Objektno orijentisano programiranje 1

Izuzeci

Pojam izuzetaka

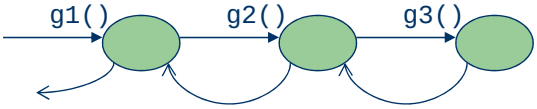
- Izuzeci (*exceptions*) su događaji koje treba posebno obraditi
- Obrada izuzetaka je izvan osnovnog toka programa
- Na primer:
  - greška koja se može javiti u nekoj obradi predstavlja izuzetnu situaciju
- Ako jezik ne podržava obradu izuzetaka, dolazi do sledećih problema:
  - problem "migracije udesno":
    - posle izvršenja dela programa (ili poziva funkcije) u kojem može doći do greške vrši se testiranje statusa
    - obrada greške se smešta u jednu, a obrada OK statusa u drugu granu `if` naredbe
  - problem "propagacije unazad":
    - u lancu poziva f-ja, ukoliko grešku treba propagirati prema prethodnom nivou, svaki nivo poziva (po povratku) treba da izvrši testiranje da li je došlo do greške
    - ako je došlo do greške koju ne može u potpunosti da obradi, funkcija u izrazu `return` naredbe treba da vrati kod greške

Ilustracije problema

- Problem "migracije udesno"

```
if (f1()) {      // status==OK
                 // dalja obrada u slucaju pozitivnog ishoda f1()
  if ((f2()) {   // status==OK
                 // dalja obrada u slucaju pozitivnog ishoda f2()
    if ((f3()) {// status==OK
                 // dalja obrada u slucaju pozitivnog ishoda f3()
    } else {     // obrada greske koju je vratila f3()
    }
  } else {       // obrada greske koju je vratila f2()
  }
} else {         // obrada greske koju je vratila f1()
}
```

- Problem "propagacije unazad"

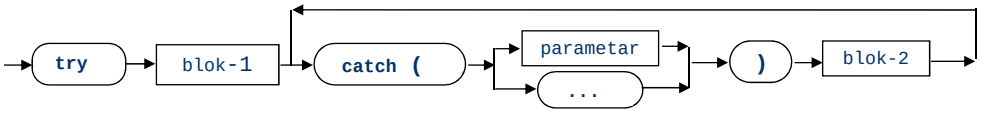


Izuzeci u jeziku C++

- C++ nudi mehanizam za efikasnu obradu izuzetaka
  - izuzeci se elegantno obrađuju izvan osnovnog toka kontrole
- Na mestu otkrivanja izuzetne situacije
  - izuzetak se "baca" (operator `throw`)
  - izuzetak može biti objekat klase ili nekog drugog tipa
  - dalja "regularna" obrada tekućeg bloka se (trajno) prekida
- Obrada izuzetka se nastavlja u posebnom bloku (`catch`)
  - rukovalac (*handler*) obradom izuzetka
  - izuzetak se dostavlja rukovaocu kao argument
  - za svaki tip izuzetka se definiše zaseban rukovalac
- Posle uspešne obrade izuzetka, obrada nastavlja regularnim tokom
- Ako ne postoji odgovarajući rukovalac
  - izuzetak se automatski propagira unatrag

Otkrivanje i obrada izuzetaka

- Obrada izuzetaka je vezana za blok naredbe `try`
- Sintaksa:



- `blok-1` je programski blok unutar kojeg mogu da se jave izuzeci
  - neposredno bačeni (`throw`) ili bačeni iz pozvane funkcije
- `parametar` se sastoji od oznake tipa i identifikatora parametra
- `...` označavaju univerzalni rukovalac
  - on se aktivira ako ne postoji rukovalac sa adekvatnim tipom izuzetka
- `blok-2` je telo rukovaoca

Definicija rukovaoca i tok obrade

- Definicija rukovaoca liči na definiciju funkcije sa tačno jednim argumentom
- Kaže se da je rukovalac tipa `T` ako je njegov `parametar` tipa `T`
  - obrađuje izuzetke tipa `T`
- `blok-2` obrađuje izuzetke koji su se javili
  - neposredno u `blok-1` (u naredbi sa operatorom `throw`)
  - u nekoj funkciji koja je pozvana iz `blok-1`
- Nakon izvršenja `blok-2`
  - kontrola se ne vraća na mesto gde se dogodio izuzetak
  - instrukcije bloka `try` koje slede iza mesta gde se dogodio izuzetak se ne izvrše
- Unutar `blok-1` ili u pozvanim funkcijama iz `blok-1` mogu da se pojave ugnježdene naredbe `try`
- Ako se u `blok-1` ne javi izuzetak
  - blok se regularno završava
  - preskaču se svi njegovi rukovaoci
  - obrada se nastavlja iza poslednjeg rukovaoca

