Obyektyönlü proqramlaşdırma

C++

ШАГ
компьютерная
АКАДЕМИЯ

Dərs №14

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

İstisna halların emalı

Sizinlə ilk proqramlaşdırma tanışlığımızda biz demişdik ki, iki əsas səhv tipləri var: kompilyasiya mərhələsində səhvlər və icra mərhələsində səhvlər. Xatırladırıq ki, birinci tip səhvlər sintaksis səhvləri nəzərdə tutur ki, xoşbəxtlikdən kompilyator bunları gözdən qaçırmır. İkinci tip səhvlər isə proqramın səhv işləməsinin nəticəsidir. Bu dərsimizdə məhz bu tip səhvlər haqqında danışacağıq.

Tamamilə təbiidir ki, proqram yazarkən biz, proqramın icrası zamanı səhv vermə hallarının olmasını nəzərdə tuturuq. Bu zaman, yəqin ki, siz standart, ümumi qəbuledilmiş variantlardan birini istifadə edirsiniz, məsələn:

1. ***Səhv baş verə biləcək funksiya özü onu təqib edir, işi datyandırır və ekrana ismarış çıxarır.*** Bu metodun çatışmayan cəhəti ondan ibarətdir ki, funksiya istifadə edən proqramçı səhvi sərbəst emal etmə və bu halda özünün hər hasnı bir fəaliyyətini həyata keçirmək istəyində olduğu zaman bunu edə bilməz. Sadəcə ona görə ki, funksiya kitabxanada yerləşə bilər və o proqramçı üçün əlyetərli olmayacaq. Nəticədə, məsələn, səhv haqqında standart ismarışdan yaxa qurtarmaq mümkün deyil.

2. Səhv baş verə biləcək funksiya, işini dayandırır və səhvin kodunu əsas proqramda çağırıldığı yerə qaytarır. Təyin edildiyi kimi, bu üsul məsələnin həllidir. Lakin, əgər funksiya müxtəlif hallarda bir neçə müxtəlif səhvlər hasil edərsə, onda proqramçının müxtəlif səhv kodları yazmağı düşünməsi lazımdır və bunun sənədlərini hazırlamalıdır ki, bu da çox münasib deyil.

Bəs nə etməli, siz söyləyin?! Yəni bütün metodlarda rast gəlinən səhvlər başdan ayağa çatışmazlıqdır?! Təbii ki, bu belə deyildir. C++ dili bizə sizinlə yuxarıda şərh edilmiş çatışmazlıqlardan asanlıqla yaxa qurtarmağa əla vasitə təqdim edir və icra mərhələsində səhvlə bağlı problemi birdəfəlik həll etməyə imkan verir. Bu metod istisnanın işlənməsi adlanır. Beləliklə:

İstisna (exception) — proqram tərəfindən, məsələn, səhvin baş verməsi anında hasil edilən istisna haldır.

Nəticədə sizi də qane etdiyi kimi, istisnanın emal edilməsi mexanizmi yuxarıda göstərilən bütün çatışmazlıqların birbaşa aradan qaldırılmasıdır. Yəni, funksiya istifadə edən və onu reallaşdıran proqramçı səhvin necə emal edilməsini özü təyin edir. Dərsin sonunda bu mexanizmin reallaşdırılması üçün istifadə edilən açar sözə baxacağıq.

Qeyd: Qeyd etmək lazımdır ki, istisnanın emal edilməsi mexanizmi yalnız səhv hallar zamanı deyil, həmcinin, proqramçının proqramda hər hansı bir kod blokunu seçmək lazım gəldiyində də istifadə oluna bilər.

Mexanizmin reallaşdırılması. try, catch, throw açar sözləri

Beləliklə, istisnanın emalı mexanizminin təşkili üçün C++ dilində üç açar sözdən istifadə edilir:

- **try** (*nəzarət etmək*) — fiqurlu mötərizələr vasitəsilə istisnanın hasil edildiyi kodun müəyyən hissəsini ayırır;
- **throw** (*atlamaq*) — istisna halların yaranması zamanı istisna hasil edən və proqramı kritik blokdan çıxardan operatordur;
- **catch** (*tutmaq*) — konkret istisna "tutan" və onu analiz edərək lazım olan reaksiyanı verən operatordur.

