Programiramabilni uređaji i objektno orijentisano programiranje

Obrada izuzetaka

Obrada izuzetaka - Potreba

- Što se dešava kada programski modul (potprogram) otkrije neku grešku (recimo, nema memorije ili dijeljenje sa nulom)?
- Programerima su na raspolaganju bile dvije osnovne strategije:
  - Prekinuti izvršavanje programa (jednom prekinuti program nije loše, ali ga prekidati svaki put kada se neregularnost dogodi to je već veliki problem);
  - Preduzeti neke korekcije uz eventualno obavještavanje korisnika o nastalom problemu. Stoga, potprogram je morao da obavjesti modul koji ga je pozvao da nije uspio da odradi posao. Ovaj je možda morao da obavjesti svoj pozivajući modul itd.

Obavještavanje modula o grešci

- Obrada izuzetaka se realizuje inače kroz dvije-tri “naredbe”, ali sam dizajn sistema za obradu izuzetaka može biti jako složen i uči se više na sopstvenim greškama nego knjiški.
- Pored toga, programski sistemi kao što je Borland Builder imaju ugrađeni predefinisani sistem za obradu izuzetaka.
- Pozvani modul može obavijestiti pozivajući da je došlo do greške putem nekog dogovorenog rezultata rada modula (recimo, malloc je vraćao NULL rezultat ako alokacija nije uspjela).
- Osnovni problem je što ovakav način rada opterećuje softver nesuštinskim stvarima, jer svaki modul mora biti opterećen pitanjima (selekcijama) da li je onaj modul koji je pozvan odradio posao.
- Stoga se kod složenih softverskih sistema na jednostavniji način moraju obrađivati situacije koje se mogu predvidjeti, a ne mogu izbjeći (recimo, nema papira u štampaču).

Što je to obrada izuzetaka?

- Izuzetak je greška koja se može predvidjeti, a ne može izbjeći.
- Obrada izuzetaka je prečica između lokacije na kojoj se izuzetak otkriva i lokacije na kojoj se obrađuje.
- Samo tri osnovne ključne riječi se koriste prilikom obrade izuzetaka:
  - **try** je dio koji uokviruje zonu u kojoj se mogu pojaviti izuzeci.
  - **throw** je mjesto gdje se otkriva izuzetak (odakle se izuzetak "baca"). Ovo mjesto se nalazi unutar **try** dijela (ili tijela f-je koja se poziva u **try** dijelu).
  - **catch** je mjesto gdje se izuzetak "hvata", odnosno obrađuje. Sekcije sa **catch**-evima se nalaze nakon **try** bloka.

Kod za obradu izuzetaka

- Sistem za obradu izuzetka obično ima oblik:
-

```
try{
//dio koda, f-je, glavni program itd.
throw(a); //baca izuzetak određenog tipa
 //definisanog objektom a
}//kraj try bloka
catch(int x){
//neka obrada
}
catch(Student st){
//neka obrada
}
```

catch dio često ne koristi objekat koji mu je argument, već samo informaciju da je izuzetak nekog tipa nastao.
catch može imati samo jedan argument!
throw prekida izvršavanje programa i u tom trenutku se preskače na najbliži catch koji odgovara po argumentu.

Sintaksa i pravila

- Uzima se najbliži catch dio koji odgovara po argumentu tipu koji je "bačen" naredbom throw, jer se ovdje ne gleda najbolje slaganje argumenata, već prvo slaganje gdje bilo kakvom konverzijom (standardnom ili onom koju korisnik specificira) može doći do preklapanja.
- Kod novijeg C++ standarda ovo se dešava samo kod dva tipa konverzija: kada je catch blok tipa koji je dostupna osnovna klasa u odnosu na tip u izuzetku, te kada su bačeni izuzetak i catch blok dio pokazivači sa definisanom konverzijom. Ovu karakteristiku treba ispitati prije nego počnete sa radom.
- Ako u tekućem try bloku nema odgovarajućeg izuzetka onda se gleda da li postoji neki try blok koji uokviruje dati try blok, pa se izuzetak traži u njemu (ponekad postoji silna hijerarhija try blokova i neki blokovi samo "malo" obrade izuzetak i prepuštaju dalju obradu blokovima koji su viši u hijerarhiji izuzetaka).

Sintaksa i pravila

- Ako želimo da ponovo postavimo isti izuzetak napišemo throw; (bez argumenata). Na ovaj način jedan catch može da obradi "malo" neki izuzetak i da ga proslijedi na dalju obradu.
- catch(...) prima izuzetke svih tipova. Uvijek mora biti na kraju spiska catch blokova, jer se u suprotnom oni catch blokovi koji slijede ne bi nikada pozivali. Zapamtite, može da se dogodi da se poziva prvi blok koji odgovara, a ne najbolji!
- Argument naredbe throw se alocira statički da bi bio dostupan na mjestu na kojem se obrađuje izuzetak.
- Premda se ugrađeni i standardni klasni tipovi mogu koristiti kao argumenti, često se kreiraju specijalni tipovi podataka samo da bi sugerisali izuzetke.

