Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Facultatea de Automatică şi Calculatoare

PROGRAMARE II

Curs 13

Excepţii

Tratarea erorilor

- Funcţie ce cauta o valoare într-un array:

```
int search_val(int v[], int n, int val)
{
	for (int i = 0; i < n; i++)
		if (v[i]==val)
			return i;
	return -1;
}
```

- Folosirea valorilor de return pentru semnalarea erorilor poate fi dificilă de înţeles pentru utilizatori.
- De cele mai multe ori, necesită parcurgerea funcţiei pentru întelegerea valorilor.

Tratarea erorilor

- Considerăm o funcţie ce returnează rezultatul împărţirii a două numere intregi

```
int imp (int a, int b)
{
	return a/b;
}
```

- Cum putem semnala eroare în cazul împărţirii la 0?

```
cout << imp (4,0);
```

- Programul se va bloca şi va da eroare

Tratarea erorilor

- Un alt caz de eroare des întâlnit este reprezentat de cazul când un utilizator introduce un alt tip de dată decât cel așteptat de program

```
int nr1, nr2;

cin >> nr1;
cin >> nr2;

cout << imp (nr1,nr2);
```

- La introducerea unei valori greșite, de exemplu un caracter, programul va afișa

```
w
1
```

Excepții

- Excepțiile reprezintă un mecanism de tratare a erorilor
- Erorile sunt transmise către unele funcții speciale numite *handle-uri*
- Se separă în acest fel metodele de tratare a erorilor de codul propriu-zis al programelor

Excepțiile în c++

- Sunt implementate cu ajutorul a trei cuvinte cheie:
  - throw
  - try
  - catch

Excepții - throw

- Lansarea (aruncarea) excepțiilor este folosită pentru a semnala existența unei excepții sau a unei erori
- Pentru a lansa o excepție se folosește cuvantul cheie `throw`
- Lansarea excepțiilor presupune transmiterea unui singur parametru
- Exemple:

```
throw -1;
throw "Impartire la 0";
```

Excepții - try

- Cuvântul cheie `try` se folosește pentru a defini un bloc de instrucțiuni.
- Acest bloc `try` se comportă ca un observator care așteaptă lansarea de excepții
- Excepțiile pot fi lansate numai in cadrul blocului definit de cuvantul cheie `try`
- Exemplu:

```
try
{
	throw -1;
}
```

Excepții - catch

- Tratarea excepțiilor se face prin definirea unui bloc de instrucțiuni prin cuvântul cheie `catch`
- Acest bloc trebuie să se afle imediat dupa blocul `try`
- Se comportă precum o funcție cu un singur parametru
- Unui block `try` ii pot corespunde mai multe blocuri `catch` in funcție de tipul parametrului transmis

```
catch (int e)
{
	cout << "Exceptie de tip int " << e << endl;
}
```

Excepții - exemplu

```
int main ()
{
  try
  {
    throw 1;
  }
  catch (int e)
  {
    cout << "Exceptie nr " << e << endl;
  }
  return 0;
}
```

Excepții - exemplu

```
int main ()
{
   double nr;
   cin >> nr;

   try
   {
       if (nr < 0.0)
          throw "Nu pot extrage radicalul unui nr negativ!";
       cout << "Radical din " << nr << " este " << sqrt (nr);
   }
   catch (char *e)
   {
       cout << "Eroare: " << e << endl;
   }
   return 0;
}
```

Excepții în cadrul funcțiilor

- Instrucțiunile `throw` pot să nu fie plasate direct în cadrul unor blocuri `try`
- Aceste instrucțiuni pot fi plasate în cadrul unor funcții apelate în interiorul unui bloc `try`
- Pentru o funcție de extragere a radicalului lansarea unei excepții poate fi plasată în interiorul funcției, dacă funcția este apelată într-un bloc `try`:

```
double radical (double nr)
{
       if (nr < 0.0)
              throw "Nu pot extrage radicalul unui nr
negativ!";
       return sqrt (nr);
}
```

Excepții în cadrul funcțiilor

```
int main ()
{
	double nr;
	cin >> nr;
	try
	{
		cout << "Radical din " << nr << " este " <<
			radical (nr);
	}
	catch (char *e)
	{
		cout << "Eroare: " << e << endl;
	}
	return 0;
}
```

Excepții în cadrul funcțiilor

- Exemplu:

http://www.learncpp.com/cpp-tutorial/153-exceptions-functions-and-stack-unwinding/

Excepții netratate

- Considerăm următoarea funcție main pentru exemplul de extragere a radicalului:

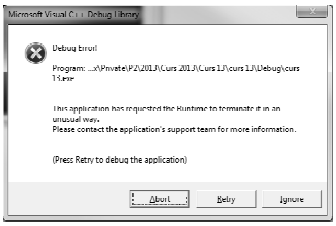
```
int main ()
{
	double nr;
	cin >> nr;

	cout << "Radical din " << nr<< " este " << radical (nr);
	return 0;
}
```

- Funcția de extragere a radicalului ramâne aceeași.

```
double radical (double nr)
{
	if (nr < 0.0)
		throw "Nu pot extrage radicalul unui nr
negativ!";
	return sqrt (nr);
}
```

Excepții netratate

- La introducerea unei valori negative, rularea programului se va opri brusc și va fi semnalată o eroare:



Excepții netratate

- Funcțiile pot lansa excepții care pot să nu aibă nici o modalitate de tratare a lor
- Pentru a preveni aceste situații, se poate introduce un handler catch-all:

```
try
{
	cout << "Radical din " << nr <<" este " << radical (nr);
}
catch (int e)
{
	cout << "Eroare: " << e << endl;
}
catch (...)
{
	cout << "Eroare necunoscuta" << endl;
}
```

Excepții în funcții membre

- Se folosesc
  - În special în constructori pentru a semnala unele erori la crearea obiectelor
  - Când se dorește semnalarea unei erori fără a duce la terminarea programului

Excepții standard

- Librăria c++ standard furnizează o clasă de bază pentru tratarea obiectelor furnizate ca excepții
- Această clasa este `std::exception`
- Conține o funcție virtuală `what` ce returnează un char *

```
int main()
{
    try {
        while (true) {
            new int[100000000ul];
        }
    } catch (const std::bad_alloc& e) {
       cout << "Allocation failed: " << e.what() << '\n';
    }
}
```

http://en.cppreference.com/w/cpp/memory/new/bad_alloc