Obyektyönlü proqramlaşdırma

C++

ШАГ
компьютерная
АКАДЕМИЯ

Dərs №13

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

Firtual funksiyalar 3
Virtual funksiyaların tətbiqinin bəzi xüsusiyyətlər 9
Virtual funksiyalar cədvəli 11
Erkən və gec əlaqələndirmə
Statik və dinamik polumorfizm 12
Abstrakt siniflər 14
Tam virtual funksiyalar 14
Nümunə 16
Virtual baza sinfi 21
Nəticə 22
Virtual destruktor 24
Təmiz virtual destruktor 28
Bəzi məsləhətlər 29
Ev tapşırığı 30

Virtual funksiyalar

Obyektyönlü proqramlaşdırmada virtual funksiya - funksiyanın çağırılması üçün konkret reallaşdırılması icra zamanı təyin ediləcək varis-sinifdə təyin edilən sinfin üzv-funksiyasıdır. Beləliklə, proqramçının obyektin firtual funksiya vasitəsilə işlənməsi üçün onun dəqiq tipini bilməsinə ehtiyac yoxdur: funksiyanın elan edildiyi sinfin varisinə aid olduğunu bilmək kifayətdir.

Virtual funksiyalar – polumorfizmin reallaşdırılması üçün mühüm üsullardan biridir. Onlar baza sinfinin obyektləri ilə, həmçinin onun istənilən varis-sinfinin obyektləri ilə işləyən kod yaratmağa imkan verir. Bu zaman baza sinfi obyektlərlə iş üsulunu təyin edir və onun istənilən varisi bu üsulun konkret reallaşdırılmasını təmsil edə bilər.

Başqa sözlə - virtual funksiyalar vasitəsi ilə obyekt özü öz davranışını (məxsusi fəaliyyətlərini) təyin edir. Obyekt sizin proqramınızda, həqiqətdə yalnız bir sinfi deyil, həmçinin əgər onlar ümumi baza sinfi vasitəsilə varislik mexanizmi ilə əlaqələndirilmiş olarsa, çox sayda müxtəlif sinifləri də təmsil edə bilər və bu sinfin obyektlərinin ierarxiyada davranışları, təbii ki, müxtəlif ola bilər.

İndi isə, həqiqət anı: C++ qaydalarına görə, baza sinfinin göstəricisi bu sinfin obyektinə, həmçinin baza sinfindən törəmiş istənilən obyektə istinad edə bilər.

Bu qaydanı anlamaq çox vacibdir. Gəlin hər hansı A, B, C siniflərinin sadə ierarxiyasına baxaq. A bizdə baza sinfi olacaq, B – A sinfindən, C isə B sinfindən törəyir.

Proqramda bu sinfin obyektləri, məsələn, növbəti şəkildə elan edilə bilər:

```
A object_A; //A obyektinin elan edilməsi
B object_B; //B obyektinin elan edilməsi
C object_C; //C obyektinin elan edilməsi
```

Bu qaydaya görə A tipində göstərici bu üç obyektin hər birinə müraciət edə bilər. Yəni, növbəti yazılış forması mümkündür:

```
A *point_to_Object;         //baza sinfinin göstəricisini
                            //elan edirik
point_to_Object=&object_C; //Göstəriciyə C obyektinin
                            //ünvanını mənimsədirik
point_to_Object=&object_B; //Göstəriciyə B obyektinin
                            //ünvanını mənimsədirik
```

point_to_Object göstəricisinin A* tipində, C* (və ya B* deyil) olmasına baxmayaraq, C (və ya B) tipində obyektə istinad edə bilər. İndi isə səhv yazılış formasına baxaq:

```
//Törəmə sinfin göstəricisini elan edirik
B *point_to_Object;

//!DİQQƏT! göstəriciyə baza sinfinin ünvanını
//mənimsətmək olmaz
point_to_Object=&object_A;
```


Qeyd: Əgər siz C obyektini A obyektinin xüsusi forması kimi düşünərsinizsə, qayda daha aydın ola bilər. Məsələn, pinqvin quşların xüsusi tipidir və o baxmayaraq ki, uçmur, yenə də quş olaraq qalır. Əlbətdə ki, bu obyektlərin göstəricilərinin qarşılıqlı əlaqəsi yalnız bir istiqamətdə olur. C tipli obyekt A obyektinin xüsusi formasıdır, A obyekti isə C obyektinin xüsusi forması deyil. Pinqvinlər nümunəsinə qayıdaraq cəsarətlə deyə bilərik ki, əgər bütün quşlar pinqvinin xüsusi tipi olsa idi, onlar sadəcə uça bilməyəcəkdilər!

Bu prinsip xüsusi ilə varisliliklə bağlı olan virtual funksiyaların təyin edildiyi siniflərdə çox vacibdir.

