



Tema 07

Upravljanje izuzecima u jeziku C++

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



Sadržaj

1. Uvod
2. Rukovanje izuzecima
3. Objekti klasa kao izuzeci
4. Funkcije koje prijavljuju izuzetke
5. Standardna biblioteka izuzetaka
6. Primeri



1. Uvod

- Obrada grešaka u programu
- Upotreba izuzetaka



Obrada grešaka u programu

- Programi se pišu tako da se predvide različite varijante njihovog izvršavanja, ustanove najčešće greške u njihovom radu i isprave se na mestu otkrivanja
 - primeri grešaka su pogrešno uneti podaci, nedozvoljene vrednosti argumenata funkcija i različite greške u računanju vrednosti izraza, npr.

```
double deljenje(float brojilac, float imenilac) {  
    if (imenilac == 0) {  
        cout << "GREŠKA: deljenje s nulom nije definisano!\n";  
        return 0; ← dvosmislen rezultat  
    }  
    else  
        return static_cast<double> brojilac / imenilac;  
}
```

- bolji *tradicionalni* način rukovanja greškama je vraćanje *koda greške* iz pozvane funkcije i prepuštanje odluke o daljim koracima funkciji pozivniku

Upotreba izuzetaka

- Poseban način rukovanja greškama koje se ne očekuju u normalnom izvršavanju programa je pomoću *izuzetaka* (*exceptions*), npr.

```
double deljenje(float brojilac, float imenilac) {  
    if (imenilac == 0)  
        throw "GREŠKA: deljenje s nulom nije definisano!\n";  
    else  
        return static_cast<double> brojilac / imenilac;  
}
```

- uzrok ovakvih greški najčešće nije u kodu programa, već u kodu koji programeru nije dostupan (biblioteke), pa se uobičajeni načini otklanjanja grešaka ne mogu koristiti
- prednost upotrebe izuzetaka za prijavu grešaka je potpuno odvajanje koda koji otklanja grešku od onog koji je grešku prouzrokovao

2. Rukovanje izuzecima

1. Prijava izuzetka
2. Rukovanje izuzecima
3. Kod koji prijavljuje izuzetak
4. Ugnježdeni blokovi za rukovanje izuzecima



2.1 Prijava izuzetka

- **Izuzetak** u jeziku C++ je privremeni objekt *bilo kog tipa* koji se koristi za prijavu greške
 - izuzetak može biti osnovnog tipa, npr. `int` ili `const char*`, ali se češće definiše kao objekt tipa neke *klase*
- Kad se u programu otkrije neki problem, *prijavljuje* se ili *baca* izuzetak (*throwing exception*)
- Kontrola se prenosi u poseban blok koda koji prihvata (*catch*) i obrađuje izuzetak
- Programski kod koji može prijaviti izuzetak mora biti u posebnoj vrsti programskog bloka (*try block*)

Sintaksa bloka za prijavu izuzetka

- Blok naredbi koji može da prijavi izuzetak i naredbe za prihvatanje i obradu izuzetka strukturiraju se kao:

```
try {  
    // Kod programa koji može da prijavi izuzetak  
}  
catch (parametar koji navodi izuzetak tipa 1) {  
    // Kod za rukovanje izuzetkom  
}  
...  
catch (parametar koji navodi izuzetak tipa n) {  
    // Kod za rukovanje izuzetkom  
}
```

- programski kod u blokovima za rukovanje izuzecima (*catch*) izvršava se samo u slučaju pojave izuzetka *odgovarajućeg tipa*

Prijava izuzetaka

- Jednostavan primer prijave izuzetaka pomoću osnovnog tipa podataka:

```
try {  
    // Kod koji može da prijavi izuzetak je u try bloku  
    if (test > 5)  
        throw "test je veće od 5"; // prijava izuzetka tipa const char*  
    // Ostatak programa koji se izvršava ako nije prijavljen izuzetak  
}  
catch(const char* poruka) {  
    // Kod za rukovanje izuzetkom, koji se izvršava  
    // ako je prijavljen izuzetak tipa 'char*' ili 'const char*'  
    cout << poruka << endl;  
}
```

