Week 14: Exception Handling

By, Mahrukh Khan

Exception

- An exception is a situation, which occured by the runtime error. In other words, an exception is a runtime error. An exception may result in loss of data or an abnormal execution of program.
- It is a new feature that ANSI C++ included in it. Now almost all C++ compilers support this feature.

Exception Handling

- Exception handling is a mechanism that allows you to take appropriate action to avoid runtime errors.
- Exceptions allow a method to react to exceptional circumstances and errors (like runtime errors) within programs by transferring control to special functions called handlers. For catching exceptions, a portion of code is placed under exception inspection. Exception handling was not a part of the original C++.
- The error handling mechanism basically consists of two parts. These are:
    - To detect errors
    - To throw exceptions and then take appropriate actions

Types Of Exceptions

- There are two types of exceptions:
    - Synchronous exceptions
    - Asynchronous exceptions
- Errors such as: out of range index and overflow fall under the category of *synchronous* type exceptions. The C++ standard supports synchronous exception handling with a termination model. Termination means that once an exception is thrown, control never returns to the throw point.
- Errors that are caused by events beyond the control of the program are called *asynchronous* exceptions. Exception handling is not designed to directly handle asynchronous exceptions such as keyboard interrupts.

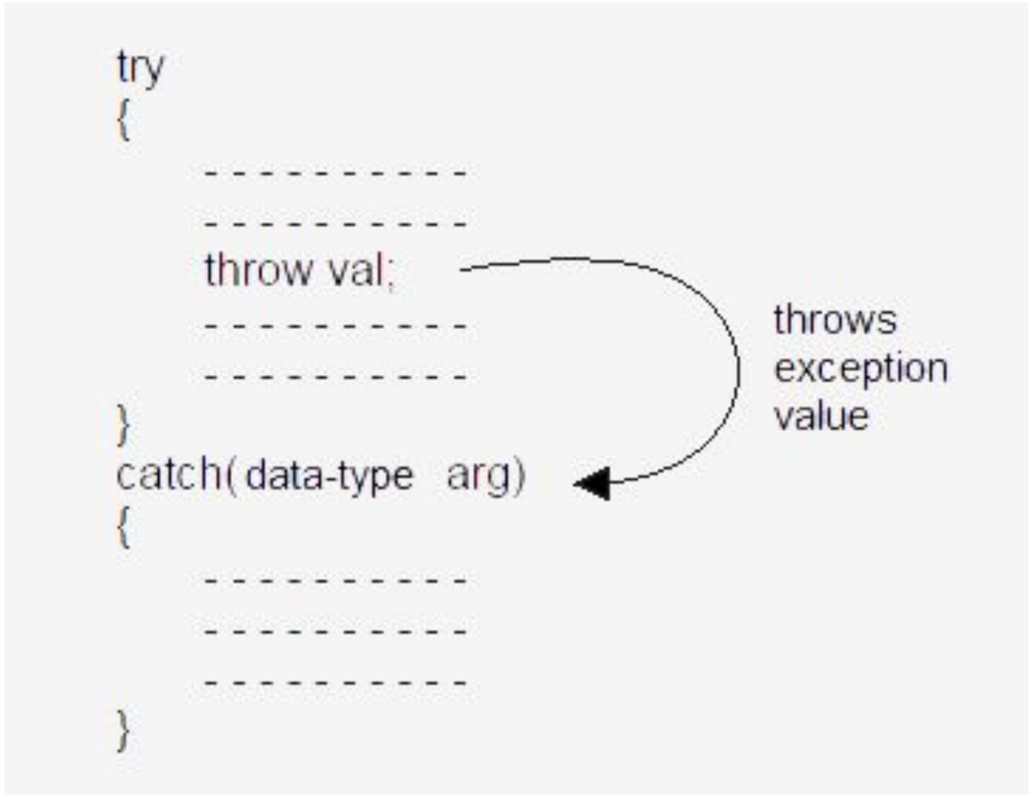
Steps

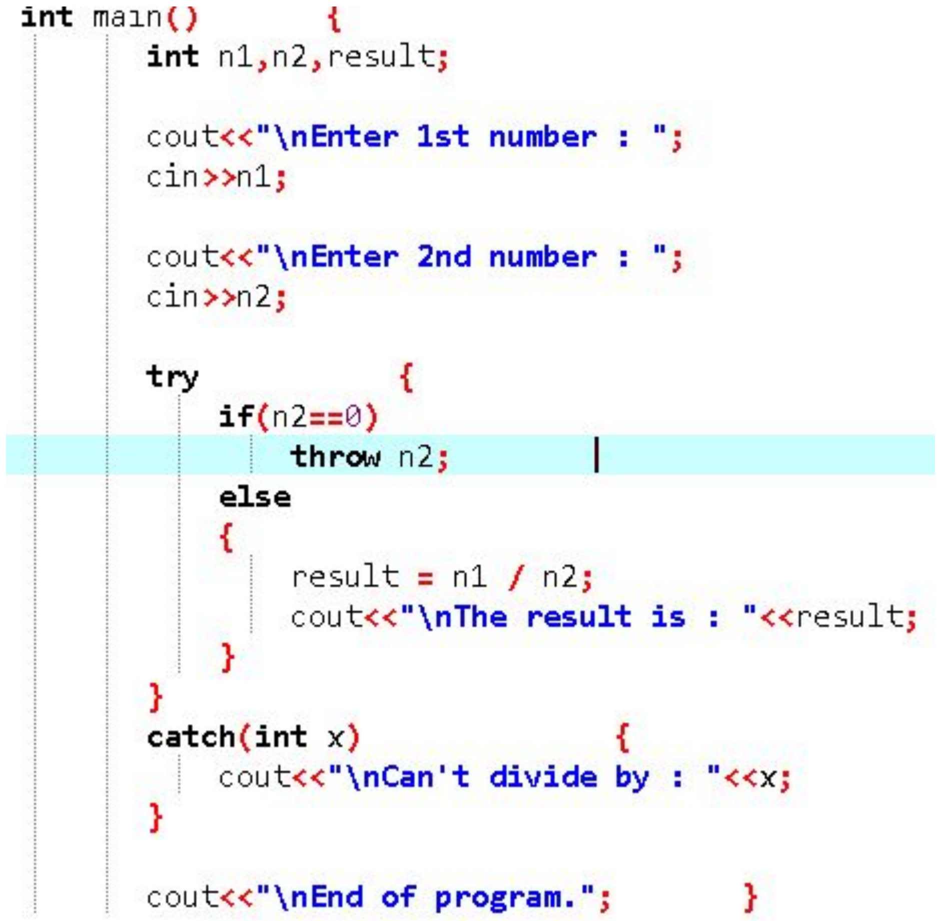
- This mechanism needs a separate error handling code that performs the following tasks:
    - Find and hit the problem (exception)
    - Inform that the error has occurred (throw exception)
    - Receive the error information (Catch the exception)
    - Take corrective actions (handle exception)

C++ Exception Handling

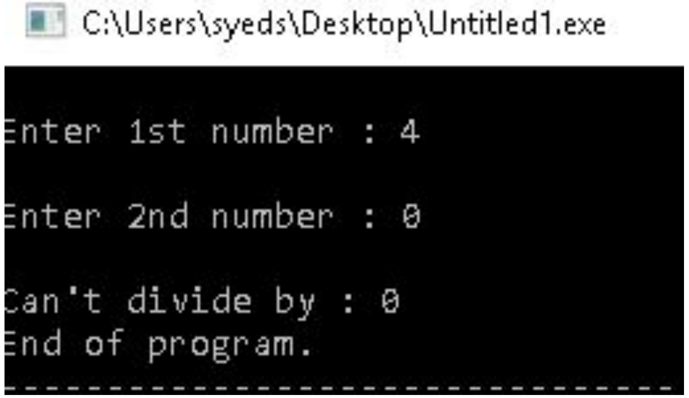
C++ provides three keywords to support exception handling.

- Try : The try block contain statements which may generate exceptions.
- Throw : When an exception occur in try block, it is thrown to the catch block using throw keyword.
- Catch :The catch block defines the action to be taken, when an exception occur.


```
try
{
        - - - - - - - - - - -
        - - - - - - - - - - -
        throw val;
        - - - - - - - - - - -
        - - - - - - - - - - -
}
catch( data-type  arg)
{
        - - - - - - - - - - -
        - - - - - - - - - - -
        - - - - - - - - - - -
}
```



```
int main()          {
        int n1,n2,result;

        cout<<"\nEnter 1st number : ";
        cin>>n1;

        cout<<"\nEnter 2nd number : ";
        cin>>n2;

        try             {
            if(n2==0)
                throw n2;
            else
            {
                result = n1 / n2;
                cout<<"\nThe result is : "<<result;
            }
        }
        catch(int x)            {
            cout<<"\nCan't divide by : "<<x;
        }

        cout<<"\nEnd of program.";        }
```



