

POO. Curs 6

Anca Dobrovăț și Andrei Păun. Document editat de Raluca Tudor

25 Martie 2020

Obs: În acest curs, exemplele vor fi luate, în principal, din cartea lui **B. Eckel - Thinking in C++**.

1 Tratarea excepțiilor în C++

- automatizarea procesarii erorilor
- try, catch, throw
- block try aruncă excepție cu throw care este prinsă cu catch
- după ce excepția este prinsă, se termină execuția din blocul catch și NU se revine la locul unde s-a făcut throw (catch nu este ca apelul de funcție - ci “se întoarce un nivel mai jos”).

Cu alte cuvinte, în momentul în care apare o eroare, se generează o excepție - excepția este, de fapt, **o instanțiere a unei clase, deci este un obiect**. Obiectul acesta este aruncat prin throw și încearcă să fie prins prin catch.

În general, este mai bine ca ‘catch’-ul să fie cu **referință** către tipul respectiv și nu prin valoare, pentru a evita constructorul de copiere.

Funcțiile de catch se mai numesc și **exception handlers**.

Există multiple catch - catch(...)

Observații:

- dacă se face throw și nu există un bloc try din care a fost aruncată excepția sau o funcție apelată dintr-un bloc try: **EROARE**
- dacă nu există un catch care să fie asociat cu throw-ul respectiv (tipuri de date egale) atunci programul se termină prin **terminate()**
- terminate() poate să fie redefinită

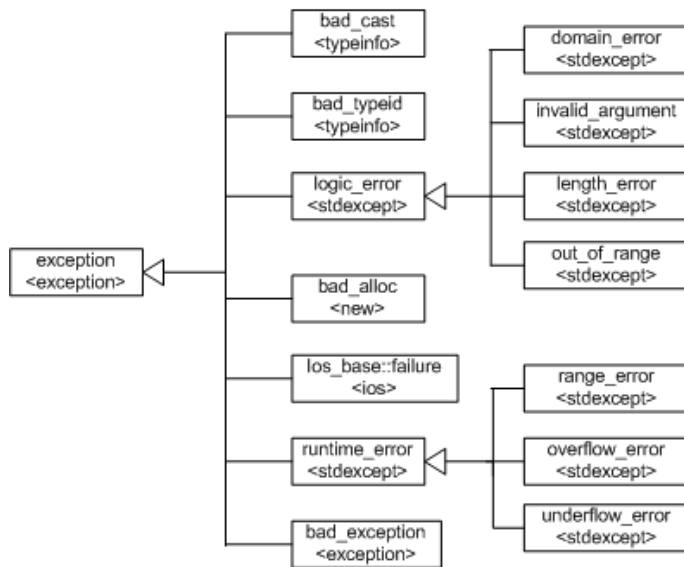
```

1 #include <iostream>
2 #include <exception>
3 #include <cstdlib>
4
5 using namespace std;
6
7 // exceptii: functia terminate()
8
9 class Unu { };
10 class Doi { };
11 class Trei { };
12 class Patru { };
13
14 void unu()
15 {
16     throw Unu();
17 }
18
19 void doi()
20 {
21     throw Doi();
22 }
23
24 void trei()
25 {
26     throw Trei();
27 }
28
29 void patru()
30 {
31     throw Patru();
32 }
33
34 void my_terminate() {
35     cout << "Revin in 5 min!" << endl;
36     abort();
37 }
38
39 void (*old_terminate)()
40     = set_terminate(my_terminate);
41
42 void main()
43 {
44     try {
45         try {
46             try {
47                 // unu();
48                 // doi();
49                 // trei();
50                 patru(); // nu gaseste niciunul din catch-urile de aici => my_terminate()
51             } catch (Trei) {
52                 cout << "Exceptie Trei.\n";
53             }
54         } catch (Doi) {
55             cout << "Exceptie Doi.\n";
56         }
57     } catch (Unu) {
58         cout << "Exceptie Unu.\n";
59     }
60 }

```

Listing 1: Exemplu Curs Dorel Lucanu

The C++ Exception Hierarchy



Cele mai importante sunt: clasa de `logic_error` și clasa `runtime_error`.

