Obyektyönlü proqramlaşdırma

C++

Dərs №2

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

Yüklənmiş konstruktorlar

Əvvəlki dərsdə siz OYP əsasları ilə tanış oldunuz və sinif anlayışını öyrəndiniz. Bugün biz sizinlə bu gözəl tanışlığı davam etdirəcəyik. Biz artıq müəyyənləşdirdik ki, konstruktorlar parametrlərə malik ola bilər. Bunun üçün konstruktorun təyin və elan edilməsinə bu parametrləri əlavə etmək, sonra isə obyektin yaradılması zamanı onları arqument kimi vermək kifayətdir. İndi isə biliklərimizə daha birini əlavə edək – konstruktorlar bir neçə dənə ola bilər. Misala baxaq:

```
#include <iostream>

using namespace std;

class _3D
{
      double x, y,
      z; public:
      _3D ();
      _3D (double initX, double initY, double initZ);
};

//_3D sinfinin parametrli konstruktoru
_3D::_3D(double initX, double initY, double initZ)
{
      x = initX;
      y = initY;
      z = initZ;
      cout << "\nWith arguments!!!\n";
}
```

```
//_3D sinfinin parametrsiz konstruktoru
_3D::_3D()
{
      x=y=z=0;
      cout << "\nNo arguments!!!\n";
}

void main()
{
      //A obyekti yaradilir,
      //parametrsiz konstruktor çağırılır
      //sinfin butun üzvləri 0 olaraq qiymətləndirilir
      //ekranda "No arguments!!!" yazısı yazılır
      _3D A;

      //B obyekti yaradılır
      //parametrli konstruktor çağırılır
      //sinfin bütün üzvləri qiymətləndirilir
      //ekranda "With arguments!!!" yazisi yazılır
      _3D B (3,4,0);
}
```

Qeyd: Konstruktordan fərqli olaraq, destruktor parametrləri olmadığı üçün yüklənə bilməz. Bu tamamilə məntiqidir, belə ki, xarici obyektə parametrlərin ötürülməsi olmur.

Verilmiş misalın şərhi və istifadəsinin xüsusiyyətləri

1. Sinfin hər bir obyektinin elan edilməsi üsuluna sinfin öz konstruktor versiyasına uyğun gəlməlidir. Əgər bu təmin edilməzsə, onda komilyasiya mərhələsində səhv aşkar edilmiş olacaq.

2. Bu misalda nəyə görə konstruktorların yüklənməsinin çağırılmasını asanlıqla başa düşmək olur (xüsusilə də yüklənmədə, belə ki, burada eyni adlı müxtəlif parametrli siyahıları olan funksiyalardan söhbət gedir). Beləliklə, konstruktorların yüklənməsinin əsas mənası proqramçının obyekti daha uyğun üsulla qiymətləndirməsindən ibarətdir.
3. Misalda konstruktorların yüklənməsinə aid daha geniş yayılmış variant verilmişdir, yəni, parametrli konstruktorlar və parametrsiz konstruktorlar. Bir qayda olaraq, proqramda bu iki formanın hər ikisinə ehtiyac olur, belə ki, parametrli konstruktor tək obyektlə işləmək üçün daha rahatdır, lakin dinamik massivin obyekt-elementlerinin qiymətləndirilməsində istifadə edilə bilməz.
4. Buna baxmayaraq konstruktora istədiyiniz qədər yükləyə bilərsiniz, yaxşısı budur ki, bundan suiistifadə etməyəsiz.

Faydalı informasiyalar

Buna diqqət edin ki, layihələndiricilərin gövdəsi sinfin xaricində verilmişdir. Bu yazılış forması sinfin adi metodları üçün də istifadə oluna bilər. Xatırladırıq ki, əvvəlki dərsdəki misallarda metodların gövdələri birbaşa sinfin təyinində yazılmışdır.

Bu üsullardan hansı daha uyğundur, bunu soruşun?! Əvvəlki dərsdə verilmiş üsul adətən sonradan dəyişikliyə uğramayacaq sadə və qısa metodlar üçün istifadə edilir.