C:\Users\syeds\Desktop\Untitled1.exe

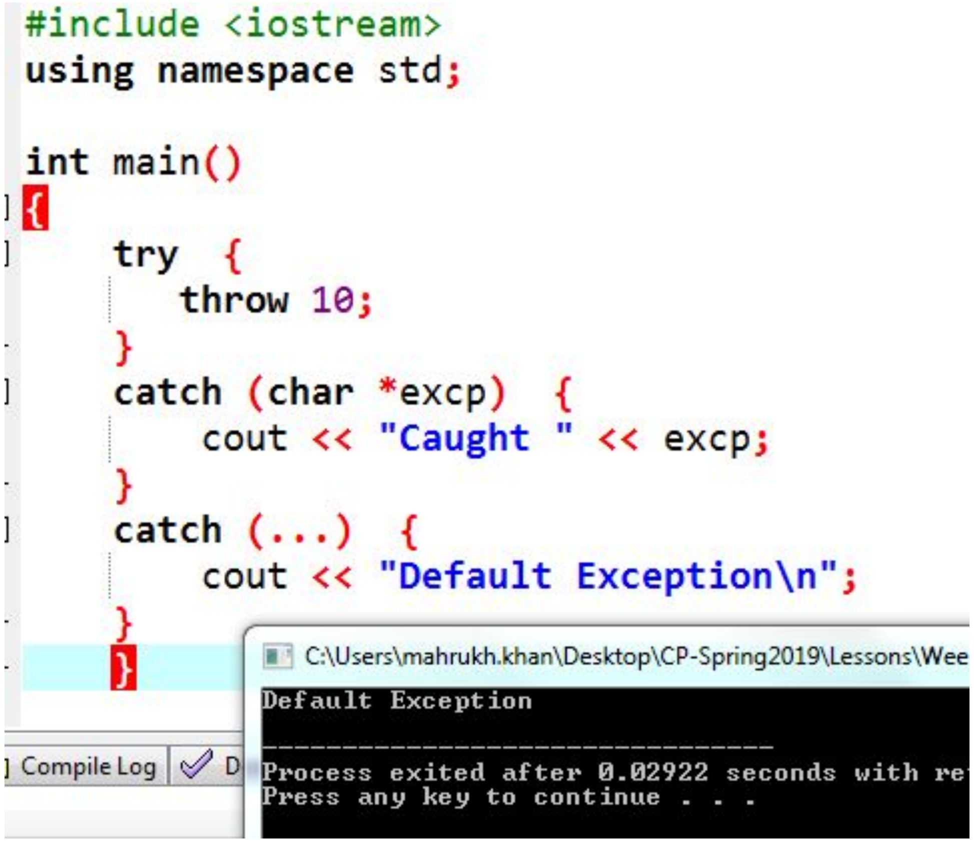
```
Enter 1st number : 4

Enter 2nd number : 0

Can't divide by : 0
End of program.
```

Advantages

- Programmers can deal with them at some level within the program
- If an error can't be dealt with at one level, then it will automatically be shown at the next level, where it can be dealt with
- Separation of Error Handling code from Normal Code
- Grouping of Error Types: In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types..