Virtual tamamilə bu cür olur və sadə funksiyalar kimi proqramlaşdırılırlar. Yalnız onların elan edilməsi **virtual** açar sözü vasitəsi ilə olur. Məsələn, bizim A baza sinfimiz **v_function()** virtual funksiyasını elan edə bilər.

```
class A
{
    public:
    virtual void v_function(); //funksiya A sinfinin
                               //hansısa davranışını
                               //təsvir edir
};
```

Virtual funksiya parametrləri ilə elan edilə bilər, o digər funksiyalar kimi qiymət qaytara bilər. Sinif daxilində sizə lazım olan qədər virtual funksiyalar elan oluna bilər.

Və onlar sinfin istənilən hissəsində ola bilər: qapalı, açıq və ya qorunmuş hissəsində. Əgər A sinfindən törəmiş B sinfində hansısa başqa bir davranış vermək lazımdırsa, onda yenə də v_function() adlanan hansısa başqa bir davranış təsvir etmək lazımdır.

```
class B: public A
{
    public:

    //əvəz olunan funksiya
    //B sinfinin hansısa yeni
    ...//davranışını təsvir edir;

};
```

B sinfi kimi sinifdə əjdad-funksiyada olan virtual funksiya ilə eyni ada malik virtual funksiya olarsa, belə funksiya əvazolunan funksiya adlanır. B sinfindəki v_function() virtual funksiyası A sinfindəki eyni adlı virtual funksiyanı əvəz edir.

B* tipində object_B obyektinə istinad edən A* tpində point_to_Object göstəricisinə qayıdaq. Gəlin point_to_Object-in göstərdiyi v_function() virtual funksiyasını çağıran operatora baxaq.

```
A *point_to_Object;//baza sinfinin göstəricisini elan edək
point_to_Object=&object_B; //göstəriciyə B obyektinin
                           //ünvanını mənimsədirik
point_to_Object->v_function(); //funksiyanı çağırırıq
```

point_to_Object göstəricisi A və ya B tipində obyektin ünvanını saxlaya bilər. Deməli icra zamanı bu operator point_to_ Object- gt; v_function();

Hazırda obyektinə istinad edilən sinfin virtual funksiyası çağırılır. Əgər point_to_Object A tipli obyektə istinad edirsə, A sinfinə məxsus funksiya çağırılmış olur. Əgər point_to_Object B tipli obyektə istinad edirsə, B sinfinə məxsus funksiya çağırılmış olur. Məhz bu proqramın icrası zamanı təyin edilən fəaliyyətdir, Başqa sözlə, polimorvizmin reallaşdırılmasıdır.

Qeyd. Bir dəqiqə hesab edək ki, biz kompüter oyunu yazmaq istəyirik. Hesab edək ki, bu silah istifadə olunduğu oyundur. Təbii ki, onu oyunda yaratmaq lazımdır. Belə ki, bizim oyunumuzda müxtəlif tip silahlar olacaq. Buna görə də, baza sinfinin əlavə edilməsi bunun tamamilə məntiqi həlli olacaq. Belə deyək:

```
class Weapon
{
    public:

    ...//burada təsvir edilə bilən üzv-verilənlər
    //olacaq, məsələn, dəyənəyin qalınlığı,
    //qranatatanda olan qranatların sayı kimi
    //bu hissə bizim üçün vacib deyildir

    virtual void Use1(void);//adətən  mausun sol düyməsi

    virtual void Use2(void);//adətən  mausun sağ düyməsi

    ...//burada bəzi üzv-verilənlər və funksiyalar
    olacaq

};
```

Bu sinfin təfərrüatına girmədən demək olar ki, əsas vacib olan bu silahın davranışını (və ya tətbiqini) təsvir edən Use1() və Use2() funksiyalarıdır. Bu sinifdən istənilən tip silahı törətmək olar. Yeni üzv-verilənlər və (güllərin sayı, atma sürəti, enerji səviyyəsi, kəsicinin uzunluğu və s. kimi) yeni funksiyalar əlavə ediləcək.

Use1() və Use2() funksiyalarını təyin edərkən biz silahların (bıçaq üçün bu zərbə ola bilər, avtomat üçün isə tək və ardıcıl atəş açma imkanı ola bilər) istifadə müxtəlifliyini təsvir edəcəyik. Silah kolleksiyasını haradasa saxlamaq lazımdır. Göründüyü kimi, bunun üçün ən asan yol Weapon* tipində göstəricilər massivi təşkil etməkdir. Sadə olması üçün hesab edək ki, bu 10 tip silah və əvvəlcədən sıfır qiyməti almış bütün göstəriciləri ehtiva edən Arms qlobal massividir.

```
Weapon *Arms[10]; // Weapon tipində obyektlərin
                  //göstəriciləri massivi
```

