрү кезигет. Толук эмес түрүндө default: m; break сабы колдонулбайт. Бул жалпы берилиште <туюнтма>-бүтүн маанилүү туюнтма же символдук туракту*улар* болушу мүмкүн. n1, n2, …, nk-белгилери б.а. бүтүн туруктуулар же бүтүн маанилүү туюнтма же символдук туракту*улар*. m1, m2, …, mk-операторлор. Тандоо операторундагы кызматчы сөздөрдүн мааниси switch-бөлүштүрүү, case-тандоо, default-дал келбөө, break-токтоо.

Операторду*н* аткарылышы адегенде <туюнтманын> мааниси эсептелинет да анын мааниси n1, n2, …, nk туракту*улары* менен салыштырылат. Салыштыруу учурунда <туюнтманын> мааниси туракту*улардын* кайсы бирине дал келсе анда ошол туракту*у* көрсөткөн оператор аткарылат. *Мисалы:* n2 дал келсе m2 операторду аткарылат ж.б.у.с. Эгерде <туюнтманын> мааниси эч бир n1, n2, …, nk туракту*ууларына* дал келбесе, анда default кызматчы сөзүнөн кийин m деген оператор аткарылат.

Тандоо операторун колдонууда белгилерди (n1, n2, …, nk) ирээтеп жазуу талап кылынбайт, ошондой эле default кызматчы сөзү сөзсүз түрдө эле тандоочу операторду*н* аягында жазылбайт, тандоочу операторунун тулкусунун каалагандай жеринде

жайгаша алат жана default сөзүн такыр эле колдонбой койсо деле болот. Бул учурда <туюнтманын> маниси эч бир n1, n2, …, nk турактуулардын маанисине дал келбейт жана default сөзүнөн кийинки m деген оператор жок дегенди билдирет б.а. switch оператору эч кандай иш аракетти аткарбай калат. Бул бөлүштүрүү операторунун толук эмес түрүнө дал келет. Тандоо операторунда n1, n2, …, nk белгилери бирин-бири кайталабаган турактуулар болушу керек. Ал эми break (үзүлүү) оператору аткарылганда башкаруу switch-тандоо операторунун тулкусунан кийинки операторго берилет. Break оператору колдонулбай калган учуру да кезигет. Эгерде break оператору кандайдыр бир mi операторлордун тобунан кийин жазылбаса, анда mi, mi+1, mi+2,... ж.б.у.с. операторлору break кайрадан жолукканга чейин аткарылат. Жалпы жазылышы төмөндөгүдөй:

```
switch <туюнтма>
{case n1:m1; break;
  case n2:m2; break;
     . . .
  case ni-1:mi-1; break;
  case ni:mi;
  case ni+1:mi+1;
     . . .
  case nk:mk; break;
  default: m; break;
}
```

Эгерде эки же андан көп иш аракеттер дал келишсе б.а. m1, m2, …, ml операторлору бирдей болсо алардын жалпы жазылышы төмөндөгүдөй болот.

```
switch <туюнтма>
```

```
{ case n1:
  case n2:
  . . .
  case nl:ml; break;
  . . .
  case nk:mk; break;
  default: m; break;
}
```

Жогорку тандоо операторуна мисал карайлы.

Мисал: Арифметикалык амалдарды аткаргыч программаны түзөлү.

```
# include <stdio.h>
 main ( )
 {float a,b;
  char z;
  printf (“\n Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:”);
  scanf (“%f %c %f”, &a,&z,&b);
  switch (z)
 { case ‘+’:printf (“%f%c%f=%f\n”,a,z,b,a+b); break;
   case ‘-’:printf (“%f%c%f =%f\n”, a,z,b,a-b); break;
   case ‘*’:printf (“%f%c%f =%f\n”, a,z,b,a*b); break;
   case ‘/’:printf (“%f%c%f =%f\n”, a,z,b,a/b); break;
   case ‘>’:;
   case ‘<’:;
   case ‘=’:printf (“\n бул катыш амалы”); break;
   default:printf (“\n бул белгисиз амал”);
  }
}
```

Жыйынтык: Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

3+5

3.000000+5.000000=8.000000

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

9-7

9.000000-7.000000=2.000000

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

5*7

5.000000*7.000000=35.000000

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

9/5

9.000000/5.000000=1.800000

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

6>2

бул катыш амалы

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

5<7

бул катыш амалы

Амал белгисинин оң жагындагы жана сол жагындагы сандарды жана белгинин арасына ачык калтырбай киргиз:

8^9

бул белгисиз амал

Токтоочу оператор

Токтоочу операторлун жалпы түрү: break;

Бул оператор кайталанууну же баяндоону токтотуу үчүн колдонулат. break оператору for, while, do, switch операторлорунун тулкусунда кезигип калса, анда кайталануу же тандоо иш аракеттери токтотулат. Токтоочу операторлун кайталануучу жана тандоочу операторлордон башка жерлерде жазылышы кашаага алынып калат, ошондой эле башкы функциянын (main) тулкусунан чыгуу үчүн колдонууга болбойт.

1-Мисал. z -тамгасын киргизгенге чейинки, мурунку киргизилген тамгаларды чыгаруучу программа.

```
# include <stdio.h>
  main()
  {char ch;
   for (; ;) {ch=getchar();
   if (ch= =’z’) break;
   printf (“%c”,ch);
        }
}
```

Жыйынтык:

```
a
a
b
b
z
```

Эгерде кирүү сабы abcdzefh түрүндө болсо z ке чейинки гана abcd маалыматы чыгарылат.

2-Мисал. 0 дөн 100 чейнки сандардын ичинен 100 дөн квадраттары кичине болгон сандарды чыгаруучу программа.

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

```
 main ()
  {int i ;
   printf (“ Сан     квадраты\n”
   for (i=0; i<100; i++)
   { if(i*i>=100)break;
   printf (“%d     %d\n”, i, i*i );

   }
  }
```

Жыйынтык :

Сан	квадраты
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81

Улантуучу оператор.

Бул операторду жалпы түрү:continue;

Улантуучу операторду кайталануу процессинин кандайдыр бир бөлүгүн калтырып, кийинки кайталануучу циклди аткарууну ишке ашырат. Эгерде кийинки кайталануу (цикл) жок болсо (түгөнсө) анда кайталануу иш аракеттери аяктайт. Демек continue оператору while жана do операторлорунда колдонулса өзүнөн кийин башкарууну while же do операторлорундагы шарттарды текшерүүгө берет, ал эми for операторунда болсо циклдин

параметиринин өзгөрүү кадамына берилет. Мисалы: 3кө эселүү болгон 20 га чейинки сандарды чыгаруучу программа.

```
# include <stdio.h>
   main()
  {int i;
   for(i=1; i<20; i++)
   { if (i%3) continue ;
    printf (“%d\n”,i);
   }
}
```

Жыйынтык:

3
6
9
12
15
18

Программага түшүндүрмө берели. Кайталануу 1 ден 20 чейин жүрөт. Эгерде санды 3 кө бөлүүнүн калдыгын табуучу (%) амал калдык 0 болсо шарттуу оператор if шартты же туюнтманы жалган деп эсептеп printf(“%d\n”,i); оператору аткарылат. Эгерде санды 3 кө бөлгөндө калдык 0 дөн айрмалуу болсо анда шарт же туюнтма чын деп эсептеп continue; оператору аткарылып, for операторунун циклинин кийнки кадамы аткарылат, мындай процесс i=20 болгонго чейин жүрөт.

Текшерүүчү суроолор

1. Си тилиндеги амалдар
2. Чоңойтуучу жана кичирейтүүчү амалдардын мааниси
3. Менчиктөө амалынын өзгөчөлүгү

4. Логикалык жана катыш амалдары
5. Разряддык амалдар
6. Операторлордун бөлүнүшү
7. Менчиктөө оператору
8. Шартуу амал
9. Шартуу оператор
10. Цикл оператору
11. Шартка чейинки цикл оператору
12. Шарттан кийинки цикл оператору
13. Тандоочу оператор
14. Токтоочу оператор
15. Улантуучу оператор

Си тилинин амалдарына жана операторлоруна көнүгүүлөр

1. int n; float x жана n=9 болсо x=n/2; x=(float)n/2 операторунун жыйынтыктары кандай болот?

2. Эгерде int x,y,z болсо z=(y=(x=5)*2+30) менчиктөө амалынын жыйынтыгын тап.

3. Эгерде x=10 болсо x+=5 менчиктөө амалынын жыйынтыгын тап.

4. x өзгөрмөсү (0, 10) интервалында жатат деген шартты Си тилинде кандай жазабыз?

5. int a,b,c жана a=3; b=5 болсо c=(a++)+(++b); оператору аткарылса a,b,c өзгөрмөлөрүнүн маанилерин тап.

6. Үч бурчтуктун бир негизги жана ага түшүрүлгөн бийиктиги берилсе, анын аянтын эсептөөчү программа түз.

7. Үч бурчтуктун үч жагы берилсе, анын аянтын эсептөөчү программаны түз.

8. Ромбунун диоганалдары берилсе, анын аянтын эсептөөчү программаны түз.

9. Трапециянын эки негизи жагы жана бийиктиги берилсе, анын аянтын эсептөөчү программаны түз.

10. Тегеректин радиусу берилсе анын аянтын эсептөөчү программаны түз.

11. Төрт орундуу натуралдык сан берилсин. Ал сандын цифралары ар түрдүүбү же андай эмеспи аныкта.

12. Үч орундуу натуралдык сан берилсин. Ал сандын биринчи жана акыркы цифрасын аныкта.

13. Борбору координата башталышында жаткан радиусу белгилүү болгон айлана жана тегиздиктен координаталары (x,y) болгон чекит берилген. Ушул чекит айлананын ичинде же айлананын өзүндө же айлананын сыртында жатаарын аныктагыла.

14. x,y жана z үч натуралдык сандары берилсин. Ушул үч сан үч бурчтуктун жактарын түзөөрүн аныктагыла.

15. a жана в сандары берилсе ах=в теңдемеси кандай чыгарылыштарга ээ болот?

16. n-суммадагы акча берилсин. 3 жана 5сомдуктар менен кантип туюнтууга болот?

17. $x^2+y^2=z^2$ теңдемесинин 100гө чейинки бүтүн чыгарылыштарын тапкыла?

18. 100гө чейинки 5ке жана 7ге бөлүнгөн сандардын суммасын тапкыла?

19. Эки орундуу сандардын ичинен 3кө бөлүнгөн сандардын көбөйтүндүсүн тапкыла?

20. Үч орундуу сандардын ичинен цифралары бирдей болгон сандардын суммасын тапкыла? 111+222+333+…

3-Бөлүм. КӨРСӨТКҮЧТӨР

3.1. Көрсөткүч түшүнүгү

Программада өзгөрмөлөрдү баяндоо – бул ошол өзгөрмөлөр үчүн эстен орун бөлүнөт дегенди билдирет жана компьютердин эсинен өзгөрмөлөрдүн мааниси үчүн да уячалар берилет. Уячалардын өлчөмү өзгөрмөнүн тибине көз каранды жана ар бир өзгөрмө адреске жана мааниге ээ болот. ***Адрес*** – бул өзгөрмө жайгашкан учурдагы эстеги орун. Адрес эстерди бөлүштүрүүдө аныкталып программа аркылуу өзгөртүлбөйт. Маани болсо өзгөрмө үчүн берилген эстин бөлүгүндөгү кармалган маалымат. Өзгөрмөнүн мааниси программа иштеши менен көп жолу өзгөрүшү мүмкүн.

Көрсөткүчтөр учурдагы эсте жайгашкан объектилерди (өзгөрмө, функция ж.б.) ордун же адресин аныктоодо колдонулат.

Көрсөткүч – деп мааниси адрес болгон өзгөрмөнүн тибин айтабыз. Көрсөткүчтөр башка өзгөрмөлөр сыяктуу эле адреске жана мааниге ээ болот. Көрсөткүчтүн мааниси катары кандайдыр бир өзгөрмөнүн адреси эсептелинет. Көрсөткүчтөр өзгөрмө катары программада баяндалат. Жалпы жазылышы:

<маалыматтардын тиби> *<көрсөткүчтүн ысымы>;

<маалыматтардын тиби> - маалыматтардын жөнөкөй типтери

<көрсөткүчтүн ысымы> - өзгөрмөлөрдүн ысымдарына окшош.

Мисалы:

int *n, *m;

char * p

Көрсөткүчтү баяндоочу өзгөрмөнүн ысымынын алдындагы * белги көрсөткүч баяндалып жатканын билгизет. Жогорудагы мисалда n жана m өзгөрмөлөрү int(бүтүн) типтеги өзгөрмөлөрдүн адрестерин кармайт, ал эми p болсо char

тибиндеги өзгөрмөнүн адресин кармайт. int n жана int *p баяндоолорун бирдиктүү жазса болот int n, *p.

Көрсөткүчтөрдү колдонуу чөйрөсү кеңири, мисалы туунду типтер болгон массивтер жана саптар менен иштөөдө жана эсти динамикалык бөлүштүрүүдө. Ошондой эле эстин каалагандай уячасын көрсөтүүдө жана түзүлүштөр менен иштөөдө колдонулат.

3.2 Адрестик амалдар.

Көрсөткүчтөр менен тыгыз байланышта болгон эки адресттик амалдар бар.

1) Адрести аныктоо амалы & - белгиси менен белгиленет же көрсөткүч амалы
2) Адрес боюнча кайрылуу амалы * - белгисин колдонот.

Эгерде a – кандайдыр бир өзгөрмө болсо, анда &a ошол өзгөрмөнүн адреси болот. Бул адрестин маниси көрсөткүчкө менчиктелиши мүмкүн. Мисалы p - көрсөткүч болсо анда *p , p көрсөткүчү кармаган адресте жаткан маани болот. Ушул сыяктуу эле sap өзгөрмөсү char тибине болгон көрсөткүч болсо, анда *sap sap-көрсөткүчү көрсөткөн символ болуп эсептелет. Төмөндөгү амалдарды карайлы:

y=&x x-өзгөрмөсүнүн адресин y өзгөрмөсүнө менчиктейт. & - амалын туракттууларга жана туюнтмаларга колдонууга болбойт. *Мисалы*: &(x+10) же &2010 ж.б.

z=*y y адреси боюнча жазылган өзгөрмөнүн маниси z ке менчиктелет.

y=&x жана z=*y болсо z=x болот. *y=7 y адреси боюнча эстин уячасына 7 санын жайгаштырат.

*x+=9 x адреси боюнча эсте жаткан мааниге 9 санын кошот (суммалайт).

(*z)++ z адреси боюнча эсте жаткан маанини бирге чоңойтот.*z++ менен (*z)++ айырмасы бар, *z++ бул адрестин өзүн бирге чоңойтот.

Көрсөткүчтөр катышкан программанын *мисалдарын* карайлы:

1-Мисал.

```
#include <stdio.h >
#include <conio.h >
  main ( )
   {int a, *p; /*a-бүтүн сан, ал эми , p-бүтүнгө болгон көрсөткүч*/
    /*б.а. бүтүн сандарды кармаган адрести кармайт */
   clrscr();
   p=&a;
   a=5;
   printf (“%d\n”, *p);  /*5 ти чыгарат*/
   a++;
   printf(“%d\n”, *p);  /*6 ны чыгарат*/
   (*p)--;
    printf(“%d\n”, *p);   /*5 ти чыгарат*/
    (*p)=9;
    printf (“%d\n”, *p);  /*9  ду чыгарат*/
   }
```

Жыйынтык:

5

6

5

9

2-Мисал.

```
#include <stdio.h >
#include <conio.h >
   main( )
  {int b, *q; /*b – бүтүн маанилүү өзгөрмө, ал эми q – болсо бүтүндү  көрсөткүч */
```

```
    q=&b;          /*q  өзгөрмөсү  b бүтүн маанилүү өзгөрмөнүн адресин кармайт*/
    clrscr();
    b=2010;
    printf("b ны жайгаштыруу: %p\n",&b);
    printf("b нын кармаган мааниси: %d\n",b);
    printf("q нун кармаган мааниси: %p\n",q);
    printf(" %p ",q);
    printf("Адресиндеги маани: %d\n",*q);
}
```

Жыйынтык:

b ны жайгаштыруу : FFF4

b нын кармаган мааниси : 2010

q нун кармаган мааниси FFF4

FFF4 Адресиндеги маани 2010

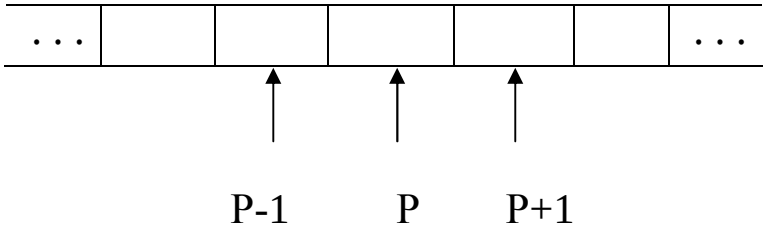
Жогорку программада b=2010 ордуна *q=2010 деп жазсак да болот, экинчи менчиктөө адрес боюнча менчиктелди дегенди билдирет.

