Obyektyönlü proqramlaşdırma

C++

Dərs №6

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

-> Operatoruna yükləmə

Biz ümid edirik ki, siz C++-da bir neçə operatordan başqa demək olar ki, bütün operatorlara yükləməyin mümkün olduğunu xatırlayırsınız. İstənilən halda -> operatoru yüklənir və bu böyük əhəmiyyət kəsb edir. Bu operator selektor (member selector) adlanır. Misala baxaq:

```
#include <iostream>
using namespace std;
//göstərici inkapsulyasiya ediləcək sinif
class Temp
{
    int TEMP;
    public:
        //konstruktor
        Temp(){TEMP=25;}
        //ekranda göstərmə funksiyası
        void TempFunction(){
            cout<<"TEMP = "<<TEMP<<"\n\n";
        }
        //qiymət vermək üçün funksiya
        void TempSet(int T){
            TEMP=T;
        }
};
//göstərici inkapsulyasiya edən sinif
class MyPtr
{   //sinif göstəricisi

    Temp Temp*ptr;
```

```
    public:
        //konstruktor
        MyPtr(Temp*p=NULL){
            ptr=p;
        }
        //inkapsulyasiya olunandan
        //inkapsulyasiya edənə tip çevirmə
        //operatoru

        operator Temp*(){
            return ptr;
        };
        // -> selector operatoru
        //«gizli» göstəriciyə birbaşa
        //müraciət etməyə imkan verir
        Temp* operator->(){
            return ptr;
        }
    //göstəricini irəli sürüşdürən ++ operatoru
        MyPtr operator++(){
            ptr++;
            return *this;
        }
};

void main ()
{   //yeni obyektin yaradılması
    Temp*main_ptr = new Temp;
    //«doğma» göstərici vasitəsilə
    //obyektin üzvlərinə sadə müraciət
    ptr->TempFunction();

    //sinif-göstəricinin obyektinin yaradılması
    MyPtr pTemp(main_ptr);
    //sinif-göstərici vasitəsilə müraciət
    pTemp->TempFunction();
```



```
    //inkapsulyasiya olunan sinfin obyektlər
    //massivinin yaradılması
    Temp*arr_=new Temp[3];

    //yuxarıda qeyd edilən massivin 0-dan 2-yə
    //qədər doldurulması
    for(int i=0;i<3;i++) arr_[i].TempSet(i);

    //sinif-göstəricinin obyektinin yaradılması və ona
    massivin ünvanının yazılması
    //(burada tipin çevirilməsi işləyir)
    MyPtr arr_temp=arr_;
    //bir element irəli
    arr_temp++;
    //nəticənin nümayiş etdirilməsi
    arr_temp->TempFunction();
    //obyektin ləğvi
    delete main_ptr;
    delete[]arr_;
}

Proqramın icrasının nəticəsi
TEMP = 25
TEMP = 25
TEMP = 1
```

Beləliklə, nəticəni müzakirə edək: Bizim bir neçə nüsxə ehtiva edə bilən Temp sinfimiz var, bir neçə TEMP üzvü və onlarla işləmək üçün minimal sayda funksiyalar var. Bundan başqa, biz obyekt sinfi-göstəricisi yaratmışıq.

Sadə göstəricini saxlandığımız MyPtr, ona müraciəti məhdudlaşdırırıq və onun üçün növbəti operatorları yükləyirik:

1. Temp tipindən MyPtr-ə çevirmə operatoru
2. -> selektor (selector) operatoru.
3. Göstəricini bir addım irəli sürüşdürən ++ operatoru.

Müsbət hallara baxaq – həqiqi siniflərin yerinə istifadə edə biləcəyimiz obyekt-göstəricilər sinfi əldə etdik. Bu rahatdir, belə ki, göstərici ilə işləmək üçün istifadə edilən bütün funksiyaları bu sinifdə inkapsulyasiya etmək olar. Lakin verilmiş konstruksiyanın geniş yayılmış daha bir tətbiqi var. Onunla biz dərsimizin növbəti bölməsində tanış olacağıq. İrəli!

«Ağıllı» göstərici anlayışı (smart pointer)

Bizim indi söhbət açacağımız, selektorun yüklənməsi vasitəsi ilə reallaşdırılır. Beləliklə.

Ağıllı göstərici (ing. smart pointer) — adi göstərici interfeysini imitasiya edən və ona nə isə bir yeni funksionallıq, (məsələn, müraciət zamanı sərhədlərin yoxlanılması və ya yaddaşın təmizlənməsi) əlavə edən sinifdir.

Başqa sözlə, ağıllı göstəricilər — həqiqi obyektin göstəricisinin və bir qayda olaraq obyektdə müraciət sayının sayğacının saxlandığı obyektlərdir.

