КӨРСӨТКҮЧТӨР

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. ***К рс тк ч*** – 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.

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 мааниси 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 мааниси 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:

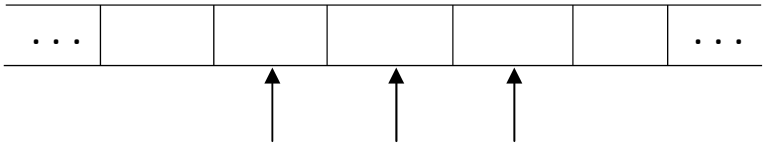
b ны жайгаштыруу : FFF4
b нын кармаган мааниси : 2010 q нун
кармаган мааниси FFF4 FFF4
Адресиндеги маани 2010

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.



P-1 P P+1

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 туруктуу 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.

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

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;
}
```