Bunu ona görə edirlər ki, sinfin təyinini adətən başlıq faylında saxlayırlar və bunu sonra tətbiqi proqrama #include direktivi vasitəsilə əlavə edirlər. Bundan başqa, bu üsulda bu funksiyalara müraciət edən zaman kompilyator tərəfindən hasil edilən maşın əmrləri translyasiya olunan mətnə avtomatik olaraq daxil edilir. Bu onların icrasına sərf edilən zaman sərfiyatını öz növbəsində istifadə edilən kodun ölçüsü hesabına azaldır, belə ki, bu cür metodların icra edilməsi funksiyanın çağırılmasından və qaytarma mexanizmindən asılı deyil (yəni, bu cür metodlar inline olur).

Yuxarıda verilən misalda istifadə edilən üsul mürəkkəb metodlar üçün məqsədəuyğundur. Bu şəkildə elan edilən funksiyalar kompilyator tərəfindən altproqramın çağırılması ilə əvəzlənir.

Obyektlərin göstəriciləri

İndiyə qədər obyektin üzvlərinə müraciət “.” Operatoru vasitəsilə olurdu. Bu, siz obyektlə işləyən zaman doğrudur. Lakin, obyektin üzvlərinə, həmçinin, obyektin göstəricisi vasitəsilə də müraciət etmək olar. Bu halda adətən "->" operatorundan istifadə edilir. Struktura oxşayır, elə deyilmi?

Obyektin göstəricisi istənilən tipli dəyişənin göstəricisi kimi elan edilir. Obyektin ünvanını almaq üçün ondan əvvəl & operatoru qoyulur.

```
#include <iostream>

using namespace std;

class _3D
{
     double x, y,
z; public:
     _3D ();
     _3D (double initX, double initY, double
     initZ); void Show(){
     cout<<x<<” “<<y<<z<<”\n”;
     }
};

//_3D sinfinin parametrli konstruktoru
_3D::_3D(double initX, double initY, double initZ)
{
     x = initX;
     y = initY;
     z = initZ;
     cout << “\nWith arguments!!!\n”;
}
```

```
//_3D sinfinin parametrsiz konstruktoru
_3D::_3D()
{
     x=y=z=0;
     cout << "\nNo arguments!!!\n";
}

void main()
{
     //A obyekti yaradılır
     //parametrli konstruktor çağırılır
     //sinfin bütün üzvləri qiymətləndirilir
     //ekranda "With arguments!!!" yazısı yazılır
     _3D A (3,4,0);

     //_3D tipində obyektin göstəricisi yaradılır
     //və bu göstəriciyə A obyektinin ünvanı yazılır
     _3D*PA=&A;

     //göstərici vasitəsilə funksiya çağırılır
     //Show()
     PA->Show();
}
```

Obyekt altında yaddaşın dinamik ayrılması

Əgər sinif arqumentsiz konstruktor ehtiva edirsə, onda `new` əməliyyatına müraciət qiyməti verilməmiş adi tipli verilənlər üçün yaddaşın ayrılmasında olduğu kimi olur.

```
#include <iostream>

using namespace std;
```



```
class Point
{
     double x,
y; public:
     Point(){
          x=y=0;
          cout << "\nNo arguments!!!\n";
     }
     void Show(){
          cout<<x<<" "<<y<<"\n";
     }
};

void main()
{
     //obyektin yaradılması
     Point A;
     //ekranda məzmunun göstərilməsi
     A.Show();

     cout<<"****************************";
     //obyektin göstəricisinin yaradılması
     Point*PA;

     //Point tipində bir obyekt üçün
     //yaddaşın dinamik ayrılması
     PA=new Point;

     //yaddaşın ayrılmasının yoxlanılması
     //və yaddaş ayrılmamışdırsa, çıxış
     if(!PA) exit(0);

     //göstərici vasitəsilə
     //Show() funksiyası çağırılır
     PA->Show();

     cout<<"****************************";
```

```
      //obyektin göstəricisinin yaradılması
      Point*PB;

      //massiv altında dinamik yaddaşın ayrılması
      //Point tipində obyekt
      PB=new Point[10];

     //yaddaşın ayrılmasının yoxlanılması
     //və yaddaş ayrılmamışdırsa, çıxış
      if(!PB) exit(0);

      //PB massivinin hər bir elementi üçün
      //Show() funksiyasının çağırılması
      for(int i=0;i<10;i++){
           PB[0].Show();
      }

      //PA obyektinin silinməsi
      PA delete PA;

      //PB massivinin silinməsi
      delete[]PB;
}
```