- ako je vrednost promenljive `test > 5`, prekida se normalno izvršavanje programa i obrađuje izuzetak (poruka o grešci "test je veće od 5")

Primer: Upotreba osnovnih tipova podataka za prijavu izuzetaka

```
#include <iostream>
using namespace std;
int main() {
    for (size_t i=0; i < 7 ; ++i) {
        try {
            if (i < 3) throw i;
            cout << "Nije prijavljen izuzetak za i= " << i << endl;
            if (i > 5) throw "Jos jedan izuzetak!";
            cout << "Kraj try bloka." << endl;
        }
        catch (size_t i) { // hvatanje izuzetka tipa size_t
            cout << "Uhvacen izuzetak za i= " << i << std::endl;
        }
        catch (const char* poruka){ // hvatanje izuzetka tipa char*
            cout << " \"" << poruka << "\" uhvacen" << endl;
        }
        cout << "Kraj petlje nakon catch blokova, i= " << i << endl;
    }
    return 0;
}
```

```
Uhvacen izuzetak za i= 0
Kraj petlje nakon catch blokova, i= 0
Uhvacen izuzetak za i= 1
Kraj petlje nakon catch blokova, i= 1
Uhvacen izuzetak za i= 2
Kraj petlje nakon catch blokova, i= 2
Nije prijavljen izuzetak za i= 3
Kraj try bloka.
Kraj petlje nakon catch blokova, i= 3
Nije prijavljen izuzetak za i= 4
Kraj try bloka.
Kraj petlje nakon catch blokova, i= 4
Nije prijavljen izuzetak za i= 5
Kraj try bloka.
Kraj petlje nakon catch blokova, i= 5
Nije prijavljen izuzetak za i= 6
"Jos jedan izuzetak!" uhvacen
Kraj petlje nakon catch blokova, i= 6
```

2.2 Rukovanje izuzecima

- Proces rukovanja izuzecima odvija se kao na prikazu:

```
try {  
    ...  
    throw izuzetak  
    ...  
}  
catch(Tip1 iz) {  
    ...  
}  
catch(TipN iz) {  
    ...  
}  
...
```

1. Prijava pomoću izraza *throw* inicijalizuje privremeni objekat izuzetak:

TipIzuzetka temp { izuzetak }

2. Za sve automatske objekte definisane u *try* bloku poziva se destruktork

4. Kopija izuzetka se koristi za inicijalizaciju parametra kao *TipN iz = temp*

3. Bira se prvi blok za rukovanje čiji se parametar slaže s tipom izuzetka

5. Nakon obrade izuzetka, izvršavanje se nastavlja iza poslednje bloka za rukovanje izuzecima, ako u kodu nije određeno drugačije (*goto* ili *return*)

- blokovi naredbi sadrže lokalne promenljive. Napuštanje bloka ih uklanja, što uključuje i objekat izuzetak, pa ga je potrebno kopirati (mora biti takvog tipa da se može kopirati)

Neobrađeni izuzeci

- Ako nije pronađen kod za rukovanje (*catch* blok) za neki prijavljeni izuzetak pokreće se funkcija `std::terminate()` definisana u zaglavlju `<exception>`
 - ova funkcija poziva *podrazumevanu* funkciju za rukovanje izuzetkom `std::abort()` definisanu u zaglavlju `<cstdlib>`, koja odmah terminira program i ne poziva destruktore automatskih i statičkih objekata
 - bolji način je promena podrazumevane funkcije u drugu *standardnu* funkciju `exit()` pomoću funkcije `set_terminate()`, npr.

```
void myHandler() {  
    exit(1); // oslobađa memoriju pre terminiranja  
}  
  
...  
terminate_handler pOldHandler = set_terminate(myHandler);
```

2.3 Kod koji prijavljuje izuzetak

- Kod koji prijavljuje izuzetak može se samo logički nalaziti u **try** bloku, a fizički može biti bilo gde u programu
- Isti programski kod (funkcija) može biti pozvan iz različitih **try** blokova, tako da se izuzetak koji se prijavljuje može, u različito vreme, obraditi u sasvim različitim **catch** blokovima