Sinifdə həmçinin, həqiqi obyekt üçün xaricdən çağırılmayan, yalnız smart pointer daxilində çağırılan destruktor da mövcuddur. Prinsip belədir ki, obyektə müraciət sayğacı 0-a bərabər olanda destruktor çağırılır.

Məsələn, obyektlərin göstəricilər massivi var. Siz obyektin nüsxəsini daha harada isə istifadə edisiniz. Sonra obyektləri ləğv edərək massivi təmizləyirsiniz, lakin harada isə hələ də göstərici saxlanılır. Təbii ki, ona müraciət zamanı proqramın icrasında kəsilmə baş verir, belə ki, massivin təmizlənməsi zamanı obyekt ləğv edilmişdir.

smart pointer-in olması halında bu baş vermir, belə ki, göstəricinin nüsxəsinin yaradılması zamanı daxili sayğacı bir vahid artıracaq. O ikiyə bərabər olacaq — biri massiv üçün, digəri isə surəti üçün. İndi biz əgər obyekti massivdən silərikə, daxili sayğac bir vahid azalacaq və

birə bərabər olacaq, yəni, nüsxə obyektlə işləməyə davam edəcək. O lazım olmayandan sonra nüsxə yaddaşı ondan azad edir və sayğac sıfra bərabər olur. Məhz bu zaman həqiqi obyekt üçün destruktor çağırılır.
Daha nə, hər şey deyilmişdir. Artıq «ağıllı göstərici» adlı anlayışı sınaqdan keçirmək olar.

```
#include <iostream>
using namespace std;
class Temp
{
    int TEMP;
    public:
        //konstruktor
        Temp(){TEMP=25;}
        //ekranda göstərmə funksiyası
        void TempFunction(){
            cout<<"TEMP = "<<TEMP<<"\n\n";
        }
        //qiymətin verilməsi funksiyası
        void TempSet(int T){
            TEMP=T;
        }
};

//ağıllı göstəricini reallaşdıran sinif
class SmartPointer
{
    //İnkapsulyasiya olunan göstərici
    Temp*ptr;
    //nüsxə sayğacı
    int count_copy;

    public:
        //konstruktor
```



```
        SmartPointer (Temp*p=NULL){
         //nüsxənin yaradılması zamanı 0 yazırıq
            count_copy=0;
            ptr=p;
        }
        //köçürmə konstruktoru
        SmartPointer (const SmartPointer&obj){
            //nösxə yaradılır - sayğacı artırırıq
            ptr=obj.ptr;
            count_copy++;
        }
        //operatora yükləmə bərabərdir
        SmartPointer operator=(const SmartPointer&obj){
            //nüsxə yaradılır - sayğacı artırırıq
            ptr=obj.ptr;
            ptr=obj.ptr;
            count_copy++;
            //növbəti hal üçün obyektin özünü
            //qaytarırıq
            a=b=c return *this;
        }
        //obyektin ləğv edilməsi
        ~SmartPointer(){
            //əgər obyekt varsa və nüsxə yoxdursa
            if(ptr!=NULL&&count_copy==0){
            cout<<"\n~Delete Object\n";
                //obyekti ləğv edirik
                delete[]ptr;
            }
            //əks halda nüsxə ləğv edilir
            else{
                //sayğacı azaldırıq
                count_copy--;
                cout<<"\n~Delete Copy\n";
            }
        }
        //köhnə sevimli selektora yükləmə
        Temp* operator->(){
            return ptr;
        }
};
```

```
void main(){
    //obyekt yaradırıq
    Temp*main_ptr=new Temp;
    //bu obyektlə ağıllı göstəricini qiymətləndiririk
    //(inisiallaşdırırıq)
    SmartPointer PTR(main_ptr);
    //ağıllı göstəricinin işini yoxlayırıq
    PTR->TempSet(100);
    PTR->TempFunction();
    //nüsxə yaradırıq (köçürmə kostruktorunun işi)
    SmartPointer PTR2=PTR;
}
Proqramın icrasının nəticəsi:
//ağıllı göstərici vasitəsilə obyektlə iş
TEMP = 100
//nösxənin ləğvi
~Delete Copy
//obyektin özünün ləğvi
~Delete Object
```

İndi biz özümüzun xüsusi göstəricimizi yaratdıq, növbəti dərslərdə STL kitabxanasının auto_ptr adlı artıq hazır olan smart pointer-ə baxacağıq.

İxtiyarı sayda və tipdə arqumentləri olan funksiyalar

Bu bolmə parametrlərin sayı funksiyaya müraciət edərkən məlum olan yeni tipli funksiyanın oyrənilməsinə həsr edilib.