Catch All

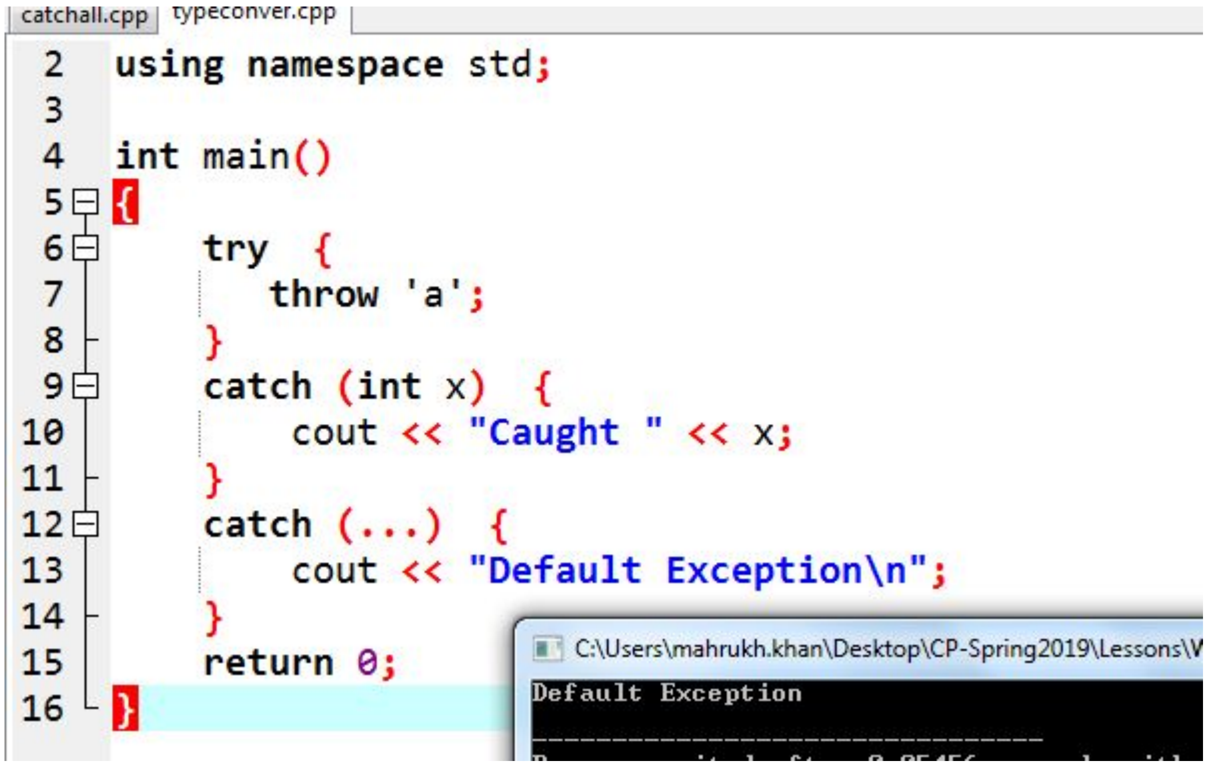
- There is a special catch block called ‘catch all’ catch(…) that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.



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

int main()
{
    try  {
        throw 10;
    }
    catch (char *excp)  {
        cout << "Caught " << excp;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
}
```

```
Default Exception

--------------------------------
Process exited after 0.02922 seconds with re
Press any key to continue . . .
```

Implicit type conversion not allowed



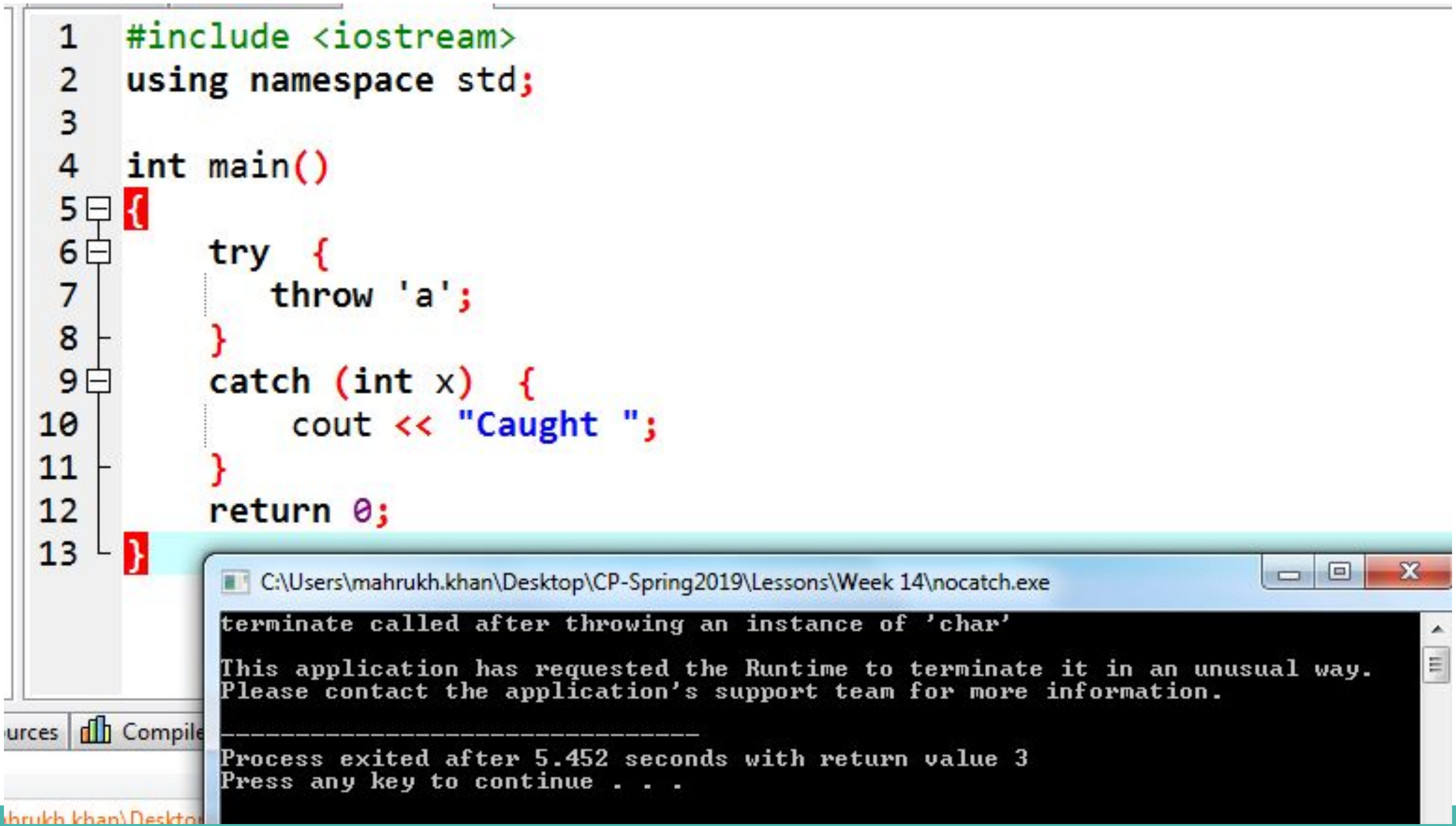
catchall.cpp | typeconver.cpp

