# Objektno orjentisano programiranje – C++

## Izuzetci

# Teme

- Pojam i definicija izuzetaka
- Uzrok izuzetaka
- Rukovanje izuzetcima
- Blokovi pokušaja i hvatanja izuzetaka
- *Try-catch* sintaksa
- Redosled navođenja blokova hvatanja
- Navođenje liste mogućih izuzetaka

# Šta su izuzetci?

- **Izuzetci** (engl. ***exceptions***) su situacije u kojima program ne može nastaviti svojim normalnim tokom već je potrebno prekinuti nit izvođenja te izvođenje preneti na neku posebnu rutinu koja će obraditi novonastalu situaciju.
- Rukovanje izuzetcima
- Odstupanje od predviđenog toka programa

# Uzrok izuzetka

- Posledica greške u radu računara
- Situacija koja je u programu nastala usled izvođenja programa
- Problemi s neispravnim pristupom korisničkim podacima

# Primer 1: Mogući problemi pri izvođenju programa

```
int main () {
    int mat [10];
    mat [130]=0;
    return 0;
}
```

# Primer 2: Moguće rešenje

```
niz a [10];
a [0]=80;
a [130]=0;
if (a.JeLIGreska ())
    cout << “Greska u pristupu nizu a.“<< endl;
```

Nedostatak – potreba da se eksplicitno proverava ispravnost nakon svakog pristupa objektu

# Rešenje problema – rukovanje izuzetcima

- Izuzetak – događaj u računaru koji sprečava normalan nastavak izvođenja programa i zahteva posebnu obradu

# Rukovanje izuzetcima - hronologija

1. Detektuje se situacija koja zahteva prekid normalnog izvođenja programa
2. Program **podiže izuzetak (*to raise an exception)***
3. Prekida se izvođenje programa
4. Izuzetak se prosleđuje rutini za **oporavak od izuzetka (*exception recovery routine)***

# Blokovi pokušaja i hvatanja izuzetaka

- Celi C++ program se razbija u niz blokova koji sadrže neke operacije koje bi mogle biti problematične prilikom izvođenja programa – **blokovi pokušaja (*try bloks*)**
- Ključna reč ***try*** – iza nje se navode problematične instrukcije
- Ključna reč ***catch*** – blok za obradu izuzetka (***hvata*** odgovarajući izuzetak iz bloka pokušaja)

# *Try-catch* sintaksa

```
try {
      // ovo je blok pokušaja
}
catch (parametar) {
     // ovo je blok hvatanja
}
```

# Primer: *try-catch*

```
niz a [10];
try {
    a [130]=0;
}
catch (NizovIzuzetci &gres) {
    cerr << gres.Greska () << endl;
}
```

# Primer 2: *try-catch*

```
try {
    // problematičan dio koda
}
catch (NizoviIzuzetci &gres) {
    cerr << gres.Greska () << endl;
}
catch (SkupoviIzuzetci &gres) {
    // obrada gresaka koje je izazvala klasa skup
}
```

Svaki blok hvatanja (***catch***) hvataće izuzetke samo određenog tipa

# Redosled navođenja blokova hvatanja:

```
try {
    // naredbe
}
catch (void *pok) {
    // svi pokazivački izuzetci će završiti ovdje
}
catch (char *pok) {
    // ovaj blok hvatanja se nikada neće dohvatiti, pa čak
    // ako se i baci objekt tipa char * jer se, pomoću
    // standardne konverzije pokazivača može svesti na
    // void * pa se ovaj blok neće nikada ni razmatrati
}
```

# *try-catch*

```
try {
    // tip problematičnog objekta
    // problematičan dio koda
}
catch (tip1 parametra) {
    // telo bloka
}
catch (tip2 parametra) {
    // telo bloka
}
catch (tip3 parametra) {
    // telo bloka
}
```

# *try-catch* (2)

- Iza pokušajnog bloka može se navesti više blokova hvatanja.
- Svaki blok hvatanja može definisati svoj tip i na taj način obraditi odgovarajuće izuzetke.
- Određivanje koji blok hvatanja će se pokrenuti se izvodi tako da se blokovi pretražuju u redosledu navođenja nakon bloka pokušaja