2.4 Ugnježdeni blokovi za rukovanje izuzecima

- Blokovi naredbi za prijavu izuzetaka mogu se gnezditi
- Svaki **try** blok za prijavu izuzetaka ima sopstvene blokove za njihovu obradu u kojima se prvo traži odgovarajući **catch** blok
- Ukoliko se odgovarajući **catch** blok za obradu izuzetka ne pronađe u unutrašnjem **try** bloku, traži se u spoljašnjem bloku



Primer: Ugnježdeni blokovi za prijavu izuzetaka (1/2)

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

void throwIt(int i) {
    throw i; // prijavljuje vrednost parametra
}

int main() {
    for (int i=0; i <= 5; ++i) {
        try {
            cout << "Spoljni try:\n";
            if (i == 0)
                throw i; // prijava int izuzetka
            if (i == 1)
                throwIt(i); // poziv funkcije koja prijavljuje int
            try { // ugnježdeni try blok
                cout << " Unutrasnji try:\n";
                if (i == 2)
                    throw static_cast<long>(i); // prijava long izuzetka
                if (i == 3)
                    throwIt(i); // poziv funkcije koja prijavljuje int
            } // kraj ugnježdenog try bloka
        }
    }
}
```


Primer: Ugnježdeni blokovi za prijavu izuzetaka (2/2)

```
catch (int n) {  
    cout << " Catch int za unutrasnji try. "  
        << "Izuzetak " << n << endl;  
}  
cout << "Spoljni try:\n";  
if (i == 4)  
    throw i; // prijava int  
throwIt(i); // poziv funkcije koja prijavljuje int  
}  
catch (int n) {  
    cout << "Catch int za spoljni try. "  
        << "Izuzetak " << n << endl;  
}  
catch (long n) {  
    cout << "Catch long za spoljni try. "  
        << "Izuzetak " << n << endl;  
}  
}  
return 0;  
}
```