```cpp
using namespace std;

int main()
{
    try  {
        throw 'a';
    }
    catch (int x)  {
        cout << "Caught " << x;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
    return 0;
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\V

```
Default Exception

-----------------------------------
```

Exception not caught anywhere



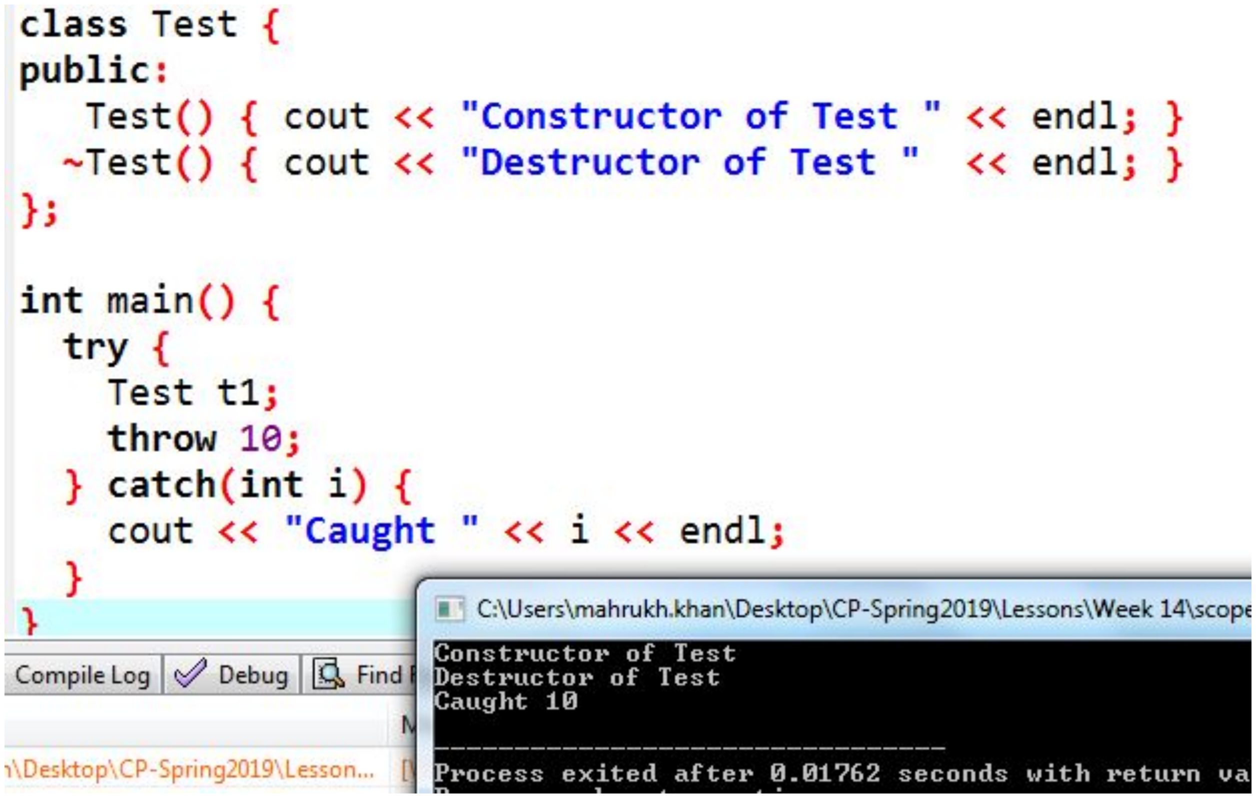
```
#include <iostream>
using namespace std;

int main()
{
    try  {
        throw 'a';
    }
    catch (int x)  {
        cout << "Caught ";
    }
    return 0;
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\Week 14\nocatch.exe

