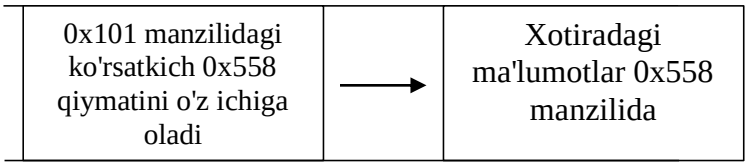
18-MA’RUZA. KO’RSATKICHLAR. ADRES OLUVCHI O’ZGARUVCHILAR

1. Ko‘rsatkichlar. Adres oluvchi o’zgaruvchilar

C++ tilining eng katta afzalliklaridan biri shundaki, u mashina darajasidan abstraktlash paytida yuqori darajadagi dasturlarni yozish, shu bilan birga kerak bo‘lganda apparatga yaqin ishlash imkoniyatini ham beradi. C++ ilova ishlashini bayt va bit darajasida sozlash imkonini beradi. Ko‘rsatkichlarning qanday ishlashini tushunish tizim resurslaridan samarali foydalanadigan dasturlarni yozishni o‘rganish bosqichlaridan biridir.

Ko‘rsatkichlar. Ko‘rsatkich – bu maydon manzilini xotirada saqlaydigan o‘zgaruvchidir. int o‘zgaruvchisi butun qiymatni saqlash uchun ishlatilgandek, ko‘rsatkich o‘zgaruvchisi xotira maydoni manzilini saqlash uchun ishlatiladi (13-rasm).



13-rasm. 0x558 manziliga ko‘rsatkich

Shunday qilib, **ko‘rsatkich** o‘zgaruvchidir va barcha **o‘zgaruvchilar** singari u ham xotiradan joy egallaydi (13-rasmda 0x101 manzilda). Ko‘rsatkichlarni maxsus ko‘rinishga keltiradigan xususiyat shundaki, ular tarkibidagi qiymatlar (bu holda, 0x558) xotira maydonlarining manzillari sifatida talqin etiladi. Demak, ko‘rsatkich - bu xotiradagi maydonni ko‘rsatadigan maxsus o‘zgaruvchidir.

Xotira manzillari odatda o‘n oltilik sanoq sistemasida ko‘rsatiladi, ya’ni 16 xil belgilar yordamida - 0-9, so‘ngra A-F. An'anaga ko‘ra 0x prefiksi o‘n oltilik sanoq sistemasidan oldin yoziladi. Shunday qilib, 0xA o‘n oltilik oltilik sanoq sistemasidagi raqam o‘nlik sanoq sistemasida 10ni ifodalaydi; 0xF - 15; va 0x10 – 16 ni ifodalaydi.

Ko‘rsatkichni e’lon qilish. Ko‘rsatkich o‘zgaruvchi bo‘lgani uchun uni boshqa har qanday o‘zgaruvchi singari e'lon qilish kerak. Odatda, ko‘rsatkich ma’lum bir turdagi qiymatga ishora qiladi (masalan, int tipiga). Bu shuni anglatadiki, ko‘rsatkichda joylashgan manzil butun sonni o‘z ichiga olgan xotiradagi maydonga ishora qiladi. Shuningdek, xotiraning tipga ega bo‘lmagan blokiga ko‘rsatkichni belgilashingiz mumkin (void ga ko‘rsatkich deb ham ataladi).

Ko‘rsatkich boshqa barcha o‘zgaruvchilar singari e'lon qilinishi kerak:

tip_nomi * Ko‘rsatkich_nomi;

Ko‘pgina o‘zgaruvchilar bilan bo‘lgani kabi, agar ko‘rsatkichni initsializatsiya qilmasangiz, unda tasodifiy qiymat bo‘ladi. Tasodifiy xotira maydoniga kirmaslik uchun ko‘rsatkich **nullptr**[1] qiymati bilan boshlanadi. Ko‘rsatkich qiymati har doim **nullptr** qiymatiga tengligini tekshirishi mumkin, bu haqiqiy xotira maydonining manzili bo‘lishi mumkin emas:

tip_nomi * Ko‘rsatkich_nomi = nullptr;

Shunday qilib, butun sonli ko‘rsatkichni e'lon qilish quyidagicha bo‘ladi:

int *pInteger = nullptr;

Ko‘rsatkich, shu paytgacha o‘rganilgan har qanday boshqa ma’lumotlar turining o‘zgaruvchisi kabi, initsializatsiyadan oldin tasodifiy qiymatni o‘z ichiga oladi. Ko‘rsatkich uchun bu tasodifiy qiymat ayniqsa xavflidir, chunki u xotira maydonining ba’zi manzillarini anglatadi. Initsializatsiya qilinmagan ko‘rsatkichlar dasturingizning yaroqsiz xotira maydoniga kirishiga olib kelishi va u dasturni ishdan chiqarishi mumkin.