Beləliklə, C++ dilində parametrlərinin sayı və onların tipləri funksiyanın kompilyasıyası zamanı təyin edilməyən funksiyanın istifadəsi mümkündür. Bu qiymətlər yalnız finksiyanın çağırılması zamanı faktik parametrlərin aşkar verilməsi halında məlum olur. Uzunluğu məlum olmayan parametrlər siyahısı və formal parametrlərin spesifikasiyası olan bu cür funksiyaların təyini və təsviri zamanı nöqtə-vergüllə tamamlanır:

```
funksiyanın_tipi funksiyanın_adı (aşkar_parametrlərin
spesifikasiyaları,...);
```

Burada aşkar parametrlərin spesifikasiyası, sayı və tipi sabitdir və kompilyasiya zamanı məlumdur. Bu parametrlər vacibdirlər. Aşkar (vacib) parametrlər siyahısından sonra vacib olmayan vergül, sonra isə nöqtələr qoyulur ki, bu da kompilyatora funksiyanın çağırılmasının emalı zamanı parametrlərin sayının və tiplərinin uyğunluğuna gələcəkdə nəzarət edilməsinə ehtiyac olmamasını bildirir.

Dəyişkən siyahılı parametrləri olan hər bir funksiyanın onların sayı və tiplərini təyinetmə mexanizmi olmalıdır. Bu məsələnin həlli üçün iki yanaşma var:

1. Birinci yanaşma, siyahının bitdiyinə işarə edən unikal qiymətli xüsusi parametr-indikatorun vacib olmayan parametrlərini siyahının sonuna əlavə edilməsini ehtimal edir. Funksiyanın gövdəsində parametrlər ardıcıl olaraq götürülür və onların qiymətləri əvvəlcədən bəlli olan sonluq əlamətinə əsasən müqayisə olunur.
2. İkinci yanaşma faktik parametrlərin sayının real qiymətinin öturulməsini nəzərdə tutur. Bu kəmiyyəti funksiyaya aşkar (vacib) verіən parametrlərdən biri vasitəsilə ötürmək olar.

Qeyt etmək lazımdır ki, hər iki yanaşmada — sonluq əlamətinin verilməsi zamanı, həm də real istifadə edilən faktik parametrlərin sayının göstərilməsi zamanı — bir faktik parametrdən digərinə keçid göstəricilər vasitəsilə, yəni, ünvan cəbrinin istifadəsi ilə icra edilir, Deyilənləri misallarla nümayiş etdirək.

Misal 1. *Toplananların sayından isitifadə etməklə verilmiş ədədlərin cəmini hesablayan proqramı tərtib etməli.*

```
#include <iostream>

using namespace std;

//Funksiyanın prototipi. long

summa (int,...);
```



```
void main()
{
    cout << "\n summa(2,4,6)=" << summa (2,4,6);

    cout << "\n summa(6,1,2,3,4,5,6)="
                                  << summa (6,1,2,3,4,5,6);
}
//Parametrlərin sayını veririk.
long summa (int k,...)
{
    //pk parametrlər siyahısının başlanğıc ünvanını
    //ehtiva edir
    //bu onunla əlaqədardır ki, parametrlər əməli
    //yaddaşda ardıcıl yerləşirlər
     int *pk=&k;

    //cəmin hesablanması
    //k - parametrlərin sayı
    long sm=0;
    for (;k;k--)

        sm+=*(++pk)

        ;

    return sm;
}
```

Proqramın şərhi

Adi halda funksiyanın təyini zamanı dəyişənlərin siyahısı – formal parametrlər dəsti (ötürülən qiymətlərə müraciətin həyata keçirildiyi) verilir. Burada belə bir siyahı yoxdur. Hansı formada ötürülən parametrlər siyahısına müraciət etmək olar? Bu göstəricilər vasitəsilə həyata keçirilir: əvvəlcə göstəriciyə aşkar parametrlər siyahısının sonunun və ya əvvəlinin ünvanı yerləşdirilir, sonra isə bu qiymətlər isitifadə

edilərək parametrlərin dəyişən siyahısında baxılır.

Misal 2. *Təyin edilmmiş qiymət (tutaq ki, bu sıfırdır) istifadə edən proqramı tərtib edək*

```
#include <iostream>
using namespace std;
//k - bu indi yalnız cəmlənən parametrlərdən biridir
 long summa (int k,...)
{
    //k-dan başlayaraq
    int *pk=&k; long
    sm=0;
    //qiymətə rast gələnə qədər davam edirik
     0 while (*pk)
        //cəmin hesablanması
        sm+=*(pk++);
    return sm;
}
void main()
{
    //yoxlayırıq
    cout << "\n summa(4,6,0)="<< summa (4,6,0);
    cout << "\n summa(1,2,3,4,5,6,0)="
                              << summa (1,2,3,4,5,6,0);
    cout << "\n summa(1,2,0,4,5,6,0)="
                              << summa (1,2,0,4,5,6,0);
}

Proqramın icrasının nəticəsi:

summa(4,6,0)=10
summa(1,2,3,4,5,6,0)=21


//Diqqət burada funksiya var
```



```
//ilk sıfra qədər çalışır
//digər    verilənlər    itəcək.
summa(1,2,0,4,5,6,0)=3
```

