# Tutoriat 5
# Try-catch

1. La ce folosește?

   Tratarea unor erori neprevăzute care apar în timpul execuției unui program. Acestea duc la oprirea neașteptată a acelui program, dacă nu sunt prinse într-un bloc **try-catch**.

   Programul compilează, însă aruncă o eroare la rulare(runtime) și se oprește din execuție la instrucțiunea care o produce.

```cpp
#include<iostream>
using namespace std;

int main(){
    int x = 10, y = 0;
    cout<<x/y;  //eroare la runtime -> impartire la 0
    return 0;
}
```

```cpp
#include<iostream>
using namespace std;

int main(){
    int x = 10, y = 0;
    try {
        if(y ==0) {
            throw runtime_error("0 division");
        }else{
            cout<<x/y;
        }
    }
    catch(...){
        cout<<"A aparut o eroare"<<endl;
    }
    cout<<"A trecut de impartire";
    return 0;
}
```

   Programul va afișa:

   A aparut o eroare
   A trecut de impartire

2. Pașii de execuție pentru un bloc **try-catch**

1. Se execută, pe rând, ce se află în blocul **try**.
2. Dacă se întâlnește o excepție sau este aruncată una(throw) se oprește execuția blocului **try**.
    2.1. Se caută un bloc **catch** care să prindă exact tipul aruncat(NU se pot produce cast-uri implicite ca: int -> double, char -> int etc..) din **try**.
    2.2. Daca acest tip se găsește, se va executa ce se află în interiorul acelui **catch** și se continuă execuția programului după blocul **try-catch**.
    2.3. Altfel, se oprește execuția programului.
3. Dacă nu se întâlnește nicio excepție în **try**, se sare peste blocul **catch.**

Obs: Instrucțiunea *catch(...){…}* prinde orice tip de eroare.

3. Tipuri de date pentru excepții

În C++, poate fi aruncat orice tip de date de la cele deja definite(int, float, string, const char* etc..), la clase definite în program/clase care moștenesc alte tipuri de excepții deja definite:

```
int main(){
    int x = 10, y = 0;
    try {
        if(y ==0) {
            throw "0 division error";
        }else{
            cout<<x/y;
        }
    }
    catch(const char * error){
        cout<<error<<endl;
    }
    return 0;
}
```

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

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

```
class E{
    string message;
public:
```

```
    E(string message): message(message){}

    const string &getMessage() const {
        return message;
    }

};
int main(){
    int x = 10, y = 0;
    try {
        if(y ==0) {
            throw E("0 divison");
        }else{
            cout<<x/y;
        }
    }
    catch(E& e){
        cout<<e.getMessage()<<endl;
    }
    return 0;
}
```

```
class Exception : public runtime_error {
public:
    // Defining constructor of class Exception
    // that passes a string message to the runtime_error class
    Exception()
          : runtime_error("0 division error")
    {
    }
};

int main(){
    int x = 10, y = 0;
    try {
        if(y ==0) {
            throw Exception();
        }else{
            cout<<x/y;
        }
    }
    catch(Exception& e){
        cout<<e.what()<<endl;
    }
    return 0;
}
```

4. Propagarea excepțiilor

Excepțiile se propagă de la o funcție la alta, în funcție de cum sunt apelate acestea.

*Obs: Propagarea prin funcții se oprește la întâlnirea primului bloc **catch** cu tipul potrivit pentru eroarea propagată.*

Programul de mai jos cere introducerea unei valori pentru y până când aceasta este diferită de 0, pentru a efectua împărțirea :

```
class Exception : public runtime_error {
    ....
};

void verify (int y) {
    if (y == 0) {
        throw Exception ();
    }
}
void readY (int& y) {
    cin >> y;
    verify(y); // (apel 2)
}
int main () {
    int x = 10, y;
    while (true) {
        try {
            readY(y); // (apel 1)
            break; // daca NU a aruncat exceptie se excuta break;
        } catch (Exception  e) {
            // a fost aruncata o exceptie, propagate in functii, prinsa in catch
            cout << e.what();
        }
    }
    cout << x / y;
    return 0;
}
```