Əgər sinfin konstruktorunun arqumentləri olarsa, onda arqumentlərin siyahısı, standart verilənlər tipləri ilə iş zamanı qiymətləndirmə ifadəsinin olduğu yerdə yerləşdirilir.

```
#include <iostream>

using namespace std;

class Point
{
      double x, y;
```



```
public:
      //konstruktor susmaya görə
      //parametrləri ilə
      Point(double iX=1,double iY=1){
            x=iX;
            y=iY;
            cout << "\nWith arguments!!!\n";
      }
      void Show(){
            cout<<x<<" "<<y<<"\n";
      }
};
void main()
{
      //obyektin yaradılması
      Point A(2,3);
      //məzmunun ekranda göstərilməsi
      A.Show();
      cout<<"****************************";
      //obyektin göstəricisinin yaradılması
      Point*PA;

      //Point tipində bir obyekt altında
      //yaddaşın dinamik ayrılması
      //Point tipində obyekt
      //Dairəvi mötərizələrdə konstruktoru üçün
      //parametrlər
      PA=new Point(4,5);
      //yaddaşın ayrılmasının yoxlanılması
      //və yaddaş ayrılmamışdırsa, çıxış
      if(!PA) exit(0);

      //göstərici vasitəsilə
      //Show()funksiyası çağırılır
      PA->Show();

      cout<<"****************************";
```

```
      //obyektin göstəricisinin yaradılması
      Point*PB;

      //Point tipində massiv obyekt altında
      //yaddaşın dinamik ayrılması
      //parametrlər ötürülmür
      //konstruktorun parametrləri
      //susmayagörə istifadə
      edilir
      PB=new Point[10];

       //yaddaşın ayrılmasının yoxlanılması
       //və yaddaş ayrılmamışdırsa, çıxış
      if(!PB) exit(0);

      //PB massiviniin hər bir elementi üçün Show()
      //funksiyasının çağırılması
      for(int i=0;i<10;i++){
            PB[0].Show();
      }
      //Deleting the PA

      object delete PA;

      //Deleting the PB
      array delete[]PB;
}
```

Qeyd: Verilmiş misalda parametrləri susmaya görə olan konstruktordan istifadə edilmişdir. Bu onunla əlaqədardır ki, massiv obyekt altında yaddaşın dinamik ayrılması zamanı konstruktora parametrlərin ötürülməsinin mümkün olmadığına diqqət edin. Bizim misalda biz bunu etmirik. Massiv obyektlər üçün parametrlər susmaya görə istifadə edilirlər. Bu problemi parametrsiz konstruktor yaratmaqla başqa cür həll etmək olar.

Statik massivlər

Dinamik massivlərdən fərqli olaraq, statik massivin yaradılması zamanı konstruktora parametrlər ötürmək olar. Bu əməliyyatın sintaksisinə misalda baxaq:

```
#include <iostream>

using namespace std;

class Point
{
      double x,
y; public:
      //parametrli konstruktor
      Point(double iX,double iY){
            x=iX;
            y=iY;
            cout << "\nWith arguments!!!\n";
      }
      void Show(){
            cout<<x<<" "<<y<<"\n";
      }
};

void main()
{
      //massiv obyektin yaradılması
      //konstruktora parametrlərin ötürülməsi
      Point AR[2]={Point(2,3),Point(4,5)};

      //AR massivinin hər bir elementi üçün Show()
      //funksiyasının çağırılması
      for(int i=0;i<2;i++){
            AR[i].Show();
      }
}
```

this göstəricisi

Əvvəlki dərsdə biz müəyyən etdik ki, sinfin istənilən metodu sərbəst təyin edilir, hər bir obyekt üçün o çağırılmışdır və sinfin digər üzvlərini onlara parametrlər göndərmədən "görür". Sual verək: bu necə baş verir?!

Bu sualın cavabı sir deyil. İş ondadır ki, sinfə aid olan funksiya müəyyən obyektin emalı üçün çağırılarsa, bu funksiyaya avtomatik və qeyri-aşkar funksiyanın çağırıldığı obyektin göstəricisi göndərilir. Bu göstərici this sabit adında olur və sinfin hər bir funksiyası üçün proqramçıdan gizli təyin edilir.