Primer izuzetka

```
try {
  ...
  radi(); //  funkcija možda baca izuzetak
  ...
} catch(const char *pz){
    // Obrada izuzetka tipa znakovnog niza
} catch(const int i){
    // Obrada izuzetka celobrojnog tipa
} catch(...){
    // Obrada izuzetka proizvoljnog tipa
    // koji nije jedan od gornjih
}
```


Izazivanje izuzetaka

- Izazivanje (prijavljivanje, bacanje) izuzetaka se vrši naredbom:
  `throw` *`izraz`*`;`
  - gde izraz svojim tipom određuje koji rukovalac će biti aktiviran
  - vrednost izraza se izračunava i prenosi rukovaocu kao argument
- Izuzetak se može izazvati iz bloka ili iz bilo koje funkcije direktno ili indirektno pozvane iz bloka naredbe `try`
- Funkcije iz kojih se izaziva izuzetak mogu biti:
  - članice klasa, operatorske funkcije, konstruktori, a i destruktori
- Za (dinamički) ugnežđene naredbe `try`:
  - rukovalac ugnežđene naredbe može da izazove izuzetak
  - takav izuzetak se prosleđuje rukovaocu spoljašnje naredbe `try`
- Unutar rukovaoca ugnežđene naredbe `try`:
  - izuzetak može da se izazove i pomoću naredbe `throw;` (bez izraza)
  - takav izuzetak ima tip rukovaoca u kojem je izazvan

Specifikacija izuzetaka funkcije

- U deklaraciji ili definiciji funkcija može da se navede spisak tipova izuzetaka koje funkcija izaziva
- Navođenje spiska tipova izuzetaka se postiže pomoću `throw(niz_identifikatora)` iza liste argumenata
- Ako se stavi `throw()` klauzula:
  - kad funkcija izazove izuzetak tipa koji nije nabrojan – greška
- Ako se izostavi `throw()` klauzula:
  - funkcija sme da prijavi izuzetak proizvoljnog tipa

Primer `throw()` klauzule

```
void radi(...)throw(char *, int){
  if(...) throw "Izuzetak!";
  if(...) throw 100;
  if(...) throw Tacka(0,0);
    // GRESKA: nije naveden tip izuzetka Tacka
}
```

- Virtuelna metoda u izvedenoj klasi:
  - ne sme da proširi listu izuzetaka iz `throw()` klauzule
  - sme da suzi listu izuzetaka `throw()` klauzule
- Standard C++11 više ne preporučuje `throw(`*`niz_id`*`)` klauzulu
  - umesto specifikacije pojedinih tipova izuzetaka, samo informacija o tome da li funkcija baca ili ne izuzetke

Modifikator i operator noexcept

- Modifikator:
  - `noexcept(`*`izraz`*`)`
  - navodi se iza liste parametara
  - izraz je konstantan logički, izvršava se u toku prevođenja
  - vrednost `true` označava da funkcija neće bacati izuzetke
  - `noexcept ≡ noexcept(true) ≡ throw ()`
- Operator:
  - `noexcept(`*`izraz`*`)`
  - rezultat je logičkog tipa
  - izraz može biti proizvoljnog tipa, čak i `void`
  - proverava se da li bi u slučaju izvršenja izraza moglo doći do izuzetka
  - izraz se ne izračunava, samo se proverava u toku prevođenja
  - može se koristiti u izrazu `noexcept` modifikatora

Primer noexcept

```
void   f() noexcept {}                  // ne baca
int    g() noexcept(false){return 0;} // moze da baca
double h() {return 0.0;}                // moze da baca
int    i() throw(){return 0;}           // ne baca
void   j() throw (int, double){}        // moze da baca
                                        //    int i double
bool p=noexcept(f());                   // true
bool q=noexcept(g());                   // false
bool r=noexcept(h());                   // false
bool s=noexcept(i());                   // true
bool t=noexcept(j());                   // false
bool v=noexcept(new int);               // false
```

Prihvatanje izuzetaka – pravila

- Rukovalac tipa `R` može da prihvati izuzetak tipa `I` ako:
  - `R` i `I` su isti tip
  - `R` je javna osnovna klasa za izvedenu klasu `I`
  - `R` i `I` su pokazivački/upućivački tipovi i `I` standardno može da se konvertuje u `R`
    - `R` je tip pok./ref. na klasu `B` koja je javna osnovna klasa za izvedenu klasu `D`, a `I` je pok./ref. na `D`
- Prilikom navođenja rukovaoca treba se držati sledećih pravila:
  - rukovaoce tipa izvedenog iz neke osnovne klase treba stavljati ispred rukovaoca tipa te osnovne klase
  - univerzalni rukovalac treba stavljati na poslednje mesto

