Sveučilište u Zagrebu
PMF – Matematički odjel

Objektno programiranje (C++)

Vježbe 05 – Izuzeci (Exceptions)

Vinko Petričević

Izuzeci (iznimke)

- Izuzeci (exception) su anomalije koje ne spadaju u područje normalnog djelovanja programa, pojavljuju se u tijeku izvođenja programa (run-time), te zahtjevaju trenutno reagiranje
  - npr. potrošena memorija, neočekivan ulaz, dijeljenje s nulom,...
- Ključne riječi za bacanje i rukovanje izuzecima:
  - throw
  - try
  - catch

Izuzeci

- Ideja: kao što korisniku dajemo sučelje za rad s nekom klasom (npr. stog ima metode push, pop, full, empty i display), tako bismo mu trebali dati i popis izuzetaka (tj. potencijalno opasnih situacija) koje bi se mogle dogoditi. Korisnik je onda taj koji bi trebao reagirati na izuzetke i obraditi ih (npr. push je opasan ako ga pozovemo na punom stogu). Umjesto da ispišemo grešku na cout ili cerr i prekinemo izvođenje programa, pomoću izuzetaka možemo javiti korisniku klase o kakvoj se grešci radi, te omogućiti mu da on na nju reagira po svom nahođenju.

Bacanje izuzetka

- Polazni primjer:
  - ```
    class iStack {
    public:
      iStack( int capacity ) : _stack( capacity ), _top( 0 ) { }
      bool pop( int &top_value );
      bool push( int value );

      bool full();
      bool empty();
      void display();

      int size();

    private:
      int _top;
      vector< int > _stack;
    };
    ```
    ```

Bacanje izuzetka

- Modifikacija primjera:
  - // stackExcp.h
    class popOnEmpty { /* ... */ };
    class pushOnFull { /* ... */ };
  - class iStack {
    public:
    // ...

    // funkcije vise ne vracaju vrijednost
    void pop(int &value);
    void push(int value);

    private:
    // ...
    };

Bacanje izuzetka

- Modifikacija primjera:
  - #include "stackExcp.h"
    void iStack::pop(int &top_value) {
      if (empty())
        throw popOnEmpty();

      top_value = _stack[--_top];
      cout << "iStack::pop(): " << top_value << endl;
    }
  - void iStack::push(int value) {
      cout << "iStack::push(" << value << ")\n";

      if (full())
        throw pushOnFull();

      _stack[_top++] = value;
    }

Bacanje izuzetka

- `throw` izrazi mogu bacati objekte bilo kojeg tipa
  - Primjer:

```
enum EHstate { noErr, zeroOp, negativeOp,
severeError };

int mathFunc( int i ) {
   if ( i == 0 )
        throw zeroOp; // izuzetak tipa enum

   // nastavak normalnog procesiranja
}
```

try blok

- Polazni primjer:
  - ```
    #include <iostream>
    #include "iStack.h"

    int main() {
       iStack stack( 16 );
       stack.display();
       for ( int ix = 1; ix < 21; ++ix ) {
            if ( ix % 3 == 0 )
                 stack.push( ix );
            if ( ix % 4 == 0 )
                 stack.display();
            if ( ix % 10  == 0) {
                 int dummy;
                 stack.pop( dummy );
                 stack.display();
            }
       }
       return 0;
    }
    ```
    ```

try blok

- Modifikacija primjera (1. varijanta):
  - ```
    try {
       for ( int ix = 1; ix < 21; ++ix ) {
          if ( ix % 3 == 0 )
             stack.push( ix );
          if ( ix % 4 == 0 )
             stack.display();
          if ( ix % 10  == 0 ) {
             int dummy;
             stack.pop( dummy );
             stack.display();
          }
       }
    }
    catch ( pushOnFull ) { ... }
    catch ( popOnEmpty ) { ... }
    ```
    ```

try blok

- Modifikacija primjera (2. varijanta):
  -
```
int main() {
    try {
        iStack stack( 16 ); // ok: deklaracija u try bloku
        stack.display();
        for ( int ix = 1; ix < 21; ++ix ) {
            // isto kao ranije
        }
    }
    catch ( pushOnFull ) {
        // ovdje vise ne mozemo referirati stack
    }
    catch ( popOnEmpty ) {
        // ovdje vise ne mozemo referirati stack
    }
    // ovdje vise ne mozemo referirati stack
    return 0;
}
```

try blok

- Modifikacija primjera (3. varijanta):
  - int main()
    try {
      iStack stack(16);
      stack.display();
      for (int ix = 1; ix < 21; ++ix) {
        // isto kao ranije
      }
      return 0;
    }
    catch (pushOnFull) {
      // ovdje vise ne mozemo referirati stack
    }
    catch (popOnEmpty) {
      // ovdje vise ne mozemo referirati stack
    }

Hvatanje izuzetaka

- Primjer:
  -
```
int main() {
   iStack stack( 16 );
   try {
      stack.display();
      for ( int ix = 1; ix < 21; ++ix ) {
         // isto kao ranije
      }
   }
   catch ( pushOnFull ) {
      cerr << "trying to push a value on a full stack\n";
   }
   catch ( popOnEmpty ) {
      cerr << "trying to pop a value on an empty stack\n";
   }
   // izvršavanje programa nastavlja se ovdje
   return 0;
}
```

Hvatanje izuzetaka

- Primjer:
  - class pushOnFull {
    public:
      pushOnFull(int i) : _value(i) { }
      int value() { return _value; }
    private:
      int _value;
    };
  - void iStack::push(int value) {
      if (full())
        // value se sprema u iznimci
        throw pushOnFull(value);
      // ...
    }
  - catch (pushOnFull eObj) {
      cerr << "trying to push the value " << eObj.value()
           << " on a full stack\n";
    }

Hvatanje izuzetaka

- Primjer:
  - ```
    enum EHstate { noErr, zeroOp, negativeOp, severeError };
    enum EHstate state = noErr;
    int mathFunc( int i ) {
       if ( i == 0 ) {
          state = zeroOp;
          throw state;
       }
       // ...
    }
    ```
  - ```
    void calculate( int op ) {
       try {
          mathFunc( op );
       }
       catch ( EHstate& eObj ) {
          eObj = noErr; // globalna varijabla state ovime se ne
    modificira
       }
    }
    ```

Ponovno bacanje izuzetka

- Sintaktički oblik:
  - `throw;`
- Primjer:
  - ```
    catch ( exception eObj ) {
       if ( canHandle( eObj ) )
          // baratanje izuzetkom
          return;
       else
          // ponovno bacanje istog izuzetka (rethrow)
          throw;
    }
    ```
    ```