Көрсөткүчтөрдүн үстүнөн бүтүн сандарды кошуу жана кемитүү амалдарын жүргүзүүгө болот. Мындай амалдардын натыйжасы да көрсөткүч болот. Амалдарды аткарууда көрсөткүч кандай типке көрсөтүү жүргүзүп жатканы эске алынышы керек.

Мисалы: учурдагы эстин удаалаш жайланышкан уячаларында бүтүн маанилер кармалсын.

Мейли p- көрсөткүчү эстин уячаларынын ортоңку адресин кармасын, анда

p-1 мурунку адрести, ал эми p+1 кийинки адрести кармайт.



3.3 Динамикалык өзгөрмө

Си тилинде өзгөрмөлөрдү статикалык (туруктуу) жана динамикалык өзгөрмөлөр деп экиге бөлүшөт. Статикалык өзгөрмөлөр программанын баяндоо бөлүгүндө аныкталып жана ысымга ээ болот. Ар бир статикалык өзгөрмөгө транслятор белгилүү санда эстин уячаларынан программа иштеп бүткөнгө чейин орун бөлөт жана аны сактап турат.

Динамикалык өзгөрмөлөр программада ысымга ээ болбойт, транслятор динамикалык өзгөрмөгө эстен зарыл болмоюнча орун бөлбөйт. Программанын иштөөсүндө мындай өзгөрмөлөр зарыл болгондо пайда болуп, кереги жок болуп калганда жоюлуп турат. Ошого карата транслятор учурдагы эстен динамикалык өзгөрмөгө орун бөлүп, зарылдыгы жок болгондо ал орунду бошотуп турат. Динамикалык өзгөрмөлөрдүн ысымы жок болгондуктан, аны көрсөткүчтөрдүн жардамы менен иштетишет.

Программанын аткарылышында көрсөткүчтөрдүн жардамы менен жаңы динамикалык өзгөрмөлөрдү түзүлүп, алар иштетилип, андан ары кереги жок болгондо жоюлуп турат. Программист тарабынан программада учурдагы эсти бөлүштүрүү эсти- динамикалык бөлүштүрүү деп аталат.

Динамикалык өзгөрмөлөрдү түзүү.

Динамикалык өзгөрмөлөрдү пайда кылуу үчүн Си тилинде эки функция колдонулат, бул функциялар malloc жана calloc кызматчы сөздөрү менен берилет. Жогорудагы эки функция alloc.h файлында баяндалган жана бул функцияларды колдонууда программанын башында сөзсүз # include <alloc.h> сабы болушу керек. Функциялардын жазылышы malloc(<эстин көлөмү>) бул берүүдө <эстин көлөмү> - бөлүштүрүүгө зарыл болгон учурдагы эстин бөлүгү.

Экинчи функциянын жазылышы calloc(<элементтердин саны>, <эстин көлөмү>).

<элементтердин саны> - учурдагы эстин бөлүгүн бөлүү үчүн оң бүтүн типтеги элементтердин саны.

<эстин көлөмү> - ар бир элемент үчүн зарыл болгон эстин көлөмү.

Эки функция тең учурдагы эсти бөлүштүрөт жана бөлүнгөн эстин аймагынан адресин кармаган көрсөткүчкө куру типти void ыйгарат. Эгерде эс жетишпесе жогорку функциялар NULL деген маанини берет.

Массивтерди автоматтык түзүүдө ыңгайлуу болуш үчүн calloc – функциясы эсти бөлүштүрүүдөн башка да ар бир элементке 0(нөл) деген маанини ыйгарат.

Функциялар void (куру) типке ээ болушу тиби жок көрсөткүчтү берет дегенди билдирет, мындай көрсөткүчтөргө маалыматтардын каалагандай тибин менчиктөөгө болот. Зарыл болгон эстин көлөмүн аныктоо үчүн sizeof (<өлчөм>) амалы колдонулат. Бул амал эки түрдө берилет:

1 – түрү sizeof(<туюнтма>)

2 – түрү sizeof(<тип>)

1 – түрүндөгү берилишинде туюнтма тураktуу же өзгөрмө болуп калышы мүмкүн.

Мисалы:

```
#include <stdio.h >
 #include <conio.h >
main( )
{int  b;
float d[50];
clrscr();
printf("\n бүтүнгө берилген эстин өлчөмү %d ", sizeof(int));
printf("\n b-өзгөрмөсүнө берилген эстин өлчөмү %d",sizeof(b));
printf("\n d га берилген эстин көлөмү %d ", sizeof d);
 }
```

Жыйынтык

бүтүнгө берилген эстин өлчөмү 2

b-өзгөрмөсүнө берилген эстин өлчөмү 2

d га берилген эстин көлөмү 200

Бүтүн тип 2 байт өлчөмдү алгандыктан, эстен дагы ошончо орун бөлүнөт. Ал эми float типтүү өзгөрмө 4 байт өлчөмүндө болгондуктан 50*4=200байт болот.

Эсти динамикалык бөлүштүрүүгө жана int тибиндеги бир өзгөрмөнү түзүүгө мисал:

```
# include <stdio.h>
# include <alloc.h>
  main( )
   {int  *byt;      /*бүтүн мааниге болгон көрсөткүч*/
     byt= (int*) malloc(sizeof(int));
   *byt=2010;    /*динамикалык өзгөрмөнүн мааниси 2010*/
    printf("byt-өзгөрмөнүн мааниси= %p\n",byt);  /*адрести
чыгаруу*/
     printf("Адресттеги кармалган маани=%d\n",*byt);
}
```

Жыйынтык

byt-өзгөрмөнүн мааниси = 05B6

Адресттеги кармалган маани=2010

Жогорудагы программанын 5-сабындагы byt=(int*)malloc(sizeof(int)) оператор төмөндөгү иш-аракеттерди аткарат:

1) sizeof (int) амалы бүтүн(int) типтүү өзгөрмөнү сактоо үчүн зарыл болгон эстин өлчөмүн байттардын саны менен аныктайт.Жыйынтыгы кандайдыр бир n бүтүн санын берет.

2) malloc (n) функциясы компьтердин учурдагы эсинен n сандагы удаалаш бош орунду ээлейт жана маанилерди жайгаштыруу үчүн башталыш адрести берет.

3) (int*) туюнтмасы бүтүн типтүү маалыматка болгон башталыш адрести кармаган көрсөткүч.

4) malloc функциясы аркылуу алынган адрес byt өзгөрмөсүнө менчиктелет жана бүтүн (int) типтүү динамикалык өзгөрмө түзүлөт,ага кайрылуу *byt ысымы менен жүргүзүлөт.Демек жогорку иш-аракеттерден кыскача корутунду чыгарсак:

Компьютердин эсинен бүтүн (int) өзгөрмөсү үчүн орун бөлүп, ал орундун башталыш адресин byt өзгөрмөсүнө менчиктейт. Бул өзгөрмө бүтүн типтеги өзгөрмөгө болгон көрсөткүч болуп эсептелет.

Көрсөткүчтү колдонуу эрежеси: көрсөткүчтү программада колдонууга чейин, дайыма адреске ээ болушу керек.

Эсти динамикалык бөлүштүрүүгө жана адрестик амалдарга мисал карайлы.

```
#include <stdio.h>
#include <alloc.h>
main( )
 {  int i,*p;
 p=(int*)calloc (3,sizeof (int));
    *p=19;
    *(p+1)=9;
    *(p+2)=1959;
     printf (“\n Адрестердин тизмеси:”);
    for(i=0;i<3;i++)
   printf (“  %4p”,(p+i));
   printf (“\n Маанилердин тизмеси:”);
   for (i=o; i<3; i++)
    printf (“  %4d “,*(p+i));
}
```

Жыйынтык

адрестердин тизмеси: 05D0 05D2 05D4

маанилердин тизмеси: 19 9 1959

Жогорудагы программа бүтүн (int) тибиндеги үч динамикалык өзгөрмөнү түзөт.

p-көрсөткүчү 3*2=6 байт өлчөмүндөгү эстин бөлүгүн адресин көрсөтөт, бул эстин бөлүгү бүтүн (int) тибиндеги элементтерди сактоого үчүн жетиштүү, ал эми p+i болсо p+(i*sizeof(int)) туюнтмасы аныктаган эстин бөлүгүнүн адрестерин көрсөтөт. Көрсөткүчтөрдү да программада баяндоодо мааанилештирүүгө болот.

Мисалы: int *p=(int*)malloc(sizeof(int)).

Баяндоо учурунда эле бүтүн (int) тибине болгон көрсөткүч баяндалып, анын баштапкы мааниси malloc – функциясы берген адрес болуп эсептелет.

3.4 Бөлүнгөн эсти тазалоо

Динамикалык бөлүнгөн эс зарылчылыгы жок болуп калганда, башка максаттарга колдонуу үчүн аны тазалоого туура келет. Динамикалык бөлүнгөн эсти тазалоо free – функциясы аркылуу жүргүзүлөт. Бул функциянын баяндалышы alloc.h функциясында баяндалган. Жалпы жазылышы free (<көрсөткүч>);

<көрсөткүч> - тазалоочу эстин бөлүгүнүн башталыш адресин кармаган куру(void) типтүү көрсөткүч.

free – функциясынын иштөөсүндө динамикалык өзгөрмө үчүн бөлүнгөн эс тазаланган болсо, андай динамикалык эске көрсөтүү катага алып келинет, мындай көрсөтүүгө өтө сак болуш керек. Динамикалык бөлүнгөн эсти тазалоого мисал карайлы:

```
# include <stdio.h>
# include<alloc.h>
  main( )
   {int  *p,i;
    p=malloc(10*sizeof(int));
    if(!p) {printf("Эстин көлөмү жетишсиз \n");
          exit(1);
         }
  for(i=0; i<10; ++i) *(p+i)=i
```

```
    for(i=0; i<10; ++i) printf("%d",*(p++));
    free(p);
    return 0;
}
```

Текшерүүчү суроолор

1. Динамикалык жана статикалык өзгөрмөлөр
2. Адрес жана маани түшүнүгү
3. Көрсөткүч түшүнүгү
4. Көрсөткүчтү колдонуу
5. Адрести аныктоо амалы
6. Адрес боюнча кайрылуу амалы
7. Көрсөткүчтөрдүн үстүнөн жүргүзүлгөн амалдар
8. Динамикалык өзгөрмөлөрдү түзүү
9. Динамикалык өзгөрмөлөрдүн үстүнөн аткарылуучу функциялар
10. Бөлүнгөн эсти тазалоо

Көрсөткүчтөргө берилген көнүгүүлөр

1. a=&в менчиктөөсү эмнени билдирет?
2. y=&x; y=&(x+10) жана y=&2003 жазууларынын туурасын тап.
3. y=&x жана z=*y болсо z тин маанисин тап.
4. *a=7 эмнени билдирет?
5. *в+=5 кандай натыйжа берет?
6. (*y)++ жана *y++ айырмасын тап?
7. # include <alloc.h> буйругунун мааниси кандай?
8. byt=(int*) malloc (sizeof (int)); оператору эмне иш аракеттерди аткарат?
9. int *p;
   p=(int*) calloc (3, sizeof (int)); оператору кандай иш аракеттерди аткарат?
10. int *p;
    free (p)-функциясы кандай иш аракет аткарат?

4-Бөлүм ТУУНДУ ТИПТЕР

4.1 Массивтер

Маалыматтардын туунду типтери деп, негизги типтерден маалыматтардын куралып түзүлүшүн айтабыз. Программа түзүүдө бир типтеги көп өзгөрмөлөр менен иштөөгө туура келет. Көп өзгөрмөлөрдүн ысымдарын программага колдонуу, анын көлөмүн чоңойтуп иштөөсүнүн тездигин азайтат, ушул себептен улам массив түшүнүгү программа түзүүдө колдонулат.

Массив - деп бир ысымдуу, бир типтеги, чектелген, ирээттелген жана өлчөмгө ээ болгон маалыматтардын жыйындысын айтабыз.

Массив түшүнүгү беш мүнөздөмөгө ээ болот: массивтин ысымы, элементтеринин номери, элементтеринин типтери, өлчөмү жана элементтеринин саны.

Массивтин элементтеринин номерин индекс деп да аташат. Си тилинде массив төмөндөгүдөй жарыяланат:

<тип><массивтин ысымы.> [n1][n2]…[nk]

Бул жазууда <тип>-массивтин элементтеринин тиби. n1,n2,…nk – массивтин өлчөмдөрү. Массив бир өлчөмдүү болсо a[n]; эки өлчөмдүү болсо матрица деп аталып b[n] [m], массив үч өлчөмдүү болсо c[n] [m] [k] деп жазылат. Массивтердин индекстери 0 дөн башталат.Мисалы int a[15] ; берилсе анда:

a[0] - 1 элементи,

a[1] - 2 элементи,

a[2] - 3 элементи

….

a[13]- 14 элементи

a[14]- 15 элементи деп эсепетелинет.

Ал эми эки өлчөмдүү массивти же матрицаны карасак.

Мисалы float b[2] [3] .

Бул матрица эки сап жана үч мамычадан турат. Элементтери калкыма чекиттүү болуп, учурдагы эсте төмөндөгүдөй тартип менен жайгаштырылат b[0][0]; b[0][1]; b[0][2]; b[1][0]; b[1][1]; b[1][2].

Си тилинде массив үчүн учурдагы эстен дайыма орун бөлүнөт жана массивтин индекстери чектен чыгып кетүүсү текшерилбейт. Мисалы: int a[50] деп жарыяланса, ал эми программада a[100] көрсөтүлсө, бул катага алып келбейт. Себеби a[100] элементи катары массивтин башынан баштап кайрадан менчиктелүү жүрөт. Индекс жок массивтин ысымы массивтин башталышынын адресин берген туракту (константа) болуп эсептелет.

Эки өлчөмдүү массивтин учурдагы эсте жайгашаарын карайлы.

Мисалы: int a[2][3] анда жогорудагыдай a массивинин элементтери төмөндөгүдөй жайгаштырылат:

a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2] эгерде a=1000 болсо, анда a[0][1] элементинин адреси 1002 болот, себеби int тиби эсте 2 байт өлчөмдү ээлейт, ал эми массивтин кийинки элементи a[0][2] болсо1004 адреске ээ болот ж.у.с.

Массивтер менен иштөөдө ага маанилештирүү иш-аракеттерин да жүргүзүүгө болот. ***Маанилештирүү***- деп массивти баяндоодо анын маанилерин б.а. элементтерин кошо берүүнү айтабыз.

Мисалы: int b[3]={1,2,3} бул учурда b[0]= 1; b[1]=2; b[2]=3 маанилерин алышат. Көп өлчөмдүү массивтерди маанилештирүүгө *мисал:*

int a[2][3]={1,2,3,4,5,6} бул маанилештирүү төмөндөгү менчиктөө операторлорунун тобуна тең күчтүү болуп калат.

a[0][0]=1; a[0][1]=2; a[0][2]=3; a[1][0]=4; a[1][1]=5; a[1][2]=6

Көпчүлүк учурда көп өлчөмдүү массивтерге маанилештирүүдө кабатталган маанилештирүүнү да колдонушат.

Мисалы: int a[2][3]={{1,2,3},{4,5,6}}.

Кээ бир учурда программада окулушу жөнөкөй болуш үчүн төмөндөгүдөй да маанилештирүүнү колдонушат.

```
int mas[5][3]={{1,1,1},
               {2,2,2},
               {3,3,3},
               {4,4,4},
               {5,5,5}};
```

Символдук массивтерди да маанилештирсе болот.

Мисалы: char s[10]={‘t’,’u’,’r’,’b’,’o’,’ ‘,’c’}

Сап маанилүү өзгөрмөлөрдү да маанилештирүүнү карайлы. *Мисалы:* char sap1[7]=”Салам!”