Proqramın əvvəlində silah tipində dinamik obyektlər yaradaraq onların göstəricisini massivə əlavə edəcəyik. Hansı silahın istifadə edildiyini göstərmək üçün, qiymətini seçilmiş silahın tipindən asılı olaraq dəyişəcəyimiz dəyişən-massiv indeksini təyin edək.

```
int TypeOfWeapon;
```

Bu cəhdlərin nəticəsində oyunda silahın tətbiqini təsvir edək, məsələn, növbəti şəkildə olacaq:


```
if("mausun sol düyməsi basılmışdr") Arms[TypeOfWeapon]-
>Use1(); else Arms[TypeOfWeapon] ->Use2();
```

Bu da hamısı. Biz silahın hansı tipinin istifadə ediləcəyinə qərar verməmişdən əvvəl kod yaratmışıq. Bundan başqa, Bizdə ümumiyyətlə, hələ heç bir real silah tipi yoxdur! Əlavə (bəzən də çox vacib) mənfəət – bu kodu ayrıca kompilyasiya etmək və kitabxanada saxlamaqdır. Sonra da siz (və ya başqa bir proqramçı) Weapon-dan yeni siniflər xaric edə bilər, onları Arms[] massivində saxlaya və istifadə edə bilər. Bu zaman sizin kodu yenidən kompilyasiya etməyinizə ehtiyac yoxdur. Xüsusilə diqqət edin ki, bu kod sizdən Arms[] göstəricilərinin istinad etdiyi obyektlərin verilənlər tiplərini dəqiq vermənizi tələb etmir. Obyektlər icra zamanı onların hansı Use() funksiyasını çağıracağını təyin edirlər.

Tətbiqin bəzi xüsusiyyətləri

İndi isə gəlin əvvələ - A, B, C siniflərinə qayıdaq.

C sinfi hazırda bizdə ierarxiyanın ən aşağısında, varislilik xəttinin sonunda yerləşir. C sinfində eynilə belə əvəzləyici virtual funksiya yerləşdirmək olar. Bundan başqa virtual açar sözü tətbiq etmək vacib deyil, belə ki, bu varislilik xəttində sonuncu sinifdir.

Funksiya elə beləcə də işləyəcək və virtual seçiləcək. Əgər sizə C sinfindən D sinfini çıxartmaq və hətta v_function() fuksiyasının davranışını dəyişdirmək lazım gələrsə, bu baş tutmayacaq.

Bunun üçün C sinfində v_function() funksiyası virtual elan edilməlidir. Burdan növbəti qayda alınır:

virtual açar sözünü yaxşı olar ki, atmayaq, lazım gələ bilər?

Törəmə funksiyada funksiyanı baza sinfinin virtual funksiyasındakı eyni ad və eyni parametrlər dəsti ilə təyin etmək olmaz, lakin fərqli verilənlər tipi ilə olar. Əks halda proqramın kompilyasiya mərhələsində səhv baş verəcək.

Əgər törəmə sinifdə baza sinfinin virtual funksiyası ilə eyni adda və qaytarma tipində, lakin başqa parametrlər dəsti olan funksiya daxil edəriksə, onda törəmə sinfinin bu funksiyası artıq virtual olmayacaqdır. Hətta, əgər siz onu virtual açar sözü ilə müşayiət etsəniz də, o belə olmayacaq. Bu halda baza sinfinin göstəricisi vasitəsi ilə bu göstəricinin istənilən qiymətində baza sinfinin funksiyasına müraciət ediləcək. Funksiyaya yükləmə qaydasını xatırlayın! Bu sadəcə fərqli funksiyalardır. Sizdə tamamilə başqa virtual funksiya alınacaq. Burdan daha bir qayda çıxır:

Virtual funksiyaların əvəzlənməsi zamanı baza və törəmə siniflərdə parametrlərin tiplərinin, funksiayalrın adlarının və qaytarılan qiymətlərin tipləri eyni olmalıdır.

Hekayəmizin sonunda əlavə edək ki, firtual funksiya yalnız sinfin statik olmayan üzv-funksiyası ola bilər. Qlobal funksiya virtual ola bilməz. Virtual funksiya başqa bir sinifdə dost (friend) elan edilə bilər.

Virtual funksiyalar cədvəli

Heç olmazsa bir virtual funksiyası olan hər bir sinif üçün virtual funksiyalar cədvəli tərtib edilir. Hər bir obyektdə öz sinfinin firtual funksiyalar cədvəlinin göstəricisi saxlanılır. Virtual funksiyaların çağırılması zamanı növbəti mexanizmdən istifadə edilir: obyektdən uyğun cədvəlin göstəricisi götürülür, ondan isə sabit yerdəyişməyə - verilmiş sinif üçün funksiyanın reallaşdırılmasının göstəricisinə. Mürəkkəb varislilikdən istifadə zamanı virtual funksiyalar cədvəli qeyr-xətti olduğu üçün vəziyyət bir qədər çətinləşir.