Tipuri de excepții predefinite

Din clasa `exception` sunt derivate două clase, `logic_error`, pentru raportarea erorilor detectabile înainte de execuția programului și `runtime_error`, pentru raportarea erorilor din timpul execuției programului.

Principalele clase derivate din `logic_error` sunt următoarele:

- `domain_error` – raportează violarea unei precondiții
- `invalid_argument` – argument invalid pentru o funcție
- `length_error` – un obiect cu o lungime mai mare decât NPOS (valoarea maximă reprezentabilă)
- `out_of_range` – în afara domeniului
- `bad_cast` – operație `dynamic_cast` eronată

Principalele clase derivate din `runtime_error` sunt următoarele:

- `range_error` – violarea unei postcondiții
- `overflow_error` – operație aritmetică eronată (depășire)
- `bad_alloc` – eroare de alocare

Aruncarea de erori din clase de bază și derivate

- un catch pentru tipul de bază va fi executat pentru un obiect aruncat de tipul derivat
- să se puna catch-ul pe tipul derivat primul și apoi catchul pe tipul de bază

```
1 class B { };
2 class D: public B { };
3 int main()
4 {
5     D derived;
6     try {         throw derived;    }
7
8     catch(B b) {      cout << "Caught a base class.\n";    }
9
10    catch(D d) {      cout << "This won't execute.\n";    }
11 return 0;
12 }
```

Listing 2: Aruncarea de erori din clase de baza si derivate

```
class Vector {
private:
    float t[MAX];
    int n;
public:
    Vector(int n);
    int getNrElemente() const;
    void setElement(int i, float val);
    float getElement(int i) const;
    void citire();
    void afisare();
};

Vector::Vector(int n){
    this->n = n;
    if (n>MAX)
        throw out_of_range("Depasire
dimensiune maxima");
    for (int i = 0; i < n; i++){
        this->t[i] = 0;
    }
}

int main(){
    int n;
    Vector v(2);
    v.setElement(0,4);
    try{
        v.setElement(3,9);
    } catch (exception &e){
        cerr<<"Eroare:"<<e.what()<<endl;
    }
    v.afisare();
    getch();
    return 0;
}

OUTPUT:
Eroare:Depasire limite
4 0
```

Observăm că se aruncă `out_of_range` (obiectul derivat) și folosim un `catch` care prinde obiectul - `exception` (vezi tabel).

Observații: void Xhandler(int test) throw(int, char, double)

- se poate specifica ce excepții aruncă o funcție
- se restricționează tipurile de excepții care se pot arunca din funcție
- **un alt tip nespecificat termină programul: - apel la unexpected() care apelează abort() - se poate redefini**
- re-aruncarea unei excepții: throw; // fără excepție din catch