Саптын узундугу 7, себеби саптын аягын билдирүүчү символго бир орун берилет. Саптын узундугун чарчы кашаада жазбай койсо да болот.

Мисалы: char sap2[ ]={‘с’,’а’,’л’,’а’,’м’,’!’,’\o’};

Жогорудагы сап массивин маанилештирүүнүн стандарттык формасы болуп эсептелет. Бул формадагы ‘\o’ белгисинин бар болушу сап массиви экендигин далилдеп турат, эгер бул белги жок болсо анда белгилердин гана массиви болуп калмак.

Массивтерди түзүүгө жана анын элементтерин иштетүү карата мисалдарды карайлы. Мейли бир өлчөмдүү массивтин эң чоң элементтин жана терс элементтеринин санын аныктоочу программаны түзөлү.

```
#include <stdio.h>
#include <conio.h >
    main( )
    {int a[15],      /*а масивин баяндоо*/
    n,               /*а массивин элементтерин санын аныктоо*/
```

```
    i,                  /*массивтин индекси*/
    max,                /*а массивинин элементтеринин максималдуу
мааниси*/
    k;                  /*а массивинин терс элементтеринин саны*/
    /*а массивинин элементтерин киргизүү*/
    clrscr();
    do
    {
    printf(“15 тен ашпаган элементтердин санын бер n=”);
    scanf(“%d”,&n);
    }
    while(n<1|| n>15);
    for(i=0;  i<n;  i++){  printf(“Масситин  %d-элементин
киргиз”,i+1);
                        scanf(“%d”,&a[i]);};
    /*а массивинин эң чоң элементин аныктоо*/
    max=a[0];
    for(i=1;i<n;i++);
    if(max<a[i])max=a[i];
    printf(“\n Массивтин эң чоң элементи=%d” , max);
    /*а массивинин терс элементтеринин санын аныктоо*/
    k=0;
    for (i=0;i<n ;i++) if (a[i]<0)k++;
    printf (“\n Массивтин терс элементтеринин саны=%d”,k);
    }
```

Жыйынтык:

```
15 тен ашпаган элементтердин санын бер n=5
Масситин 1-элементин киргиз 2
Масситин 2-элементин киргиз 25
Масситин 3-элементин киргиз -8
Масситин 4-элементин киргиз 7
```

Масситин 5-элементин киргиз -4

Массивтин эң чоң элементи=25

Массивтин терс элементтеринин саны=2

Массивтерди маанилештирүүгө мисал. Мейли a жана b матрицалары (эки өлчөмдүү массивтер) берилсин бул матрицалардын суммасын тапкыла:

```
#include <stdio.h>
#include <conio.h >
    main( )
    {int  a[3][4]={{1,1,1,1},{2,2,2,2},{3,3,3,3}},
    b[3][4]={{4,4,4,4},{5,5,5,5},{6,6,6,6}}, c[3][4],i,j;
    clrscr();
    for(i=0;i<3;i++)
    for(j=0;j<4;j++)  c[i][j]=a[i][j]+b[i][j];
    /*C матрицасын чыгаруу */
    for (i=0; i<3; i++);
    {for (j=0; j<4; j++);   printf("c[%d] [%d]=%d", i, j,c[i][j]);
    printf("\n"); };
    }
```

Жыйынтык:

```
c[0][0]=5
c[0][1]=5
c[0][2]=5
c[0][3]=5
c[1][0]=7
c[1][1]=7
c[1][2]=7
c[1][3]=7
c[2][0]=9
c[2][1]=9
```

```
c[2][2]=9
c[2][3]=9
```

4.2 Көрсөткүчтөр менен массивтердин ортосундагы байланыш

Си тилинде массивтер менен көрсөткүчтөр тыгыз байланышта. Массивтин ысымын массивтин башталыш элементинин адресин көрсөтүүчү катары караса болот. Тактап айтканда массивтин ысымы индекси жок жазылса мисалы a – бир өзгөрмөлүү массив болсо “a” жазуусу массивтин нөлүнчү элементинин адресин көрсөтөт б.а. “& a[0]” жазуусун билдирет. Ал эми көрсөткүчтөрдүн арифметикасы боюнча “a+i” –жазуусу i-элементтин адресин билдирет, ошондой эле “*(a+i)” – жазуусу “a[i]” жазуусуна тең күчтүү. Жалпылаганда төмөндөгү тең күчтүүлүк орун алат:

```
a==&a[0]
(a+i )==&a[i]
*(a+i)==a[i]
```

Бул жазуулар солдон оңду жана оңдон солду көздөй колдонуу туура болуп эсептелет. Массивтин ысымынын же көрсөткүч же массив катары баяндалышынын негизги айырмасы, массив өзүнүн жайланышына карата болот. Эгерде өзгөрмө массив катары жарыяланса, анда программа автоматтык түрдө керектүү өлчөмдө эстен орун бөлөт, ал эми массивтин ысымы көрсөткүч катары жарыяланса, анда программа calloc функциясын колдонуу менен эстен ушул массив үчүн орун бөлөт же мурун аныкталган эстин сегментинен массивтин ысымы үчүн адрес менчиктейт.

Өзгөрмөнүн массив катары баяндалышына мисал карайлы:

```
#include <stdio.h>
#include <conio.h >
    main ( )
    {int  a[3],i;
    clrscr();
    a[0]=30;
    a[1]=7;
    a[2]=2003;
    printf(“адрестин тизмелери:”);
    for(i=0;i<3;i++) printf(“%4p  ”,&a[i]);
    printf(“\n маанилердин тизмеси:”);
    for(i=0; i<3; i++) printf (“%-4d”,a[i]);
    }
```

Жыйынтык:

```
адрестин тизмелери: FFF0  FFF2  FFF4
маанилердин тизмеси:   30    7       2003
```

Кийинки *мисалда* а массиви көрсөткүч катары баяндалышын карайлы:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h >
    main ( )
    {int *pa,i;
    clrscr();
    pa=(int*)calloc(3,sizeof(int));
    *pa=30;
    *(pa+1)=7;
    *(pa+2)=2003;
    printf( “адрестердин тизмеси:”);
```

```
for(i=0;i<3;i++) printf(“%4p ”,(pa+i));
printf(“\n маанилердин тизмеси:”);
for (i=0;i<3;i++) printf(“%-4d ”, *(pa+i));
}
```

Жыйынтык:

```
адрестин тизмелери: 05D0  05D2  05D4
маанилердин тизмеси: 30     7       2003
```

Жогорудагы эки *мисалды* бир программа катары кароого болот:

```
#include <stdio.h>
#include <alloc.h>
  main ( )
 {int  *pa, a[3],i;
    pa= a;
   *pa=30;
   *(pa+1)=7;
   *(pa+2)=2003;
   printf(“адрестердеги маанилердин тизмеси:”);
   for(i=0;i<3;i++) printf(“%-4d”,*(pa+i));
}
```

Жыйынтык:

```
адрестердеги маанилердин тизмеси: 30     7       2003
```

Бул мисалда pa=a; оператору pa көрсөткүчүнө a статикалык массивинин башталышынын адресин менчиктейт, ал эми a=pa; жазылышы туура эмес жана каталыка алып келет, себеби бул учурда a статикалык массивинин адресин өзгөртүү аракети жүргүзүлүп калат.

Массивтин ысымы дайыма адреске болгон шилтеме катары каралат. Эгерде мындай шилтеменин көчүрмөсүн көрсөткүчкө

ыйгарсак, көрсөткүчтүн маанисин чоңойтуу же кичирейтүү менен массивтин кандайдыр бир элементине шилтеме жүргүзүүгө болот

Мисалы:

```
int  b[5], *pb, i;
pb=b;       /*pb  көрсөткүчү  b[0] дү көрсөтүп турат*/
pb++;       /*pb  көрсөткүчү  b[1] ди көрсөтүп турат*/
pb++;       /*pb  көрсөткүчү  b[2] ни көрсөтүп турат*/
for (i=0;i<5; i++) *pb++=0   /*b  массивинин элементинин
баарысы 0 болот*/
```

Жогорку программанын фрагментиндеги акыркы сабындагы *pb++=0 операторунун аткарылышын карайлы. Адегенде * - адреске кайрылуу амалы pb адреси боюнча маанини берет. Бул маани 0 гө барабар болот. Циклдин биринчи айлампасында b[0] элементи нөлгө айланат, андан ары pb адреси “++” амалы аркылуу бирге чоңоёт.

Массивтер жана көрсөткүчтөр менен иштөөдө олуттуу каталар болуп маанилештирилбеген көрсөткүчтү колдонуу жана массивтин чегинен чыгып кетүү эсептелинет int *p көрсөткүчтү баяндаса аны маанилештирүү деп программада

p=(int *) malloc(sizeof(int)); сабынын жазылышын түшүнөбүз.

Эки өлчөмдүү массивтер жана көрсөткүчтөр.

Эки өлчөмдүү массив бул ар бир элементи сап болгон бир өлчөмдүү массивти түшүнөбүз.

Мисалы: a[3] [4] болсо:

a – массивтин нөлүнчү (0) сабынын көрсөткүчү.

(a+1) - массивтин 1- сабынын көрсөткүчү.

(a+2) - массивтин 2- сабынын көрсөткүчү ж. у. с.

a[2] [3] жазылышы *(a[2]+3) же *(*(a+2)+3) жазылыштары менен эквиваленттүү.

Жалпылаганда a[i] [j] матрицасы компилятор тарабынан *(*(a+i)+j) – эквиваленттүү жазылышына которулат.

4.3 Саптар

Жалпы маалымат.

Си тилинде сап катары символдордун удаалаштыгын алышат жана ар бир саптын аягына компилятор тарабынан нөлдүк символ (\0) коюлат, бул символду 0 цифрасы менен алмаштырбоо керек. Мисалы “Си тили” деген сапты карасак 8 элементтен турган символдук массив катары берилет, эгерде массивтин ысымы a болсо, анда a[0]=’С’, a[1]=’и’, a[2]=’ ‘, a[3]=’т’, a[4]=’и’, a[5]=’л’, a[6]=’и’, a[7]=’\0’

Эгерде сап туракту катары берилсе б.а. тырмакчанын ичинде жасылса, мисалы “Информатика” анда саптын аягына компилятор нөлдүк символду (‘\0’) автоматтык түрдө кошот. Ал эми калган учурда, сапты айкын түрдө берүүдө программист саптын аягына нөлдүк символду (‘\0’) коюуга тийиш.

Мисалы:

char sap[2]={‘С’, ‘и’, ‘\0’}.

Сапты символдордун массиви же символдордун удаалаштыгы катары карашат. Сапты символдордун массиви катары кароодо, массивтерди маанилештирүүнү колдонсо болот.

Мисалы:

char str[10]={‘T’, ‘u’, ‘r’, ‘b’, ‘o’, ‘ ‘, ‘C’,‘\0’}

Ал эми символдордун удаалаштыгы катары берилгенде,

Мисалы:

char str[10]={“Turbo C”}

саптын аягына 8-символ катары нөлдүк символ (‘\0’) автоматтык түрдө коюлат.

Саптарды киргизүүдө Си тилинде эки функция каралган scanf жана gets. Эки функциянын айырмасы: scanf – функциясы ачык (пробел) символу кезиккенге чейин же Enter клавишасын басканга чейин сапты окуп киргизет, ал эми gets – функциясы сапты <Enter> клавишасын басканга чейин окуп киргизет, ал эми ачык (пробел) символун саптын мааниси катары эсептейт.

Сапты чыгарууну карайлы. Мейли char a[10] берилсе printf(“%s”, a) функциясынын иштөөсү төмөндөгүдөй, бул функция аткарылганда, а көрсөткөн биринчи элементтин адреси берилет. Эгерде биринчи элемент нөлдүк символ (‘\0’) болсо, чыгаруу аяктайт, ал эми нөлдүк символ болбосо, ал элемент экранга чыгарылат да учурдагы адреске бир кошулуп кийинки элементин маниси текшерилет, ушундай иш аракеттер сап экранга толук чыгарылып бүтмөйүнчө улана берет. Саптарды киргизүү мисалдары:

```
# include <stdio.h>
#include <conio.h >
    main( )
    {
    char  sap[20];
    clrscr();
    printf(“\n Ысымыңыз ким ?”);
    scanf(“%s”, sap);
    printf(“Саламатсыңбы,  %s \n”, sap);
    }
```

Жыйынтык:

Ысымыңыз ким? Нуртегин

Саламатсыңбы, Нуртегин

Жогорку программада sap символдордун массиви ошол эле учурда sap массивдин адреси болгондуктан scanf – киргизүү функциясында адрести аныктоочу & амалы, sap ысымынын алдына коюлбайт.

Программанын иштеши: Экранга, ысымыңыз ким? деген суроо чыгат. Бул суроого “Акмат“ деп жооп берсеңиз, анда программа, экранга “Саламатсыңбы Акмат“ деген сүйлөмдү чыгарат. Эгерде суроого “Акмат Асанов” деп жооп берилсе, анда

экранга “Саламатсыңбы Акмат” деген эле кабар чыгарылат. Себеби Акматтан кийинки коюлган ачык (‘ ‘) белги scanf – функциясы үчүн киргизүүчү саптын аягын билдирет. Мындай кемчиликти жоюуш үчүн: Эки сапты киргизүү менен же gets сапты киргизүү функциясын колдонуу менен жойсо болот. Бул учурлар үчүн мисал карайлы:

Мисал 1.

```
 # include <stdio.h>
#include <conio.h>
main ( )
{char  sap1 [15], sap2 [20];
clrscr();
printf(“\n Ысымыңызды жана фамилияңызды киргиз:”);
scanf(“%s%s”, sap1, sap2);
printf(“Саламатсыңбы,  %s %s\n”, sap1,sap2);
}
```

Жыйынтык:

Ысымыңызды жана фамилияңызды киргиз: Нуртегин Максатбеков

Саламатсыңбы, Нуртегин Максатбеков

Бул программада эки саптык өзгөрмө sap1, sap2 колдонулду.

Мисал 2.