```
Spoljni try:  
Catch int za spoljni try. Izuzetak 0  
Spoljni try:  
Catch int za spoljni try. Izuzetak 1  
Spoljni try:  
Unutrasnji try:  
Catch long za spoljni try. Izuzetak 2  
Spoljni try:  
Unutrasnji try:  
Catch int za unutrasnji try. Izuzetak 3  
Spoljni try:  
Catch int za spoljni try. Izuzetak 3  
Spoljni try:  
Unutrasnji try:  
Spoljni try:  
Catch int za spoljni try. Izuzetak 4  
Spoljni try:  
Unutrasnji try:  
Spoljni try:  
Catch int za spoljni try. Izuzetak 5
```


3. Objekti klasa kao izuzeci

1. Prihvatanje izuzetka
2. Prihvatanje izuzetaka izvedene klase rukovaocem osnovne klase
3. Ponovljeno prijavljivanje izuzetaka (rethrowing)
4. Prihvatanje svih izuzetaka



3.1 Prihvatanje izuzetka

- Izuzetak je objekt bilo koje klase, čija je osnovna namena *prenos informacija* programu za obradu izuzetka
- Može se definisati posebna *korisnička klasa izuzetaka*, koja može da sadrži informacije o problemu, kod greške i neke dodatne informacije, npr.

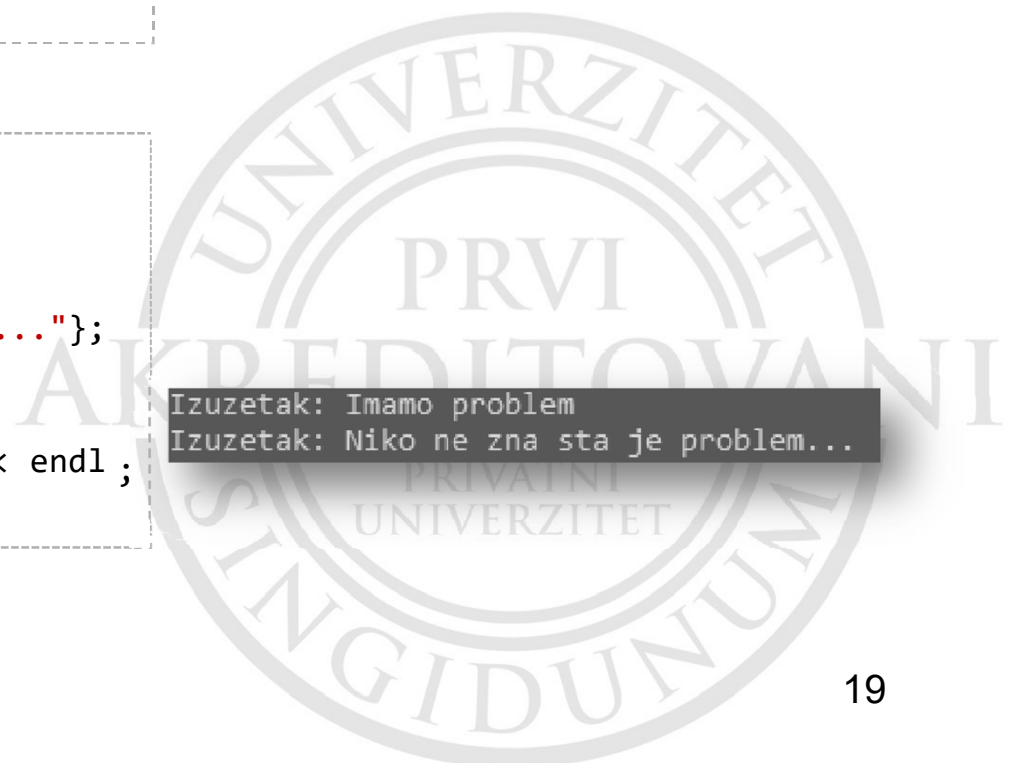
```
class Problem {  
    private:  
        string poruka;  
    public:  
        Problem(string s="Imamo problem") : poruka {s} {}  
        string prikaziProblem() const { return poruka; }  
};
```

Primer: Upotreba korisničke klase izuzetaka

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

class Problem {
private:
    string poruka;
public:
    Problem(string s="Imamo problem") : poruka {s} {}
    string prikaziProblem() const { return poruka; }
};

int main() {
    for(int i=0; i < 2 ; ++i) {
        try {
            if (i == 0)
                throw Problem {};
            else
                throw Problem {"Niko ne zna sta je problem..."};
        }
        catch (const Problem& t) {
            cout << "Izuzetak: " << t.prikaziProblem() << endl;
        }
    }
    return 0;
}
```



```
Izuzetak: Imamo problem
Izuzetak: Niko ne zna sta je problem...
```

Proces prihvatanja izuzetka

- Prilikom prihvatanja izuzetka neophodno je *poklapanje* tipa izuzetka i parametra u **catch** bloku
- Prilikom poklapanja su moguće situacije:
 - tip parametra je *isti* kao tip izuzetka (ako se zanemari oznaka **const**)
 - tip parametra je *osnovna klasa* tipa klase izuzetaka ili *referenca* na osnovna klasu tipa klase izuzetaka, direktno ili indirektno (ako se zanemari oznaka **const**)
 - izuzetak i parametar su *pokazivači*, a tip izuzetka se može *implicitno konvertovati* u tip parametra (ako se zanemari oznaka **const**)
- Ako postoji hijerarhija klasa izuzetaka, potrebno je prvo navesti blok za najnižu izvedenu klasu, a na kraju za osnovnu klasu, inače će se uvek poklopiti osnovna, a ne izvedena klasa

3.2 Prihvatanje izuzetaka izvedene klase rukovaocem osnovne klase

- Izuzeci izvedene klase izuzetaka implicitno se konvertuju u tip *osnovne* klase radi poklapanja parametra **catch** bloka, tako da se više prijavljenih izuzetaka može uhvatiti jednim rukovaocem
- Parametar se inicijalizuje pomoću konstruktora kopije *osnovne* klase, čime se gube svojstva *izvedene* klase
- Pojava gubitka informacija o izvedenim klasama naziva se *object slicing* i predstavlja opšti izvor grešaka prilikom prenošenja objekata po vrednosti
- Zbog toga je u **catch** blokovima neophodno uvek koristiti parametre tipa *reference*