Adres oluvchi o‘zgaruvchilar. & adres olish amali. O‘zgaruvchilar - bu til tomonidan xotirada ma’lumotlar bilan ishlashni ta’minlaydigan imkoniyat.

Agar **varName** o‘zgaruvchi bo‘lsa, &varName uning qiymati saqlanadigan xotira adres o‘rnini qaytaradi. Shunday qilib, agar sintaksisdan foydalangan holda butun o‘zgaruvchini e'lon qilgan bo‘lsangiz, sizga tanish bo‘lgan

int age = 30;

u holda &age ifodasi belgilangan qiymat 30 joylashtirilgan xotira maydonining manzilini qaytaradi.

Yuqorida saqlangan qiymatga kirish uchun foydalaniladigan butun sonli o‘zgaruvchi xotira manzilini olish ko‘rsatilgan tushunchasi ko‘rsatilgan.

Misol. O‘zgaruvchi manzilini olish

```
#include <iostream>
using namespace std;
```

[1] Ushbu qiymat - bo'sh ko'rsatkich C ++ da C++11 standartida paydo bo'ldi. Ilgari, C ga mos keladigan **null** qiymat ishlatilgan (bu hali ham ishlatilishi mumkin, garchi **nullptr** faqatgna yangi dasturlar uchun tavsiya etilsa ham).

```
int main()
{
   int age = 30;
   const double Pi =3.1416;
   cout <<"age manzili: "<<&age<<endl;
   cout <<"Pi manzili: "<<&Pi<<endl;
   return 0;
}
```

Adreslarni saqlash uchun ko‘rsatkichlardan foydalanish. Ko‘rsatkichlarni qanday e'lon qilishni va o‘zgaruvchining manzilini aniqlashni, shuningdek, ko‘rsatkichlar xotira maydoni manzilini saqlash uchun ishlatiladigan o‘zgaruvchini bilasiz.

Ushbu ma’lumotni birlashtirish va & adres olish operatori yordamida olingan adreslarni saqlash uchun ko‘rsatkichlardan foydalanish vaqti keldi.

Allaqachon ma’lum bir turdagi o‘zgaruvchini e'lon qilish sintaksisini yaxshi bilasiz:

```
tip O‘zgaruvchi_nomi = Boshlang‘ich qiymat;
```

Ushbu o‘zgaruvchining adresini ko‘rsatkichda saqlash uchun siz belgilangan tip orqali ko‘rsatkichni e'lon qilishingiz va uni adresni olish operatori yordamida initsializatsiyalashingiz kerak:

```
tip* Ko‘rsatkich = &O‘zgaruvchi_nomi;
```

int tipidagi **age** o‘zgaruvchisini shunday e'lon qildingiz deylik:

```
int age = 30;
```

Keyinchalik foydalanish uchun **age** o‘zgaruvchisi qiymatining manzilini saqlaydigan int ko‘rsatkichi quyidagicha e'lon qilinadi:

```
int *pointsToInt = &age; //age butun sonli o‘zgaruvchisiga ko‘rsatkich
```

Misol. Quyidagi misolda & operatori bilan olingan manzilni saqlash uchun ko‘rsatkichdan foydalanishni keltirilgan.

```
#include <iostream>
using namespace std;
```

```
int main()
{
   int age = 30;
   int* pointsToInt = &age;
   //Ko‘rsatkich qiymatini chiqarish
   cout<<"age manzili: "<<pointsToInt<<endl;
   return 0;
}
```

Endi adres ko‘rsatkich o‘zgaruvchisida qanday saqlashni bilganingizdan so‘ng, quyidagi misolda ko‘rsatilgandek, xuddi shu ko‘rsatkichga boshqa xotira adresi berilishi va keyin boshqa qiymatga ishora qilishi mumkin deb taxmin qilish mantiqan to‘g‘ri keladi.

```
#include <iostream>
using namespace std;
int main()
{
   int age = 30;

   int* pointsToInt = &age;
   cout<<"pointsToInt age ga ko‘rsatkich"<<endl;

   cout<<"pointstoInt = "<<pointsToInt<<endl;
   int dogsAge = 9;
   pointsToInt = &dogsAge;
   cout<<"pointsToInt dogsAge ga ko‘rsatkich"<<endl;
   cout<<"pointstoInt = "<<pointsToInt<<endl;
   return 0;
}
```

Ajratish operatori * yordamida ma’lumotlarga kirish. Sizda to‘liq manzilni o‘z ichiga olgan ko‘rsatkich bor deylik. Qanday qilib biz o‘z ichiga olgan ma’lumotlarni yozish yoki o‘qish uchun ushbu maydonga kirishimiz mumkin? Buning uchun ajratish operatori * ishlatiladi.