```
# include <stdio.h>
#include <conio.h>
main( )
{char  sap[20];
clrscr();
printf(“\n Ысымыңыз ким:”);
gets(sap);
printf(“Саламатсыңбы,  %s\n”, sap)
```

}

Жыйынтык:

Ысымыңыз ким: Балтатыр

Саламатсыңбы, Балтатыр

Экинчи программанын иштешинде gets – функциясы, колдонуучу киргизүү баскычын (“Enter”) басканга чейинки терген символдорунун баардыгын окуйт, бирок киргизилген саптын аягына ‘\0’ нөлдүк символун жайгаштырат. Эгерде сапты киргизүү мезгилинде саптын узундугу, берилген сандан ашып кетсе, ашыкча символдор сапка берилген эстин областынын башка бөлүгүнө жазылат. Сап – символдордун массиви болгондуктан, массивдин ысымы массивтин биринчи элементинин көрсөткүчү болот. Ошондуктан программада саптарды же символдордун массиви же символго болгон көрсөткүч катары баяндаса болот. Мисалы char m[20], *m1 бул жерде “char m[20]” m массиви 20 символдон турат жана бул сап үчүн учурдагы эстен орун бөлүнөт. Ал эми “char *m1” болсо сапты символго болгон көрсөткүч катары баяндайт. Бул учурда эстен айкын түрдө орун бөлүнүп, m1 – символго болгон көрсөткүч болгондуктан m1 же символдордун массиви же баштапкы адреси m1 болгон сап. Бир эле программада сапты массив катары да көрсөткүч катары да караса болот. Мындай колдонуу сапты функциянын аргументи катары берген учурда өзгөчө маанилүү болуп эсептелет.

Мисалы:

```
#include <stdio.h>
#include <conio.h >
      main( )
     {char  sapmass[10], *p;
    clrscr();
      sapmass[0]=’C’;
```

```
  sapmass[1]=’а’;
  sapmass[2]=’л’;
  sapmass[3]=’а’;
  sapmass[4]=’м’;
  sapmass[5]=’\0’;
  p= sapmass;
  puts(p);
  printf(“\n p=%s\n”, p);
  puts(sapmass);
  while(*p!=NULL) putchar(*p++);
}
```

Жыйынтык:

```
Салам
p= Салам
Салам
Салам
```

Бул программада 3 – сапта символдордун удаалаштыгын массив жана көрсөткүч түрүндө баяндап жатат, ал эми 4 – саптан 9 – сапка чейин символдордун массиви айкын түрдө маанилештирилген 10- сапта p символдук массивти менчиктеп алат. 11 – саптагы puts(p) функциясы көрсөткүчтүн мааниси боюнча символдук массивти экранга чыгарат б.а. “Салам” деген сөз чыгат. 12 – сапта printf функциясы сапты толугу менен чыгарат. 13 – саптагы puts функциясы sapmass сабын экранга чыгарат.

14 – саптагы while цикл оператору p көрсөткүчү кармаган адресте жайгашкан символ NULL туректуусунан айырмалуу болсо putchar функциясы *p++ адрести бирге чоңойтуу менен улам кийинки адресте жаткан символдорду чыгара берет. Чыгаруу процессинде ал адрестеги символдорду жоюп салат.

Саптардын өзгөчөлүктөрү.

Саптарды туруктуу жана өзгөрмөнүн мааниси катары кароого болот.

Саптарды туруктуу катары караган учурда аларды берүүнүн эки жолу бар.

1. Программанын текстинде сап туруктуусу кош тырмакчага алынып жазылат. Бул учурда компилятор сап туруктуусунун аягына “\0” нөлдүк символун автоматтык түрдө кошот.

2. define директивасынын жардамы менен *мисалы:*

```
#define sap “Си тили”
…
puts(sap);
```

Бул программанын фрагментинде sap туруктуу катары каралып анын мааниси “Си тили” экранга puts –функциясы менен чыгарылат.

Саптарды өзгөрмөнүн мааниси катары караган учурда, сапты аныктоонун да эки жолу болот.

1. Символдук массивти колдонуу.

2. Символго болгон көрсөткүчтү колдонуу.

Символдук массивти колдонуу менен сапты берүүгө *мисал:*

```
#include <stdio.h>
#include <conio.h >
#define n 10
    main( )
    {char sap[n];
    clrscr();
    sap[0]=’С’;
    sap[1]=’и’;
    sap[2]=’ ’;
    sap[3]=’т’;
    sap[4]=’и’;
```

```
sap[5]=’л’;
sap[6]=’и’;
sap[7]=’\0’;
puts(sap);
}
```

Жыйынтык:

Си тили

Символго болгон көрсөткүчтү колдонуу менен сапты берүүгө мисал:

```
#include <stdio.h>
#include <conio.h >
main( )
{char *sp;
clrscr();
sp=”Си тили”;
puts(sp);
printf(“sp=%s\n”, sp);
}
```

Жыйынтык:

Си тили

Саптар менен иштөөдө байкалбаган каталар кетип калышы мүмкүн. Мисалы:

```
#include <stdio.h>
main( )
{char *sap;
char sp[10];
printf(“\n Сапты киргиз:”);
scanf(“%s”, sap);
```

```
sp=”Киргизилген сап”;
printf(“%s%s\n”, sp,sap);
}
```

Жыйынтык:

```
Сапты киргиз: тттт
Null pointer. Assigment
```

Жогорку программадагы 6 саптагы scanf (“%s”,sap); функциясынан ката кеткен. Кетирилген катанын маниси төмөндөгүдөй sap үчүн эстен орун бөлүнбөйт, киргизилген сап кандайдыр бир башка адреске жазылат. Мындай ката программанын иштешин токтотпойт.

Экиничи ката sp=”Киргизилген сап” менчиктөө операторун колдонгондон кеткен. Компилятор “Киргизилген сап” деген саптык туракту улуктун адресин sp - өзгөрмөсүнүн маанисине алмаштырууга аракет катары эсептейт. Мындай иш аракет туура эмес, себеби массивтин ысымы туракту улук (константа) катары каралат б.а. туракту ууга өзгөрмөнү (9=a) менчиктөө менен тең күчтө болуп калат, мындай менчиктөө туура эмес.

Жогорку ката жазылган программанын туура жазылышы.

```
# include <stdio.h>
#include <conio.h >
    main ( )
    { char sap [10];
    char *sp;
    clrscr();
    printf (“\n Сапты киргиз:”);
    scanf (“%s”, sap);
    sp=”Киргизилген сап:”;
    printf (“%s%s\n”, sp, sap);
```

}

Жыйынтык:

Сапты киргиз: Манас

Киргизилген сап: Манас

Саптарды иштетүүчү программаларга токтололу.

1-Мисал: strlen камтылган программаны колдонбой киргизилген саптын узундугун аныктоочу программа.

```
# include <stdio.h>
#include <conio.h >
    main ( )
    { char  *sap;
    int i ;
    clrscr();
    printf (“\nСапты киргиз \n”);
    gets (sap);
    for (i=0; sap[i] !=0; i++);
    printf (“Киргизилген саптын узундугу=%d”, i);
    }
```

Жыйынтык:

Сапты киргиз: Манас эпосу

Киргизилген саптын узундугу=11

2-Мисал: символдук массивди колдонуу менен бир сапты экинчи сапка менчиктөө.

```
# include <stdio.h>
#include <conio.h >
    main ( )
    { char  sap1[10], sap2[10];
    int i ;
    clrscr();
    printf(“\n Сапты киргиз sap1=”);
    gets (sap1);
```

```
i=0;
do
{sap2[i]=sap1[i];}
while (sap1[i++]!=0);
printf(“sap2=”);
puts (sap2);
}
```

Жыйынтык:

Сапты киргиз sap1=Тагай бий

sap2= Тагай бий

3-Мисал: сапты киргизүүнүн жана чыгаруунун ар түрдүү жолдору.

```
# include <stdio.h>
#include <conio.h >
main ( )
{ char  smas [10], *sap;
clrscr();
printf (“\n Сапты киргиз smas=”);
gets (smas);
sap=smas; /* sap көрсөткүчү smas сабынан башталыш адресин менчиктейт */
printf (“smas=”);

puts (smas);
printf (“sap=”);
puts (sap);
printf (“sap=”);
while (*sap!=NULL) putchar (*sap++);
}
```

Жыйынтык:

Сапты киргиз smas=Барсбек каган

smas=Барсбек каган

sap= Барсбек каган

sap= Барсбек каган

3-мисалда сапты киргизүүнүн эки жолу жана чыгаруунун үч жолу берилген, киргизүү gets(smas) функциясы жана sap=smas менчиктөө оператору менен берилсе. Чыгаруу puts(smas), puts(sap) жана адрес аркылуу чыгаруу putchar (*sap) функциялары менен берилген.

Саптар менен иштөөчү стандарттык функциялар.

Саптар менен иштөөчү стандарттык функциялар Турбо-Си системасында string.h файлында баяндалган функцияларды баяндоонун жалпы түрү.

<жыйынтыктын тиби> <функциянын ысымы> (<формалдуу параметрлер);

Саптарды бириктирүүчү функциянын эки түрү бар.

1. char *strcat(sap1, sap2); бул функция sap1 сабынын аягына sap2 сабынын көчүрмөсүн бириктирет.

Мисалы sap1=”эсен” sap2=”аман” анда strcat(sap1, sap2) функциясынын жыйынтыгы «эсенаман» деген маанини берет.

2. char *strncat(sap1, sap2, k); sap2 сабынан k сандагы символдун күчөрмөсүн бириктирип келип чыккан саптын аягына “\0” нөлдүк символду коёт.

Мисалы sap1= “Асан”, sap2=”баймат”

strncat (sap1, sap2, 3) функциясынын жыйынтыгы “Асанбай” болот

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

```
{ char *sap1, *sap2, *ch1, *ch2;
clrscr();
sap1=”Эсен”;
sap2=”аман”;
strcat (sap1, sap2);
puts (sap1);
ch1=”Асан”;
ch2=”баймат”;
strncat (ch1, ch2, 3);
puts (ch1);
}
```

Жыйынтык:

Эсенаман

Асанбай

Саптарды салыштыруу функциясы

int strcmp (sap1, sap2)

sap1 менен sap2 саптары салыштырылат. Салыштыруу, саптарды түзгөн символдордун коддору менен жүргүзүлөт. Салыштырууда салыштырылып жаткан символдордун коддору кайсынысы чоң болсо ошол сапты чоң деп эсептейт. Эгерде ал символдордун коддору барабар болсо кийинки символдор салыштырылат. Мындай иш аракеттер символдордун коддорунун кайсынысы чоң же кичине болгонго чейин жүргүзүлөт.

Си тилинде саптарды салыштыруу үчүн салыштыруу функциясынын үч түрү каралган. Бул функциялардын жыйынтыктарынын бардыгы бүтүн типте болушат.

1. int strcpm (sap1, sap2); sap1 менен sap2 салыштырылат, эгерде sap1<sap2 болсо функциянын жыйынтыгы терс болот, sap1==sap2 болсо strcpm функциясынын маниси 0 болот, ал эми sap1>sap2 болсо функциянын маниси оң болот.

2. int stricmp (sap1, sap2) бул функция саптагы баш (чоң) тамгалар менен кичине тамгаларды айырмалабай sap1 жана sap2 салыштырылат.

3. int strncpm (sap1, sap2, k) бул функция саптагы баш (чоң) тамгалар менен кичине тамгаларды айырмалабай, бирок k сандан ашпаган гана символдорду салыштырат.

1-Мисал:

```
# include <stdio.h>
# include <string.h>
# include <conio.h >
      main ( )
      {
      clrscr();
      printf (“%d \n”, strcmp (“A”, “A”)); /* 0 саны чыгарылат*/
      printf (“%d \n”, strcmp (“A”, “B”)); /* -1 саны чыгарылат*/
      printf (“%d \n”, strcmp (“C”, “B”)); /* 1 саны чыгарылат*/
      printf (“%d \n”, strcmp (“D”, “A”)); /* 3 саны чыгарылат*/
      }
```

Жыйынтык:

```
      0
      -1
      1
      3
```

2-Мисал:

```
# include <stdio.h>
# include <string.h>
# include <conio.h >
  clrscr();
  main ( )
  { char sap1[20], sap2[20];
```

```
   printf (“\n sap1 жана sap2 саптарын 20дан ашпаган узундукта
киргиз \n”);
   printf (“\n sap1 киргиз:”);
   scanf (“%s”, sap1);
   printf (“\n sap2 киргиз”);
   scanf (“%s”, sap2);
   if (strcmp (sap1, sap2)<0) printf (“\n sap1<sap2”);
   if (strcmp (sap1, sap2)>0) printf (“\n sap1>sap2”);
   if (strcmp (sap1, sap2)==0)printf (“\n sap1=sap2”);
  }
```

Жыйынтык:

```
sap1 жана sap2 саптарын 20дан ашпаган узундукта киргиз
sap1 киргиз: AAA
sap2 киргиз: DDD
sap1<sap2
```

Саптарды көчүрүү.

Си тилинде саптарды көчүрүү үчүн strcpy-функциясынын эки түрү каралган.

1-түрүнүн жалпы берилиши char * strcpy(sap1, sap2); sap2 сабынын баарысын sap1 сабына көчүрөт.

2-түрүнүн жалпы берилиши char * strcpy(sap1, sap2, k); sap2 сабынан k сандагы символду sap1 сабына көчүрөт.

1-түрдөгү көчүрүү функциясына мисал.

```
# include <stdio.h>
# include <string.h>
# include <conio.h>
main ( )
  {char s1[10], s2[10];
   clrscr ( );
```

```
  printf (“s2 сабын киргиз:”);
  scanf (“%s”, s2);
  strcpy(s1, s2);
  printf (“s1=%s\n”, s1);
 }
```

Жыйынтык:

s2 сабын киргиз: Атилла

s1= Атилла

Көчүрүү функциясынын 2-түрүнүн саптын ортосундагы символдорду көчүрүүгө мисал карайлы.

```
# include <stdio.h>
# include <string.h>
# include <conio.h>
    main ( )
    {char s1[6], s2[10];
    clrscr ( );
    printf (“s2 сабын киргиз:”);
    scanf (“%s”, s2);
    strncpy(s1, &s2[2], 6);
    printf (“%s\n”, s1);
    }
```

Жыйынтык:

s2 сабын киргиз: informatika

format

Саптын узундугу.

Саптын узундугун strlen-кызматчы сөзү менен берилген функция аркылуу табылат. Бул функциянын жалпы берилиши: int strlen (sap)

sap канча символдордон турса ошол символдордун санын чыгарып берет.

Мисалы: strlen(“Си программалоо тили”) жыйынтыгы 20санын берет, себеби көрсөтүлгөн сап 20 символдон турат. Сапты аяктоочу нөлдук символ саптын узундугунун санына кирбейт.

Мисалы: Клавиатура аркылуу киргизилген саптын узундугун берүүчү программа.

```
# include <stdio.h>
# include <string.h>
# include <conio.h>
    main ( )
    {char s[80];
    clrscr ( );
    printf (“Сапты киргиз:”);
    gets (s);
    printf (“Берилген сап: %s \n %d символдон турат \n”, s, strlen(s));
    }
```

Жыйынтык:

Сапты киргиз: Информатика

Берилген сап: Информатика

11 символдон турат

Саптан символдорду издөө.

Символдорду издөө Си тилинде strchr-кызматчы сөзү менен берилген функция аркылуу жүргүзүлөт. Бул функциянын эки түрү бар.

1-түрүнүн жалпы берилиши char *strchr (<берилген сап>, <изделүүчү символ>). Бул функция берилген сапта <изделүүчү символ> биринчи кезиккен орундан баштап саптын аягына чейин чыгарып берет. Эгерде <изделүүчү символ> кезикпесе анда функция NULL деген маанини чыгарат.

Мисалы: char *strchr (sap, ‘C’). мисалдагы функция ‘C’ тамгасы сапта биринчи кезиккен орундан баштап аягына чейин символдордун удаалаштыгын чыгарып берет.

2-түрүнүн жалпы берилиши char *strrchr (<берилген сап>, <изделүүчү символ>). Бул функция берилген сапта <изделүүчү символ> акыркы кезиккен орундан баштап саптын аягына чейин чыгарып берет.

Мисалы char *strrchr (sap, ‘C’) бул функция берилген сапта ‘C’ тамгасы акыркы кезиккенинен баштап саптын аягына чейинки бөлүгүн чыгарат.

```
# include <stdio.h>
# include <string.h>
 # include <conio.h >
    main ( )
     {char sap[10], *p, *q;
       clrscr();
       printf (“Сапты киргиз: ”);
       gets (sap);
       p=strchr(sap, ‘m’);
       q= strrchr(sap, ‘m’);
       printf (“Бул сап:  %s \n”, p);
       printf (“%s \n”, q);
     }
```

Жыйынтык:

```
Сапты киргиз: zxcvmasmkkk
Бул сап: masmkkk
mkkk
```

Бир саптагы символдорду экинчи бир саптан издөөчү функция strpbrk-кызматчы сөзү менен берилет. Жалпы берилиши char *strpbrk(sap1, sap2) бул функция sap2- биринчи символу sap1

сабында кезиккен ордунан баштап sap1 сабынын аягына чейин чыгарып берет. Эгерде sap1 ден sap2нин бир дагы символу кезикпесе функция NULL деген маанини чыгарып берет.

Мисалы: strpbrk

```
# include <stdio.h>
# include <string.h>
# include <conio.h>
    main ( )
    {char sap1[10], sap2[10], *p;
      clrscr();
      printf (“sap1 ди киргизгиле: ”);
      gets (sap1);
      printf (“sap2 ни киргизгиле: ”);
      gets (sap2);
      p=strpbrk(sap1, sap2);
      printf (“%s \n”, p);
    }
```

Жыйынтыгы:

```
sap1 ди киргизгиле: zxcvb
sap2 ни киргизгиле: rtyxkum
xcvb
```

Түшүндүрмө: sap1дин мааниси катары “zxcvb” берсек ал эми sap2 мааниси катары “rtyxkum” киргизсек жообу: “xcvb”-сабы болот. Издеп табылган саптардын узундуктары. int strspn (sap1, sap2)-функциясы менен табылат. Бул функцияга мисал карайлы.