Prosleđivanje i obrada izuzetka (1)

- Na mestu izazivanja izuzetka (npr. `throw X();`) formira se objekat koji se prosleđuje rukovaocu na obradu
  - objekat koji se prosleđuje rukovaocu je (po pravilu) kopija objekta rezultata izraza (operanda) operatora `throw`
  - prevodilac može da odluči da ne pravi kopiju
    - tada se za objekat izuzetka koristi objekat rezultata izraza
    - to može biti bezimeni objekat ili čak lokalni automatski objekat
    - ipak, mora postojati kopirajući ili premeštajući konstruktor i destruktor
  - životni vek prosleđenog objekta je do kraja poslednjeg pozvanog rukovaoca
- Objekat izuzetka se prosleđuje i obrađuje u najbližem (prvom odgovarajućem na koji se naiđe) rukovaocu
- Ako se ne pronađe odgovarajući rukovalac, objekat izuzetka se prosleđuje prethodnom nivou `try` bloka

Prosleđivanje i obrada izuzetka (2)

- Ako su `try` naredbe (statički ili dinamički) ugnežđene:
  - ako se pronađe odgovarajući rukovalac tekuće naredbe `try`
    - on obrađuje izuzetak
    - ako se iz tog rukovaoca ne baci izuzetak,
      obrada nastavlja naredbom posle poslednjeg `catch` bloka
  - ako se ne pronađe odgovarajući rukovalac tekuće naredbe `try`
    - izuzetak se prosleđuje odgovarajućem rukovaocu
      prethodnog (spoljnog) nivoa naredbe `try`
    - eventualne naredbe u sekvenci iza poslednjeg `catch` bloka se neće izvršiti
- U slučaju da se izuzetak baci iz `catch` bloka naredbom `throw;`
  - i on se prosleđuje rukovaocu prethodnog nivoa naredbe `try`
  - objekat izuzetka (argument) ostaje živ i u `catch` bloku tog `try`

Uništavanje lokalnih objekata

- Predaja kontrole rukovaocu podrazumeva
  definitivno napuštanje bloka u kojem se dogodio izuzetak
  - napuštanje bloka podrazumeva uništavanje svih lokalnih objekata
    u tom bloku i ugnežđenim blokovima
  - uništavanje redosledom obrnutim od stvaranja
    (poslednji iz tekućeg `try` bloka)
- Izuzetak bačen iz konstruktora
  - uništavaju se prethodno stvoreni klasni atributi i nasleđeni podobjekti
- Nije dobro da rezultat izraza `throw` pokazuje/upućuje na lokalni objekat
  - taj objekat će se uništiti pre dohvatanja iz rukovaoca

Funkcijska naredba `try`

- Telo `try` bloka se poklapa sa telom funkcije
- Sintaksa:

```
tip fun(parametri) try { /* telo funkcije */ }
catch (parametar1) { /* telo rukovaoca 1 */ }
catch (parametar2) { /* telo rukovaoca 2 */ }
```

- Modifikatori metode (npr. `const`) i `throw` klauzula – ispred `try`
- U rukovaocima (`catch`)
  - mogu da se koriste parametri funkcije
  - ne mogu da se koriste lokalne promenljive funkcije
  - ako funkcija nije `void`, mora se izvršiti `return`, ako se na baci izuzetak

Primer funkcijske naredbe `try`

```
int f(int x) throw(double) try {
  int y=0;
  if (…) throw 1;     // baca se i obradjuje
  if (…) throw 2.0;   // baca se i propagira dalje
  …
  return x+y;   // regularan rezultat funkcije
} catch (int g) {
  int a=x;      // u redu
  int b=y;      // ! GRESKA
  return -1;    // rezultat u izuzetnoj situaciji
}
```

Funkcijski `try` u konstruktoru

- Omogućava hvatanje izuzetaka koji se bacaju iz
  - inicijalizatora atributa primitivnog tipa
  - konstruktora atributa klasnog tipa
  - konstruktora osnovnih klasa
- Lista inicijalizatora u definiciji konstruktora se piše iza `try`
- Ako se iz rukovaoca pristupa atributima ili nasleđenom podobjektu
  - posledice su nepredvidive
  - neki još nisu inicijalizovani, a one klasne koji su već bili konstruisani odgovarajući destruktor je uništio pre izvršenja rukovaoca
- Dolazak do kraja rukovaoca izaziva ponovno bacanje izuzetka
  - kao da je rukovalac završen naredbom `throw;` (bez operanda)
  - konstruktor može regularno da se završi samo ako uspešno stvori objekat
  - rukovalac može i da se završi pozivom funkcije `exit(int)`
  - u konstruktoru sa funkcijskim `try` nije dozvoljen `noexcept` ili `throw()`