Aslida, agar **pData** ko‘rsatkichi mavjud bo‘lsa, *pData ifodasi ushbu ko‘rsatkichda joylashgan manzilda saqlangan qiymatga kirishga imkon beradi.

***** operatoridan foydalanish quyidagi dasturda ko‘rsatilgan.

```
#include <iostream>
```

```
using namespace std;
int main()
{
   int age = 30;
   int dogsAge = 9;

   cout<<"age ="<<age<<endl;
   cout<<"dogsAge = "<<dogsAge<<endl;

   int* pointsToInt = &age;
   cout<<"pointsToInt age ga ko‘rsatkich"<<endl;
   //Ko‘rsatkich qiymatini chiqarish
   cout<<"pointstoInt = "<<pointsToInt<<endl;
   //Ko‘rsatilgan sohadan qiymatni chiqarish

   cout<<"*pointsToInt="<<*pointsToInt<<endl;

   pointsToInt = &dogsAge;
   cout<<"pointsToInt dogsAge ga ko‘rsatkich"<<endl;
   cout<<"*pointstoInt = "<<*pointsToInt<<endl;
   return 0;
}
```

Yuqoridagi misoldagi ko‘rsatkich u ko‘rsatgan xotira maydonidan qiymatni o‘qish uchun ishlatilgan. Quyidagi dastur esa *pointsToInt operatoridan birinchi qiymat sifatida foydalanilganda nima sodir bo‘lishini ko‘rsatadi - ya’ni qiymatni o‘qish uchun emas, balki belgilash uchun.

```
#include <iostream>
using namespace std;
int main()
{
   int dogsAge = 30;
   cout<<"Boshlang‘ich qiymati dogsAge = "<< dogsAge<<endl;
   int* pAge = &dogsAge;
   cout<<"pAge DogsAgega ko‘rsatkich"<<endl;
   cout<<"dosAgening qiymatini kiriting: ";
   //pAgening manzili bo‘yicha xotira sohasiga qiymatni saqlash
   cin>>*pAge;
   //Manzilni kiritish
```

```
    cout<<"Qiymat quyidagi manzilda saqlanadi: "<<hex<<pAge<<endl;
    cout<<"Endi dogsAge = "<<dec<<dogsAge<<endl;
    return 0;
}
```

Ko‘rsatkich uchun sizeof() qiymati. Yuqoridagi fikrlardan bilganimizdek, ko‘rsatkich faqat xotira maydoni manzilini o‘z ichiga olgan o‘zgaruvchidir. Shuning uchun, qaysi turiga ishora qilmasin, ko‘rsatkichning tarkibi manzilning raqamli qiymatidir. Manzilning uzunligi - uni saqlash uchun zarur bo‘lgan baytlar soni; u ma’lum bir tizim uchun doimiydir. Shunday qilib, **sizeof()** ko‘rsatkichi bo‘yicha bajarilish natijasi dastur tuzilgan kompilyatorga va operatsion tizimga bog‘liq bo‘lib, quyidagi dasturda ko‘rsatilgandek u ko‘rsatadigan ma’lumotlarning tabiatiga bog‘liq emas.

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"tiplar uchun sizeof:"<<endl;
    cout<<"sizeof(char) = "<<sizeof(char)<<endl;
    cout<<"sizeof(int) = "<<sizeof(int)<<endl;
    cout<<"sizeof(double) = "<<sizeof(double)<<endl;

    cout<<"Ko‘rsatkichli tiplar uchun sizeof:"<<endl;
    cout<<"sizeof(char*) = "<<sizeof(char*)<<endl;
    cout<<"sizeof(int*) = "<<sizeof(int*)<<endl;
    cout<<"sizeof(double*) = "<<sizeof(double*)<<endl;
    return 0;
}
```

2. Xotirani dinamik ravishda taqsimlash

Quyidagicha ko‘rinishdagi statik massivlarni e’lon qilishda bizda muammolar paydo bo‘ladi:

```
int Numbers[100]; //100 ta butun son uchun statik massiv
```

1-muammo: Bu yerda dasturimizning imkoniyatlarini chegaralaymiz, chunki u 100 dan ortiq raqamni saqlay olmaydi.

2-muammo: Masalan, faqat 1 ta raqamni saqlash kerak bo‘lganda va 100 ta raqam uchun xotira ajratilganda resurslardan samarasiz foydalanyapmiz.

Ushbu muammolarning asosiy sababi kompilyator tomonidan massiv uchun statik bo‘lgan, doimiy xotirani ajratishdir.

Dastur foydalanuvchidan o‘ziga xos ehtiyojlariga qarab xotiradan maqbul foydalanishi uchun xotirani dinamik taqsimotidan foydalanish zarur. Bu sizga kerak bo‘lganda ko‘proq xotira ajratish va kerak bo‘lmaganda bo‘shatish imkonini beradi. C++ dasturida xotiradan foydalanishni boshqarish imkonini beradigan ikkita operator, **new** va **delete** mavjud. Xotira manzillarini saqlaydigan ko‘rsatkichlar xotirani samarali dinamik ravishda taqsimlashda hal qiluvchi rol o‘ynaydi.

Xotirani ajratish va bo‘shatish uchun new va delete operatorlaridan foydalanish. new operatori yangi xotira bloklarini ajratish uchun ishlatiladi. **new** operatorining eng ko‘p ishlatiladigan versiyasi, u so‘ralgan xotira maydoniga ko‘rsatkichni qaytaradi va aks holda istisno qiladi. new operatoridan foydalanishda siz xotira ajratiladigan ma’lumotlar turini ko‘rsatishingiz kerak:

tip* Ko‘rsatkich = new Tip; //Xotiraga bitta element uchun so‘rov

Shuningdek, siz xotirani ajratmoqchi bo‘lgan elementlar sonini belgilashingiz mumkin (agar siz elementlar massivi uchun xotira ajratishingiz kerak bo‘lsa):

tip* Ko‘rsatkich = new Tip [Miqdor] // Belgilangan elementlar soni uchun xotirani so‘rash

Shunday qilib, xotirada butun sonlarni joylashtirish kerak bo‘lsa, quyidagi koddan foydalanishimiz mumkin:

int* pointToAnInt = new int; //Butun songa ko‘rsatkich
int* pointToNums = new int[10]; //10 ta butun sondan iborat massivga ko‘rsatkich

new operatori bilan ajratilgan har bir xotira maydoni tegishli delete operatori tomonidan bo‘shatilishi kerak:

```
tip* Ko‘rsatkich = new Tip;
delete Ko‘rsatkich;
```

Bu bir nechta element uchun xotira ajratilganda ham shu usul yordamida o‘chirish mumkin:

```
tip* Ko‘rsatkich = new Tip[Miqdor];
delete p[] Ko‘rsatkich;
```

Agar siz ajratilgan xotirani tugatgandan so‘ng bo‘shatmasangiz, u *ajratilgan* bo‘lib qoladi va keyinchalik sizning yoki boshqa ilovalaringizga ajratish uchun mavjud bo‘lmaydi. Xotiraning bunday *sarflanishi* hatto dastur yoki umuman kompyuter ishini sekinlashtirishi mumkin va bunga har qanday holatda yo‘l qo‘ymaslik kerak.

Quyidagi dasturda xotirani dinamik ajratish va taqsimlash ko‘rsatilgan.

```
#include <iostream>
using namespace std;
int main()
{
  //int tipi uchun xotira ajratish
  int* pointsToAnAge = new int;

  //Ajratilgan xotiradan foydalanish
  cout<<"Yoshni kiriting: ";
  cin>> *pointsToAnAge;

  //* Ajratish operatorini qo‘llash
  cout<<"Yosh "<< *pointsToAnAge<<" " <<hex<<pointsToAnAge<<" adresida saqlanadi"<<endl;
  delete pointsToAnAge; //Xotirani bo‘shatish


   return 0;
}
```

E'tibor bering, new [] operatoridan foydalangan holda bir qator elementlar uchun xotirani ajratganda, quyidagi dasturda ko‘rsatilgandek, uni delete [] operatori yordamida bo‘shatish kerak.