```
 # include <stdio.h>
 # include <string.h>
  # include <conio.h>
   main ( )
   {char *sap1=”d8656bn8”;
```

```
char *sap2=”2td86bnc”;
int t;
clrscr();
t=strspn (sap1, sap2);
printf (“sap2де кезиккен sap1деги символдордун удаалаштыгынын  узундугу =%d\n”,t)
}
```

Жыйынтыгы:

sap2де кезиккен sap1деги символдордун удаалаштыгынын узундугу =3 Түшүндүрмө: sap2деги ”2td86bnc” саптан d86 удаалаштыгы sap1деги “d8656bn8” маанинин биринчи 3 символу болгондуктан узундугу 3кө барабар.

4.4 Түзүлүштөр

Түзүлүш-деп компоненттери (элементтери) ар түрдүү тип болгон курама объектини айтабыз.

Түзүлүш (структура) - бир канча объектилердин биригишин берет. Массив сыяктуу эле маалыматтардын жыйындысын берет. Массив менен түзүлүштүн айырмасы: массивтин элементтери бир типте жана ирээттелген болот, ал эми түзүлүштүн элементтери ар түрдүү типте болуп жана ирээтелбеши мүмкүн. Паскаль тилинде бул түшүнүк аралаш типтер же жазылыштар деп берилет жана төмөндөгүдөй жазылат.

```
Type <жазылыш ысымы>=RECORD
<1-талаа> : <тип-1>;
<2-талаа> : <тип-2>;
. . . . . .
<k-талаа>:<тип-n>;
END;
```

Си тилинде түзүлүштөр 3жол менен берилет. Түзүлүштөрдү баяндоодо struct деген кызматчы сөз колдонулат.

1-жолу:

struct {

<баяндоолордун тизмеси>

} <түзүлүштүн аталыштары>;

<баяндоолордун тизмеси>-түзүлүштүн элементтерин (компоненттерин бөлүктөрүн) баяндайт.

<түзүлүштүн аталыштары>-өзгөрмөлөрдүн, массивтердин, көрсөткүчтөрдүн жана функциялардын ысымдары.

Мисалы:

```
struct {
        double x, y;
       } a, b, c[9];
```

Бул түзүлүштө a, b өзгөрмөлөрүнүн ар бири эки компоненттен турган түзүлүш катары аныкталып жатат, ал эми c-массив катары элементтери эки компоненттен турат.

a=xa+ya*i, b=xb+yb*i, c[0]=x0+y0*i, c[1]=x1+y1*i ж.у.с.

```
struct {
          int ai, күн, gil;
        } data1, data2;
```

2-жолу:

Түзүлүштүн тибин колдонуучу да аныкташы мүмкүн. Бул учурда typedef-кызматчы сөзү колдонулат бул типтин ысымын өзгөрмөлөрдү аныктоодо колдонсо болот. Жалпы жазылышы:

```
typedef struct {
          <баяндоолордун тизмеси>;
          }<типтин ысымы>;
```

Мисалы:

```
typedef struct {
                  char name [30];
                  int d;
                 } cmp;
```

cmp e1, e2; /* e1, e2 өзгөрмөлөрү cmp тибиндеги түзүлүштөр*/

3-жолу:

Кандайдыр бир үлгүнү же белгини (метка) колдонууга негизделген. Жалпы жазылышы:

struct <белги> {

<баяндоолордун тизмеси>

};

<белги>-бул кандайдыр бир ысым

Мисалы:

```
struct student {
                char name [20];
                int kurs;
                char gruppa [10];
                } s1,s2;
```

Түзүлүштүн компоненттерине (бөлүктөрүнө) жетиш үчүн же иштетиш үчүн <түзүлүштүн ысымы> <компоненттин ысымы> struct student s1, s2; бул мисалда student <белги> катары, ал эми s1, s2 тиби student белгиси менен берилген түзүлүштөр s1.name [20], s1.kurs, s1.gruppa[10]

Мисалы: Машинанын маркасы, чыккан жылы жана баасы берилген. Берилген акчадан ашпаган, жана берилген жылдан эски болбогон жеңил машиналардын тизмесин чыгаргыла.

```
# include <stdio.h>
# include <conio.h>
 main ( )
{
struct {
char marka [10];
int gil;
int baasi;
```

```
} avto [20];
int n, i, y,m;
clrscr();
printf (“\n n<20 болгондой машиналардын санын киргиз:”);
scanf (“%d”, &n);
printf (“%d машинадан турган тизмени киргиз: \n”, n);
for (i=0; i<n; i++)
{fflush (stdin);
printf (“Машинанын маркасын киргиз: ”);
gets(avto [i]. marka);
printf (“Машинанын жылын киргиз: ”);
scanf (“%d”, &avto [i]. gil);
printf (“Машинанын баасын киргиз: ”);
scanf (“%d”, &avto [i]. baasi);
}
printf (“Керектүү жылды жана бааны үтүр менен киргиз:”);
scanf (“%d,%d”, &y, &m);
printf (“Сизге ылайык келүүчү машина:”);
for (i=0; i<n; i++)if (avto [i]. gil>=y&&avto [i]. baasi <=m)
printf (“\n%12S%nd%7.3f”, avto [i]. marka, avto [i]. gil, avto [i].
baasi);
}
```

Жыйынтык:

```
n<20 болгондой машиналардын санын киргиз: 3
3 машинадан турган тизмени киргиз:
Машинанын маркасын киргиз: mmmm
Машинанын жылын киргиз: 2001
Машинанын баасын киргиз: 56000
Машинанын маркасын киргиз: zzzz
Машинанын жылын киргиз: 2002
Машинанын баасын киргиз: 65000
```

Машинанын маркасын киргиз: ffff
Машинанын жылын киргиз: 2003
Машинанын баасын киргиз: 85000
Керектүү жылды жана бааны үтүр менен киргиз:2002,67000
Сизге ылайык келүүчү машина:
zzzz 2002 65000

Бул программа жеңил машина сатып алуучунун акчасынын баасынан чоң эмес, берилген жылдан эски болбогон жеңил автомашиналардын маркаларынын чыккан жылдарын жана бааларынын тизмелерин чыгарып берет. Түзүлүштүн элементтерине жетүү көрсөткүчтүн жардамы менен да ишке ашырылат. Бул учурда түзүлүштүн элементине кайрылууда “→” жебе белгиси колдонулат.

Мисалы: жогорудагы дата түзүлүшүн колдонуучунун тибин аныктоочу typedef-кызматчы сөзүн колдонуп аныктайлы.

```
   typedef struct
{ int күn;
  char ai[10];
   int gil ;
 }data;
data d1, d2;
data *kor;
kor=& d1;
kor→күн=19;
kor→ai=”сентябрь”;
kor→gil=1959;
```

Бул программанын бөлүгүндө түзүлүшкө болгон көрсөткүч жана түзүлүштүн элементерине көрсөткүч аркылуу жетүү баяндалган. Көрсөткүчтү колдонуп түзүлүштүн элементерине жетүүнүн башкача жолу.

```
typedef struct
{ int күн;
char ai[10];
int gil ;
}data;
data d1, d2;
data *kor;
kor=& d1;
(*kor).күн=19;
(*kor).ai=”сентябрь”;
(*kor).gil=1959;
```

Түзүлүштүн элементтерин иштетүүдө массивти колдонууга мисал карайлы. Мисалы: студенттин тизмедеги номери боюнча ал жөнүндө информация чыгарып берүүчү программа.

```
# include <stdio.h>
 # include <conio.h>
main ( )
{int i, k;
typeaef  struct
{int num;
char fam [20];
char isim [15];
}student;
student mas[3];
clrscr();
printf (“\n Студенттердин номерин, фамилиясын жана ысымын киргиз”);
for (i=0; i<3; i++)
{printf (“\n %d-студенттин номерин  киргиз: ”);
scanf (“%d”, &mas[i].num);
```

```
printf (“\n %d-студенттин фамилиясын  киргиз: ”);
scanf (“%s”, &mas[i].fam)
printf (“\n %d-студенттин  ысымын  киргиз : “);
scanf (“%s”, &mas[i]. isim)}
printf (“Студенттин тизмедеги номерин киргиз\n”);
scanf (“%d”, &n);
for (i=0; i<3; i++);
if (mas[i].num==k);
printf (“Студент: %s %s\n”, mas[i].fam, mas[i].isim);
}
```

Жыйынтык:

```
Студенттердин номерин, фамилиясын жана ысымын киргиз
 1-студенттин номерин  киргиз: 555
 1-студенттин фамилиясын  киргиз: Максатбеков
 1-студенттин  ысымын  киргиз : Нуртегин
 2-студенттин номерин  киргиз: 777
 2-студенттин фамилиясын  киргиз: Эркинбаева
 2-студенттин  ысымын  киргиз : Айзат
 3-студенттин номерин  киргиз: 999
 3-студенттин фамилиясын  киргиз: Ибраева
 3-студенттин  ысымын  киргиз : Гулмира
 Студенттин тизмедеги номерин киргиз: 555
 Студент: Максатбеков Нуртегин
```

Түзүлүштүн элементтери анык бир маанини кармап, ар кайсы учурларда өзгөрбөгөн формага ээ болсо анда мындай түзүлүштү-туруктуу деп айтабыз.

4.5 Бирикме

Си тилинде учурдагы эсте бир канча ар түрдүү типтеги өзгөрмөлөрдү жайгаштыруу үчүн өзүнчү тип каралган. Бул типти бирикме деп айтабыз. Бирикме union кызматчы сөзү менен

берилет жана түзүлүшкө окшош эле жарыяланат. Түзүлүштөн бирикменин айырмасы болуп бирикменин баардык элементтери эстен бир эле оорунду ээлейт. Ал эми түзүлүштүн ар бир элементтери үчүн учурдагы эстен орун бөлүнүп берилет. Бирикменин элементтеринин өлчөмү чоңуна карата учурдагы эстен орун берилет, калган элементтери ушул орунга кабатталып жайланышкан сыяктанат. Бирикменин элементтерине кайрылуу түзүлүшкө окшош эле чекит же жебе менен ишке ашырылат. Бирикме төмөнкүдөй баяндалат:

```
union {
        <1-элементтин баяндалышы>
        <2-элементтин баяндалышы>
              . . . . . .
       <n-элементтин баяндалышы>
      }<бирикменин аталышы>;
```

Бирикмени биринчиден кандайдыр бир убакыттын ичинде көп объектилердин ичинен бир гана объект активдүү болсо, эсти үнөмдөөдө экинчиден негизги бир типтеги объектиге башка бир типти ыйгаруу үчүн колдонушат.

Бирикмеге *мисал* карайлы.

```
union {
        float radius;    /*Айлана*/
        float a[2];      /*Тик бурчтук*/
        int b[3];       /*үч бурчтук*/
       orun p;        /*p-чекит, orun-колдонуучунун тиби*/
      } geomfig ;
```

Жогорудагы мисалда кандайдыр бир элемент маниге ээ болсо ал активдүү элемент деп аталат. Калгандары каралбайт.

Мисалы radius элементине маани берилсе ошол эле учурда a[2] же в[3] ж.у.с. элементтерине кайрылууга болбой калат.

4.6 Өзгөрмөлүү түзүлүштөр

Түзүлүштөрдү колдонуп программа түзүүдө Си тилинде өзгөрмөлүү түзүлүш түшүнүгү да колдонулат. Өзгөрмөлүү түзүлүштөр деп –түзүлүштүн элементтери ар кайсы учурда, ар түрдүү маанилерди кармаган түзүлүштү айтабыз. Өзгөрмөлүү түзүлүш жалпы компоненттеринин жыйындыларын кармайт. Жалпы учурда өзгөрмөлүү түзүлүш үч бөлүктөн турат. Жалпы компоненттердин жыйындысы, активдүү компоненттин белгиси жана өзгөрмөлүү компоненттеринин бөлүктөрү. Өзгөрмөлүү түзүлүштөрдү ишке ашырууда түзүлүштөрдүн жана бирикмелердин комбинациясы колдонулат. Өзгөрмөлүү түзүлүштүн жалпы берилиши.

```
struct { <жалпы компоненттер>;
        <активдүү компоненттин белгиси>;
  union { <1-компоненттин баяндалышы>;
          <2-компоненттин баяндалышы>;
                . . . . . .
          <n-компонентин баяндалышы>;
          } <бирикменин аталышы>;
          } <өзгөрмөлүү түзүлүштүн ысымы>;
```

Активдүү компоненттин белгисинин учурдагы мааниси бирикменин өзгөрмөлүү компоненттеринин кайсынысы ошол учурда активдүү болоорун көрсөтүп турат. Мисал катары 4.5. Бирикме бөлүмүндөгү геометриялык фигуралар жөнүндө маселени карайлы. Бул маселеде фигуралар үч параметр менен берилсин деп алалы: аянт, периметр жана өлчөм. Үч фигура үчүн жалпы мүнөздүү параметрлер болуп аянт жана периметр эсептелинет. Ал эми өлчөм болсо тегерек үчүн радиусу, тик

бурчтук үчүн анын эки жагы, үч бурчтук үчүн үч жагы эсептелет. Программанын фрагментин карайлы:

```
typedef struct {
  float aiant, perimetr;
  int t;
  union {float r;
  float a[2];
  float b[3];
  } geomfig;
  } fig;
  fig f;
```

бул жерде f өзгөрмөсү fig тибиндеги өзгөрмө болуп эсептелет.

Жогорудагы мисалда fig тибиндеги ар бир өзгөрмө үч туруктуу aiant, perimeter жана t деген компоненттерден турат. t компоненти geomfig бирикмесинин кайсыл компонентти активдүү болоорун көрсөтүп турат.

Мисалы: f.t=1 болсо анда fig тибиндеги f өзгөрмөсүнүн t компоненти 1ге барабар болду дегенди билдирет жана геометриялык фигуралардын ичинен айлананы иштетүү активдештирилди деп эсептелет жана f.geomfig.r=5.0 же айлананын радиусуна башка деле маанилер берилип айлана иштетиле баштайт. Ушул эле сыяктуу f.t=2 болсо тик бурчтук иштетилет жана ага тиешелүү эки жагы мисалы: f.geomfig.a[0]=2.0 жана

f.geomfig .a[1]=3.0 дон же башка деле маанилер берилип тик бурчтук иштетиле берет. Ошондой эле f.t=3 болсо анда үч бурчтук иштетилет.

4.7 Маанилештирүү

Программада өзгөрмөлөрдүн маанилерин берүү

программанын текстинде же клавиатура аркылуу киргизүү сабында же маалыматтар файлы аркылуу киргизүү менен жүргүзүлөт. Өзгөрмөлөрдүн маанилерин ал өзгөрмөлөрдү баяндоодо баштапкы маанилери менен кошо берүү – маанилештирүү деп аталат жана өзгөрмөнүн маанисин программанын текстинде берүү жолун билдирет. Маанилештирүүнү жөнөкөй өзгөрмөлөргө, массивдерге жана түзүлүштөргө жүргүзүүгө болот.

Мисалы:

```
int i=0, j=1;
int mas[5]={1,3,5,7,9};
int mat [2][3]={0,2,4,6,8,10}
```

Мындай маанилештирүү төмөнкү менчиктөө операторлорунун тобуна тең күчтүү:

```
mas [0]=1; mas [1]=3; mas 2]=5; mas [3]=7; mas [4]=9;
mat [0][0]=0; mat [0][1]=2; mat [0][2]=4; mat [1][0]=6; mat [1][1]=3; mat [1][2]=10;
```

Көп өлчөмдүү массивтерди маанилештирүүдө, анын кабатталган түрү колдонулат.

Мисалы:

```
int mat [2][3]={{0,2,4},{6,8,10}}
```

Түзүлүштөрдү маанилештирүүнү карайлы:

```
typedef struct {
                char atalysh [30];
                char avtor [20];
                 int baasy; } kitep;
main( )
{static kitep sap={“Айкөл Манас”, “Ж. Садыков”, 930}
}
```

kitep тибиндеги sap өзгөрмөсүн маанилештирүү жүргүзүлдү.

Текшерүүчү суроолор

1. Маалыматтардын туунду типтери
2. Массив жана анын мүнөздөмөлөрү
3. Си тилинде массивтерди баяндоо жана маанилештирүү
4. Көрсөткүч менен массивтердин байланышы
5. Си тилинде сап түшүнүгү
6. Саптарды киргизүү
7. Саптарды тыракту катары киргизүүнүн жолдору
8. Саптарды өзгөрмөнүн мааниси катары киргизүүнүн жолдору
9. Саптарды бириктирүү функциялары
10. Саптарды салыштыруу функциялары
11. Саптарды көчүрүү функциялары
12. Саптын узундугу жана символдорду издөө функциялары
13. Түзүлүштөрдүн аныктамасы
14. Си тилинде түзүлүштөрдү баяндоо жолдору
15. Түзүлүшкө көрсөткүчтү колдонуу
16. Бирикме түшүнүгү
17. Өзгөрмөлүү түзүлүш

Туунду типтерге көнүгүүлөр

1. a_1, a_2, …, a_{10} бүтүн сандардын удаалаштыгы берилген 5 эселүү болгон сандардын суммасын тапкыла
2. a_1, a_2, …, a_n бүтүн сандардын удаалаштыгы берилген. Жуп жана оң сандардын суммасын тапкыла
3. n жана p натуралдык сандары берилген. a_1, a_2, …, a_n бүтүн сандардын ичнен p санына эселүү болгон сандардын көбөйтүндүсүн тапкыла
4. a_1, a_2, …, a_n бүтүн сандары берилген. Ушул сандардын ичинен (0, 10) интервалына тиешелүү болгон сандарды 0 гө алмаштыргыла
5. a_1, a_2, …, a_n бүтүн сандарынын ичинен оң бүтүн сандар көпбү же терс бүтүн сандар көпбү аныктагыла
6. Элементтери бүтүн сан болгон квадраттык матрицасы берилген, ар бир саптын орточо арифметикалык санын тапкыла

7. Элементтери бүтүн сан болгон квадраттык матртца берилген. Матрицанын эң чоң элементи жаткан саптын жана мамычанын номерин аныктагыла

8. Элементтери чыныгы сан болгон nxm өлчөмүндө матрица берилген, эң кичине элементи жаткан саптын элементтеринин суммасын аныктагыла.

9. Элементтери чыныгы сан болгон nxm өлчөмүндө матрица берилген, ар бир сабынын эң чоң элементтерин аныктагыла

10. Элементтери чыныгы сан болгон квадраттык матрица берилген, диоганалдык элементтеринин суммаларын аныктагыла. Кайсы диоганалдык элементтеринин суммасы чоң экендигин тапкыла

11. s_1, s_2, …, s_n символдорунун удаалаштыгы берилген. а тамгасы канча жолу кезигээрин тапкыла

12. s_1, s_2, …, s_n символдорунун удаалаштыгы берилген, бул удаалаштыкта, арифметикалык амалдардын белгилери бар же жок экендигин аныктагыла

13. “Информатика” сөзүнөн ”формат” сөзүн бөлүп алуучу программаны түз

14. Эки ысымдан бир ысымды алуучу программаны түз

15. Берилген сүйлөмдө канча сөз бар экендигин аныкта

16. Студенттер жөнүндө маалыматтар берилген. информатика сабагынан “5” деген баа алган студенттердин аты-жөнүн чыгаруучу программа түз

17. Спорттун түрлөрү боюнча спортсмендердин тизмеси берилген. тогуз коргоол боюнча спортсмендердин тизмесин чыгар.

18. Окутуучулар жөнүндө маалымат берилген. “программалоо тили” боюнча сабак өтүүчү окутуучулардын аты жөнүн чыгаруучу программа түз.

19. Орто мектептин окуучулары жөнүндө маалымат берилген. Эки кружокко катышкан окуучулардын аты жөнүн чыгаруучу программаны түз.

20. Студенттер жөнүндө маалымат берилген. Баардык экзамендерден “5” деген баа алган студенттердин аты жөнүн чыгаруучу программаны түз.

5-Бөлүм. КАМТЫЛГАН ПРОГРАММА. ФУНКЦИЯ

5.1 Камтылган программа түшүнүгү

Программа түзүүдө программанын кандайдыр бир бөлүгү, программанын ар кайсы жеринде кайталанып колдонулушу көп кезигет. Программанын ошол бөлүгүн кайталап жазып отурбай, өзүнчө белги же ысым берип, программанын кайсыл жеринде кайталанса ошол жерден коюлган белги же ысым аркылуу кайрылып туруу ыкмасы колдонулат. Көпчүлүк учурда көп кайталанып колдонгон программанын бөлүктөрү арбын болот, ошолорду бири-биринен көз каранды болбогондой өзүнчө программалык бирдик катары чакан программа түрүндө түзүп, зарыл болгондо ага кайрылып турууга болот.

Си тилинде камтылган программанын баардык түрлөрүн функция катары карашат. Функциялар эки топко бөлүнөт:

1-топко: кайрылуу болгон учурда негизги программага маани берүүчү функциялар.

2-топко; кайрылуу болгон негизги программага маани бербеген функциялар.

Си тилинде функцияларды аныктоо жана баяндоо түшүнүктөрү колдонулат. ***Функцияларды аныктоо деп*** – функциялар кандай иш аракеттерди аткараарын көрсөткөн баяндамаларды айтабыз. ***Функцияларды баяндоо*** - деп программадан функцияга кандайча кайрылуулар жүргүзүүлөрүн берүүчү маалыматтарды айтабыз. Функцияны аныктоодо жана баяндоодо эстин класстары колдонулат.

5.2 Эстин класстары

Программадагы өзгөрмөлөрдүн эстеги кармалуу мөөнөтү жана иш аракеттер области эстин класстары менен аныкталат. Функциялар жана өзгөрмөлөр үчүн эстин класстары төмөндөгү кызматчы сөздөр менен берилет: auto, extern, static жана register. Ар бир эстин классына токтололу.

1) auto менен эстин автоматтык классы аныкталат, пайда болуу мөөнөтү убактылуу жана иш аракеттер областы чакан чөйрөгө гана таандык. Мындай эстин классы чакан чөйрөнүн өзгөрмөлөрү жана функциялары үчүн программа иштеген учурда эстен орун бөлөт, ал эми программа токтогон учурда бул орун бошотулат.

2) register менен эстин регистирлик классы аныкталат. Бул класстагы өзгөрмөлөрдүн маанилери болуп ЭЭМдин регистирине жайланышкан маалыматтар эсептелет, мындай ыкма маалыматтарды тез иштетүүдө колдонулат.

3) static кызматчы сөзү менен эстин статикалык классы аныкталат. Өзгөрмөлөр менен функциялардын эстеги кармалуу мөөнөтү туруктуу жана иш аракеттер областы чакан чөйрөгө гана таандык. Чакан чөйрөдөгү өзгөрмөлөр жана функциялар программанын иштөө процессинин башында пайда болуп жана булар үчүн эстен бир гана жолу орун бөлүнөт, ал орун программа аткарылып бүтмөйүнчө сакталат.

4) extern кызматчы сөзү менен эстин сырткы классы аныкталат. Өзгөрмөлөр менен функциялардын эстеги кармалуу мөөнөтү туруктуу жана иш аракеттер областы кеңири чөйрөгө (глобалдык) таандык. Сырткы өзгөрмөлөр функциялардын ортосундагы байланыш үчүн жана ар түрдүү файлдарда жаткан көз карандысыз компиляцияланган функциялар үчүн колдонулат, сырткы өзгөрмөлөргө бөлүнгөн эстеги орун туруктуу, бирок анын кармаган маанилери өзгөрүп турат.

5.3 Функцияны аныктоо

Функцияны аныктоонун жалпы жазылышы.

[static][<жыйынтык тиби>]<функциянын ысымы>(<формалдык параметрлер>)

[<формалдуу параметрди баяндоо>]

{

<функциянын тулкусу>

}

static кызматчы сөзү эстин классы. static классындагы эске ээ болгон функциялар өзүн кармаган файлдын ичинде гана иштетилет. Эгерде static сөзү жазылбаса анда функция extern (сырткы) деген эстин классына ээ болот. Мындай функциялар каалагандай файлдарда иштетилет. Ошондой эле <жыйынтык тиби> коюлбаса анда ал бүтүн тип int деп эсептелет. Дайыма айкындык үчүн <жыйынтык тибин> көрсөтүп жазуу талапка ылайык. Маани кайрып бербеген функциялар үчүн <жыйынтык тиби> куру <void> деп берилет. Функциянын жыйынтык мааниси катары маассивти же функцияны кайрып бербейт. Формалдуу параметрлерди баяндоо өзгөрмөлөрдү баяндаганга окшош эле болот. Баяндоо учурунда register классында эсти гана айкын көрсөтүү зарыл формалдуу параметр массивтерди баяндоодо массивтин биринчи индексин жазбай койсо болот.

Мисалы: int a[ ], b[ ], d[ ] [4]

Ал эми int x[ ] [ ]; баяндоосу тура эмес, себеби x массивинин экинчи индексинин максималдык мааниси берилген эмес. Маанилерин кайтарып берген функциялар сөзсүз түрдө return <туюнтма>; операторү менен аякталышы керек. Бул операторду функция аткарганда, анын иштөөсү аякталат жана return каралган туюнтманын мааниси функциянын жыйынтыгы болуп калат. Мындай операторлордон функциянын тулкусунда бир канча болушу мүмкүн. Эгерде return операторү <туюнтмасы> жок жазылса анда функция жыйынтык маанисин кайтарып бербейт. Камтылган программа түрүндөгү функцияга *мисал:*

Эки сандын чоңун табуучу программаны функция түрүндө жазгыла. Функцияны аныктоо.

int max (a,b);

int a,b;

```
{
return (a>b)?a:b;
}
```

Жогорку функцияны аныктоонун экинчи түрү.

```
int max (int a, int b);
{
return (a>b)?a:b;
}
```

5.4 Функцияны баяндоо

Программада функцияга анын аныктоосунан мурун ага кайрылуу же функция параметр катары берилиши мүмкүн. Бул учурда функцияга кайрылууга чейин анын баяндалышы болушу керек. Функциянын баяндоосунун эки түрү бар:

1) Функциянын маниси кайтарылып берилген учурда

[static же extern] <жыйынтык тиби> <функциянын ысымы> ( );

2) Функциянын маниси кайтарылбаган учуру.

[static же extern] void <функциянын ысымы> ( );

Эгерде баяндоодо эстин ысымы берилбесе ал класс extern деп эсептелет. Функциянын баяндалышында функциянын ысымынан кийин сөзсүз; (үтүрлүү чекит) белгиси коюлат, ал эми функцияны аныктоодо анын ысымынан кийин; (үтүрлүү чекит) коюлбайт.

Функцияны баяндоонун классикалык жана түпкү түспөлү (прототип) аркылуу баяндоо деген эки түрү бар. Функцияны баяндоонун классикалык түрүнө *мисал:*

```
int mars ( );
extern char *strcpy( );
```

Түпкү түспөлүн колдонуп функцияны баяндоодо функциянын параметрлери жөнүндө кошумча да информацияларды беришет. Бул информациялар функциянын ысымынан кийин тегерек кашаага алынып жазылат. *Мисалы:* int imax (int x1, int x2); long uzun (long x, long y); түпкү түспөлү

боюнча функцияны баяндоо uzun функциясына кайрылуусу бар башкы программанын бөлүгү.

```
void main (void);
{ int lim=52;
char ch=’M’;
long uz;
uz=uzun (lim, ch);
}
```

Бул программада uzun функциясына кайрылуу үчүн lim жана ch параметрлерин тизмекке (стек) жайгаштыруудан мурун, программа uzun функциясы үчүн түпкү түспөлү ыкмасы боюнча lim жана ch параметрлери long тибине өзгөртүлөт.

Эгерде функцияга түпкү түспөлү аркылуу баяндоону колдонбосок анда lim жана ch параметрлери бүтүн жана символдук маани катары тизмекке (стек) жайгаштырылып функцияга кайрылуу жүрмөк, бул учурда жыйынтык катага алып келмек. Түпкү түспөлдү колдонуп функциянын параметрлерин баяндоодо төмөндөгү эки эреже сакталышы керек.

1. Параметрлерди баяндоодо алар бири биринен үтүр (,) аркылуу ажыратылат, эч качан үтүрлүү чекит (;) белгиси коюлбайт.

2. Бир канча параметрлерди бир тип менен баяндоого болбойт, ар бир параметр үчүн ар кандай тип болушу керек. *Мисалы:* long a, b, c деп баяндоого болбойт.

Түпкү түспөлү аркылуу функциянын параметрин баяндоодо void тиби берилсе, анда ал функция параметрге ээ эмес дегенди билдирет.

Мисалы: int func (void); func функциясынын тиби бүтүн (int), ал эми параметрлери жок дегенди билдирет.

5.5 Функцияга кайрылуу

Си тилинде кабатталган функциялар колдонулбайт. Бирок функциялардын төмөндөгүдөй жайланышы мүмкүн:

```
static int f (……)
{…
}
extern int g (……)
{…
}
```

Бул учурда f-функциясына g-функциясынан кайрылууга болот, а бирок башка файлда жаткан эч бир функция f-функциясына кайрыла албайт. Маанисин кайтаруучу функцияга кайрылуунун жалпы жазылышы.

<функциянын ысымы> (<аныкталган параметрлер тизмеси>); аныкталган параметрлер туюнтма да болушу мүмкүн. *Мисалы:*

```
x=max (a, 52);
y=max (max (b, 52), a)+c;
```

Маанисин кайтарбаган функцияга кайрылуунун жалпы жазылышы:

<функциянын ысымы> (<формалдуу параметрлердин тизмеси>)

```
bery (0.5);
func (a,b, &i, &j);
```

Си тилиндеги функция Паскаль тилиндеги процедура жана функцияга окшош келет маанисин кайтаруучу функция бул Паскаль тилиндеги функцияга, ал эми маанисин кайтарбаган функция Паскаль тилиндеги процедурага окшош болот.

Си тилинде параметрлерди берүү маани боюнча гана берүү жолу менен жүргүзүлөт, демек Си тилинде параметрлердин баардыгы кирүүчү болуп эсептелет.

Мисалы: бөлчөктөрдү кыскартууга программа түзгүлө. 6/10=3/5 ж.б.у.с.

```
# include <stdio.h>
 # include <conio.h>
int funk (int x, int y);
void main ( )
  {int a, b,z;
   clrscr();
   printf (“Бөлчөктүн алымын жана бөлүмүн үтүр менен ажыратып  киргиз: ”);
   scanf (“%d,%d”, &a, &b);
   z= funk (a,b);
   printf (“\n %d %d=%d %d”, a, b, a/z, b/z);
  }
 /* Эң чоң жалпы бөлүүчүсүн табуучу функция */
   int funk (int x, int y)
{while (x!=y)
if (x>y) x-=y;
else y-=x;
return x;
}
```

Жыйынтык:

Бөлчөктүн алымын жана бөлүмүн үтүр менен ажыратып киргиз: 6,8

6/8=3/4

Бөлчөктүн алымын жана бөлүмүн үтүр менен ажыратып киргиз: 15,9

15/9=5/3

Функция өзүнүн аныкталган параметрин өзгөртүүсүз түрдө колдонуп, өзүнүн маанисин кайтарган учурда, параметрлерди маани боюнча берүү ыңгайлуу.

Көп сандагы маалыматтарды, параметрди маани боюнча берүү жолу менен жиберүү эффективдүү болбой калат, ошондуктан параметрлерди шилтеме (көрсөткүч) аркылуу берүүгө болот. Функцияга кайрылууда өзгөрмөнү чыгуу параметри катары колдонгон учурда өзгөрмөнүн өзү көрсөтүлбөстөн анын адреси көрсөтүлөт. Мисалы: scanf функциясына кайрылган учурда б.а. scanf функциясы аткарылганда анда кармалган өзгөрмөлөргө маани менчиктелет жана ал өзгөрмөлөр чыгуу параметрлери болуп калат. Бул функцияга кайрылган учурда өзгөрмөнүн ысымынын алдына & (амперсенд) белгиси коюлат, себеби бул функцияга кайрылуу учурунда өзгөрмөнүн адреси сөзсүз көрсөтүлөт. & белгиси адрести билдирет.

Мисалы: эки өзгөрмөнүн маанилерин алмаштыруучу функция түзөлү.

```
 # include <stdio.h>
void almash (int *a, int *b);
 main ( )
 {int i, j;
 i=30;
 j=19;
 printf (“Өзгөрмөнүн маанилери алмашканга чейин:i=%4d, j=%4d\n”, i, j);
 almash (&i, &j);
 printf (“Өзгөрмөнүн маанилери алмашкандан кийин:i=%4d, j=%4d\n”, i, j);
 }
 void almash (int *a, int *b);
 {int ;
 t=(*a);
 *a=(*b);
```

```
*b=t;
}
```

Жыйынтык:

Өзгөрмөнүн маанилери алмашканга чейин: i=30, j=19

Өзгөрмөнүн маанилери алмашкандан кийин: i=19, j=30

Бул мисалда almash функциясы a жана b эки формалдуу параметрди int бүтүн тибине болгон көрсөткүч катары жарыялайт. Ошондуктан almash функциясы бүтүн маанилүү өзгөрмөлөрдүн адресстери менен түздөн-түз иштейт жана almash функциясына кайрылган учурда анык параметр катары эки адрес көрсөтүлгөн almash функциясы бул эки адрести өзгөртө албайт, бирок бул адрестерде жаткан маанилерди өзгөртөт.

5.6 Массивтерди параметрлер катары берүү

Си тилинде баардык параметрлер маанилери боюнча берилет, бирок массивтин ысымы массивке болгон көрсөткүч болгондуктан б.а. массивтин биринчи элементинин адресин (дарегин) билдиргендиктен массивти параметр катары берүү көрсөткүч (шилтеме) аркылуу аткарылат. *Мисалы:* бир өлчөмдүү массивтин элементтерин киргизүүнү функция катары түзүп, массивти параметр катары берүүнү ишке ашыралы.