Erkən və gec əlaqələndirmə. Statik və dinamik polimorfizm

Biz yenicə virtual funksiyalarla tanış olduq, indi isə biz erkən və gec əlaqələndirmə anlayışı ilə tanış olmalıyıq. Başlayaq.

Bir kilo portağalın alınması nümunəsində iki yanaşmaya baxaq. Birinci halda əvvəlcədən bilirik ki, biz bir kiloqram portağal almalıyıq. Buna görə biz bu kiloqramın kifayət etməsi üçün qiyməti dəqiq təyin edilmiş daha böyük paket götürürük. İkinci halda, evdən çixarkən bizim nəqədər alacağımızı əvvəlcədən bilmirik. Buna görə də biz avtomobili götürürük (ola bilər çəki çox olsun) ehtiyyat üçün böyük və kiçik ölçülü paketlər və daha çox pul götürürük. Bazara gedirik və məlum olur ki, yalnız bir kiloqram portağal almaq lazımdır.

Verilmiş nümunə müəyyən dərəcədə uyğun olaraq erkən və gec əlaqələndirmənin mənasını əks etdirir. Aydındır ki, verilmiş nümunə üçün birinci variant optimaldır. İkinci halda isə biz daha çox şeyi diqqətə almışıq, lakin bu bizə lazım deyil. Digər yandan, bazara gedərkən yolda qərara gəlsək ki, bizə portağal lazım deyil və 10 kq alma alacağıq, onda birinci halda biz bunu edə bilməyəcəyik. İkinci halda isə bu asandır.

İndi isə, bu nümunəyə proqramlaşdırma nöqteyi-nəzərindən baxmaq lazımdır. Erkən əlaqələndirmənin tətbiqi zamanı biz sanki kompilyatora deyirik ki: "Mən nə istədiyimi dəqiq bilirəm. Buna görə də butun funksiyaların çağırılmasını statik əlaqələndir. Gec əlaəqləndirmə mexanizminin tətbiqi zamanı biz kompilyatora deyirik ki: "Mən hələ nə istədiyimi bimirəm. Vaxtı gələndə mən nə istədiyimi bildirəcəm".

Beləliklə, erkən əlaqələndirmə zamanı çağırılan və çağıran metodlar ilk uyğun halda, adətən kompilyasiya zamanı əlaqəlndirilirlər.

Çağırılan və çağıran metodların gec əlaqələndirmə zamanı onlar kompilyasiya zamanı əlaqələndirilə bilməzlər. Buna görə də çağırışın faktik baş verməsi halında çağıran və çağırılan metodların necə əlaqələndiriləcəklərini təyin edən xüsusi mexanizm reallaşdırılmışdır. Məhz bu mexanizm virtual funksiyaları reallaşdırır.

Aydındır ki, erkən əlaqələndirmə zamanı surət və səmərəlilik gec əlaqələndirmənin istifadəsinə nisbətən daha yüksəkdir. Eyni zamanda, gec əlaqələndirmə əlaqəliliyin bəzi universallılığını təmin edir.

Nəticə olaraq, obyektyönlü proqramlaşdırmanın reallaşdırılması əlaqələndirmə olan xüsusiyyətini dəqiq formalaşdıraq:

Polimorfizm — baza sinfinin üzv-funksiyasının varisliliyinin təyin edilməsidir. Polimorfizm çağırılan funksiya icra zamanı təyin edildiyində dinamikdir (erkən əlaqələndirmə) və statikdir (gec əlaqələndirmə).

Abstrakt (mücərrəd) siniflər

Gəlin virtual funksiyaların istifadəsinə baxılmasına davam edək. Lakin başlanğıc üçün bir qədər nəzəriyyə.

Təmiz virtual funksiyalar

"Təmiz" sözü bu halda "boş" mənasında istifadə edilir. Başqa sözlə, təmiz virtual funksiya – boş funksiyadır. Onun yaradılma sintaksisi növbəti şəkildədir:

```
class A { public: //təmiz virtual funksiya
virtual void v_function()=0; };
```

Göründüyü kimi, bütün fərq ondan ibarətdir ki, «təmiz spesifikator» adlanan «=0» konstruksiyası yaranmışdır. Təmiz virtual funksiya tamamilə heç bir iş görmür və çağırılma üçün əlyetərsizdir. Onun təyinatı törəmə sinifdə əvəzlənən funksiya üçün əsasdır (istəsəz, şablondur).