Funksiya daxilində arqumentləri analiz etmək üçün daha bir üsul vardır. Məhz bu üsul yalnız ixtiyari sayda deyil, həmcinin müxtəlif tipli arqumentləri analiz etməyə imkan verir. Başlanğıc üçün stdarg.h kitabxanasını qoşuruq, məhz onda analiz üçün vacib avtomatik vasitələr vardır. Növbəti fəaliyyətləri addım-addım gözdən keçirək:

1. va_list (char* kimi təyin eidilmiş) tipində dəyişən elan edirik. Məhz bunda bizim funksiyanın parametrlərinin siyahısının göstəricisini saxlayacağıq.
2. Funksiyanı vahid şəkildə çağırırıq:

```
void va_start( va_list arg_ptr, prev_param );
```

İlk parametr — əvvəlcədən va_list tipində yaradılmış dəyişən, ona bu funksiya ilə yaradılmış parametrlərin siyahısının göstəricisi yazılacaq.
Funksiyamızın ikinci parametri arg_ptr və siyahının əlaqələndirilməsi zamanı va_list-in nəyə isə əsaslanması üçündür.

3. Funksiyanın çoxsaylı çağırılması vasitəsi ilə parametrləri analiz edirik:

```
type va_arg( va_list arg_ptr, type );
```

Verilmiş funksiya siyahıdan parametr alır ki, bu da ona birinci parameter kimi ötürülən göstəicidir.

Bu göstəricini növbəti elementə yerləşdirir. İkinci parametr kimi siyahıdan əldə edilməlidir.

4. Parametrlərin analizini tamamlayırıq və funksiya vasitəsilə daxili göstəriciyə sıfır yazmaqla onu silirik:

```
void va_end( va_list arg_ptr );
```

Misala baxaq:

```
#include <iostream>
using namespace std;
#include <stdarg.h>
//Funksiya count sayı ilə ədədlərin cəmini hesablayır
//typeof elementin hansı tipdə olacağını təyin edir
//true – tam tipli parametrlər (int)
//false – həqiqi tipli parametrlər(double)
void AnyType(int count,bool typeof,...){
    //tam tipli parametrlər üçün cəm
    int sumi=0;
    //həqiqi tipli parametrlər üçün cəm
    double sumd=0.0;
    //parametrlər siyahısının göstəricisini yaradırıq
    va_list arg_ptr;
    //ilk faktik parametrə əsaslanaraq bu
    //göstəricini əldə edirik
    va_start(arg_ptr,count);
    //ikinci faktik parametri buraxırıq
    va_arg(arg_ptr,bool);
    //növbəti parametrləri count dəfə sayırıq
    while(count--){
        //əgər typeof - true, onda int tipində
        //parametrləri oxuyuruq
        //əgər typeof - false, onda double tipində
        parametrləri oxuyuruq
```



```
        (typeof)?sumi+=va_arg(arg_ptr,int):sumd+=va_ar
                                g(arg_ptr,double);
    }
    //parametrlər siyahısını bağlayırıq
    va_end(arg_ptr);
    //əgər typeof - true, onda int tipində cəmi
    //göstəririk
    //əgər typeof - false, onda double tipində cəmi
    //göstəririk
    (typeof)?cout<<"Integer sum = "<<sumi:cout<<"Double
                    sum = "<<sumd;
    cout<<"\n\n";
}
void main()
{
    //int tipində parametrlərin cəmlənməsi üçün
    //funksiyanı çağırırıq
    AnyType(4,true,3,8,9,4);
    //double tipində parametrlərin cəmlənməsi üçün
    //funksiyanı çağırırıq
    AnyType(3,false,2.5,1.1,3.6);
}
```

Siz artıq diqqət etdinizsə, qeyri-məhdud sayda parametrləri olan funksiyaların bəzi çatışmayan cəhətləri var. Xüsusilə, parametrlərin analizi çətindir və çox zaman hətta böyük kodu olur. Bundan başqa, bu funksiyalar üçün icra mərhələsində səhvlərin olma ehtimalı artir, belə ki, parametrlərin düzgün ötürülməsini izləmək çox çətindir. Buna görə də, Biz bu aparatın reallaşdırılmasında çox diqqətli olmanızı tövsiyyə edirik.

Ev tapşırığı

1. Kompleks ədədlərlə işi təmin edən sinif təyin edin.
2. Blek-Jek kart oyununu siniflər əsasında reallaşdıran proqram yazın.
3. Ekrana formatlaşdırılış çıxışı reallaşdıran proqram yazın.