```
#include <iostream>
using namespace std;
int main()
{
  cout<<"Massiv miqdorini kiriting?"<<endl;
  int numEntries = 0;
  cin>>numEntries;

  int* myNumbers = new int[numEntries];
```

```
  cout<<"Ajratilgan xotira manzili: "<<myNumbers<<hex<<endl;
  //Xotirani bo‘shatish

  delete[] myNumbers;
    return 0;
}
```

3. Ko‘rsatkichlarda const kalit so‘zidan foydalanish

Oldingi mavzularda o‘zgaruvchini const deb e'lon qilish, ishga tushirilgandan so‘ng o‘zgaruvchining qiymati o‘zgarmas turishini bilib olgandik. Bunday o‘zgaruvchining qiymatini o‘zgartirish mumkin emas.

Ko‘rsatkichlar ham o‘zgaruvchidir, shuning uchun const kalit so‘zi ham ularga mos keladi. Biroq, ko‘rsatkichlar - bu xotira maydonlarining manzillarini o‘z ichiga olgan va xotiradagi ma’lumotlarni o‘zgartirishga imkon beradigan o‘zgaruvchilarning maxsus turi.

Shunday qilib, ko‘rsatkichlar va doimiylar haqida gap ketganda, quyidagi kombinatsiyalar bo‘lishi mumkin.

1) Ko‘rsatkichda joylashgan manzil doimiy bo‘lib, uni o‘zgartirish mumkin emas, ammo u ko‘rsatgan ma’lumotlar o‘zgarishi mumkin:

```
  int daysM = 30;
  int* const pdaysM = &daysM;
  *pdaysM = 31;
  int daysMK = 30;
  pdaysM = &daysMK //Xatolik: adresni o‘zgartirish mumkin emas
```

2) Ko‘rsatkich ko‘rsatgan ma’lumotlar doimiy va ularni o‘zgartirish mumkin emas, lekin ko‘rsatkichda joylashgan manzilning o‘zi o‘zgarishi mumkin (ya’ni ko‘rsatkich boshqa joyga ishora qilishi mumkin):

```
int hoursInDay = 24;
  const int* pointsToInt= &hoursInDay;
  int monthsInYear = 12;
  pointsToInt = &monthsInYear; //OK!
  *pointsToInt = 13; //Kompilyatsiya vaqtida xatolik
                     //Ma’lumotni o‘zgartirish mumkin emas
  int* newPointer = pointsToInt; //Kompilyatsiya vaqtida xatolik:
                                 //doimiy ko‘rsatkichga
                                 //doimiy ko‘rsatkichni belgilay olmaysiz
```

3) Koʻrsatkichda joylashgan manzil ham, u koʻrsatadigan qiymat ham doimiy boʻlib, ularni oʻzgartirish mumkin emas (eng cheklovchi variant):

```
int hoursInDay = 24;
  const int* const pHoursInDay= &hoursInDay;
  *pHoursInDay = 25; //Kompilyatsiya xatoligi:
                     //ushbu koʻrsatkich tomonidan koʻrsatilgan
                     //qiymatni oʻzgartirish mumkin emas
  int daysInMonth = 30;
  pHoursInDay = &daysInMonth;
```

Ushbu turli xil konstruksiyalar, ayniqsa funksiyalarga koʻrsatkichlarni uzatishda foydalidir. Funksiya parametrlari funksiyani koʻrsatkich tomonidan koʻrsatilgan qiymatni oʻzgartira olmasligini ta’minlash uchun agar funksiyada bunday oʻzgartirish nazarda tutilmagan boʻlsa, eng cheklangan darajadagi barqarorlikni ta’minlash uchun e'lon qilinishi kerak. Bu dasturchining koʻrsatkichning qiymatini yoki u koʻrsatgan ma’lumotni notoʻgʻri oʻzgartirishiga yoʻl qoʻymaydi.

4. Ko’rsatkichlar va adres oluvchi o’zgaruvchilar funksiya parametri sifatida

Koʻrsatkichlar - bu qiymatlarni oʻz ichiga olgan va natijani oʻz ichiga oladigan xotiraning funksional sohalariga oʻtishning samarali vositasi. Funksiyalar bilan koʻrsatkichlardan foydalanishda, chaqirilgan funksiyani faqat siz oʻzgartirishni xohlagan parametrlarni oʻzgartirishga ruxsat berilishini ta’minlash kerak. Masalan, koʻrsatkich orqali oʻtgan radiusdan doiraning maydonini hisoblaydigan funksiyaga ushbu radiusni oʻzgartirishga yoʻl qoʻyib boʻlmaydi. Bunday holda, doimiy koʻrsatkichlar yordam beradi, bu qanday funksiyalarni oʻzgartirishga ruxsat berilishini va nimani oʻzgartirish mumkin emasligini samarali boshqarish imkonini beradi (pastdagi dasturga qarang).

