Osnovi računarskih sistema - Izuzeci

Jelena Tomašević

Čas 1

Vežbe—— 01.04.2005

Izuzeci su anomalije koje program otkriva u vreme svog izvršavanja, kao što su na primer deljenje nulom, pristupanje nizu van njegovih granica i drugo. Obzirom da ovakvi izuzeci ne predstavljaju deo normalnog funkcionisanja programa, očekuje se da program trenutno reaguje na njih. Upravljanje izuzecima je mehanizam koji omogućava komunikaciju, odnosno prenošenje izuzetka između dva nepovezana (često nezavisno razvijana) dela programa. Jezik C++ poseduje već gotove alatke za proglašavanje izuzetaka kao i ispravno upravljanje njima.

Naime, kada se izuzetak uoči, deo programa koji ga je otrkrio može da proglašavanjem ili ispaljivanjem (engl. throwing) izuzetka saopšti da se taj izuzetak pojavio. Izuzetak se ispaljuje pomoću izraza za ispaljivanje koji se sastoji od ključne reči **throw** iza koje zatim dolazi izraz čiji tip odgovara tipu ispaljenog izuzetka. Sve naredbe koje mogu da ispale izuzetak moraju se nalaziti u okviru bloka try. Ovaj blok započinje ključnom rečju **try** iza koje, unutar vitičastih zagrada, sledi niz programskih naredbi. Iza bloka try sledi lista hvatača koji se nazivaju klauzule za hvatanje (engl. catch clause). Blok try povezuje skup naredbi koje mogu da ispale izuzetak sa skupom hvatača koji na odgovarajući način obrađuju te izuzetke. Klauzula za hvatanje se sastoji iz tri dela: ključne reči **catch**, deklaracije jednog tipa ili jednog objekta u zagradama (to se zove joši deklaracija izuzetka) i skupa naredbi u okviru jedne složene naredbe.

Kada dođe do izuzetka, sve naredbe koje dolaze iza naredbe koja je proizvela izuzetak se preskaču a program nastavlja izvršenje u klauzuli za hvatanje koja upravlja tim izuzetkom. Tada se izvršava složena naredba klauzule za hvatanje koja je izabrana a zatim se izvršavanje programa nastavlja u naredbi koja sledi iza poslednje klauzule za hvatanje u listi. Ukoliko ne postoji odgovarajuća klauzula

za hvatanje koja je u stanju da rukuje izuzetkom, izvršavanje se nastavlja u funkciji terminate() koja je definisana u standardnoj biblioteci jezika C++.

Blok try uvodi lokalnu oblast važenja tako da se promenljivim deklarisanim u bloku try ne može pristupati izvan tog bloka pa čak ni iz klauzula za hvatanje.

Postoji i hvatač (klauzula za hvatanje) koja obrađuje sve izuzetke i ona ima deklaraciju izuzetka u formi (...). U ovu klauzulu za hvatanje se ulazi za bilo koji tip izuzetka. Klauzule za hvatanje se uvek ispituju jedna po jedna onim redosledom kojim su navedene iza bloka try. Kada odgovarajuća klauzula bude pronađena ostale se ne ispituju. To znači da ako se klauzula catch(...) koristi u kombinaciji sa drugim klauzulama za hvatanje, ona se mora postaviti da bude poslednja u listi hvatača, inače će kompajler prijaviti grešku u fazi prevođenja.

Primer 1 *Primer koji ilustruje jednostavno upravljanje izuzecima.*

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

main()
{
    cout << "pre bloka try" << endl;
    try {
        cout << "pre throw" << endl;
        throw "namerni izuzetak";
        cout << "posle throw" << endl;
    }catch( char* s ){
        cout << "Doslo je do greske: " << s << endl;
    }catch(...){
        cout << "Nepoznata greska!" << endl;
    }
    cout << "posle bloka try" << endl;
    return 1;
}

/* Izlaz:
pre bloka try
pre throw
Doslo je do greske: namerni izuzetak
posle bloka try
*/
```

Primer 2 *Primer koji ilustruje funkciju koja proizvodi izuzetke i program koji njima upravlja.*

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

int f( int x )
{
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>10)
        throw "Parametar funkcije f je veci od 10!";
    return x * x;
}

main()
{
    cout << "pocetak" << endl;
    for( int i=-1; i<=12; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }
        catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
    cout << "kraj" << endl;
    return 1;
}

/* Izlaz:
pocetak
GRESKA: Parametar funkcije f je manji od 0!
f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9
f(4) = 16
f(5) = 25
f(6) = 36
f(7) = 49
f(8) = 64
```

```
f(9) = 81
f(10) = 100
GRESKA: Parametar funkcije f je veci od 10!
GRESKA: Parametar funkcije f je veci od 10!
kraj
*/
```

Primer 3 *Primer koji ilustruje korišćenje klauzule za hvatanje koja obrađuje sve izuzetke.*

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

int f( int x )
{
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>10)
        throw x;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 12 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
```