3.3 Ponovljeno prijavljivanje izuzetaka (*rethrowing*)

- Kada rukovalac unutrašnjeg bloka prihvati izuzetak, može ga ponovo prijaviti radi omogućavanja obrade izuzetka u spoljašnjem bloku pomoću posebne forme naredbe, koja sadrži samo reč
`throw;`
- Tada se ponovo prijavljuje tekući izuzetak, odnosno *postojeći objekt* izuzetka, bez ponovnog kopiranja, što je važno ako je u pitanju izvedena klasa izuzetaka, koja inicijalizuje parametar osnovne klase (referencu)
- Ponovljenu prijavu izuzetka ne koriste ostali rukovaoci unutrašnjeg bloka

3.4 Prihvatanje svih izuzetaka

- Specifikacija parametara pomoću *tri tačke* u *catch* bloku (...) označava da blok može da rukuje bilo kojim izuzetkom:

```
try {  
    // Programski kod koji može da prijavi izuzetak  
}  
catch (...) {  
    // Programski kod za rukovanje bilo kojim izuzetkom  
}
```

- Rukovalac koji prihvata sve izuzetke treba da bude poslednji u bloku

4. Funkcije koje prijavljuju izuzetke

1. Funkcije za prijavu izuzetaka
2. Funkcije koje ne prijavljuju izuzetke
3. Prihvatanje izuzetaka u konstruktoru
4. Izuzeci i destruktori



4.1 Funkcije za prijavu izuzetaka

- Svaka funkcija može da prijavi izuzetak, uključujući konstruktore klasa. Izuzetak koji funkcija prijavi može se prihvatiti u funkciji *pozivniku*, ali je tada potrebno da se u samoj funkciji *ne* prihvati ili se ponovo prijavi (*rethrow*)
 - izuzetak je potrebno negde prihvatiti da se spreči terminiranje programa zbog neobrađenih izuzetaka
 - poziv funkcije koja prijavljuje izuzetke neophodno je zatvoriti u **try** blok koji prihvata izuzetke
 - telo funkcije može da sadrži **try** i **catch** blokove. Svaki izuzetak koji nije prihvaćen prosleđuje se do tačke gde je funkcija pozvana
- Ponekad je pogodno da celo telo funkcije bude **try** blok sa skupom rukovalaca (*function try block*)

Ilustracija: Funkcionalni blokovi za prihvatanje izuzetaka (*function try blocks*)

- Ključna reč **try** može se postaviti *ispred*, a **catch** blokovi *iza* tela funkcije, tako da je kompletno telo funkcije *try-catch* blok
 - ako se izvršavanje programa ne prekida, **catch** blok treba da završi naredbom **return**
 - za tip **void** kraj **catch** bloka je ekvivalent izvršavanja naredbe **return**

```
void funkcijaUradi(int argument)
try {
    // Telo funkcije (kod)
}
catch (VelikiProblem& iz){
    // Obrada izuzetka VelikiProblem
}
catch (VeciProblem& iz){
    // Obrada izuzetaka VeciProblem
}
catch (Problem& iz) {
    // Obrada izuzetaka tipa Problem
}
```

4.2 Funkcije koje ne prijavljuju izuzetke

- Pomoću specifikacije `noexcept` u zaglavlju funkcije može se označiti da funkcija prihvata, a ne ponavlja prijave izuzetaka, npr.

```
void funkcijaUradi(int argument) noexcept
try {
    // Programski kod funkcije
}
catch ( ... ) {
    // Obrada svih izuzetaka (ne prijavljuju se ponovo)
}
```

- Funkcija obrađuje sve izuzetke koje može da uhvati. Ako se *prijavi* izuzetak koji nije uhvaćen u funkciji, neće se proslediti funkciji pozivniku, već će se odmah pozvati `std::terminate()`