Primjer

- Posmatrajmo sljedeći primjer:

```
#include <iostream.h>
class DijeljenjeSaNulom{
public:
DijeljenjeSaNulom(){/*neko obavještenje*/}
};
double dijeljenje(double m, double n){
    if(n==0) throw DijeljenjeSaNulom();
    //poziv konstruktora klase za izuzetak
    return m/n;}
```

Ovo je klasa samo za izuzetke. Obično su veoma jednostavne. Npr. Builder ima mnoštvo predefinisanih klasa asociranih sa VCL-om i njihova imena počinju sa E.

Primjer - Nastavak

- Dio glavnog programa:

```
main(){
        double n,m,r;
        cin>>n>>m;
        try{r=dijeljenje(m,n);
        cout<<r;}
        catch(DijeljenjeSaNulom x)
        {cout<<"Doslo je do dijeljenja sa nulom!";}
        }
```

- Ključna riječ try se koristi unutar f-je, ali može da uokviruje jednu ili više f-ja.
- Ovdje nakon uvida da u funkciji ne može da se obradi izuzetak, vraća se na mjesto poziva i izuzetak obrađuje catch try dijela unutar kojeg je poziv funkcije.

Što obrađuje catch?

- catch blok može da obradi svaki izuzetak istog tipa koji je postavio throw, tipa koji je iz tipa u izuzetku izveden javnim izvođenjem, tipa podatka koji se može konvertovati u tip podatka u izuzetku. Kod catch-a su referenca i podatak nekog tipa potpuno ekvivalentni.
- U slučaju da nema catch bloka koji odgovara tipu izuzetka onda se prelazi u try blok koji uokviruje tekući blok i tamo traži izuzetak odgovarajućeg tipa.
- catch(...) obrađuje izuzetke svih tipova i mora biti na kraju liste catch blokova.

Funkcije i izuzeci

- U zaglavlju funkcije se mogu navesti tipovi izuzetaka koje funkcija može postaviti. Npr.
  `void f() throw(X,Y) {/**/}`
- Specifikacija tipova izuzetaka ne čini dio specifikacije tipa funkcije.
- Često se izostavlja `throw` blok kod funkcija koje postavljaju izuzetke i tada se pretpostavlja da funkcija može postaviti izuzetak bilo koga tipa.
- Funkcija koja postavlja izuzetak nekog tipa može postaviti i izuzetak tipa koji je javno izveden iz predmetnog tipa.
- Prazan `throw` dio u zaglavlju funkcije sugeriše da funkcija ne postavlja izuzetke.

Odmotavanje steka

- Ako tip izuzetka koji je funkcija postavila ne odgovara specificiranim tipovima u zaglavlju funkcije dolazi do poziva specijalne funkcije unexpected(), o kojoj ćemo više riječi reći kasnije.
- Pretpostavimo da je izuzetak detektovan u dijelu programskog koda (nekoj funkciji). Ako se ne pronađe catch koji odgovara, tada se predmetna funkcija završava zajedno sa dealokacijom te funkcije sa steka (stoga se ovaj mehanizam naziva **odmotavanje steka**), pa se kontrola programa vraća na poziciju funkcije koja je pozvala predmetnu funkciju. Ako ova funkcija nije u try bloku, ili se izuzetak ne uhvati, dolazi do nastavka odmotavanja steka (prelazi se na narednu funkciju).

Neobrađeni izuzetak

- Ako se izuzetak ne obradi dolazi do pozivanja specijalnog metoda terminate, o kojem će više riječi biti nešto kasnije.
- Sada ćemo kroz nekoliko generičkih primjera ilustrovati priču o izuzecima.

```
class gException{/**/};
//klasa samo za izuzetke
void g(){
//…
if(nesto) throw gException();
//...
}
```

Izuzetak – Primjer 1

- Neka f-ja f() sada poziva f-ju g():

```
void f(){
    try{ //try blok unutar f-je
    //…
    g();
    //…
    throw "U pomoc";
    //…
}catch(char *p) {cout<<p;}
//izuzetak postavljan u f-ji f koji koristi
//podatak p o izuzetku
catch(gException){
//iz f-je g ne koristi podatak o izuzetku
}
}
```

- Ukoliko dođe do izuzetka u funkciji g, kontrola se prosleđuje funkciji f, koja ima try dio, i odgovarajući catch - catch(gException)

Izuzetak – Primjer 2

- Primjer korišćenja catch(...) i throw;:

```
try{ //spoljni try
    try{ //unutrašnji try
    //…
    }
    catch(…){
    //djelimično obradi izuzetak
    throw; //ponovo postavi posljednji izuzetak
    //da bi ga mogli obraditi "viši" (spoljni)
    //nivoi try blokova
    }
} //catch-evi za spoljni try
```