```
}

/* Izlaz:
pocetak
GRESKA: Parametar funkcije f je manji od 0!
f(0) = 0
f(1) = 1
f(2) = 4
f(3) = 9
f(4) = 16
f(5) = 25
f(6) = 36
f(7) = 49
f(8) = 64
f(9) = 81
f(10) = 100
Nepoznata greska!
kraj
*/
```

Primer 4 *Primer koji ilustruje kako se nakon ispaljivanja izuzetka u okviru neke funkcije ta funkcija napusta i pozivaju se podrazumevani destruktori objekata deklarisanih u okviru te funkcije.*

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

class A
{
public:
    A( int a )
        : _a(a)
        { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    // Ukoliko dodje do ispaljivanja izuzetka,
```

```
    // f-ja f() se napusta i vrsi se implicitno pozivanje
    // destruktora klase A radi unistavanja objekta a.
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>3)
        throw x;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

/* Izlaz:
pocetak
A::A(-1)
A::~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::~A(0)
f(0) = 0
A::A(1)
A::~A(1)
```

```
f(1) = 1
A::A(2)
A::~A(2)
f(2) = 4
A::A(3)
A::~A(3)
f(3) = 9
A::A(4)
A::~A(4)
Nepoznata greska!
kraj
*/
```

Primer 5 *Primer koji ilustruje kako se može desiti da usled ispaljivanja izuzetka neki resursi ostanu zauvek neoslobođeni.*

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

class A
{
public:
    A( int a )
        : _a(a)
        { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    //Ukoliko funkcija moze da dobije odredjeni resurs
    //(otvori datoteku ili da alocira memoriju na hipu)...
    A* p = new A(100+x);
    //...i ukoliko dodje do ispaljivanja izuzetka, izlazi se iz funkcije...
    if( x<0 )
        throw "Parametar funkcije f je manji od 0!";
    if( x>3)
        throw x;
    //...i oslobadjanje ovog resursa se nece obaviti.
```

```
    delete p;
    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}

/* Izlaz:
pocetak
A::A(-1)
A::A(99)
A::~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::A(100)
A::~A(100)
A::~A(0)
f(0) = 0
A::A(1)
A::A(101)
A::~A(101)
A::~A(1)
```

```
f(1) = 1
A::A(2)
A::A(102)
A::~A(102)
A::~A(2)
f(2) = 4
A::A(3)
A::A(103)
A::~A(103)
A::~A(3)
f(3) = 9
A::A(4)
A::A(104)
A::~A(4)
Nepoznata greska!
kraj
*/
```

Primer 6 *Primer koji ilustruje kako se korišćenjem hvatača koji obrađuje sve izuzetke može sprečiti da usled ispaljivanja izuzetka neki resursi ostanu zauvek neoslobođeni.*

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

class A
{
public:
    A( int a )
        : _a(a)
        { cerr << "A::A(" << _a << ")" << endl; }
    ~A()
        { cerr << "A::~A(" << _a << ")" << endl; }

private:
    int _a;
};

int f( int x )
{
    A a(x);
    A* p = new A(100+x);
    try {
        if( x<0 )
```

```
            throw "Parametar funkcije f je manji od 0!";
        if( x>3)
            throw x;
        cout << "Poziv destruktora u bloku try" << endl;
        delete p;
        }
    catch(...){
        cout << "Poziv destruktora u hvatacu izuzetaka" << endl;
        // Obezbedjuje da se memorija na koju pokazuje p oslobodi
        // i u slucaju ispaljivanja izuzetka.
        delete p;
        //Ispaljuje izuzetak dalje prosledjujuci ga drugoj
        //klauzuli za hvatanje radi obradjivanja.
        throw;
    }

    return x * x;
}

void g( int a, int b)
{
    for( int i=a; i<=b; i++ ){
        try {
            int y = f(i);
            cout << "f(" << i << ") = " << y << endl;
        }catch( char* s ){
            cerr << "GRESKA: " << s << endl;
        }
    }
}

main()
{
    cout << "pocetak" << endl;
    try {
        g( -1, 5 );
    }catch(...){
        cerr << "Nepoznata greska!" << endl;
    }
    cout << "kraj" << endl;
    return 1;
}
```

```
/* Izlaz:
pocetak
A::A(-1)
A::A(99)
Poziv destruktora u hvatacu izuzetaka
A::~A(99)
A::~A(-1)
GRESKA: Parametar funkcije f je manji od 0!
A::A(0)
A::A(100)
Poziv destruktora u bloku try
A::~A(100)
A::~A(0)
f(0) = 0
A::A(1)
A::A(101)
Poziv destruktora u bloku try
A::~A(101)
A::~A(1)
f(1) = 1
A::A(2)
A::A(102)
Poziv destruktora u bloku try
A::~A(102)
A::~A(2)
f(2) = 4
A::A(3)
A::A(103)
Poziv destruktora u bloku try
A::~A(103)
A::~A(3)
f(3) = 9
A::A(4)
A::A(104)
Poziv destruktora u hvatacu izuzetaka
A::~A(104)
A::~A(4)
Nepoznata greska!
kraj
*/
```