Heç olmazsa, bir təmiz virtual funksiya ehtiva edən sinif abstrakt sinif adlanır. Bu onunla əlaqədardır ki, belə sinfin sərbəst obyektini yaratmaq olmaz. Bu yalnız başqa siniflər üçün hazırlıqdır. Abstrakt siniflər mexanizmi gələcəkdə dəqiqləşdirilməsi ehtimal edilən ümumi anlayışların təqdim edilməsi üçün işlənib hazırlanmışdır. Bu ümumi anlayışları adətən birbaşa istifadə etmək mümkün deyildir, lakin onların əsasında mümkündür: bazada olduğu kimi, konkret obyektlərin təsviri üçün törəmə özəl siniflər yaratmaq.

Qeyd: Nümunə göstərək. Bütün heyvanların öz davranışlarında «yemək», «içmək», «yatmaq», «səs çıxartmaq» kimi funksiyaları var. Bütün bu funksiyaları elan etmək və onları təmiz virtual etmək üçün baza sinfi təyin etmək olar. Sonra isə bu sinifdən konkret heyvanları onların xüsusi davranışları ilə təsvir edən siniflər çıxartmaq. Baza sinfi isə bu zaman həqiqətən də abstrakt olacaq. Belə ki, o az-çox konkret heyvanı təsvir etmir (hətta heyvan tipini də). Bu balıq və ya quş da ola bilər.

Adi siniflərlə müqayisədə, məhdud hüquqlardan istifadə edirlər.

- Hər hansı bir sinif kimi, aşkar təyin edilmiş konstruksiyaya malik ola bilər. Konstruktordan sinfin metodunu çağırmaq olar. Lakin konstruktordan təmiz virtual funksiyaya müraciət etmək üçün proqramın icrasi zamanı səhv baş verəcək.
- Artıq qeyd edildiyi kimi, abstrakt sinfin obyektini yaratmaq mümkün deyil.
- Abstrakt sinfi funksiyanın parametrinin tipinin verilməsi və ya qaytarılan qiymətin tipi kimi tətbiq etmək olmaz.
- Onu tiplərin aşkar çevrilməsi zamanı istifadə etmək olmaz. Buna baxmayaraq abstrakt siniflərin istinad və göstəricisini təyin etmək olar.

Nümunə

Hansısa heyavanları təsvir edən siniflər ierarxiyasına aid nümunəyə baxaq. Nümunəni sadələşdirmək üçün hər bir heyvanın təsvirində onun adı və çıxardığı səs ilə kifayətlənək. Proqramın əsas imkanı kimi heyvanların adlarının onların çıxardıqları səslərin siyahısını ekrana çıxartmaq olacaq.

```
#include <iostream>
#include <string.h>
using namespace std;

//abstrakt baza sinfi
class Animal
{
    public:
        //heyvanın adı
        char Title[20];
        //sadə konstruktor
        Animal(char *t){
            strcpy(Title,t);
        }
        //təmiz virtual funksiya
        virtual void speak()=0;
};

//qurbağa sinfi
class Frog: public Animal
{
    public:
        Frog(char *Title): Animal(Title){};
        virtual void speak(){
            cout<<Title<<" say "<<"\'kwa-kwa\'\n";
        }
};
```



```
//it sinfi
class Dog: public Animal
{
    public:
        Dog(char *Title): Animal(Title){};
        virtual void speak(){
            cout<<Title<<" say "<<"\'gav-gav\'\n";
        }
};

//pişik sinfi
class Cat: public Animal
{
    public:
        Cat(char *Title): Animal(Title){};
        virtual void speak(){
            cout<<Title<<" say "<<"\'myau-myau\'\n";
        }
};

//aslan sinfi
class Lion: public Cat
{
    public:
        Lion(char *Title): Cat(Title) {};
        /*virtual void speak(){
            cout<<Title<<" say "<<"\'rrr-rrr\'\n";
        }*/

        /*virtual int speak(){
            cout<<Title<<" say "<<"\'rrr-
            rrr\'\n"; return 0;
        }*/

        virtual void speak(int When){ cout<<Title<<"
            say "<<"\'rrr-rrr\'\n";
        }
};
```

```
void main ()
{
    //Animal baza sinfi göstəricilər massivinin
    //tutumu
    //onu heyvanlar siyahısı obyektini yaradaraq
    //doldururuq
    Animal *animals[4] = {new Dog("Bob"),
                          new Cat("Murka"),
                          new Frog("Vasya"),
                          new Lion("King")};
    for(int k=0; k<4; k++)
        animals[k]->speak();
}
```

Baza sinfi kimi Animal abstrakt sinfi yaradılmışdır. O heyvanların adını təsvir edən Title yeganə üzv ehtiva edir. Onda heyvana onun «adı»nı mənimsədən xüsusi konstruktor və heyvanın hansı səsləri çıxartdığını təsvir edən speak() adında yeganə təmiz virtual funksiya var.

Bu sinifdən bütün digər siniflər törənir. Yalnız birindən başqa. «Aslan» sinfi «pişik» sinfindən törəyib (aslanlar da pişik kimilərdir). Bu virtual siniflərin nümayiş etdirilməsinin incəlikləri üçün edilmişdir. Lakin bu sinif haqqında bir qədər sonra.