Primer funk. `try` u konstruktoru

```
class A {
public:
  A(int x) {… if (…) throw 'x';…}
  A(char x) {… if (…) throw 2;…}
}
class B {
  A a1=A(3);
  A a2;
public:
  B() try : a2('a'){     // izuzetak tipa int
     … if(…) throw 4.0; // izuzetak tipa double
  } catch (double g) {  // rukovalac za tip double
    A a3(a1);           // ! GRESKA – a1 unisten
  } catch (char g) {    // rukovalac za tip char
    …
  }                     // nije obradjen tip int
}
```

Neprihvaćeni izuzeci

- Ako se za neki izuzetak ne pronađe rukovalac koji može da ga prihvati - izvršava se sistemska funkcija:

```
void terminate();
```

- Podrazumeva se da ova funkcija poziva funkciju `abort()`
  - funkcija `abort()` vraća kontrolu operativnom sistemu
- Ovo se može promeniti pomoću funkcije `set_terminate()`
  - njoj se dostavlja pokazivač na funkciju koju treba da pozove funkcija `terminate()` umesto funkcije `abort()`
  - pokazana funkcija mora biti bez argumenata i bez rezultata (`void`)
  - vrednost funkcije `set_terminate()` je pokazivač na staru funkciju koja je bila pozivana iz `terminate()`

Zamena za `abort()`

- Tip korisničke funkcije koja zamenjuje funkciju `abort()`

```
typedef viod (*PF) ();
```

- Prototip funkcije `set_terminate()`:

```
PF set_terminate(PF pf);
```

- Iz korisničke funkcije `(*pf)` treba pozvati `exit(int)`, `terminate()` ili `abort()` za povratak u operativni sistem
- Pokušaj povratka sa `return` iz korisničke funkcije `(*pf)`
  - dovešće do nasilnog prekida programa sa `abort()`

Neočekivani izuzeci

- Ako se u nekoj funkciji izazove neočekivani izuzetak (koji nije na spisku naznačenih izuzetaka), izvršava se funkcija:

```
void unexpected();
```

- Podrazumeva se da ova funkcija poziva funkciju `terminate()`
- Ovo se može promeniti pomoću funcije `set_unexpected()`
  - njoj se dostavlja pokazivač na funkciju koju treba da pozove funkcija `unexpected()` umesto `terminate()`
  - pokazana funkcija mora biti bez argumenata i bez rezultata (`void`)
  - vrednost funkcije `set_unexpected()` je pokazivač na staru funkciju koja je bila pozivana iz `unexpected()`

Zamena za `terminate()`

- Tip korisničke funkcije koja zamenjuje funkciju `terminate()`

```
typedef viod (*PF) ();
```

- Prototip funkcije `set_unexpected()`:

```
PF set_unexpected(PF pf);
```

- Iz korisničke funkcije treba:
  - pozvati `exit(int)`, `terminate()` ili `abort()` ili
  - baciti izuzetak sa `throw` liste dozvoljenih ili
  - baciti `bad_exception`
- Pokušaj povratka sa `return` iz korisničke funkcije `(*pf)` dovešće do nasilnog prekida programa sa `abort()`

Standardni izuzeci

- Klasa `exception` u standardnom zaglavlju `<exception>`
  - klasa je predak (koren hijerarhije) svih standardnih izuzetaka
  - metode standardnih klasa i neki operatori prijavljuju izuzetke klasa izvedenih iz klase `exception`
  - preporučuje se da se i korisnički izuzeci izvode iz klase `exception`
  - klasa omogućava da se projektuje hijerarhija izuzetaka takva da se pojedini tipovi izuzetaka mogu obrađivati:
    - pojedinačno
    - u srodnim grupama
    - svi zajedno (ali ipak ne kao `...`)

Definicija klase `exception`

```
class exception {
public:
  exception() noexcept;
  exception(const exception &) noexcept;
  exception& operator=(const exception &) noexcept;
  virtual ~exception() noexcept;
  virtual const char* what() const noexcept;
};
```

- `what()` vraća pokazivač na tekstualni opis izuzetka (std. ne propisuje tekst poruka)
- Ni jedna metoda ne sme da prijavi ni jedan izuzetak:
  - obezbeđuje se sa `noexcept`
  - to automatski važi i za date metode u izvedenim klasama (lista izuzetaka se ne sme proširiti)