```
# include <stdio.h>
# define n 20 /*массивтин элементтеринин санынын жогорку чеги*/
void kirgiz (int a[],int s);/*функцияны түпкү түспөлү аркылуу баяндоо*/
main ( ) /*негизги программа*/
{int k, i, mas [n]; /*mas-массив, k-элементтеринин саны*/
printf (“Массивтин элементтеринин санын бер:”);
scanf (“%d”, &k);
kirgiz(mas,k); /*mas-массивин киргизүү үчүн kirgiz функциясына кайрылуу*/
```

```
for (i=0; i<k; i++) printf (“mas[%2d]=%d \n”, i+1, mas[i]);
}
void kirgiz (int a[ ], int s) /* kirgiz функцияысын аныктоо*/
{
int j;
for (j=0; j<s; j++)
{printf (“ Массивтин %d-элементин киргиз:”,j);
 scanf (“%d”, & a[j]);
 };
}
```

Жыйынтык:

```
Массивтин элементтеринин санын бер:3
Массивтин 1-элементин киргиз: 15
Массивтин 2-элементин киргиз: 10
Массивтин 3-элементин киргиз: 2009

mas[ 1]=15
mas[ 2]=10
mas[ 3]=2009
```

Жогорудагы программаны талдасак, kirgiz функциясын дагы int a [ ] жазуусу формалдуу a[ ] параметринин тибин гана белгилейт жана массивке эстен эч кандай орун бөлүнбөйт. Компилятор a [ ] белгисин, a массивинин башталыш адреси катары гана карайт. Демек kirgiz функциясында mas массивинин башталыш адреси берилди дегендин a массивинде кандайдыр бир өзгөрүүлөр болсо анда mas массивинде да өзгөрүүлөр болот дегенди билдирет.

Жогорудагы kirgiz функциясын баяндоонун классикалык түрүн карасак ал төмөндөгүдөй баяндалат: int kirgiz ал эми функциянын аныкталышы болсо мындай жазылат:

```
void kirgiz (a, s)
int a, s;
```

Эки өлчөмдүү массивди же матрицаны параметр катары берүү мисалын карайбыз:

```
# include <stdio.h>
# define s1 20
# define s2 5
  void kirgiz (int a [ ] [s2], int s);
  main ( )
     {int i, j, r, mas [s1] [s2];
       printf (“ Матрицанын биринчи өлчөмүн киргиз: ”);
       scanf (“%d”, &r);
       kirgiz (mas, r);
       printf (“ Киргизилген матрицанын элементери: \n”);
       for (i=0; i<r; i++)
       {for (j=0; j<s2; j++)printf (“%4d”, mas [i] [j]);
        printf (“\n”);
       }
       }
   void kirgiz (int a [ ] [s2], int s);
       {int i, j;
        for (i=0; i<s; i++)
        for (j=0; j<s2; j++){ printf (“a[%d] [%d]=”,i,j);
         scanf (“%d”, &a[i] [j]);};
       }
```

Жыйынтык:

```
   Матрицанын биринчи өлчөмүн киргиз: 2
a[0] [0]=1
a[0] [1]=2
a[0] [2]=3
a[0] [3]=4
a[0] [4]=5
a[1] [0]=5
```

```
a[1] [1]=6
a[1] [2]=7
a[1] [3]=8
a[1] [4]=9
Киргизилген матрицанын элементери:
  1   2   3   4   5
  5   6   7   8   9
```

Жогорку мисалда эки өлчөмдүү массивти баяндоодо, анын экинчи өлчөмүн жазбай коюга болбойт, себеби шилтеменин ысымын индексирлөө мүмкүн болбой калат. Эгерде бир өлчөмдүү массивти баяндоону жана аныктоону колдонсок б.а. kirgiz (int a[ ], int s), анда mas [r] [s2] эки өлчөмдүү массиви менен иштөөдө, функцияга кайрылуу kirgiz (mas, r*s2) түрүндө болот. Бул жерде kirgiz-функциясы mas-массивин r*s2 элементтен турган бир өлчөмдүү массив катары карайт.

Матрицаны параметр сапатында берүү менен, матрицалар менен иштөөчү камтылган программаны уюштурууну эки ыкма менен жүргүзүүгө болот. Биринчи ыкмада камтылган программага кайрылууда матрицанын ысымы көрсөтүлөт, ал эми камтылган программаны аныктоодо параметр, матрица катары баяндалып жана ал матрица катары иштетилет.

Экинчи ыкмада камтылган программага кайрылууда жогорудагыдай эле матрицанын ысымы көрсөтүлөт б.а. матрицанын баштапкы элементинин адреси көрсөтүлөт, ал эми камтылган программаны аныктоодо параметр көрсөткүч катары баяндалат. Ошондуктан камтылган программанын текстинде бул параметр менен адрестик арифметиканы колдонуп, көрсөткүч катары иштөө керек.

Массивтерди параметрлер катары берүүдө бул эки ыкманы аралаштырбоо керек. Биринчи ыкма түшүнүктүү жана көрсөтмөлүү, экинчи ыкма болсо ийкемдүү.

Бул эки ыкмага мисал келтирели. Мейли n*m өлчөмүндөгү матрица берилсин, ал матрицанын эң чоң элементин аныктагыла. Матрицаны mat-ысымы менен берели. Эң чоң элементин max-менен, саптардын санын n-менен, мамычалардын санын m-менен белгилейли.

Матрицанын элементтерин киргизүүчү функцияны kirel-менен ал эми эң чоң элементин аныктоочу функцияны matmax-менен белгилейли. Камтылган программаны биринчи ыкма менен түзөлү.

```
# include <stdio.h>
/* mat (n,m)матрицанын элементтерин киргизүүчү kirel функциясы */
void kirel (int n, int m, int mat [ ] [10])
{ int i, j;
for (i=0; i<n; i++)
for (j=0; j<m; j++){ printf (“\n mat [%d] [%d]= ”,i,j);
scanf (“%d”, & mat [i] [j]);};
}
/* mat (n,m) матрицанын эң чоң элементин аныктоочу matmax функциясы */
void matmax (int n, int m, int mat [ ] [10], int *max)
{int i, j;
*max=mat [0] [0];
for (i=0; i<n; i++)
for (j=0; j<m; j++)
if (*max<mat[i] [j]) *max=mat[i] [j];
}
/*негизги программа*/
void main (void)
{int n,m, mat [10] [10], maximum;
printf (“\n Матрицанын сабынын санын киргиз n=”);
```

```
scanf (“%d”, &n);
printf (“\n Матрицанын мамычаларынын  санын бер m=”);
scanf (“%d”, &m);
kirel (n,m, mat);
matmax (n,m,mat, &maximum);
printf (“\n mat-матрицасынын эң чоң элементи maximum=%d”, maximum)
}
```

Жыйынтык:

Матрицанын сабынын санын киргиз n=2

Матрицанын сабынын санын киргиз m=2

mat[0][0]=1

mat[0][0]=9

mat[0][0]=6

mat[0][0]=3

mat-матрицасынын эң чоң элементи maximum=9

Камтылган программаны 2-ыкма менен түзөлү.

```
# include <stdio.h>
/* mat-матрицанын элементтерин киргизүүчү функция*/
void kirel (int n, int m, int *mat)
{ int i, j;
for (i=0; i<n; i++)
for (j=0; j<m; j++)
scanf (“%d”,  mat ++);
}
/* mat -матрицанын эң чоң элементин аныктоо */
void matmax (int n, int m, int *mat, int *max)
{int i, j;
  *max=*mat
  for (i=0; i<n; i++)
  for (j=0; j<m; j++)
```

```
 if (*(mat+i*m+j)>*max)*max=*(mat+i*m+j);
 }
void main (void)
 {int n,m, mat [10] [10], maximum;
  printf (“\n Матрицанын сабынын санын бер n=”);
  scanf (“%d ”, &n);
  printf (“\n Матрицанын мамычасынын санын бер m=”);
  scanf (“%d ” &m);
   printf (“\n Матрицанын элементтерин сабы боюнча киргизгиле:”);
   kirel (n,m,&mat[0][0]);
   matmax (n,m, ,&mat[0][0], &maximum);
   printf(“\nМатрицанын эң чоң элементи maximum=%d”, maximum);
 }
```

Жыйынтык:

Матрицанын сабынын санын бер n=2

Матрицанын мамычасынын санын бер m=2

Матрицанын элементтерин сабы боюнча киргизгиле:19 3 30 15

Матрицанын эң чоң элементи maximum=30

Эскертүү: *(mat+i*m+j) жазуусу компьютердин эсинде матрицанын элементтери удаалаш саптар менен жайланышат дегенди билдирет б.а. адегенде 0 сабы, андан кийин 1-сабы ж.б.у.с. *Мисалы*: mat [2] [3] матрицасын карасак *(mat+i*m+j) жазуусу циклды эске алганда mat[0][0] mat+0 адреси, mat[0][1] элементи mat+1 адреси, mat[0][2] элементи mat+2, mat[1][0] элементи mat+3, mat[1] [1] элементи mat+4 адреси жана mat[1][2] элементи mat+5 адреси боюнча жайгашкан дегенди билдирет.

5.7 Функцияга болгон көрсөткүч

Си тилинде функцияга болгон көрсөткүчтү түзүүгө болот. *Мисалы:* эки ар түрдүү типтеги эки параметри болгон бүтүн int

типтүү funk-функциясына болгон көрсөткүчтүү баяндоону карайлы:

type 1, type 2;

int (*funk)(type1 p1, type2 p2)

функцияга болгон көрсөткүч биринчи байттын адресин же функция аткарган сөздүн кодунун адресин кармайт. Функцияга болгон көрсөткүчтүн үстүнөн арифметикалык амалдарды жүргүзүүгө болбойт. *Мисалы:*

```
#include <stdio.h>
#include <math.h >
double f (double x);
void main (void)
{double (*funk)( double x);
double z=1, y, y1;
funk=f;
y=(*funk)(z);
y1=funk(z);
printf (“z=%f, y=%f, y1=%f\n”, z,y,y1);
}
/* f-функциясын аныктоо*/
double f (double x)
{printf (“f-функциясы иштөөдө\n”);
return x+2;
}
```

Жыйынтык:

```
f-функциясы иштөөдө
z=1.000000, y=3.000000, y1=3.000000
```

Функцияны параметр катары берүүгө да болот. Бул учурда аныкталган берилүүчү маани функциянын адреси болуп эсептелет. *Мисалы:*

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

```
int kosh (int x, int y);
int aiyrma (int x, int y);
int esep (int x, int y, int(*kor) (int x, int y)); /* бул жерде kor функцияга болгон көрсөткүч*/
void main (void)
{int x,y,z;
printf (“\n Эки бүтүн санды киргиз:”);
 scanf (“%d %d”, &x, &y);
z=esep(x,y,kosh);
printf (“%d+%d=%d\n”, x,y,z);
z=esep(x,y,aiyrma);
printf (“%d-%d=%d\n”, x,y,z);
}
/* esep функциясын аныктоо*/
int esep (int x, int y, int(*kor) (int x, int y))
{
return (*kor)(x,y);
}
/* kosh функциясын аныктоо */
int kosh (int x, int y);
{
 return x+y;
}
/* aiyrma функциясын аныктоо */
int aiyrma (int x, int y);
{
return x-y;
}
```

Жыйынтык:

```
Эки бүтүн санды киргиз: 7 2
7+2=9
7-2=5
```

Жогорку программадагы esep, kosh жана aiyrma функцияларына тиешелүү түрдө nat, cosh жана kemit аралык өзгөрмөлөрүн колдонуп программа түзсө да болот натыйжа бирдей эле болот.

```
#include <stdio.h>
int kosh (int x, int y);
int aiyrma (int x, int y);
int esep (int x, int y, int(*kor) (int x, int y)); /* бул жерде kor функцияга болгон көрсөткүч*/
void main (void)
{int x,y,z;
printf (“\n эки бүтүн санды киргиз:”);
 scanf (“%d %d”, &x, &y);
z=esep(x,y,kosh);
printf (“%d+%d=%d\n”, x,y,z);
z=esep(x,y,aiyrma);
printf (“%d-%d=%d\n”, x,y,z);
}
/* esep функциясын аныктоо*/
int esep (int x, int y, int(*kor) (int x, int y))
{int nat;
nat=(*kor)(x,y);
return nat
}
/* kosh функциясын аныктоо */
int kosh (int x, int y)
{int cosh;
cosh=x+y;
return cosh;
}
/* aiyrma функциясын аныктоо */
int aiyrma (int x, int y)
{int kemit;
```

```
kemit=x-y;
return kemit;
}
```

Жыйынтык :