Bütün törəmə siniflərdə ekranda konkret heyvanın çıxardığı səsi əks etdirən speak() əvəzləyici virtual funksiya təsvir edilmişdir.

Proqramın əsas gövdəsində Animal* göstərici tipində animals[4] massivi elan edilmişdir və siniflərin dinamik obyektləri yaradılmış və göstəricilər massivi doldurulmuşdur.

for() dövründə isə göstəriciyə görə speak() virtual funksiyası çağırılır.

Proqramın icrasının nəticəsi növbəti şəkildədir:

```
Bob say 'gav-gav'
Murka say 'myau-myau'
Vasya say 'kwa-kwa'
King say 'rrr-rrr'
```

İndi isə, Lion (aslan) sinfinin təsvirinə diqqət edək. Onda bir speak() virtual funksiyasının yerinə eyni zamanda üçü var. Onlardan ikisi şərh edilmişdir.

Əgər siz birinici funksiyanı şərh edərsinizsə, ikincisinə isə yenidən baxarsınızsa, onda virtual əvəzləyici funksiyanı başqa tipli qaytarılan qiymətlə silahlandırmağa cəhdin olduğu variantı yoxlaya bilərsiniz. Bu halda ikinci (düzgün olmayan) funksiya int tipi yerinə baza sinfinin speak() funksiyasında olduğu kimi void tipini qaytarır. Proqramı kompilyasiya etməyə çalışın və kompilyasiya mərhələsində səhv baş verəcək.

İndi isə, üçüncü funksiyanı şərh etməyə çalışın, ilk ikisini isə yenidən nəzərdən keçirin. Kompilyator bu dəfə yalnız xəbərdarlıq ismarışı verəcək. Bu o haldır ki, əvəzləyici virtual funksiya qaytarılan qiymətin tipi, lakin başqa parametrlər dəsti ilə elan edilir. Baxaq, nə alındı:

```
Bob say 'gav-gav'
Murka say 'myau-myau'
Vasya say 'kwa-kwa'
King say 'myau-myau'
```

Aslan bizdə artıq nərildəmir, miyovlayır. Bu ona görədir ki, artıq tamamilə başqa funksiya işləyir. Yəni, verilmiş sinifdə düzgün təyin edilmiş funksiya olmadığı üçün göstəriciyə görə baza sinfindən speak() virtual funksiya çağırılır. Bizim halda isə Lion sinfi üçün baza sinfi Cat sinfidir. Buna görə də Aslan miyovlayır.

Virtual baza sinfi

Bəzən mürəkkəb varislikdə baza siniflərinin necə miras buraxdıqlarına nəzarət etmək halları yaranır. Nümunəyə baxaq.

```
 class A {

    public:
    int val;
};

class B : public A {...};
class C : public A {...};
class D : public B, public C{

    public:
    int Get_Val(){
        return val; //səhv!
    }
};
```

Yuxarıda göstərilmiş nümunədə val üzvünə müraciət birmənalı deyil. Kompilyasiya val-in hansı nüsxəsinə istinad etməyi anlamaz və səhv haqqında xəbərdarlıq edir. Çoxmənalılğı aradan qaldırmaq üçün ya görünməyə icazə verən operatordan istifadə etmək, məsələn növbəti şəkildə:

```
int Get_Val(){

    return B::val;

 }
```

ya da virtual baza sinfindən istifadə etmək lazımdır.
Nümunə əsasında bunu necə etmək lazım gəldiyini gözdən keçirək. İerarxiya ağacını növbəti şəkildə təyin edək:

```
class A {
    public:
        int val;
};

class B : public virtual A {...};

class C : public virtual A {...};

class D : public B, public C {

    public:
        int Get_Val() {
            return val; //hamısı düzgün işləyir
        }
};
```

Baza sinfinin virtual elan edilməsi kompilyatoru törəmə sinfinin elan edilməsində baza sinfinin yalnız bir nüsxəsini tətbiq etməyə məcbur edir. Buna görə də D sinfində val üzvünün yalnız bir nüsxəsi olur və görünmə oblastına icazə operatoru dəqiqləşdirmək üçün tələb olunmur. Virtual baza sinifləri yalnız mürəkkəb varislilik zamanı istifadə edilir.

Nəticə

Beləliklə, virtual baza sinfi, törəmə sinif iki və daha çox sinfi törətdiyində və onların da hər biri eyni bir baza sinfindən miras aldığında lazım olur.

Virtual baza sinfi olmadan sonuncu törəmə sinifdə ümumi baza sinfinin iki (və daha çox) nüsxəsi ola bilərdi. Lakin cari baza sinfinin virtual edilmsinə görə sonuncu törəmə sinfdə baza sinfinin yalnız bir nüsxəsi təmsil edilmişdir.