Beləliklə, this haqqında bir neçə məlumat

1. this göstəricisi metodun çağırıldığı obyektin ünvanı ilə bu metodun kodunun icrasından əvvəl qiymətləndirilir.
2. this adı xidməti (açar) sözdür.
3. this göstəricisini aşkar yazmaq və ya təyin etmək olmaz.
4. Qeyri-aşkar təyinə uyğun olaraq this sabit göstəricidir, yəni, onu dəyişdirmək olmaz, lakin sinfə aid hər bir funksiya üçün o yalnız funksiyanın çağırıldığı obyektin göstəricisi olur.

5. this göstəricisi ilə ünvanlanan obyekt məhz this göstəricisi vasitəsilə sinfə məxsus funksiya daxilində əlyetərlidir.
6. Sinfin üzv funksiyası daxilində bu göstəricini aşkar istifadə etmək olar.

this göstəricisi çox faydalıdır, bəzən isə əvəzedilməzdir. Məsələ, növbəti kodda this göstəricisi sinfin üzvünün adı metoda daxil olan formal parametrin adı ilə üst-üstə düşərsə, kompilyatora bu vəziyyətdən çıxmağa imkan verir:

```
#include <iostream>
using namespace std;
class Student //tələbə sinfi
{
      char name[50]; //ad
      char surname[50]; //soyad
      int age; //yaş
public:
      //Konstruktor:
      Student(char name[],char surname[],int age)
      {
            //Üzvlər və eyniadlı parametrlər:
            strcpy(this->name,name);
            strcpy(this->surname,surname); this-
            >age=age;
      }
      void Show()
      {
            //burada this vacib deyil,
            //lakin onu istifadə etmək olar
            cout << "\nNAME - " << this->name;
            cout << "\nSURNAME - " << this->surname;

            cout << "\nAGE - " << this->age;
```

```
            cout << "\n\n";
      }
};

void main(void)
{
      Student A("Ivan","Sidoroff",25);
      A.Show();
}
```

Hazırda biz this konstruktoru ilə tanış olduq. Növbəti dərslərdə o daha geniş tətbiq ediləcək.

Köçürmə konstruktoru

"Sirli" konstruktoru müzakirə etməzdən əvvəl gəlin sadə həqiqətdən danışaq, Beləliklə...

Obyektin funksiyaya ötürülməsi

Sinfin obyektlərini digər tipli verilənlərin ötürülməsi arqument kimi funksiyaya ötürmək olar. Lakin, yadda saxlamaq lazımdır ki, C və C++ dillərində parametrlərin ötürülməsi metodu susmaya görə obyektlərin qiymətə görə ötrülməsidir. Bu o deməkdir ki, funksiya daxilində obyekt-arqumentin nüsxəsi yaradılır və bu nüsxə (obyektin özü yox) funksiyada istifadə edilir. Nəticə olaraq, funksiya daxilində obyektin nüsxəsinin dəyişmdirilməsi obyektin özünə təsir etmir.

Bu halda obyekt üzərində baş verən əməliyyatları xarakterizə edən bir neçə təsdiq:

Obyektin funksiyaya ötürülməsi zamanı yeni obyekt yaranır. Obyektin ötürüldüyü funksiyanın işi başa çatdıqdan sonra arqumentin nüsxəsi ləğv edilir.

Obyektin nüsxəsi ləğv edildiyi zaman nüsxənin destruktoru çağırılır, belə ki, bu nüsxə öz görünmə sahəsindən çıxır. Funksiya daxilində obyekt – ötürülən obyektin bitlərlə nüsxəsidir, bu isə o deməkdir ki, əgər obyekt, məsələn yaddaşın dinamik ayrılmış sahəsinin göstəricisini ehtiva edirsə, onda köçürmə zamanı elə o yaddaş sahəsinin göstəricisi olan obyekt yaradılır.

