

Section 18 : Exception Handling

15 April 2022 16:49

© Rajat Kumar

<https://www.linkedin.com/in/imRajat/>

<https://github.com/im-Rajat>

Section 18 : Exception Handling

3 Types of Errors :

- Syntax Error (Removed using help of **compiler**)
- Logical Error (Program run successfully but result is different. **Debugger** helps us to run the program line by line, statement by statement.)
- Runtime Error (Reasons - Bad input, unavailability / problem with resources)

Runtime error :

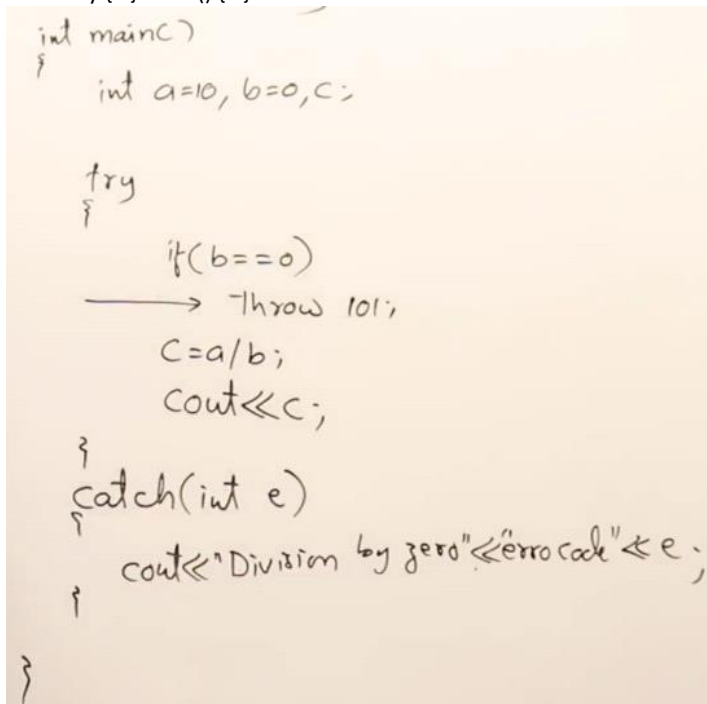
- In case of runtime error. Program crashes, program stops abruptly, abnormally without completing its execution. Suddenly it will stop.
- And that's how the runtime errors are dangerous for the user.
- User is responsible because as a developer he did his job perfectly and user is not providing proper resources or proper input. So the program is failing. So the failure of a program responsibility goes to user.

Exception :

- These runtime errors are called exceptions means why we are using an exception term, because of exceptional cases.
- So what is the objective of exception? Handling, giving a proper message to the user, informing about the exact problem, and also providing him guidance to solve that problem.

Exception Handling :

- Construct/Structure of Exception Handling :
 - Try {...} catch() {...}



```
int main()
{
    int a=10, b=0, c;

    try
    {
        if(b==0)
            → throw 101;
        c=a/b;
        cout<<c;
    }
    catch(int e)
    {
        cout<<"Division by zero" <<"error code" << e;
    }
}
```

- If there is any error (on any line - up to their lines with execute) inside try block then it will jump to the catch block and execute the statements inside catch block.
- And if there are no error in try block, then catch block will not execute.
- It's similar to if else.
- In line throw 101, we are throwing an exception. So we have to throw exception in C++. C++ compiler doesn't have any built-in mechanism for throwing exceptions.
 - Throw 101 will be caught by catch block, where int e will be 101.
- In Java compiler will check for it and write the code for taking an exception. But C++ compiler doesn't do that.

Example :

```

int main()
{
    int x = 10, y = 0, z;

    try {
        if (y == 0) {
            throw y;
            // without throw it will execute
            // complete try block
        }
        z = x/y;
        cout<<z<<endl;
    }
    catch(int e) {
        cout<<"Division by "<<e<<" not possible\n";
    }

    cout<<"Bye"<<endl;

    return 0;
}

```

Throw and Catch between Functions :

- It's basically to communicate between the functions.
- Exception Handling is more useful in between the function, otherwise the errors we can check just using if else also.

```

int division(int a,int b)
{
    if(b==0)
        throw 1;
    return a/b;
}

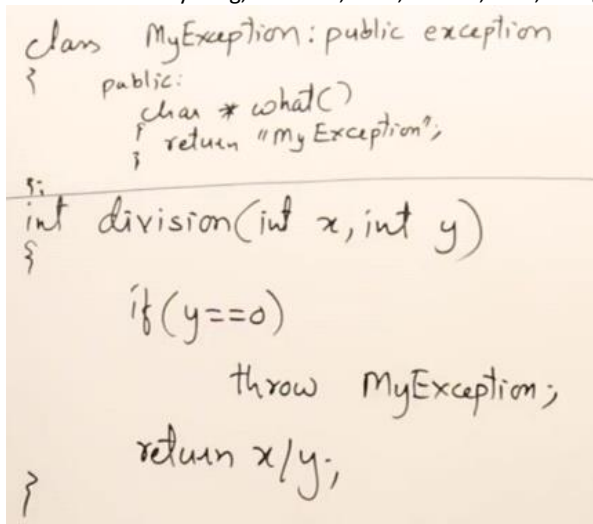
int main()
{
    int x=10,y=2,z;

    try
    {
        z=division(x,y);
        cout<<z<<endl;
    }
    catch(int e)
    {
        cout<<"Division by zero "<<e<<endl;
    }
    cout<<"Bye"<<endl;
}

```

More about Throw :

- We can throw anything, int value, float, double, char, string, or even our own class object.



```

class MyException: public exception
{
public:
    char * what()
    {
        return "MyException";
    }
}

int division(int x,int y)
{
    if(y==0)
        throw MyException;
    return x/y;
}

```

- We can also mention that the function is throwing the exception.
 - `int division(int x, int y) throw(MyException) {...}`

```

int division(int x, int y) throw(int)
{
    if (y == 0)
        throw 4;
    return x/y;
}

```

- A programmer has developed some class than the other programmers can know from its signature that this function through some exceptions. So we are supposed to catch the exception.

More about Catch :

- We can have multiple catch blocks for each type of data.

```

try
{
    1 _____ int
    2 _____ ✓
    3 _____ float
    4 _____ ✓
    5 _____ ✓
}
catch(int e)
=
catch(float e)
=
catch(...)
=
    Catch All Exception
}

```

- Catch(...) can catch all exception, it's called ellipse. It can handle any type of exception.
 - If we are using this means we are not interested in giving a clear message to the user for every type.
 - Its recommend to have multiple catch block for different data types.
 - This catch(...) has to be last catch block.
- We can have try block inside try block and so on.
- We can do nesting of try and catch blocks.
- We must write for the child class then parent class for if the exception classes are defined in the hierarchy,
 - Write child class exception first then parent class.
- Void fun() throw() can throw no exception.
- Exception are raised by program.
- A try block can have multiple catch blocks .
- Catch-All must be defined as last block.
- Classes in inheritance can be used in catch block as child class first then base class.