4.3 Prihvatanje izuzetaka u konstruktoru

- Konstruktorske funkcije mogu da prijave izuzetak i bez eksplicitne naredbe **throw** u telu konstruktora
 - npr. zbog upotrebe funkcija iz standardne biblioteke ili nekih funkcija klase **string**, koje *mogu da prijave* izuzetke, ako konstruktor koristi **new**
- Ako je izuzetak prijavljen i prihvaćen u konstruktoru, **catch** blok treba da ponovi prijavu izuzetka (*rethrow*), da se pozivnik obavesti o tome da objekt nije bio izgrađen
- Ako izvršavanje programa dođe do kraja **catch** bloka bez ponavljanja prijave izuzetka, ponoviće se prijava originalnog izuzetka

Izuzetak u listi inicijalizacije

- Ako postoji mogućnost da konstruktor prilikom inicijalizacije prijavi izuzetak, *lista inicijalizacija* se postavlja u **try** blok, koji se navodi odmah nakon liste parametara, npr.

```
Primer::Primer(int brojac) try : OsnovnaKlasa(brojac) {  
    // Konstruktor osnovne klase može da prijavi izuzetak  
}  
catch(...) // blok prihvata sve izuzetke {  
    // Obrada izuzetka  
    throw; // nije neophodno  
}
```

- Klasa je izvedena iz klase osnovne klase, pa konstruktor klase **Primer** poziva konstruktor osnovne klase u *listi inicijalizacije*

4.4 Izuzeci i destruktori

- Destruktori ne treba da prijavljuju izuzetke, po definiciji su `noexcept`, tako da pojava izuzetka terminira program
- Za automatski kreirane objekte može se, prilikom pojave izuzetka, dogoditi da se destruktor klase pokrene *pre* obrade izuzetka u `catch` bloku
- Destruktor može sam da ustanovi situaciju da je pozvan usled pojave izuzetka pre izvršenja `catch` bloka funkcijom `std::uncaught_exceptions()`
- Ako je neophodno sprečavanje pojave izuzetaka izvan granica destruktora, kod destruktora se može zatvoriti u `try` blok čiji `catch` blok prihvata sve izuzetke

