КАМТЫЛГАН ПРОГРАММА. ФУНКЦИЯ

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.

C++ 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.

C++ 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.

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 области 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 мааниле ри 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 области 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 области 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.

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 х массивинин экинчи индексинин максималдык мааниси берилген эмес. Маанилерин кайтарып берген функциялар сөзсүз түрдө return
<туюнтма>; оператору 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 оператору <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;
}
```

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 мааниси кайтарылып берилген учурда
[static же extern] <жыйынтык тиби> <функциянын ысымы> ( );

2) Функциянын мааниси 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.

Функцияга кайрылуу

C++ 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);
```

C++ 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.

C++ тилинде параметрлерди берүү маани боюнча гана берүү жолу менен жүргүзүлөт, демек C++ 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.

Массивтерди параметрлер катары берүү

C++ 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

0Матрицанын 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.

Функцияга болгон көрсөткүч

C++ 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
```

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.