Izuzetak – Primjer 3

```
class B_E {
/**/};
class DE:public B_E {/**/};
//izvedena klasa za izuzetke
try{
//...
} catch(B_E &b) {/*obrada 1*/}
catch(DE &d){/**/}
//...
```

- Greška se ogleda u činjenici da će uvijek biti izvršen prvi catch blok bez obzira na navedeni tip izuzetka, jer se biri prvi catch blok koji odgovara tipu izuzetka, a ne najbolji. Stoga, ako se želi da rješenje dozvoljava da svaki catch blok može biti pozvan treba zamjeniti redosljed catch blokova.

Izuzetak – Primjer 4

- Ovaj primjer demonstrira odmotavanje steka.

```
#include <iostream.h>
void f3() throw (char *)
      {throw "greška u f-ji 3";}
void f2() throw (char *)
      {f3();}
void f1() throw (char *)
      {f2();}
main(){
      try{ f1();}
      catch (char *e)
            {cout<<e;}
}
```

Objašnjenje primjera

- U predmetnom primjeru sve f-je mogu postaviti izuzetak tipa char *, što smo specificirali u zaglavljima f-je. try uokviruje poziv funkcije f1(). Ova funkcija poziva f2(), a ova f3(). Poziv f-je f3() nije uokvirena sa try, pa se ova funkcija dealocira sa steka i kontrola programa zajedno sa izuzetkom vraća funkciji f2(), a ova funkcija to vraća funkciji f1(). Kako je poziv funkcije f1() uokvirena try-om, catch može da obradi izuzetak koji je postavila f-ja f3(), te dolazi do odgovarajuće obrade.
- Ovo je zgodan mehanizam koji nam dozvoljava da izbjegnemo pisanje try-eva oko pojedinačnih f-ja, već kompletnu obradu izuzetaka možemo povezati sa glavnim programom koji može obrađivati izuzetke koje su postavile pojedine funkcije.

Izuzeci i konstruktor

- Ako se objekat ne konstruiše iz bilo kog razloga zgodno je obavijestiti ostatak koda putem izuzetka.
- Ovdje definitivno ne možemo koristiti rezultat funkcije, jer konstruktor nema rezultat, a nezgodno bi bilo pretpostaviti da će djelovi koda koji zahtjevaju konstrukciju objekata klasnog tipa na osnovu "nedostataka" kod konstruisanog objekta znati da objekat nije konstruisan.
- Kada se postavi izuzetak u konstruktoru dolazi do destrukcije djelova objekta koji je konstruisan prije postavljanja izuzetka. Takođe, u bilo kom try bloku destruktor se poziva za sve objekte koji su nastali od početka try bloka do pozicije na kojoj je postavljen izuzetak.

Izuzetak i nasljeđivanje

- Klase koje se koriste samo za izuzetke često kreiraju čitavu hijerahriju nasljeđivanja. To je, naravno, i dozvoljeno i poželjno, samo imajte na umu da ako catch "hvata" izuzetak koji je referenca ili pokazivač na osnovnu klasu on može uhvatiti i referencu i pokazivač na javno izvedenu klasu. Stoga, catch-eve koji hvataju osnovnu klasu treba postaviti na dnu liste catch-eva (iza svih catch-eva koji hvataju izuzetke koji su pokazivači ili reference na izvedene klase).

Izuzetak i new

- Uobičajeno nakon dinamičke alokacije sa new vršili smo provjeru da li je aktuelni pokazivač jednak 0, odnosno da li je alokacija uspjela. C++ pored ovoga dozvoljava i druge tehnike za provjere uspjeha dinamičke alokacije.
- Kao i kod neuspjeha konstrukcije, dosta dobra tehnika je korišćenjem izuzetka. Po ANSI C++ standardu, ako dinamička alokacija ne uspije "baca" se izuzetak klasnog tipa bad_alloc. bad_alloc je izuzetak definisan u zaglavlju new.h.
- Jedan poseban problem koji se može dogoditi u ovoj situaciji je da se objekat dinamički alocira, a da se prije dealokacije dogodi izuzetak što može dovesti do ne-dealokacije dinamički kreiranih objekata.
- C++ ovo može prevazići upotrebom pokazivača auto_ptr koji je definisan u zaglavlju memory.h. Detalje (ove i neke druge praktično važne) možete naći na stranama 685-690 engleskog izdanja knjige Deitel-Deitel.