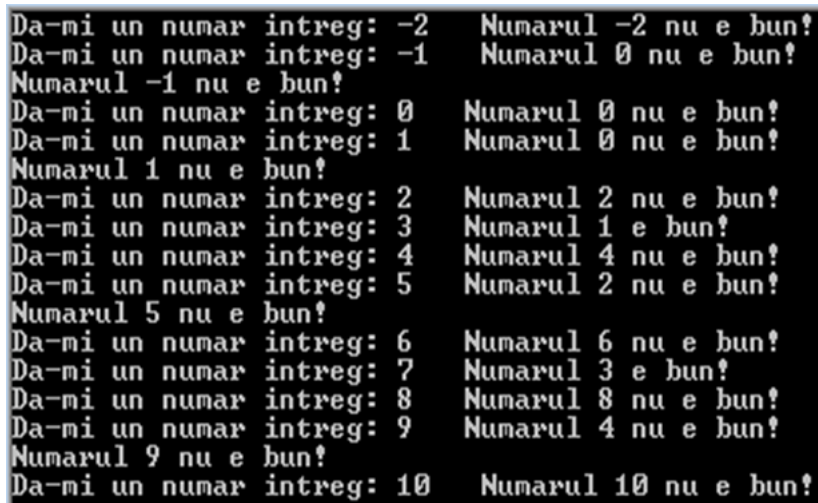
```
1 #include <iostream>
2 #include <exception>
3 #include <stdlib.h>
4
5 using namespace std;
6
7 // exceptii si functia "unexpected"
8 // ... merge numai sub g++
9
10 class Unu { };
11
12 class Doi { };
13
14 void g()
15 {
16     throw "Surpriza.";
17 }
18
19 void fct(int x) throw (Unu, Doi)
20 {
21     switch (x)
22     {
23         case 1:
24             throw Unu();
25         case 2:
26             throw Doi();
27     }
28     g();
29 }
30
31 void my_unexpected() {
32     cout << "Exceptie neprevazuta.";
33     exit(1);
34 }
35
36 int main()
37 {
38     set_unexpected(my_unexpected);
39     try {
40         fct(3);
41     } catch (Unu) {
42         cout << "Exceptie 1" << endl;
43     } catch (Doi) {
44         cout << "Exceptie 2" << endl;
45     }
46     return 0;
47 }
```

Listing 3: Exemplu Curs Dorel Lucanu

Exemplu examen:

XVIII. Spuneți dacă programul de mai jos este corect. În caz afirmativ, spuneți ce afișează pentru o valoare întreagă citită egală cu 7, în caz negativ spuneți de ce nu este corect.

```
#include <iostream.h>
float f(int y)
{ try
  { if (y%2) throw y/2;
  }
  catch (int i)
  { if (i%2) throw;
    cout<<"Numarul "<<i<<" nu e bun!"<<endl;
  }
  return y/2;
}
int main()
{ int x;
  try
  { cout<<"Da-mi un numar intreg: ";
    cin>>x;
    if (x) f(x);
    cout<<"Numarul "<<x<<" nu e bun!"<<endl;
  }
  catch (int i)
  { cout<<"Numarul "<<i<<" e bun!"<<endl;
  }
  return 0;
}
```



Da-mi un numar intreg: -2	Numarul -2 nu e bun!
Da-mi un numar intreg: -1	Numarul 0 nu e bun!
Numarul -1 nu e bun!	
Da-mi un numar intreg: 0	Numarul 0 nu e bun!
Da-mi un numar intreg: 1	Numarul 0 nu e bun!
Numarul 1 nu e bun!	
Da-mi un numar intreg: 2	Numarul 2 nu e bun!
Da-mi un numar intreg: 3	Numarul 1 e bun!
Da-mi un numar intreg: 4	Numarul 4 nu e bun!
Da-mi un numar intreg: 5	Numarul 2 nu e bun!
Numarul 5 nu e bun!	
Da-mi un numar intreg: 6	Numarul 6 nu e bun!
Da-mi un numar intreg: 7	Numarul 3 e bun!
Da-mi un numar intreg: 8	Numarul 8 nu e bun!
Da-mi un numar intreg: 9	Numarul 4 nu e bun!
Numarul 9 nu e bun!	
Da-mi un numar intreg: 10	Numarul 10 nu e bun!

2 Proiectarea descendentă a claselor. Moștenirea în C++

- Controlul accesului la clasa de bază.
- Constructori, destructori și moștenire.
- Redefinirea membrilor unei clase de bază într-o clasa derivată.
- Declarații de acces.

