# Exception Handling


**Mohammad Tasin (Tasin Coder)**

# Agenda

- What is exception handling
- Why does an exception occur?
- try, catch, throw, catch all
- C++ Standard Exceptions
- Demo of exception one by one
- Define New Exceptions
- Handle Any Type of Exceptions

# What is exception handling

- Exception handling is a mechanism in C++ used to handle the run time error so that the program's normal flow can be maintained.
- If an exception occurs in your code, then the rest of the code is not executed.
- Therefore, the C++ compiler creates an exception object.
- This exception object directly jumps to the default catch mechanism; there is a default message that prints on the screen and our program gets terminated.

# Why does an exception occur?

- Exceptions in C++ can occur due to the following reasons.
  - Incorrect data entered by the user, e.g., dividing a no. by zero.
  - Opening a file that doesn’t exist in the program.
  - Network connection problem
  - Server down problem

# try, catch, throw

- **try : -** Represents a block of code that can throw an exception. **try is a Keyword.**
- **catch :-** Represents a block of code that is executed when a particular exception is thrown. **catch is a Keyword.**
- **throw :-** used to throw an exception. Also used to list the exceptions that a function throws but doesn`t handle itself. **throw is a Keyword.**

```
int main()  {
    int a, b;
    cout<<"Enter two Number : ";
    cin>>a>>b;
    try{
        if(b == 0)
            throw -1;
        if(b == 1)
            throw "Error";
        cout<<"Division is "<<a/b<<endl;
    }
    catch(int a)    {
        cout<<"Division By Zero"<<endl;
    }
    catch(...)  {
        cout<<"Error"<<endl;
    }
    cout<<"TasiNCoder"<<endl;
    return 0;
}
```

# C++ Standard Exceptions

- invalid_argument
- length_error
- out_of_range
- runtime_error
- overflow_error
- range_error
- underflow_error
- exception
- bad_alloc
- bad_cast
- bad_exception
- bad_typeid
- logic_error
- domain_error

| S.No | Exception & Description |
|---|---|
| 1 | **std::exception**<br>An exception and parent class of all the standard C++ exceptions. |
| 2 | **std::bad_alloc**<br>This can be thrown by new. |
| 3 | **std::bad_cast**<br>This can be thrown by dynamic_cast. |
| 4 | **std::bad_exception**<br>This is useful device to handle unexpected exceptions in a C++ program. |
| 5 | **std::bad_typeid**<br>This can be thrown by typeid. |
| 6 | **std::logic_error**<br>An exception that theoretically can be detected by reading the code. |
| 7 | **std::domain_error**<br>This is an exception thrown when a mathematically invalid domain is used. |

| | |
|---|---|
| 8 | **std::invalid_argument**<br>**This is thrown due to invalid arguments.** |
| 9 | **std::length_error**<br>**This is thrown when a too big std::string is created.** |
| 10 | **std::out_of_range**<br>**This can be thrown by the 'at' method, for example a std::vector and std::bitset<>::operator[]().** |
| 11 | **std::runtime_error**<br>**An exception that theoretically cannot be detected by reading the code.** |
| 12 | **std::overflow_error**<br>**This is thrown if a mathematical overflow occurs.** |
| 13 | **std::range_error**<br>**This is occurred when you try to store a value which is out of range.** |
| 14 | **std::underflow_error**<br>**This is thrown if a mathematical underflow occurs.** |

# Demo of exception one by one

```
int main()  {
    int a, b;
    cout<<"Enter two numbers : ";
    cin>>a>>b;
    try{
        if(b==0)
            throw "Division By Zero";
        cout<<"Div is "<<a/b<<endl;
    }
    catch(const char *ch)   {
        cout<<ch<<endl;
    }
    cout<<"TasiNCoder"<<endl;
    return 0;
}
```

# Define New Exceptions

```
int main()     {
    int a,b;
    cout<<"Enter two Number : ";
    cin>>a>>b;
    try{
        if(b == 0)
            throw DivisionByZero(b);
        cout<<"Division is "<<a/b<<endl;
    }
    catch(DivisionByZero d)  {
        cout<<"Division By Zero "<<d.what()<<endl;
    }
    cout<<"TasiNCoder"<<endl;
    return 0;
}
```

```
class DivisionByZero    {
    int a;
public:
    DivisionByZero(int a):a(a){}
    int what(){return a;}
};
```

# Handle Any Type of Exceptions

```
int main()  {
    int a, b;
    cout<<"Enter two numbers : ";
    cin>>a>>b;
    try{
        if(b==0)
            throw out_of_range("Invalid_argument");
        cout<<"Div is "<<a/b<<endl;
    }
    catch(std::exception &ch)   {
        cout<<ch.what()<<endl;
    }
    cout<<"TasiNCoder"<<endl;
    return 0;
}
```