```
#include <iostream>
using namespace std;

void CalcArea(const double* const pPi, const double* const pRadius, double* const pArea)
{
   //Ishlatishdan oldin koʻrsatkichlarning toʻgʻriligini tekshirish!
   if (pPi && pRadius && pArea)
```

```
    *pArea = (*pPi) * (*pRadius) * (*pRadius);
}

int main()
{
  const double Pi = 3.1416;
  cout<<"Aylananing radiusini kiriting: ";
  double radius = 0;
  cin>>radius;

  double area = 0;
  CalcArea (&Pi, &radius, &area);
  cout<<"Yuza:"<<area<<endl;
  return 0;
}
```

C++da ishlaydigan parametrlar koʻrsatkichlarni koʻrsatishi mumkin. Koʻrsatkichlar funksiyaga qiymat boʻyicha uzatiladi, ya'ni funksiya koʻrsatkichning nusxasini oladi. Shu bilan birga, koʻrsatkich nusxasi asl koʻrsatkich bilan bir xil manzilga ega boʻladi. Shuning uchun parametr sifatida parametrlardan foydalanib, biz argument qiymatiga kirishimiz va uni oʻzgartirishimiz mumkin.

Masalan, bizda sonni birga oshiradigan oddiy funksiya mavjud deylik:

```
#include <iostream>
using namespace std;
void increment(int x)
{
   x++;
   cout << "Funksiya natijasi x=: " <<  x << endl;
}

int main()
{
   int n = 10;
   increment(n);
   cout << "Bosh funksiyada n= " <<  n << endl;
   return 0;
}
```

Bu yerda n o‘zgaruvchisi x parametriga argument sifatida uzatiladi. U qiymat bilan uzatiladi, shuning uchun inkrement funksiyasidagi x parametrining har qanday o‘zgarishi n qiymatiga ta’sir qilmaydi. Dasturni ishga tushirsak:

Funksiya natijasi x= 11;
Bosh funksiyada n= 10;

Parametr sifatida ko‘rsatkichni ishlatish uchun inkrement funksiyasini o‘zgartiraylik:

```
#include <iostream>
using namespace std;
void increment(int *x)
{
   (*x)++;
   cout << "Increment funksiyasida x=: " <<  *x << endl;
}
int main()
{
   int n = 10;
   increment(&n);
   cout << "Bosh funksiyada n=: " <<  n << endl;
   return 0;
}
```

Parametr qiymatini o‘zgartirish uchun keyingi inkrement bilan ajratish jarayoni qo‘llaniladi: (*x)++. Bu x ko‘rsatkichida saqlangan manzildagi qiymatni o‘zgartiradi.

Endi funksiya parametr sifatida ko‘rsatkichni oladi, uni chaqirganda siz o‘zgaruvchining manzilini kiritishingiz kerak: inkrement (&n);.

Natijada x parametrining o‘zgarishi n o‘zgaruvchiga ham ta’sir qiladi.

Shu bilan birga, argument funksiyaga qiymat bo‘yicha uzatilganligi sababli, funksiya manzilning nusxasini oladi, agar funksiya ichida ko‘rsatkich manzili o‘zgartirilsa, bu tashqi ko‘rsatkichga ta’sir qilmaydi:

```
#include <iostream>
using namespace std;
void increment(int *x)
{
   int z = 6;
```

```
    x = &z;     // x ko‘rsatkichining manzilini qayta o‘rnating
    cout << "inkrement funksiyasida x= " <<  *x << endl;
}
int main()
{
    int n = 10;
    int *ptr = &n;
    increment(ptr);
    cout << "main funksiyasida: " <<  n << endl;
    return 0;
}
```

ptr ko‘rsatkichi inkrement funksiyasiga uzatiladi. Qachon chaqirilsa, funksiya bu ko‘rsatkichning x parametri sifatida nusxasini oladi. Funksiyada x ko‘rsatkichining manzili o‘zgartirilgan. Ammo bu ptr ko‘rsatkichiga hech qanday ta’sir qilmaydi, chunki u boshqa nusxani anglatadi. Natijada, manzilni tiklash maydoni, x va ptr ko‘rsatkichlari turli manzillarni saqlaydi.

Dastur natijasi:

Inkrement funksiyasida: x=6
Asosiy funksiyada: n=10