Mexanizmin təşkili sintaksisi:

```
try{
    kod bloku;
    ...
    throw müəyyən_tip_ifadəsi;
    ...
}

catch(istisna_tipi ad)
{
    analiz_bloku;
}
```

Sintaksisin analizi

1. bir try blokunda müxtəlif istisnalar doğuran bir neçə throw ola bilər.
2. throw işləyən zaman, onda olan ifadə vasitəsilə yaddaşda ifadənin tipi ilə üst-üstə düşən obyekt yaradılır. Onun formalaşdırılmasından sonra throw obyektin idarəsini try blokunun xaricinə ötürür. Obyekt eyni verilənlər tipi olan uyğun catch blokuna düşür.
3. Catch blokları, throw kimi bir neçə ola bilər, lakin onların sayının eyni olması vacib deyil, yəni, birbirindən asılı deyillər.
4. Bir catch digərindən verilənlərin tipinə görə fərqlənməlidirlər, yəni iki eyni tipli catch ola bilməz.

Bu materialı asan qavramanız üçün, bütün sonrakı xüsusiyyətlərə biz sadə nümunələr əsasında baxacağıq.

Nümunə 1. *İstisnaya aid sadə nümunə.*

```
# include  <iostream>

using namespace std;

void main(){
    //Verilmiş ifadə proqram icra edilən
    // kimi ekrana çıxacaq
    cout<<"\nStart!!!\n";
```



```
//istisna blokundan çıxış
    try{
        //ekrana çıxışın icrası
        cout<<"\nBefore!\n";

        //try blokunun kəsilməsi
        //int tipində istisnanın oyanması
        //try blokundan çıxış
        throw 100;

        //verilmiş ismarış ekranda heç zaman
        //görünməyəcək

        cout<<"\nAfter!\n";
    }

    //int tipində istisnanın təsbit edilməsi
    //100 qiyməti g-yə mənimsədilir
     catch(int g){

        //istisnanın analizinin nəticəsi
        cout<<"\nException!!!!\n";
    }

    //tamamlanan sətrin ekranda
    //göstərilməsi
    cout<<"\nBye!!!\n";
}
_____________________________
Proqramın icrasının nəticəsi:
_____________________________
Start!!!

Before!

Exception!!!!

Bye!!!
```

Nümunə 2. *Sıfra bölmə istisnası*

```
# include  <iostream>

using namespace std;

void main(){

    //istisna blokuna giriş
    try{
        //dəyişənlərin, sorğuların yaradılması
        //və verilənlərin klaviaturadan daxil edilməsi

        float a,b;

        cout<<"\nPut digit a:\n";
        cin>>a;

        cout<<"\nPut digit b:\n";
        cin>>b;

        //sıfra bölmənin yoxlanılması
        if(b==0){

            //əgər bölən sıfırdırsa -
            //float tipində istisnanın oyanması
            //try blokundan çıxış
            throw b;

        }

        //əks halda blok müvəffəqiyyətlə
        //tamalanmışdır
        cout<<"\nResult = "<<(a/b)<<"\n\n";
    }
    //buraya throw float tipində qiymət ötürür

    catch(float g){

        //analiz və səhv haqqında ismarış
        cout<<"\nError – Divide by "<<g<<"\n\n";
    }
}
```


Nümunə 3. *Bir neçə istisnaya aid nümunə.*

```
# include  <iostream>

using namespace std;

void main(){

    //try bloku üç istisna döğura bilir
    //üç istisna
    try{

        //dinamik massiv göstəricisi və ölçüsü
        int*ptr=0;
        int size;

        //ölçünün daxil edilməsi
        cout<<"\nPut size:\n";
        cin>>size;

        //əgər ölçü verilmiş diapazonun xaricinə
        //çıxarsa
        if(size<1||size>500)
            //char* tipində istisna doğururuq
            //try bloku kəsilir
            throw "\n\nErr Size!!!\n\n";

        //əks halda massiv yaradırıq
        ptr=new int [size];


        //yaddaşın ayrıldığını yoxlayırıq
        if(!ptr)
            //əks halda char* tipində istisna
            //doğururuq
            //try bloku kəsilir

            throw "\n\nErr Memory!!!\n\n";
```

```
        //əks halda a test dəyişəni yaradırıq
        //verilənlərin klaviaturadan daxil
        //edilməsini həyata keçiririk
        int a;
        cout<<"\nPut digit a:\n";
        cin>>a;

        if(a==0)
            //əgər a sıfra bərabərdirsə,
            //int tipində istisna yaradırıq
            //try bloku kəsilir
            throw a;

    }

    //int tipində bütün istisnaların təsbit edilməsi
    catch(int s){
        cout<<"\nError - A = "<<s<<"\n\n";
    }

    //char* tipində bütün istisnaların təsbit edilməsi
    catch(char*s){
        cout<<s;
    }
}
```

Nümunə 4. *Universal catch.*

Əgər catch blokunda ekvivalenti olmayan verilənlər tipi ilə throw hasil edilərsə, onda avtomatik bağlanacaq. Bu zaman səhv baş verəcək və (Skip düyməsi sıxıldığı zaman) ekranda növbəti ismarış peyda olacaq:

This application has requested the Runtime to terminate it in an unusual way. Please contact the application's support team for more information.