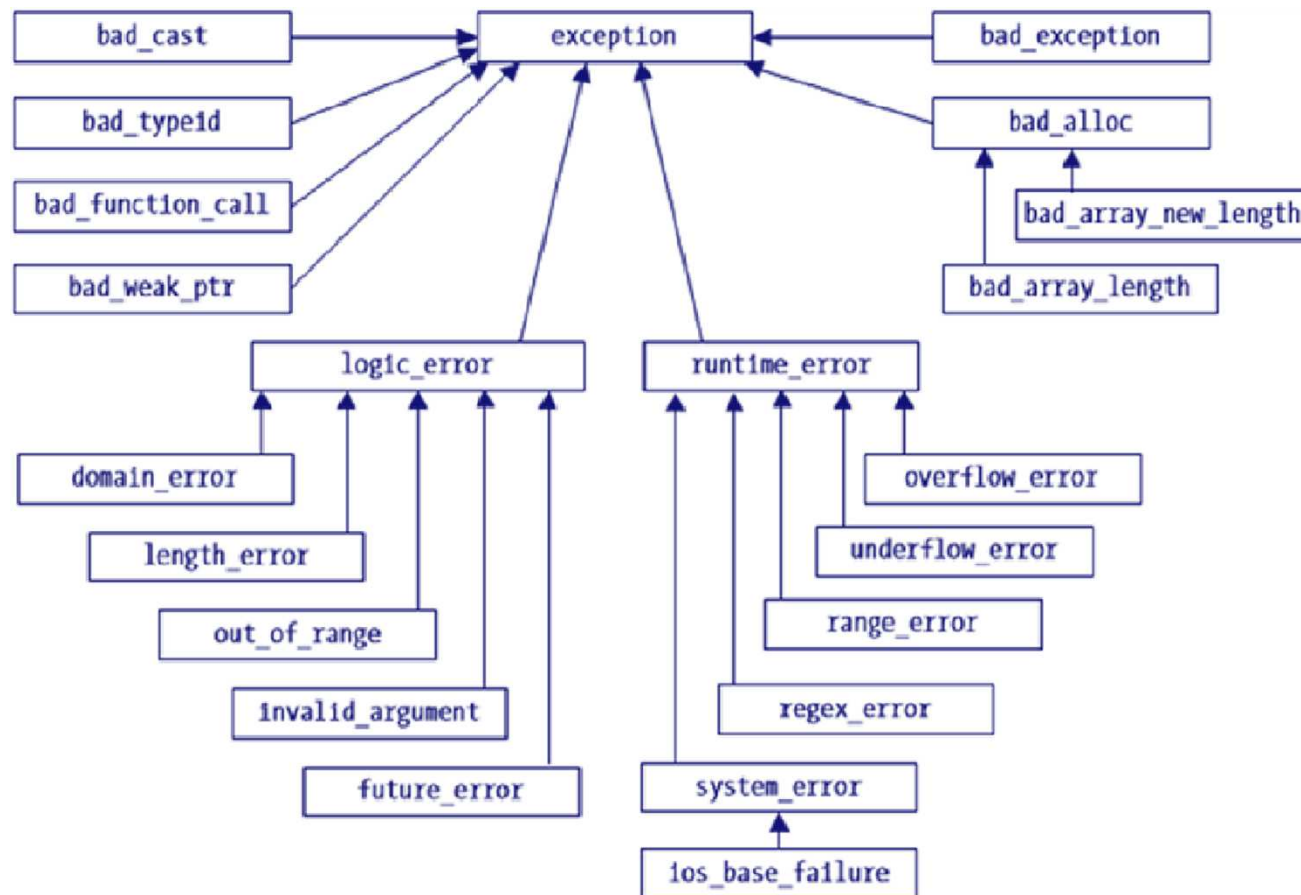
5. Standardna biblioteka izuzetaka

1. Standardni tipovi klasa izuzetaka
2. Upotreba standardnih izuzetaka



5.1 Standardni tipovi klasa izuzetaka

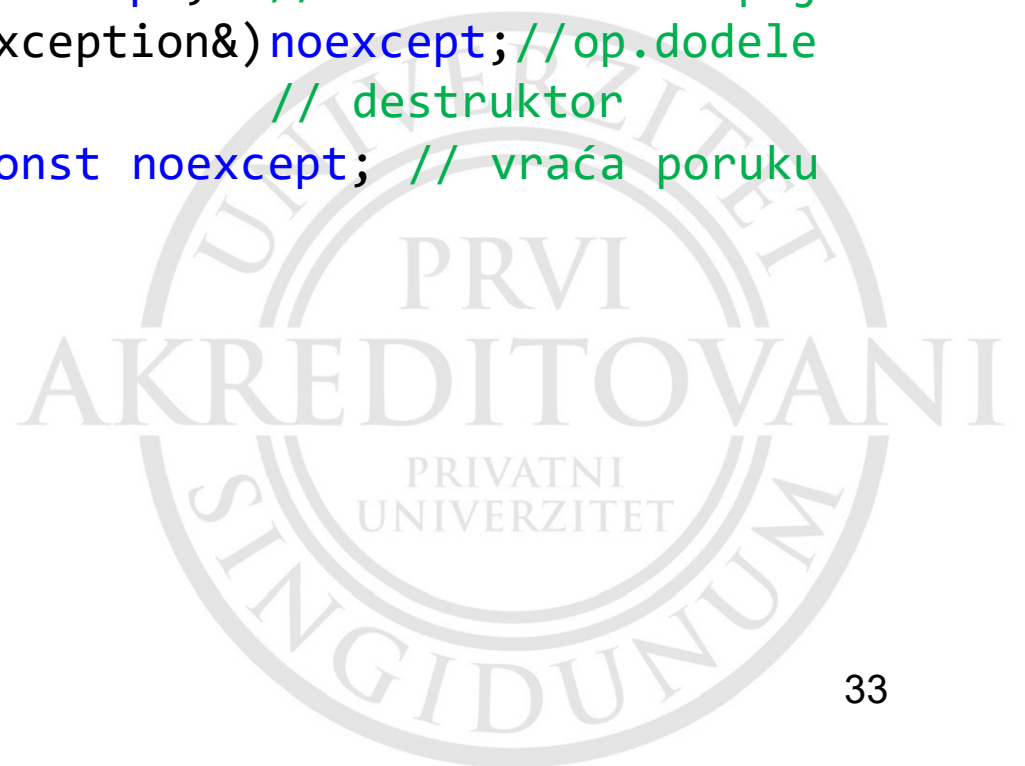
- Standardna biblioteka sadrži definicije nekih klasa izuzetaka, a izvedene klase definisane su u zaglavlju `<stdexcept>`