Hvatanje svih izuzetaka

- Polazni primjer:
  - void manip() {
    resource res;
    res.lock(); // zakljucavanje resursa

    // koristenje resursa
    // neka akcija koja uzrokuje bacanje izuzetka

    res.release(); // ako je bacen izuzetak ovo se nece izvrsiti
    }

Hvatanje svih izuzetaka

- Primjer:
  - ```
    void manip() {
        resource res;
        res.lock();
        try {
            // koristenje resursa res
            // neka akcija moze uzrokovati bacanje izuzetka
        }
        catch (...) { // ovime hvatamo sve izuzetke
            res.release();
            throw; // proslijedimo izuzetak dalje
        }
        res.release(); // ako je bacen izuzetak ovo se nece izvrsiti
    }
    ```
    ```

Hvatanje svih izuzetaka

- Primjer:
  - ```
    try {
       stack.display();
       for ( int ix = 1; ix < 21; ++ix )
       {
          // isto kao ranije
       }
    }
    catch ( pushOnFull ) { }
    catch ( popOnEmpty ) { }
    catch (...) { } // posljednja catch klauzula
    ```
    ```

Specifikacija izuzetaka

- Primjer:
  - class iStack {
    public:
      // ...

      void pop(int &value) throw(popOnEmpty);
      void push(int value) throw(pushOnFull);

    private:
      // ...
    };

Izuzeci

- Primjer: new ne vraća NULL pokazivač ako nije uspio alocirati memoriju, nego baca iznimku.
- Zadatak: Napišite program koji alocira 100kB memorije i ispisuje pointer na rezervirani blok.
- Zadatak: Napišite program koji alocira 1GB memorije i ispisuje pointer koji je dobio. Što se dogodilo?
- Zadatak: Napišite program koji alocira 1GB i hvata iznimku koju je bacio new.

Izuzeci

- Standardne biblioteke za rad sa izuzecima:
- #include <stdexcept>
  - exception
  - runtime_error
  - range_error
  - overflow_error
  - itd.
- #include <exception>
- #include <new>
- #include <type_info>

Izuzeci

- Zadatak: Bitset operacija to_ulong baca overflow_error iznimku ako je bitset prevelik za veličinu unsigned long. Napišite program koji generira i obrađuje navedenu iznimku.
- Zadatak: napišite program koji učitava dva broja koja treba podijeliti. Pogledajte što se događa ako dijelite s nulom. Prepravite program tako da baci grešku Dijeljenje_s_nulom(), te uhvatite i obradite navedenu iznimku.

try blok i kontrola kopiranja

- Prilikom bacanja iznimke, prvo se kreira klasa koja se šalje kao iznimka
- nakon toga unište se sve varijable kreirane od odgovarajučeg try bloka
- ako je u catch bloku klasa navedena bez reference, desit će se njezin copy-konstruktor

- pa ipak, desi li se throw tijekom izvršavanja nekog destruktora, u VS ga nećemo moći uhvatiti, te će nam se program srušiti

try blok i kontrola kopiranja

```
struct Ex {
    Ex() { cout<<"kreiran Ex "<<this<<endl;}
    Ex(const Ex& e) { cout<<"kopiran Ex "<<this
                        <<" <- "<<&e<<endl;}
    ~Ex() { cout<<"unisten Ex "<<this<<endl;}
};
```

```
struct S {
    S() { cout<<"kreiran S "<<this<<endl;}
    S(const S& e) { cout<<"kopiran S "
                        <<this<<" <- "<<&e<<endl;}
    ~S() { cout<<"unisten S "<<this<<endl;}
};
```

```
int main() {
    try {
            S s;
            cout<<"program ide"<<endl;
            throw Ex();
    }
    catch (Ex e) {
            cout<<"hvatanje greske"<<endl;
    }
    return 0;
}
```

```
kreiran S 0012FF53
program ide
kreiran Ex 0012FE7B
kopiran Ex 0012FF47 <- 0012FE7B
unisten S 0012FF53
hvatanje greske
unisten Ex 0012FF47
unisten Ex 0012FE7B
```