Beləliklə, yalnız nüsxənin destruktoru çağırılır. Burada bir qayda olaraq, obyekt-"əsli"n göstərdiyi yaddaşı azad etmək lazım gəlir, bu da cari obyektin ləğv edilməsinə səbəb olur.

```
#include <iostream>

using namespace std;
class ClassName
{
public:
     ClassName ()
     {
          cout << "ClassName!!!\n";
     }
     ~ClassName ()
     {
          cout << "~ClassName!!!\n";
     }
};

void f (ClassName o)
{
     cout << "Function f!!!\n";
}

void main()
{
     ClassName
     c1; f(c1);
}

Proqram icrasının nəticəsi:

ClassName!!!
Function f!!!
~ClassName!!!
~ClassName!!!
```


Konstruktor yalnız bir dəfə çağırır. Bu c1 yaradılarkən baş verir. Lakin destruktor iki dəfə işləyir: bir dəfə nüsxə üçün, növbəti dəfə c2 obyektinin özü üçün. Destruktorun iki dəfə çağırılması faktı problemin, məsələn obyekt üçün dinamik ayrılmış yaddaş sahəsinin destruktorun ləğv etməsi zamanı potensial problem mənbəyi ola bilər.

Funksiyadan obyektin qaytarılması

Oxşar problem obyektin qaytarılan qiymət kimi istifadəsi zamanı meydana çıxır.

Funksiyanın obyekt qaytara bilməsi üçün: birincisi, funksiyanı elə elan etmək lazımdır ki, onun qaytardığı qiymət sinif tipində olsun, ikincisi, obyekt adi return operatoru ilə qaytarılsın. Lakin əgər qaytarılan obyekt destruktor ehtiva edirsə, onda bu halda obyektin gözlənilməz ləğv edilməsi ilə bağlı problem ortaya çıxır.

```
#include <iostream>

using namespace std;
class ClassName
{
public: ClassName
      ()
      {
           cout << "ClassName!!!\n";
      }
      ~ClassName ()
      {
          cout << "~ClassName!!!\n";
      }
};
```

```
ClassName f()
{
      ClassName obj;
      cout << "Function
      f\n"; return obj;
}

void main()
{
      ClassName
      c1; f();
}

Proqram icrasının nəticəsi:

ClassName!!!
ClassName!!!
Function f
~ClassName!!!
~ClassName!!!
~ClassName!!!
```

Konstruktor iki dəfə çağırılır: c1 və obj üçün. Lakin destrtuktor burada iki dəfə çağırılır. Bu necə olur? Aydındır ki, destruktor c1-i, daha bir obj-ni ləğv edir. Destruktorun (ikinci dəfə) "lazımsız" çağırılması zamanı qaytarılan obyektin nüsxəsi olan müvəqqəti obyekt çağırılır. Funksiya obyekt qaytararkən, bu nüsxə formalaşır. Bundan sonra, funksiya qiymət qaytaran kimi müvəqqəti obyektin destruktoru icra olunur. Təbii olaraq, əgər destruktor, məsələn, dinamik ayrılmış yaddaşı azad edirsə, müvəqqəti obyektin ləğv edilməsi qaytarılan obyektin ləğv edilməsinə səbəb olur.

Obyektin yaradılması zamanı digəri ilə qiymətləndirilməsi

Proqramlaşdırmada bitlərlə köçürməyin daha bir halı var – bir obyektin yaradılarkən digəri ilə qiymətləndirilməsi:

```
#include <iostream>

using namespace std;

class ClassName
{
public:
      ClassName ()
      {
            cout << "ClassName!!!\n";
      }
      ~ClassName ()
      {
            cout << "~ClassName!!!\n";
      }
};

void main()
{
      ClassName c1;
      //O budur!!! Bitlərlə köçürmə anı.
      ClassName c2=c1;
}

Proqramın icrasının nəticəsi:

ClassName!!!
~ClassName!!!
~ClassName!!!
```

Konstruktor bir dəfə çağırılır: c1 üçün. c2 üçün konstruktor işləmir.

Lakin destruktor hər iki obyekt üçün işləyir. Belə ki, c2 c1-in dəqiq nüsxəsi olduğu üçün dinamik ayrılmış yaddaşı azad edir, bu yadaşın eyni fraqmenti üçün iki dəfə çağırılır.