# Bačeni objekat će se predati bloku hvatanja ako je zadovoljen jedan od uslova:

1. Tip objekta i tip argumenta bloka su potpuno jednaki. U tom slučaju se ne provodi nikakva konverzija.
2. Tip parametra je klasa koja je javna osnovna klasa bačenog objekta. Tada se bačeni objekt svodi na objekt osnovne klase.
3. Ako je bačeni objekt pokazivač i ako je argument bloka pokazivač. Podudaranje će se postići ako se bačeni pokazivač može svesti na pokazivač parametra standardnom konverzijom pokazivača.

# Blok hvatanja koji preuzima sve izuzetke:

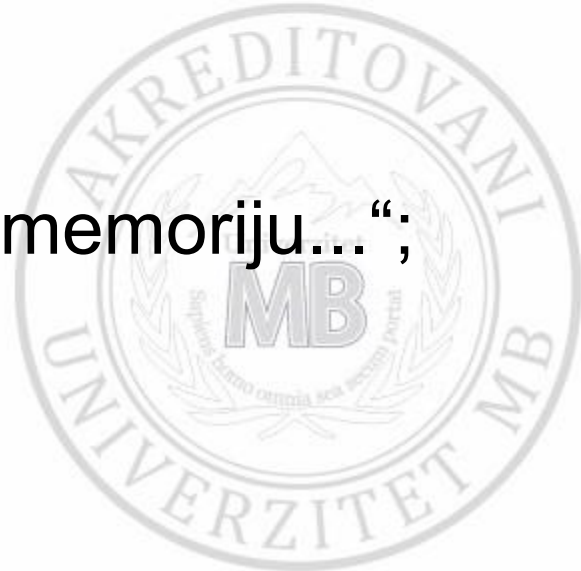
```
try {
    // naredbe
}
catch (...) {
    // svi pokazivački izuzetci će završiti ovde
}
catch (char *pok) {
    // ovaj neće nikada biti izvršen
}
```

- Ako postoji više blokova za hvatanje, ovakav blok mora doći kao poslednji.

# Drugi način podizanja izuzetka

- **Bacanje izuzetka** (***throwing an exception***)
- Ključna riječ ***throw***

```
char *Zn = new char [10000];
if (!Zn) throw “Ne mogu alocirati memoriju...“;
```



# Primer: *throw*

```
inline int &Niz :: operator (int indeks) {
    if (0 <= indeks && indeks <= duzina)
        return pokNiz (indeks);
    else
        throw NizovIzuzetci (“Pogresan pristup nizu.“);
}
```

# Navođenje liste mogućih izuzetaka

- U složenim programima često nije moguće pratiti koji izuzetak se baca u pojedinoj funkciji.
- Zbog toga se uvodi mehanizam kojim svaka funkcija obaveštava ostatak programa koje izuzetke ona može baciti.
- Lista mogućih izuzetaka se navodi iza liste parametara funkcije tako da se navede ključna riječ throw iza koje se u zagradama popišu svi tipovi izuzetaka koje ta funkcija može baciti.

# Navođenje liste mogućih izuzetaka (2)

```
void Funkc1 () throw (int, const char*, NizoviIzuzetci) {
        // definicija funkcije
}
Void Funkc2 () throw () {
        // funkcija koja ne baca nikakav izuzetak
}
```

# Navođenje liste mogućih izuzetaka (3)

- Ako slučajno neki izuzetak drugačijeg tipa osim navedenih ipak promakne obradi i bude bačen iz funkcije, poziva se predefinisana funkcija unexpected () koja će izazvati završetak programa

# Zaključak

- Izuzetci – odstupanje od predviđenog toka programa
- Posledica greške u radu računara
- Program podiže (baca) izuzetak, prekida izvršenje programa i prosleđuje izuzetak rutini za oporavak
- *try-catch* sintaksa
- Navođenje liste mogućih izuzetaka - throw

# Kraj prezentacije

# HVALA NA PAŽNJI!