2 modalități (COMPUNERE (HAS A) și MOȘTENIRE (IS A));

- “compunere” - noua clasă “este compusă” din obiecte reprezentând instanțe ale claselor deja create (Embedded Object - îl voi folosi ca și câmp al obiectului din noua clasă);
- “moștenire” - se creează un nou tip al unei clase deja existente

Moștenirea în C++

Prin derivare se obțin clase noi, numite clase derivate, care moștenesc proprietățile unei clase deja definite, numită clasă de bază. Clasele derivate conțin toți membrii clasei de bază, la care se adaugă noi membrii, date și funcții membre.

Dintr-o clasă de bază se poate deriva o clasă care, la rândul său, să servească drept clasă de bază pentru derivarea altora. Prin această succesiune se obține o **ierarhie de clase**.

Se pot defini clase derivate care au la bază mai multe clase, înglobând proprietățile tuturor claselor de bază, procedeu ce poartă denumirea de **moștenire multiplă** - *de ex., noi suntem obiecte din moștenire multiplă - provenim din 2 baze: clasa mamă și clasa tată.*

Clasa de bază se mai numește clasa parinte sau superclasa, iar clasa derivată se mai numește subclasa sau clasa copil.

Moștenire vs Compunere

Moștenirea este asemănătoare cu procesul de includere a obiectelor în obiecte (procedeu ce poartă denumirea de compunere), dar există câteva elemente caracteristice moștenirii:

- codul poate fi comun mai multor clase;
- clasele pot fi extinse, fără a recompila clasele originare;
- funcțiile ce utilizează obiecte din clasa de bază pot utiliza și obiecte din clasele derivate din această clasă.

Exemplu 1 Curs

```
using namespace std;

class Baza
{
public:
    void afis() {cout<<"B";}
};

class Derivata: public Baza{
public:
    //void afis() {cout<<"D";}
    void functie_proprie(){}
};

int main()
{
    Derivata ob;
    ob.afis();
    ob.functie_proprie();

    ///
    Baza *p = &ob;
    p->
    (C) afis
```

```

1 class Baza {
2 protected:
3     int x;
4 public:
5     Baza(){x = 10;}
6 };
7
8 class Deriv1 : public Baza{};
9
10 class Deriv2 : public Deriv1 {
11 public:
12     void afis() {cout<<x;}
13 };
14
15 int main()
16 {
17     Deriv2 ob;
18     ob.afis();
19 }

```

Listing 4: Exemplu Anca Dobrovăț

Output: 10

Analog pt. class Deriv1 : protected Baza;

În schimb, dacă am private:

```

1 class Baza {
2 protected:
3     int x;
4 public:
5     Baza(){x = 10;}
6 };
7
8 class Deriv1 : Baza{};
9
10 class Deriv2 : public Deriv1 {
11 public:
12     void afis() {cout<<x;}
13 };
14
15 int main()
16 {
17     Deriv2 ob;
18     ob.afis();
19 }

```

Listing 5: Exemplu Anca Dobrovăț

EROARE

Inițializare de obiecte

Foarte important în C++: garantarea inițializării corecte => trebuie să fie asigurată și la compoziție și moștenire. La crearea unui obiect, compilatorul trebuie să garanteze apelul TUTUROR subobiectelor.

Problema: - cazul subobiectelor care nu au constructori implicați sau schimbarea valorii unui argument default în constructor.

De ce? - constructorul noii clase nu are permisiunea să acceseze datele private ale subobiectelor, deci nu le pot inițializa direct.

Rezolvare: - o sintaxa speciala: lista de initializare pentru constructori!


```

1 class Bar {
2     int x;
3     public:
4         Bar(int i) {x = i;}
5 };
6
7 class MyType: public Bar {
8     public:
9         MyType(int);
10 };
11
12 MyType :: MyType(int i) : Bar(i) { ... }

```

Listing 6: Exemplu Slide 25

```

1 class Alta_clasa { int a;
2     public:
3         Alta_clasa(int i) {a = i;}
4 };
5 class Bar { int x;
6     public:
7         Bar(int i) {x = i;}
8 };
9 class MyType2: public Bar {
10     Alta_clasa m; // obiect m = subobiect in cadrul clasei MyType2
11     public:
12         MyType2(int);
13 };
14 MyType2 :: MyType2(int i) : Bar(i), m(i+1) { ... } // !!!
15 // Modalitate prin care pot sa accesez datele private lor locale din interiorul altor
    clase

```

Listing 7: Exemplu Slide 26

Nu folosesc numele clasei aici, în cazul compunerii, când deja am un obiect (folosesc numele clasei la moștenire - exemplu slide 25). Este ca și cum folosesc constructorul de copiere.