```
Эки бүтүн санды киргиз: 7 2
7+2=9
7-2=5
```

5.8 Локалдык жана глобалдык маанилер

Функциянын тулкусунда баяндалган өзгөрмөлөр, турактуулар жана маалыматтардын типтери локалдык маанилер деп аталат. Локалдык маанилер ошол функцияда гана колдонулат. Ар кандай функцияларда бирдей ысымдуу локалдык өзгөрмөлөр колдонулушу мүмкүн, бирок бул өзгөрмөлөр бири-бири менен байланышы болбойт. Функциянын параметрлери да локалдуу болушат.

Негизги программада баяндалган өзгөрмөлөр, турактуулар жана маалыматтардын типтери глобалдык маанилер деп аталат. Глобалдык маанилерди баяндагандан кийинки баардык функцияларга колдонууга болот. Глобалдык өзгөрмөлөрдүн жардамы менен программанын негизги объектилери баяндалат.

Текшерүүчү суроолор

1. Камтылган программа түшүнүгү
2. Функциялар камтылган программа катары
3. Эстин класстары
4. Функцияны аныктоо
5. Функцияны баяндоо
6. Функцияга кайрылуу
7. Массивтерди параметр катары берүү
8. Функцияга болгон көрсөткүч
9. Параметрлүү жана параметрсиз функциялар
10. Глобалдык жана локалдык маанилер

Камтылган программага көнүгүүлөр

1. a(4, 5) матрицасы берилген, ар бир сабынын эң чоң элементтерин аныкта. Эң чоң элементтерди чыгарууну, камтылган программа катары түзгүлө.

2. Квадраттык матрица берилген. Эң чоң элементи менен эң кичине элементинин ордуларын алмаштыр. Эң чоң жана эң кичине элементтерин жана алардын индексттерин аныктоону камтылган программа катары түзгүлө.

3. a (m,n) матрицасы берилген. Кайсы сабында 0 дү көп кармаарын аныктоочу программаны түз. Саптагы 0дүк элементтин санын чыгарууну камтылган программа катары түзгүлө.

4. Квадраттык матрица берилген, кайсы диоганалынын элементтеринин суммасы чоң. Элементтеринин суммасын эсептөөнү камтылган программа катары түзгүлө.

5. a(m,n) матрицасы берилген. Ар бир сабынын элементтеринин көбөйтүндүлөрүн аныкта жана алардын ичинен эң чоң көбөйтүндүсүн чыгар. Эң чоң көбөйтүндүнү чыгарууну камтылган программа катары түзгүлө.

6. a(5, 4) матрицасынын 2 жана 4 сабынын чоң элементтерин аныкта. Чоң элементтерди чыгарууну камтылган программа катары түзгүлө.

7. Сүйлөм берилген, ал сүйлөмдөгү эң акыркы сөзү канча тамгадан турат. Тамганын санын эсептөөнү камтылган программа катары түзгүлө.

8. Сүйлөмдүн эң узун сөзүн тапкыла. Сөздүн узундугун эсептөөнү камтылган программа катары түзгүлө.

9. Сүйлөмдүн кайсыл сөзүндө “a” тамгасы көп? Сөздөрдү эсептөөнү камтылган программа катары түзгүлө.

10. Сүйлөмдүн кайсыл сөзүндө цифралар бар? Цифраларды аныктоочу камтылган программа катары түзгүлө.

6-Бөлүм. ФАЙЛДАР

Си тилинде файл катары логикалык байланышта болгон информациялардын атайын уюштурулган жыйындысын айтышат. Файлдардын мисалы катары: баштапкы модулдун тексттин, аткарууга даяр болгон б.а. курагычтан (компоновкадан) кийинки программаны жана программа үчүн баштапкы маалыматтардын тобун алсак болот. Файл катары эстин кандайдыр бир информация кармаган бөлүгүн да алышат б.а. файлга маалыматтарды киргизип чыгаруу иш аракети катары карашат. Си тилинде киргизип-чыгаруу библиотекалык функциялар аркылуу жүргүзүлөт. Киргизүү-чыгарууда файл (file) жана агым (stream) деген эки түшүнүктү айырмалап кароо керек. Си системасы киргизип-чыгаруунун кайсыл түзүлүшү пайдаланаарына көз каранды болбогон киргизип-чыгарууну да колдонот.

Ушундай абстрактуу киргизип-чыгаруунун логикалык түзүлүшү – ***агым*** деп аталат.

Маалыматтарды белгилүү болгон ыкмалар менен сактоого жөндөмдүү болгон, аныкталган физикалык түзүлүштүн (дискалар, лазердик түзүлүштөр ж.б.) бөлүгүн ***файл*** деп атайбыз.

Демек баардык агымдар бирдей болот, ал эми файлдар дайыма эле бирдей боло бербейт. Агым-анык бир физикалык түзүлүшкө көз каранды болбогондуктан, агымга маалыматтарды сактоочу бир эле стандарттык камтылган программа ошол эле маалыматтарды дискага да, экранга да ж.б. түзүлүштөргө да чыгара алат.

Си тилинде эки түрдөгү тексттик (text) жана экилик код түрүндөгү (binary) файлдар каралат.

6.1 Агым түшүнүгү

Көпчүлүк программалар киргизилүүчү маалыматтарды агым деп аталган атайын файлдардан окутат. Ошондой эле чыгарылуучу маалыматтар да агымга жазылат.

Агым деп – колдонуучунун деңгээлинде ийкемдүү киргизип-чыгаруу ишке ашыруучу буфер менен биргелешкен файлды айтабыз.

Буфер – бул учурдагы эстин бир бөлүгү. Буфердик киргизүүдө маалымат адегенде буферге келип түшөт андан кийин аны колдоно турган программага берилет. Ушундай эле буфердик чыгарууда маалымат программадан буферге берилет, андан кийин гана сырткы сактоочу түзүлүштөргө жиберилет.

Агымдарды аныктоо.

Агымдарды башкаруу үчүн агым көрсөткүчү колдонулат (stream pointer). Агым көрсөткүчү маалыматтарды кармаган файл жайгаштырылган түзүлүшкө болгон көрсөткүч (же шилтеме) катары эсептелет. Агым көрсөткүчү алдын ала аныкталган FILE типтүү түзүлүштүн жардамы менен аныкталат.

Төмөндөгүдөй: FILE * <агым көрсөткүчүнүн ысымы>

Агым көрсөткүчүнүн ысымы-бул FILE түзүлүшүнө болгон көрсөткүчтүн ысымы. FILE тибинин баяндалышы stdio.h файлында берилет.

Стандарттык агымдар.

Си системасында үч стандарттык агымдар бар:

stdin-стандарттык киргизүү файлы.

stdout-стандарттык чыгаруу файлы.

stderr-каталар жөнүндө кабарлардын стандарттык файлы.

Бул файлдар дайыма колдонуучулардын терминалдарын (дисплей жана клавиатура) байланыштырат. Стандарттык файлдар программада иштегенде автоматтык түрдө ачылып, ал эми иштөөсүн токтотқондо жабылат.

Стандарттык агымдар stdio.h файлында жатат жана аларды аныктоо FILE *stdin, *stdout, *stderr; түрүндө болот. Маалыматтарды стандарттык агымдардан окуу үчүн төмөндөгү

функцияларды пайдаланса болот: getchar, gets жана калыптап окуучу scanf, ал эми маалыматтарды жазуу үчүн putchar, puts жана printf функциялары колдонулат.

Стандарттык эмес агымдар.

Программист аныктаган агымдар стандарттык эмес агымдар деп аталат. Эгерде колдонуучу стандарттык эмес агымдарды колдонсо, анда ал өзүнүн программасында ал агымды ачуу жана жабуу мүмкүнчүлүгүн карашы керек. Агымдар FILE тибин колдонуу менен аныкталат. Мисалы: FILE *shilteme

Файлды агым катары ачуу менен бул файл үчүн буферди уюштурууда fopen функциясы колдонулат. Файлды башкача жол менен да уюштурууга болот. Адегенде open функциясынын жардамы менен ачып, андан кийин аны fopen функциясы менен агымга өзгөртүп түзсө болот.

Файлдын көрсөткүчү маалыматтарды киргизүүдө жана чыгарууда агымдан кийинки элементинин абалын белгилейт. Файлдын көрсөткүчүн керектүү орунга орнотуу үчүн fseek-функциясы колдонулат. Ал эми файлдын көрсөткүчүн файлдын башына жылдырыш үчүн rewind-функциясы колдонулат.

Агымды ачууда fopen-функциясы колдонулат, жалпы жазылышы FILE *fopen (char *fisim, char *type); Бул функция fisim сабы менен берилген ысым менен файлды ачат, ал эми type параметри файлды ачуудагы колдонуучу режимдерди кармайт. Функция ачылган файлга көрсөткүчтү пайда кылат. Бул көрсөткүч FILE түзүлүшүн көрсөткөн көрсөткүч катары алдын ала баяндалышы керек. Эгерде жок файлды ачуу иш аракети каралса же кандайдыр бир ката келип чыкса бул функция NULL деген маанини берет. Ошондуктан fopen-функциясына кайрылуудан кийин программада шартуу if-оператору менен NULL мааниси барбы же жокпу текшерүү зарыл.

Мисалы:

FILE *shilteme;

shilteme=fopen (“misal.txt”, “w”);

if shilteme==NULL) printf (“файл ачылган жок”) else printf (“файл ачылды”)

Бул мисалда учурдагы каталогко misal.txt файлы маалыматтарды жазуу үчүн

(“w”-маалыматтарды жазуу үчүн файлды түзүү режими) түзүлөт.

6.2 Файлды уюштуруу

Файлды уюштуруу үчүн fopen-функциясы колдонулат. Жалпы жазылышы:

file * fopen (char * fisym, char * type);

Бул функция fisym-ысымдуу файлды ачат, ал эми type параметри ачылган файл колдонгон режимди аныктоочу сапты көрсөтөт. Файлды уюштуруу катары маалыматтарды сактоо үчүн файлды түзүүнү, маалыматтарды окуу үчүн файлды ачууну, маалыматтарды толуктоо үчүн файлды ачуу, файлды өркүндөтүү ж.у.с. файлдарды уюштуруунун параметрлери.

“r” – файлды окуу үчүн ачуу

“w”- файлга жазуу үчүн аны түзүү

“a”- файлга маалыматтарды толуктоо

“r+”- файлды өркүндөтүү б.а. окуу жана жазуу үчүн файлды ачуу

“w+”- файлды түзүү менен өркүндөтүү

“a+”- файлды толуктоо менен өркүндөтүү

“r” – параметрин колдонгондо:

1) сырткы эсте бар болгон файл ачылат.

2) эгерде сырткы эсте файл жок болсо fopen функциясы NULL деген маанини берет

“w”- параметрин колдонгондо:

1) сырткы эсте файл бар болсо ал файлдагы маалыматтар толугу менен өчүрүлөт да жаңы маалыматтар башынан баштап жазылат.

2) эгерде сырткы эсте андай ысымдуу файл жок болсо анда жаңы файл түзүлөт.

“a”- параметрин колдонгондо сырткы эстеги бар болгон файлга маалыматтарды аягынан тартып толуктайт, эгер жаңы файл болсо башынан баштап маалыматтар жазылат.

Өркүндөтүү параметрлерин (r+, w+, a+) колдонгондо файлды ачуу, өркүндөтүлбөгөн параметрлердегидей (r, w, a) эле болот. Бирок, биринчиден киргизүүдөн кийин ошол эле замат чыгаруу fseek же rewind функцияларын колдонгондон кийин гана ишке ашып fseek-функциясы файлдын көрсөткүчүн орнотот. Rewind-функциясы файлдын башталышына көрсөткүчтү орнотот. Экинчиден fseek же rewind функцияларын колдонбой туруп чыгаруудан кийин ошол эле замат киргизүүнү ишке ашырууга болбойт.

6.3 Агымды жабуу жана тазалоо

Агымды иштеткенден кийин аны жабыш керек. Жабуу үчүн fclose функциясына кайрылуу жүргүзүлөт. Бул функциянын жазылышы int fclose (FILL * agym); бул функция agym ысымдуу агымды жабат. Агымды жабуу алдында, бул агым менен байланышкан баардык буферлер тазаланат. Агымды тазалоо fflush-функциясы аркылуу жүргүзүлөт. Бул функциянын баяндалышы stdio.h файлында жайгашкан жана жалпы берилиши:

int fflush (FILE * agym)

Эгерде agym киргизүүнүн ачылган агымы болсо, agym ачылган бойдон калат. Ал эми чыгаруунун ачылган агымы болсо, бул агым менен байланышкан буферлер тазаланат.

Мисалы программанын бир бөлүгүн карайлы.

scanf (“%d”, &a);

```
fflush (stdin);
scanf (“%d”, &b);
fflush (stdin);
gets (sap);
```

Бул программанын бөлүгүндө биринчи колдонулган fflush (stdin) функциясы киргизүү туура эмес болгондо stdin-киргизүү агымын тазалайт. Экинчи scanf-функциясын колдонгон учурда да stdin-агымы тазаланат, себеби gets-функциясы так аткарылышы үчүн анын алдында fflush функциясын колдонуу керек.

Агымга маалыматтарды калыптап жазуу.

Маалыматтарды агымга жазуу үчүн fprintf-функциясы колдонулат. Жалпы жазылышы:

int fprintf (FILE * agym, char * format[, belgi, …,]);

fprintf-функциясы берилген чыгаруу агымына маалыматтарды жазат. Агымдын ысымы agym параметри менен берилет.

Мисалы:

```
FILE *kor;
kor=fopen (“mat1.mal”, “w”);
fprintf (kor, “%d %d”, n,m);
fprintf (kor, “%d”, x);}
        fclose (kor);
```

Агымдан маалыматтарды калыптап окуу үчүн fscanf-функциясы колдонулат. Жалпы жазылышы:

int fscanf (FILE * agym, char * format[, belgi, …,]);

fscanf-функциясы берилген кирүү агымынан маалыматтарды окуйт.

Мисалы:

```
FILE *kor;
```

```
kor=fopen (“mat1.mal”, “r”);
fscanf (kor, “%d %d”, &n, &m);
for (i=0; i<n; i++)
for (j=0; j<m; j++)
fscanf (kor, “%d”, &mat [i] [j]);
fclose (kor);
```

Агымга маалыматтарды калыптап киргизүүгө жана чыгарууга мисал карайлы. Мейли баштапкы матрица (mxn) өлчөмүндө берилсин жана бул матрица mat1.mal ысымдуу файлына жазылсын, ар бир сабынын элементтеринин суммасын эсептеп, ал суммалардын өсүү тартиби боюнча саптар ирээтелип алынган жаңы матрица mat2.mal файлына жазылсын. Бул мисалдын программасы төмөндөгүдөй.

```
# include <stdio.h>
# include <conio.h>
main()
{
FILE *kor;/*баштапкы матрицаны кармаган mat1.mal файлын түзүүчү*/
  int n,m,i, j, x;
  clrscr();
  kor=fopen(“d:mat1.mal”,“w”); /* d дискетине mat1.mal файлын түзүү*/
  printf (“\n n<10, m<10 болгондой mat[n][m] матрицанын өлчөмдөрүн үтүр менен ажыратып киргиз”);
  scanf (“%d,%d”, &n,&m);
  printf(“Бүтүн маанилүү өлчөмдөрү (%d*%d) болгон матрицаны сап боюнча киргиз:\n”, n,m);
   for (i=0; i<n; i++)
   for (j=0; j<m; j++)
```

```
    {scanf ("%d", &x);
      fprintf (kor, "%2d", x);}
    fclose (kor);
 }
```

Жыйынтык:

n<10, m<10 болгондой mat[n] [m] матрицанын өлчөмдөрүн киргиз: 4,3

бүтүн маанилүү өлчөмдөрү (4*3) болгон матрицаны сап боюнча киргиз:

7 8 9
4 5 6
1 2 3
9 9 9

D: дискетинде mat1.mal файлы түзүлөт. Ал файлды тексттик редакторлор менен ачып көрсөк болот mat1.mal файлы кармаган маалыматтар төмөндөгүдөй түрдө болот.

Текшерүүчү суроолор

1. Агым түшүнүгү
2. Стандарттык агымдар
3. Стандарттык эмес агымдар
4. Файлдын категориялары
5. Файлды ачуу функциясы
6. Файлдын колдонуу режимдери
7. Файлды жабуу функциясы
8. Агымды тазалоо
9. Агымга маалыматтарды калыптап жазуу
10. Агымдан маалыматтарды калыптап окуу

Файлдарга көнүгүүлөр

1. Элементтери бүтүн сан болгон f файлын түзгүлө. Бул файлдын 3кө так бөлүнгөн элементтерин g файлына жазгыла.

2. f файлы бүтүн сандарды кармайт. Бул файлдан баардык оң элеменнтерин g файлына жана баардык терс элеменнтерин h файлына жазгыла.

3. Элементтери бүтүн сан болгон f файлы берилген. Бул файлдын кайталанган элементеринен башкасын g файлына жазгыла.

4. Элементтери бүтүн сан болгон f файлы берилген. [10,20] аралыгында жаткан сандарды g файлына жазгыла.

5. Квадраттык бүтүн типтүү матрица берилген, саптарынын элементеринин суммасын f файлына жазгыла.

6. Квадраттык бүтүн типтүү матрица берилген, саптарынын эң чоң элементерин f файлына жазгыла.

7. Сүйлөм берилген «а» тамгалары бар сөздөрдү f файлына жазгыла.

8. Сүйлөм берилген эң узун сөзүн f файлына жазгыла.

9. «Информатика» сөзүндөгү маани берүүчү(формат, ат, тик) сөздөрүн f файлына жазгыла.

10. «Си программалоо тили» сүйлөмүндөгү ар бир тамга канча жолу кезигет, тамганы жана кезигүү санын f файлына жазгыла.

АДАБИЯТТАР

1. Аммерал Л. Программирование графики на Турбо Си. -1992, -221 с.

2. Березин Б.И., Березин С.Б.Начальный курс С и С++. –М.: ДИАЛОГ -МИФИ, 1996-288 с.

3. Больски М.И. Язык программирование Си. Справочник. -М.: Радио и связь 1988. -96 с.

4. Берри Р.,Микинз Б. Язык Си. Введение для программистов. - М.: Финансы и статистика, 1988-139 с.

5. Бочков С.О., Субботин Д.М. Язык программирование Си для персональных компьютеров. -М.: СП ДИАЛОГ, 1991-383 с.

6. Джехани Н. Программирование на язык Си. -М.: Радио и связь. 1988.-192 с.

7. Керниган Б., Ритчи Д., Фьюэр А. Язык программирование Си. Задачи по языке Си. - М.: Финансы и статистика, 1989-279 с.

8. Прата С. Язык программирование Си. Лекции и упражнение. -М.: Издательский дом «Вильямс», 2006-960 с.

9. Техника программирование на Турбо-С. -М.: ИВК-СОФТ, 1991. -180 с.

10. Уинер Р. Язык Турбо -Си. -М.: Мир, 1991 -380 с.

11. Фьюэр А. Задачи по языку Си. -М.: Финансы и статистика, 1985 -120 с.

Мазмууну

Эркинбаев М.А.

Си тилинде программалоонун негиздери
Окуу куралы

Тех. редактор: Жакыпова Ч.А.
Компьютерге калыпка салган: Жумашева Ж.Ж.

К.Тыныстанов атындагы ЫМУнун
полиграфиялык комплексинде басылды
Заказ 414 Тираж 35.
Тел.: (03922) 52696.