Tratarea excepțiilor

Vîlculescu Mihai-Bogdan

19 aprilie 2020

Contents

1 Introducere

1.1 Definiție

Erori neprevăzute care apar **în timpul execuției** unui program. De obicei, acestea duc la **oprirea neașteptată** a acelui program și apar din *neatenția developer-ului*.

1.2 Exemple

```cpp
int main() {
    int x, y;
    cin >> x >> y;

    // x = 5, y = 0
    cout << x / y; // impartire la zero
    return 0;
}
```

Exemplu exceptie (împărțire la zero)

Exemplul acesta este unul *simplist*, dar **în practică** vom întâlni probleme mai grave, precum:

Exemple de excepții din practică

- **Conexiunea la baza de date a eșuat**
- **Fișierul nu a putut fi deschis cu succes**

Conversia nu a fost realizată cu succes

Input-ul transmis este invalid

2 Block-ul try-catch-throw

2.1 Introducere

Pentru a putea **trata** aceste probleme, evitând **oprirea execuției programului**, a fost introdus acest bloc de cod: **try-catch**.

2.2 Sintaxă

Sintaxa seamănă mult cu cea folosită pentru **do-while**.

```
try {
    // cod care poate sa arunce o exceptie
} catch (exceptie) {
    // cod care trateaza exceptia prinsa ca programul sa
    // poata continua
}
```

În interiorul **block-ului try**, se poate **arunca manual** o excepție prin intermediul cuvântului cheie **throw**.

```
try {
    throw exceptie;
} catch (exceptie) {
    // trateaza exceptia
}
```

Vor fi mai jos exemple în care vom observa mai bine de ce avem nevoie de aceste instrumente.

2.3 Mod de funcționare

Acum, partea importantă este să vedem cum funcționează de fapt acest **mecanism de tratare al excepțiilor**.

Pași de execuție pentru un block try-catch

```
1. Se execută, pe rând, ce se află în block-ul try.

2. Dacă se întâlnește o excepție sau este aruncată
una (throw) se oprește execuția block-ului try.

    2.1. Se caută un block catch care să prindă
        exact tipul aruncat din try.

    2.2. Dacă acest tip se găsește, se va executa
        ce se află în interiorul acelui catch.

        2.2.1. Dacă eroarea nu este una prea mare, se
            continuă execuția după block-ul try-catch.

        2.2.2. Altfel, se oprește execuția programului.

    2.3. Altfel, se oprește execuția programului.

3. Dacă nu se întâlnește nicio excepție în try,
se sare peste block-ul catch.
```

Pași de execuție pentru try-catch

2.4 Tipuri de excepții

Ce am observat până acum este importanța **tipurilor excepțiilor**. Să urmărim ce poate fi *aruncat* și ce poate fi *prins*:

2.4.1 Tipuri de date obișnuite

În C++, poate fi aruncat **orice tip de date** cu care se lucrează în mod obișnuit:

- **int**.
- **float**.
- **double**.
- **string**.
- **const char***.
- **char**.

```
int x, y;
cin >> x;
while (true) {
    try {
        cin >> y;
        if (y == 0) {
            // se cauta un catch care prinde "const char*"
            throw "Numar invalid. y trebuie sa fie > 0";
        }

        // daca am ajuns aici => nu a fost aruncata exceptia
        break; // avem un y valid => iesim din while
    } catch (const char* e) {
        cout << e;
    }
}
cout << x / y;
```

Excepții ca tipuri de date obișnuite

IMPORTANT

Când aruncați ceva de forma **"String literal"**, tipul de date corespondent este **const char***.

```
throw "String literal" => catch (const char*)
throw string("String literal") => catch (string)
```

2.4.2 Obiecte excepții

Am considerat importantă distincția asta. Când vorbim de **obiecte excepții**, ne referim la obiecte ale **claselor derivate** din **clasa exception**.

Le dezvoltăm separat, pentru că sunt mai folosite **în practică** decât aruncatul de tipuri de date obișnuite.

```
class DivideByZeroException : public exception {
    public:
        // suprascriem metoda "what()" din clasa exception
        const char* what() {
            return "Numitorul unei impartiri e diferit de 0";
        }
};

int main() {
    int x, y;
    cin >> x;
    while (true) {
        try {
            cin >> y;
            if (y == 0) {
                throw DivideByZeroException();
            }

            break;
        } catch (DivideByZeroException e) {
```

```
            cout << e.what();
        }
    }

    cout << x / y;
    return 0;
}
```

Excepții ca obiecte

2.5 Propagarea excepțiilor

Excepțiile **se propagă de la o funcție la alta**, în funcție de cum sunt apelate acestea. În exemplul de mai jos, vom considera **clasa DivideByZeroException** implementată.

```
void verifica (int y) {
    if (y == 0) {
        throw DivideByZeroException();
    }
}

void citeste (int& y) {
    cin >> y;
    verifica(y);
}

int main () {
    int x, y;
    cin >> x;
    while (true) {
        try {
            citeste(y);
            break;
        } catch (DivideByZeroException e) {
            cout << e.what();
```

```
        }
    }

    cout << x / y;
    return 0;
}
```

Propagarea excepțiilor

Să urmărim cum *se propagă excepția* din exemplul de mai sus:

Propagarea excepțiilor

1. Se apelează funcția "citeste(y)" din main
din interiorul unui block try.

2. Se apelează funcția "verifica(y)".

3. Dacă y=0, se aruncă excepția "DivideByZeroException".

    3.1. Excepția este aruncată fără să se afle
    într-un block try => excepția este aruncată
    mai departe către funcția apelantă.

    3.2. În funcția apelantă ("citeste"), apelul
    "verifica(y)" nu se află într-un block try
    => excepția este aruncată mai departe către
    următoarea funcție apelantă ("int main()").

    3.3. În int main(), apelul "citeste(y)" se află
    într-un block try => se caută un catch care
    să corespundă tipului de date aruncat.

```
3.3.1. Dacă se găsește un
"catch (DivideByZeroException)"
=> se execută instrucțiunile din catch
și programul continuă.

3.3.2. Dacă nu se găsește acel catch
=> programul întoarce o eroare.
```

IMPORTANT

- Dacă nu ar fi existat **block-ul try** din **int main** programul ar fi întors o **eroare la executare (runtime)**.

```
int main() {
    citeste(y); // eroare la runtime
}
```

- Dacă de-a lungul **lanțului de propagare** ar fi fost găsit un **block try-catch**, atunci propagarea ar fi **încetat**.

```
void citeste (int& y) {
    // exceptia nu se mai propaga in main
    try {
        verifica(y);
    } catch (DivideByZeroException e) {
        cout << e.what();
    }
}
```

3 Resurse

- https://www.geeksforgeeks.org/exception-handling-c/
- http://www.cplusplus.com/doc/tutorial/exceptions/
- TutorialsPoint
- Niste exemple mai dragute de pe site-ul Microsoft