Problemin həlli

Bu cür problemdən yan keçməyin üsullarından biri xüsusi tipli konstruktorun – köçürmə konstruktorunun yaradılmasıdır. Köçürmə konstruktoru və ya nüsxə konstruktoru obyektin nüsxəsinin yaradılması ardıcıllığını dəqəq təyin etməyə imkan verir. İstənilən köçürmə konstruktoru növbəti şəkildədir:

```
class_name (const class_name & obj)
{
... //konstruktorun gövdəsi
}
```

Burada obj – bu obyektə və ya obyektin ünvanına istinaddır. Köçürmə konstruktoru obyektin nüsxəsinin yaradılması zamanı hər dəfə çağırılır. Beləliklə, köçürmə konstruktorunda yeni obyekt altında öz yaddaşını ayırmaq olar.

```
#include <iostream>

using namespace std;

class ClassName
{
public:
     ClassName ()
     {
          cout << "ClassName!!!\n";
     }
     ClassName (ClassName&obj){
          cout << "Copy ClassName!!!\n";
     }
     ~ClassName ()
     {
          cout << "~ClassName!!!\n";
     }
};

void f(ClassName o){
     cout<<"Function f!!!\n";
}

ClassName r(){
     ClassName o;
     cout<<"Function
     r!!!\n"; return o;
}

void main()
{
     //bir obyektin digəri ilə qiymətləndirilməsi
     ClassName c1;
     ClassName c2=c1;

     //obyektin funksiyaya ötürülməsi
     ClassName

     a; f(a);

     //obyektin funksiyadan
     qaytarılması
     r();
}
Proqramın icrasının nəticəsi:
```

```
//c1 obyekti yaradıldı
ClassName!!!

//c2 obyektinin c1 obyekti ilə
//qiymətləndirilməsi
Copy ClassName!!!

//a obyektinin yaradılması
ClassName!!!

//a-nın funksiyaya qiymətcə ötürülməsi
//o nüsxəsinin yaradılması
Copy ClassName!!!

//f funksiyası işlədi
Function f!!!

//o-nun nüsxəsi ləğv edildi
~ClassName!!!

//r funksiyasının daxilində
//o obyekti yaradıldı
ClassName!!!

//r funksiyası işlədi
Function r!!!

//funksiyadan qayıtma
//o obyektinin nüsxəsi yaradıldı
Copy ClassName!!!

//o obyekti ləğv edildi
~ClassName!!!

//o-nun nüsxəsi ləğv edildi
~ClassName!!!
//a obyekti ləğv edildi
~ClassName!!!
```



```
//c2 obyekti ləğv edildi
~ClassName!!!

//c1 obyekti ləğv edildi
~ClassName!!!
```

Qeyd: Köçürmə konstruktoru A=B şəklində mənimsətmə operatoruna təsir etmir. Burada, həmçinin bitlərlə köçürmə baş verir, lakin bu problemi C++-da başqa cür həll edirlər.

İndi, köçürmə konstruktoru var ikən, obyektləri funksiyaya parametr kimi göndərmək və qaytarmaq olar. Bu zaman konstruktorların çağırılması sayı destruktorların çağırılması sayı ilə eyni olacaq, belə ki, nüsxənin yaradılması prosesi indi nəzarətoluna biləndir, obyektin gözlənilməz ləğv edilməsi ehtimalı nəzərəçarpacaq dərəcədə azalmşdır.

Qeyd: Lakin!!! Köçürmə konstruktorunun yaradılmasından başqa obyekt ötürən proqram və funksiya arasında qarşılıqlı əlaqənin təşkilinin başqa üsulu var. Bu üsul – obyektin istinada görə və ya göstəriciyə görə otürülməsidir.

Ev tapşırığı

1. Saaxlamaq üçün növbəti üzvləri olan Person sinfini yaratmalı:
   - adlar,
   - yaşı,
   - cinsi və
   - telefon nömrəsi.

   Bu üzv verilənləri fərdi olaraq dəyişə bilən üzv-funksiyalar tərtib edin. İnsan haqqında formatlı verilənləri daxil edən `Person::Print()` üzv funksiyasını yaradın.
2. Sətirlərlə işləmək üçün istifadə ediləcək String sinfini tərtib edin. Sinfə daxil olmalıdır:
   - uzunluğu 80 simvol olan sətir yarada bilən susmaya görə konstruktor;
   - ixtiyarı uzunluqda sətir yaratmağa imkan verən konstruktor;
   - istifadəçinin daxil etdiyi sətirlə qiymətləndirilən sətir yaradan konstruktor.

   Klass sətirlərin klaviaturadan daxil edilməsi və ekrana verilməsi metodları ehtiva etməlidir.