Specijalne f-je za obradu izuzetaka

- Postoji nekoliko f-ja koje se mogu pozvati implicitno ili eksplicitno prilikom rada sa izuzecima.
- Funkcija terminate() se poziva ako ne postoji catch blok koji bi prihvatio izuzetak datog tipa.
- Ova funkcija podrazumijevano poziva funkciju abort() koja prekida izvršavanje programa bez obavještavanja operativnog sistema.
- Na ponašanje funkcije terminate() može se djelovati preko funkcije set_terminate(p).
- Ako se pozove f-ja set_terminate(p), gdje je p pokazivač na neku f-ju, funkcija terminate() neće pozivati funkciju abort(), već funkciju na koju pokazuje p.

Specijalne f-je za obradu izuzetaka

- Funkcija terminate() mora da prekine izvršavanje programa (uz možda još neke dodatne korektivne operacije) jer su prije poziva ove funkcije pobrisani svi automatski objekti nastali od početka posljednjeg try bloka do postavljanja izuzetka.
- Funkcija void unexpected() se poziva ako funkcija postavi izuzetak koji nije u skladu sa specificiranim izuzecima iz zaglavlja funkcije.
- Ova funkcija predefinisano poziva funkciju terminate, ali se to može promijeniti funkcijom set_unexpected koja radi na isti način kao set_terminate.

Biblioteke sa izuzecima

- Obično su samo ključne riječi try, catch i throw dio definicije programskog jezika C++.
- Za ostale naredbe koje smo pominjali, a i one koje se koriste za ove namjene, a nijesmo ih pominjali, moraju se uključiti programska zaglavlja sa odgovarajućim funkcijama i klasama.
- Pogledajte da li na vašem sistemu postoje biblioteke exception.h, terminate.h i unexpected.h.
- Kompajler može da posjeduje i neke druge biblioteke za rad sa izuzecima.

Nedostatak finally

- Često se kao nedostatak C++-a navodi nedostatak dijela sistema za obradu izuzetaka koji bi se obavljao u slučaju bilo kakve pojave izuzetka uz obradu koja je asocirana konkretnim izuzetkom.
- Recimo, programski jezik C# ima takvu riječ, odnosno blok finally.
- Ova funkcionalnost se može postići i u C++ tako što svaki dio u kojem se obrađuje izuzetak ponovi throw; a na kraju vršimo obradu sa catch(...).
- Ovo nije idealno riješenje, jer moramo da vodimo računa i o nivoima try blokova, pa je, stoga, riječ finally dosta pogodna.

Obrada izuzetaka u Borland Builderu

- Nedostatak finally-a se u Borlandu prevazilazi ključnom riječju __finally (dvije podvlake koje prethode nekoj naredbi ukazuju da je riječ o proširenju C++ koje je za svoje potrebe uveo Borland).
- Ako u Builder-u pogledate Project/View Source vidjećete da je vaš program uokviren u try blok, te da već postoji odgovarajući catch blok koji hvata izuzetak tipa reference na Exception.
- Sve klase koje predefinisano postoje u Builderu i koje počinju sa **E** su klase predviđene za obradu izuzetaka.

Obrada izuzetaka u Builderu

- Jednostavnim brisanjem onoga što vam smeta možete isključiti korišćenje predefinisanih izuzetaka.
- Napominjemo da uz sve VCL komponente postoje specificirani tipovi izuzetaka tako da je to za ove standardne komponente urađeno bez vaše kontrole.
- Ako vam trebaju dopune možete ih sami napisati, a možete i izbjeći izuzetke koje postavljaju VCL klase.

Dizajniranje sistema za obradu izuzetaka

- Sintaksa sistema za obradu izuzetaka je jednostavna.
- Jedva tri ključne uz još par dodataka i pet-šest pravila koje treba poštovati.
- Međutim, odluka o načinu na koji ćete zapravo dizajnirati obradu i što ćete smatrati izuzetkom nije jednostavna.
- Obično se savjetuje da se sistem za obradu izuzetaka dizajnira nezavisno od samog softvera kao dodatak koji se “kalemi” na softver.
- Drugi savjet je da svaki izuzetak bude signaliziran preko neke od posebno dizajniranih klasa koje se koriste samo za tu namjenu.

Dizajn sistema za izuzetke

- Za ostale zahtjeve treba početi sa programiranjem da bi se došlo do odgovarajućih dilema.
- Tu mogu biti od pomoći knjige, kao što je Deitel & Deitel, koja daje nekoliko rudimentarnih zahtjeva vezanih za obradu izuzetaka (istina, nešto širih nego što je dato na ovim predavanjima).
- Koliko je nama poznato u šire dostupnoj literaturi koja se u prevodu može naći i na našem jeziku ova problematika je razmatrana u vidu poglavlja knjige posvećene OO konstruisanju (modelovanju, analizi i dizajnu) autora B. Meyera.
- Kuriozitet ove, vjerovatno najbolje knjige o OO programiranju, je da nema nijedne linije pravog koda (isključujući pseudokod) na oko 1300 strana knjige.