Virtual destruktor

Bu gözəl mövzu ilə biz virtual funksiyaların istifadəsinin baxılmasına davam edəcəyik. Biz ümid edirik ki, siz siniflərin obyektlərinin necə yaradılmasını və ləğv edilməsini, konstruktor və destruktorun nə olduğunu xatırlayırsınız. Buna görə də, bu məsələnin öyrənilməsinə sadə nümunəyə baxmaqla başlayaq.

Sətir qiymətini saxlaya bilən bir sinif yaradaq və bu sinif bizdə baza sinfi (abstrakt deyil, belə ki, bu hazırki halda lazım deyil) olsun, bundan biz digərlərini çıxaracağıq.

```
class Base
{
    private:
        char *sp1;
        int size;

    public:
        //konstruktor
        Base(const char *S, int s){
            size=s;
            sp1=new char[size];
        }

        //destruktor
        ~Base(){
            cout<<"Base";
            delete[]sp1;
        }
};
```


Beləliklə, sinfin konstruktoru new konstruksiyasına müraciət edərək sətir üçün yaddaş ayırır və yeni sətrin ünvanını sp1 göstəricisində saxlayır. Sinif destruktoru Base sinfinin obyekti görünmə oblastından çıxdığı zaman yaddaşı boşaldır. Bundan sonra baza sinfindən yeni sinif çıxardaq. Növbəti şəkildə:

```
class Derived: public Base
{
    private:
        char *sp2;
        int size2;

    public:
        //konstruktor
        Derived(const char *S1,int s1,
                const char *S2, int s2): Base(S1,s1){

            size2=s2;
            sp2=new char[size2];
        }

        //destruktor
        ~Derived(){

            cout<<"Derived";
            delete[]sp2;
        }
};
```

Bu sinif onun sp2 göstəricisinin istinad etdiyi ikinci sətiri saxlayır. Yeni konstruktor baza sinfinin konstruktorunu şətri baza sinfinə ötürərək çağırır, həmçinin ikinci sətir üçün yaddaş ayırır və yeni sətrin ünvanını sp2 göstəricisində saxlayır.

İndi proqramın hansısa bir yerində belə sinfin obyektini yarada bilərik:

```
Derived MyStrings(“string 1”,9,“string 2”,9);
```

Bu obyekt görünmə oblastından çıxdığı zaman əvvəlcə Derived sinfinin destruktoru, sonra isə Base baza sinfinin destruktoru çağırılır. Bütün yaddaş səliqə ilə azad ediləcək. Nəzəriyyəyə görə hər şey əladır.

Başqa bir varianta baxaq. Hesab edək ki, biz Base baza sinfinn göstəricisini elan etmişik, ona Derived sinfinin obyektinin ünvanını mənimsətmişik. Bu tamamilə yolverilən dir, biz artıq bu məsələni əvvəl müzakirə etmişik. Yəni, bu proqramda növbəti şəkildə görünəcək:

```
Base *pBase; //baza sinfinin göstəricisi
pBase=new Derived(“string 1”,9,“string 2”,9);
```

pBase göstəricisinin istinad etdiyi obyekt silindiyində nə baş verəcək?

```
delete pBase;
```

Kompilyator "görür" ki, pBase göstəricisi Base sinfinin obyektlərinə istinad etməlidir (o necə bilməlidir ki, bu göstəriciyə məhz nə mənimsədilib?) və tamamilə doğrudur ki, proqram yanız baza sinfinin destruktorunu çağırır və o bir sətri silir, lakin yaddaşda digərini saxlayır. Belə ki, Derived sinfinin destruktoru çağırılmamışdır. Yaddaşın klassik azalması baş verir. Məhz burada virtual destruktor peyda olur.

Bu halı aradan qaldırmaq üçün siniflərdə virtual açar sözü ilə destruktorlar elan etmək lazımdır. Beləliklə, destruktorlar növbəti şəkildə olacaq:

```
virtual ~Base(){
    cout<<"Base";
    delete[]sp1;
}

virtual ~Derived(){
    cout<<"Derived";
    delete[]sp2;
}
```

Mənası belədir. Belə ki, destruktorlar virtual elan eildikləri üçün onların çağırılması proqramın icrası zamanı baş verəcək. Yəni, obyektlərin özləri hansı destruktorun çağırılacağını təyin edirlər. Belə ki, bizim pBase göstəricimiz əslində Derived sinfinin obyektinə istinad etdiyi üçün bu sinfin destruktoru baza sinfinin destruktoru kimi çağırılacaq. Baza sinfinin destruktoru avtomatik olaraq törəmə sinfinin destruktorundan sonra icra ediləcək.

Təmiz virtual destruktor

Nəhayət, virtual funksiyalar haqqında sonuncu informasiya. Bəzi hallarda sinif daxilində təmiz virtual destruktor təyin etmək çox münasibdir.