Bu haldan qurtulmaq üçün adətən universal catch istifadə edilir. Onunsintaksisi belədir: **catch(...){analiz;}**. Diqqətinizi ona yönəltmək istəyirik ki, verilmiş konstruksiya bütün sadalanan catch blokları arasında ən sonuncu olmalıdır!!! Əgər hansısa istisna üçün uyğun catch tapılmazsa, universal konstruksiya icra ediləcək.

```
# include  <iostream>
using namespace std;
void main(){

    //bloka giriş
    try{
        //dəyişənlərin elanı və qiymətləndirilməsi

        int a;
        cout<<"\nPut digit a:\n";
        cin>>a;

        //əgər dəyişən sıfra bərabərdirsə
        if(a==0)
            //char* tipində istisnanın hasil edilməsi
            throw "URRRRRRA!!!";
    }

    //universal catch
    catch(...){
        cout<<"\nSome Error!!!!\n\n";
    }
}
```

Nümunə 5. *Funksiya daxilində istisnanın hasil edilməsi. (Butun emal funksiya daxilindədir).*

```
# include  <iostream>

using namespace std;

void Some(){

    //sıfra bölmə analizi
    int a;
    int b;

    try{

        cout<<"\nPut digit a:\n";
        cin>>a;
        cout<<"\nPut digit b:\n";
        cin>>b;

        //əgər bölən sıfra bərabərdirsə,
        //istisna hasil edirik

        if(b==0)
            throw "\tZerro!!!\n";

    }
    //istisnanın təsbit edilməsi və analizi
    catch(char*s){
        cout<<"\n Error!!!!"<<s<<"\n\n";
    }
}
void main(){

    cout<<"\nFirst!!!\n";
    //istisna ehtiva edən funksiyanın çağırılması
    Some();
    cout<<"\nSecond!!!\n";
}
```


Nümunə 6. *Funksiya daxilində istisnanın hasil edilməsi.* (Funksiya daxilində yalnız istisnanın doğrulması baş verir).

```
# include  <iostream>

using namespace std;

void Test(int t){

    //funksiyaya giriş
    cout<<"\nInside!!!\n";

    if(t==2){
        //istisnanın doğrulması
        throw "\nError – t is 2\n";
    }

    else if(t==3){
        //istisnanın doğrulması
        throw "\nError – t is 3\n";
    }
}

void main(){
    //funksiyanın çağırılması try
    //gövdəsində verilir
    try{
        //çağırılmalar
        Test(4);
        Test(2);
    }
    //istisnanın təsbit edilməsi
    catch(char*s){
        cout<<"\n\n"<<s<<"\n\n";
    }

}
```

Nömunə 7. *İstisnanın yenidən hasil edilməsi.* catch daxilində istisnanı yenidən emal etmə imkanı var:

```
# include  <iostream>

using namespace std;

void Test(){
    try{
        //istisnanın bir dəfə doğrulması
        throw "\nHello!!!\n";
    }
    //analizin daxil edilməsi
    catch(char*s){

        cout<<"\n\nException!!!\n\n";

        //istisnanın növbəti bloka ötürülməsi
        catch throw;
    }
}
void main(){
    cout<<"\nStart\n";
    try{
        //təkara istisna ehtiva edən
        //funksiyanın çağırılması try bloku
        //daxilindədir.
        Test();
    }
    //təkrar göndərilən obyektin təsbit
    //edilməsi
     catch(char*p){
      cout<<p;
    }
}
```

Burada biz istisna ilə işin bütün xüsusiyyətlərini əhatə etməyə çalışdıq. Ümid edirik ki, siz onun istifadəsinin vacibliyini başa düşmüşsünüz.