Definicija osnovne klase izuzetaka

- Klasa izuzetaka `exception` može se koristiti za izvođenje sopstvenih klasa izuzetaka, koje nasleđuju sve članove:

```
class exception {  
    public:  
        exception() noexcept;           // podraz. konstruktor  
        exception(const exception&)noexcept; // konstruktor kopije  
        exception& operator=(const exception&)noexcept; // op. dodele  
        virtual ~exception();           // destruktor  
        virtual const char* what() const noexcept; // vraća poruku  
};
```



5.2 Upotreba standardnih izuzetaka

- Standardne klase izuzetaka mogu se koristiti na dva načina:
 - za prijavu standardnih tipova izuzetaka
 - za kreiranje sopstvenih klasa izuzetaka, koje se izvode iz standardnih
- Npr. u konstruktoru klase **Kutija** može se prijaviti izuzetak *range_error* ako se zadaju nedozvoljene dimenzije kutije:

```
Kutija::Kutija(double d, double s, double v) : duzina {d},  
sirina {s}, visina {v} {  
    if (d <= 0.0 || s <= 0.0 || v <= 0.0)  
        throw range_error("Dimenzije kutije  
        <= 0");  
}
```

- U program treba uključiti zaglavlje **<stdexcept>** u kome je definisana klasa *range_error*

Kreiranje sopstvene klase izuzetaka

- Sopstvena klasa izuzetaka može se izvesti iz standardnih klasa izuzetaka, tako da se izuzeci sopstvene klase mogu hvatati u okviru istog `catch` bloka kao i standardni izuzeci
- Npr. za klasu `Kutija` može se iz standardne klase `range_error` izvesti klasa izuzetaka `Dim_error`, s detaljnijim opisom greške

```
#include <stdexcept>
#include <string>
using namespace std;
class Dim_error : public range_error {
public:
    using range_error::range_error; // nasleđ. osn. konstr.
    Dim_error(string s, int d) range_error{s+to_string(d)};
}
```

detaljniji opis greške: tekst i pogrešna vrednost

6. Primeri

- Izuzeci kod prenosa argumenata



Izuzeci kod prenosa argumenata [4]

```
// Izuzetak za nedozvoljenu
// vrednost argumenta funkcije

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

// Funkcija računa površinu kruga
double getPovrsina(double r) {
    if (r < 0)
        throw invalid_argument(
            "Poluprecnik ne može biti < 0!");

    return r * r * 3.14159;
}
```

```
int main() {
    // Unos poluprečnika
    cout << "Unesite poluprecnik: ";
    double popuprecnik;
    cin >> poluprecnik;

    try {
        double P=getPovrsina(poluprecnik);
        cout << "Povrsina= " << P << endl;
    }
    catch (exception& izuzetak){
        cout << izuzetak.what() << endl;
    }

    cout << "Kraj." << endl;
    return 0;
}
```

```
Unesite poluprecnik: -5
Poluprecnik ne može biti < 0!
Kraj.
```

Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Liang D., *Introduction to Programming With C++*, 3rd Ed, Pearson Education, 2014
5. Horton I., *Beginning C++*, Apress, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. Horstmann C., *Big C++*, 2nd Ed, John Wiley&Sons, 2009
8. Veb izvori
 - <http://www.cplusplus.com/doc/tutorial/>
 - <http://www.learncpp.com/>
 - <http://www.stroustrup.com/>
9. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