Biz artıq bugün təmiz virtual funksiyaları müzakirə etmişik. Onlar bizə obyektin yaradılması mümkün olmayan abstrakt siniflər verir. Bu siniflərin ierarxiyasının qurulmasına əsaslanır. Lakin, bəzən elə siniflər rast gəlinir ki, onları abstrakt etmək məqsədəuyğundur, lakin bunun üçün sizin sərəncamınızda təmiz virtual funksiyalar olmaya bilər. Nə etməli? Həlli çətin deyildir. Təmiz virtual funksiya və virtual destruktor anlayışlarını birləşdirək. Sadəcə abstrakt olacaq sinif daxilində təmiz virtual konstruktor elan etmək lazımdır.

Nümunə göstərək.

```
//virtual olmayan abstrakt sinif
class Something
{

    public:
        //bu isə tam virtual destruktordur
        virtual ~Something()=0;
};
```

Bu sinif abstraktdır, ona görə ki, təmiz virtual funksiya (destruktor) ehtiva edir. Belə ki, destruktor virtual olduğu üçün gələcəkdə destruktorun çağırılması problem olmayacaq.

Bundan sonra edəcəyimiz tək şey, bu destruktorun təyinini verməkdir.

```
Something::~Something() {};
```

Bunu etmək vacibdir, belə ki, virtual destruktor növbəti şəkildə işləyir: əvvəlcə varislilik zəncirinin yuxarısında olan törəmə sinfin destruktoru çağırılır, sonra isə varislilik zəncirinin yuxarısında olan siniflərin destruktorları baza sinfinin abstrakt destruktorlarına qədər ardıcıl çağırılır. Bu o deməkdir ki, kompilyator sinif abstrakt olduğu zaman da ~Something() destruktorunun çağırılmasını hasil edəcək. Buna görə də funksiyanın gövdəsini təyin etmək vacibdir. Əgər bu edilməzsə, kompilyator simvolun çatışmadığı haqqında səhv xəbərdarlığı edəcək və bunu onsuz da etmək lazım gələcək.

Bəzi məsləhətlər

Əgər sinfin virtual funksiyaları olarsa, onun üçün virtual destruktor yaratmağın birbaşa mənası var. Hətta o bu sinif üçün tələb olunmasa belə. Ondan törəyəcək siniflər uyğun şəkildə çağırılacaq destruktorlar ehtiva edə bilərlər.

Əgər sinif virtual funksiyalar ehtiva etmirsə, onda çox güman ki, onun baza sinfi kimi istifadəsi nəzərdə tutulmayıb. Bu halda onun daxilində virtual destruktorun təyini adətən əsassızdır.

Qeyd: Yeri gəlmişkən! Konstruktorlar virtual ola bilməzlər. Diqqətli olun!

Ev tapşırığı

1. Təmiz virtual Print() funksiyasını ehtiva edən Employer (işçilər) abstrakt baza sinfini yaradın. Üç törəmə sinif yaradın: President, Manager, Worker. Hər tip işçi üçün informasiyanın ekrana verilməsi üçün Print() funksiyasını təyin edin.
2. Siyahı baza sinfini yaradın. Siyahı bazasında daxiletmə və çıxartma funksiyaları olan stek və növbə reallaşdırın.
3. Virtual sahə funksiyası ehtiva edən abstarkt baza sinfini yaradın. Törəmə siniflər yaradın: öz sahə funksiyaları olan düzbucaqlı, çevrə, düzbucaqlı üçbücaq, trapesiya. Yoxlamaq üçün müxtəlif obyektlərin ünvanlarının mənimsədildiyi abstrakt siniflərin göstəriciləri massivini təyin edin. Trapesiyanın sahəsi: S=(a+b)h/2
4. Canlılar sinfini yerləşmələrinə görə yaradın. Varis siniflər: tülkü, dovşan və ot təyin edin. Tülkü dovşan yeyir. Dovşan ot yeyir. Tülkü ölə bilər – yaşı təyin edilib. Dovşan da ölə bilər. Bundan başqa həyatın olmaması sinfi də təyin edilib. Əgər ətrafda çox ot olarsa, onda ot qalacaq, əks halda yeyilmiş olacaq. Əgər tülkü çox qocadırsa, o ölə bilər. Əgər tülkülər çoxdursa (ətrafda 5-dən çox), onda tülkülər bir daha peyda olmurlar. Əgər dovşanlar tülkülərdən azdırlarsa, onda tülkülər dovşanları yeyirlər.
5. Tənliyin kökü – virtual funksiyasını ehtiva edən abstrakt baza sinfi yaradın. Növbəti törəmə sinifləri yaradın: xətti tənliklər sinfi və kvadrat tənliklər sinfi. Tənliyin köklərini hesablayan funksiya təyin edin.