Adlar fəzaları və onların istifadəsi

"Hesab edin ki, siz naməlum torpağa yollanmışsınız və yolda həmyolçu qazandınız. Həmyolçular yoruldular və yeni çayın sahilində yeni şəhər qurmaq qərarına gəldilər. Lakin siz bilmirsiniz bu şəhəri necə adlandıracaqsınız, belə ki, bu ölkənin digər şəhərləri ilə tanış deyilsiniz və artıq mövcud adı seçmək istəmirsiniz. Aydın həlli – yeni şəhərin adının daxildə unikal olmasına zəmanət vermək üçün yeni ölkə elan etməkdir" — dərsin bu bölməsinin mövzusunun əyani təsviri üçün bu nümunə xüsusi ədəbiyyatda klassikdir. İş ondadır ki, proqramlaşdırmada elə bu cür müxtəlif obyektlərə onları adlar fəzalarına ayırmaqla eyni ad vermək olar. Tərif ilə tanış olaq.

Adlar fəzası (*namespace*) — proqramlaşdırmada adlandırılmış və ya adlandırılmamış dəyişənlərin, tiplərin, sabitlərin təyin edilməsi fəzasıdır. Adlar fəzası digər funksiyalar dəsti ilə münaqişə yaratmaması üçün bir blok daxilində verilənlər və funksiyalar dəstinin məhdudlaşdırılmasıdır. Başqa sözlə, adlar fəzası görünmə oblastını bir neçə bölgəyə ayırmaq üçündür.

Adlar fəzasının elan edilməsi

Adlar fəzasının elan edilməsi sintaksisi sinfin elan ediməsini xatırladır. Şərti olaraq – bu fəzanın komponentlərinin daxil olduğu görünmə oblastı üçün adların ifadə edilməsidir:

```
namespace ad
{
    verilənlər_siyahısı;
}
```

Adlar fəzasının tətbiqi

Eyni görünmə oblastında iki eyni ad ola bilməz. Məsələn, bir proqramda adlar fəzasından istifadə etmədən biri məşəli yandıracaq, digəri isə tapançadan atəş açacaq eyni prototipli iki fire() funksiya yaratmaq olmaz.

Adlar fəzası heç bir problem yaratmadan iki funksiyanı müxtəlif görünmə oblastında yerləşdirməyə imkan verir. Bundan başqa, müəyyən istiqamətdə əlaqələndirilmiş bütün funksiyaları başqa adlar fəzasında yerləşdirmək olar. Məsələn, məşəli yandıran fire() funksiyası digər tədqiqat funksiyaları ilə yanaşı exploration (tədqiqat) adlar fəzasına düşür, fire() funksiyası isə tapançadan atəşin açılması üçündür və combat (döyüş) adlar fəzasına düşür.

```
namespace combat{
    .void fire(){
            cout<<"Vistrel";
```



```
        }
    }

    namespace exploration{

        void fire(){
            cout<<"Ogon`";
        }

    }
```

Adlar fəzasının komponentlərinə müraciət etmək üçün görünmə oblastı icazə operatorundan istifadə edilir. Müraciət sintaksisi növbəti şəkildədir:

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

namespace combat{
    void fire(){
        cout<<"Vistrel";
    }
}

namespace exploration{

    void fire(){
        cout<<"Ogon`";
    }
}

void main(){

    combat::fire();
    exploration::fire();
}
```

Qlobal adlar fəzası

Qlobal adlar fəzası — bu qlobal dəyişənlərin mövcud olduğu ən yüksək səviyyəli görünmə oblastıdır.

Qlobal fəza üzvlərinə müraciət etmək üçün çox zaman görünmə oblastı operatorundan (::) istifadə etmək lazım gəlir. Bu lokal və qlobal fəzada eyni adlı adların mövcud olması halında vacibdir, belə ki, susmaya görə həmişə ən kiçik görünmə oblastı olan dəyişən seçilir. Qlobal fəza üzvünə müraciət sintaksisi növbəti şəkildədir:

```
::qlobal_üzv;
```

Adlar fəazasının yenidən elan edilməsi

Əgər iki adlar fəzası eyni adda olarsa, ikinci birincinin məntiqi davamıdır.

Kompilyator növbəti şəkildə konstruksiyaya uyğun hesab edir:

```
namespace x
{
    func1(){}
}

namespace x
{
    func2{}
}
```



```
və

namespace x
{
    func1(){}
    func2(){}
}
```

C++ proqramın kompilyasiyası zamanı hər iki fəzanı bir görünmə oblastında yerləşdirir, bu o deməkdir ki, təkrarlanan adlar fəzasında eyniadlı iki komponent ola bilməz.

Adlar fəzasına birbaşa müraciət

Əgər siz əminsinizsə ki, fəza təkrarlanan üzvlər ehtiva etmir, onda onu qlobal adlar fəzası ilə birləşdirə bilərsiniz. Buna görə də adlar vəzası adını və görünmə oblastı icazə operatorunu həmişə yazma vacibliyi aradan qalxmış olur. Adlar fəzasına müraciətin iki üsulu mövcuddur:

1. using elanı
2. using direktivi

using elanı

using elanı, göstərir ki, iş daha aşağı səviyyəli görünmə oblastının müəyyən üzvü üzərində aparılacaq. Nəticədə görünmə oblastının aşkar göstərilməsinin vacibliyi aradan qalxmış olacaq. Using elan edilməsinin sintaksisi növbəti şəkildədir:

```
using istifadəçi_adı::üzv;

İstifadəyə aid nümunə:

#include <iostream>
using namespace std;
namespace dragon
{
    int gold=50;
}

void main()
{
    using dragon::gold;
    cout<<gold;
}
```

Using direktivi

using direktivi onu göstərir ki, iş konkret adlar fəzasının bütün üzvləri üzərində aparılacaq. Direktiv də elan kimi işləyir, yalnız yeganə fərqi fəzanın bütün üzvlərinə birbaşa müraciəti təşkil edir. Ondan istifadə etməklə siz proqramın sonuna qədər bu fəzanın üzvlərinin adlarını dəqiqləşdirməkdən azad olursunuz. Onun tətbiqindən əvvəl əmin olmaq lazımdır ki, daxil edilən adlar fəzasının qlobal fəzanın adlarını təkrarlayan adlar yoxdur. using direktivinin sintaksisi növbəti şəkildədir:

```
using namespace fəza_adı;
```


Bu konstruksiyanın istifadə nümunəsi ilə siz artıq rastlaşmısınız, o sizin yazdığınız bütün proqramlarda vardır:

```
using namespace std;
```

Adsız adlar fəzasının yaradılması

Gəlin indi aydınlaşdıraq, fəzaya mənimsədilən adın düzgünlüyünə necə zəmanət vermək olar, belə ki, bəzi adlar fəzasına eyni adlar mənimsədilmişdir. Bu cür konflikdən necə qurtulmaq olar, siz soruşun?! Bu nə qədər təəccüblü olsa da, adsız adlar fəzası yaratmaqladır!!! Bu onunla əlaqədardır ki, əgər adsız adlar fəzası yaradarıqsa, C++ ona avtomatik olaraq bizim görə bilmədiyimiz unikal ad verəcəkdir. Lakin, adın unikal olması danılmaz faktdir. Adsız adlar fəzasının elan edilməsi sintaksisi növbəti şəkildədir:

```
namespace
{
    Üzvlər;
}
```

Bu halda C++ adsız adlar fəzasının üzvlərinə :: operatoru vasitəsi ilə adları göstərmədən müraciət etməyə imkan verir. Sizin bu fəzanı elan etməyinizə ehtiyac yoxdur (hətta imkan da yoxdur, belə ki, adları siz bilmirsiniz). using namespace qeyri-aşkar elan edilməsi adlar fəzasının özünün elan edilməsindən sonra avtomatik əlavə edilir. Ona görə də, nəticədə kod növbəti şəkildə olur:

```
namespace a //ehtimal edildiyi kimi alınan ad
            //avtomatikdir (qeyri-aşkar)
{
    üzvlər;
}
using namespace a; //avtomatik yazılıan elan
                   //etmə
```

Adsız adlar fəzası ilə iş nümunəsinə baxaq:

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

namespace
{
    void func(){cout<<"::func"<<"\n";}
}

void main()
{
    ::func();
}
```

Son olaraq, qeyd etmək lazımdır ki, adsız adlar fəzası lokal xarakter daşıyır və yalnız onların elan edildikləri faylda isitifadə edilə bilərlər.

İmtahan tapşırıqları

Bütün tapşırıqların reallaşdırılması üçün OYP istifadə etməli.

1. “Poker” kart oyununu reallaşdırılmalı.
2. “Sudoku” yapon krasvodunu reallaşdıran proqramı yazmalı. İstifadəçi artıq krasfordda mövcud olan tapılması tələb olunan ilkin verilənləri daxil edir və proqram ekrana hazır nəticəni verir.
3. Elektron təşkilatçını reallaşdıran proqramı yazmalı. Verilənlərin əlavə edilməsi, silinməsi, redaktə edilməsi və saxlanılması imkanlarını reallaşdırmalı. Bütün mümükün səhvlərin emalını nəzərdə tutmalı.