Constructorii clasei derivate

Pentru crearea unui obiect al unei clase derivate, **se creează inițial un obiect al clasei de bază prin apelul constructorului acesteia, apoi se adaugă elementele specifice clasei derivate prin apelul constructorului clasei derivate.**

Declarația obiectului derivat trebuie să conțină valorile de inițializare, **atât pentru elementele specifice, cât și pentru obiectul clasei de bază.**

Această specificare se atașează la antetul funcției constructor a clasei derivate.

În situația în care clasele de bază au definit **constructor implicit** sau **constructor cu parametri impliciti**, nu se impune specificarea parametrilor care se transferă către obiectul clasei de bază.

```

1 class Forma {
2     protected:
3         int h;
4     public:
5         Forma(int a = 0) { h = a; }
6 };
7
8 class Cerc: public Forma {
9     protected:
10         float raza;
11     public:
12         Cerc(int h=0, float r=0) : Forma(h), raza(r) {}
13 };

```

Listing 8: Exemplu Slide 31

Constructorii clasei derivate

Constructorul de copiere

Se pot distinge mai multe situații.

1) Dacă ambele clase, atât clasa derivată cât și clasa de bază, nu au definit constructor de copiere, se apelează constructorul implicit creat de compilator. Copierea se face membru cu membru.

2) Dacă clasa de bază are constructorul de copiere definit, dar clasa derivată nu, pentru clasa derivată compilatorul creează un constructor implicit care apelează constructorul de copiere al clasei de bază.

3) Dacă se definește constructor de copiere pentru clasa derivată, acestuia îi revine în totalitate sarcina transferării valorilor corespunzătoare membrilor ce aparțin clasei de bază.

```
1 #include <iostream>
2
3 using namespace std;
4
5 class Baza {
6 protected:
7     int x;
8 public:
9     Baza() {
10         x = 10;
11     }
12     Baza(const Baza& ob){x = 22*ob.x;}
13 };
14
15 class Deriv: public Baza {
16     int y;
17 public:
18     Deriv()//:Baza() - el oricum isi face trimiterea catre constructorul din clasa de
        baza pt. ca obiectul derivat nu stie sa se creeze pana nu este creata clasa de
        baza
19     {
20         y = 20;
21     }
22     void afis()
23     {
24         cout<<x<<" "<<y<<endl;
25     }
26     //Deriv(const Deriv& ob){y = ob.y;} - dar ne revine sarcina "transferarii
        valorilor corespunzatoare membrilor ce apartin clasei de baza"
27     Deriv(const Deriv& ob):Baza(ob){y = ob.y;} // ii spunem ca vrem sa folosim
        explicit constructorul de copiere din clasa de baza
28
29     // altfel se facea trimitere catre constructorul de initializare din clasa de
        baza!
30 };
31
32 int main()
33 {
34     Deriv ob1;
35     ob1.afis();
36
37     Deriv ob2(ob1); //daca nu aveam constr. de copiere definit - caz 1)
38     // daca aveam constr. de copiere definit doar in clasa de baza - caz 2)
39     // acum - caz 3)
40     ob2.afis();
41 }
```

Listing 9: Exemplu Anca Dobrovăț

Dacă am făcut trimitere către constructorul de copiere din clasa de bază, nu pot să fac trimitere și către operatorul de atribuire din clasa de bază? Bineînțeles! (slide 36)