```
terminate called after throwing an instance of 'char'

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

--------------------------------
Process exited after 5.452 seconds with return value 3
Press any key to continue . . .
```

Concepts

- If an exception is thrown and not caught anywhere, the program terminates abnormally. We can change this abnormal termination behavior by writing our own unexpected function.
- A derived class exception should be caught before a base class exception. In Java, catching a base class exception before derived is not allowed by the compiler itself. In C++, compiler might give warning about it, but compiles the code.
- When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block.



```
class Test {
public:
   Test() { cout << "Constructor of Test " << endl; }
  ~Test() { cout << "Destructor of Test "  << endl; }
};

int main() {
  try {
    Test t1;
    throw 10;
  } catch(int i) {
    cout << "Caught " << i << endl;
  }
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\Week 14\scope

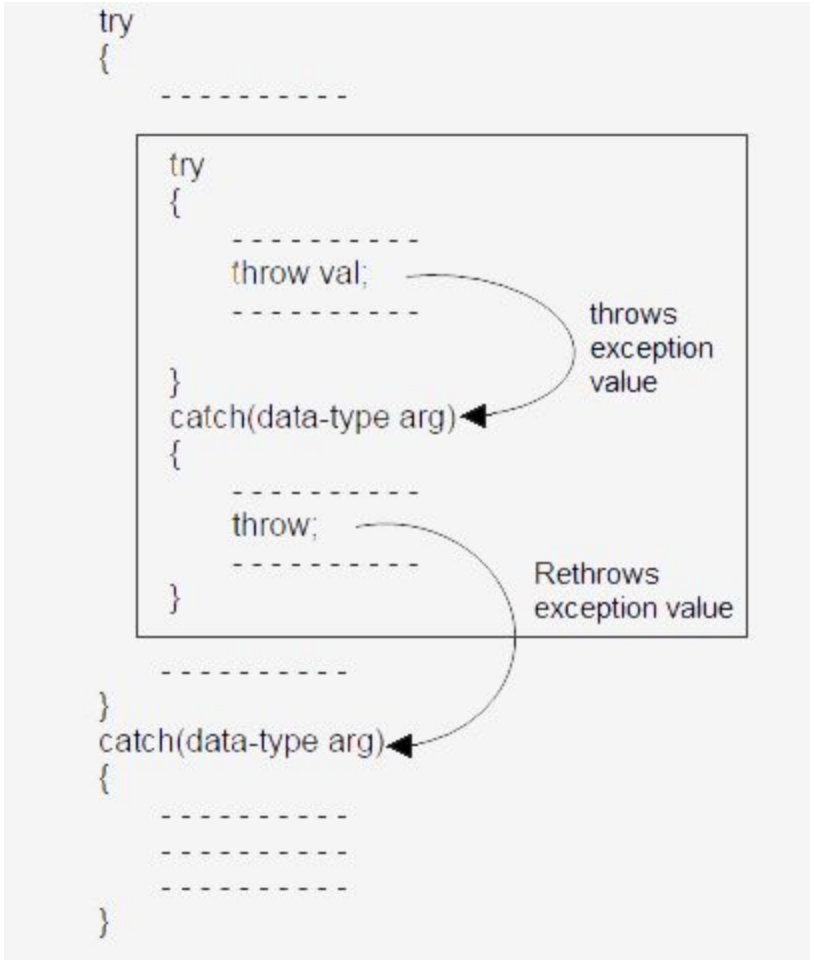
```
Constructor of Test
Destructor of Test
Caught 10

--------------------------------
Process exited after 0.01762 seconds with return va
```

Multiple Catch Blocks

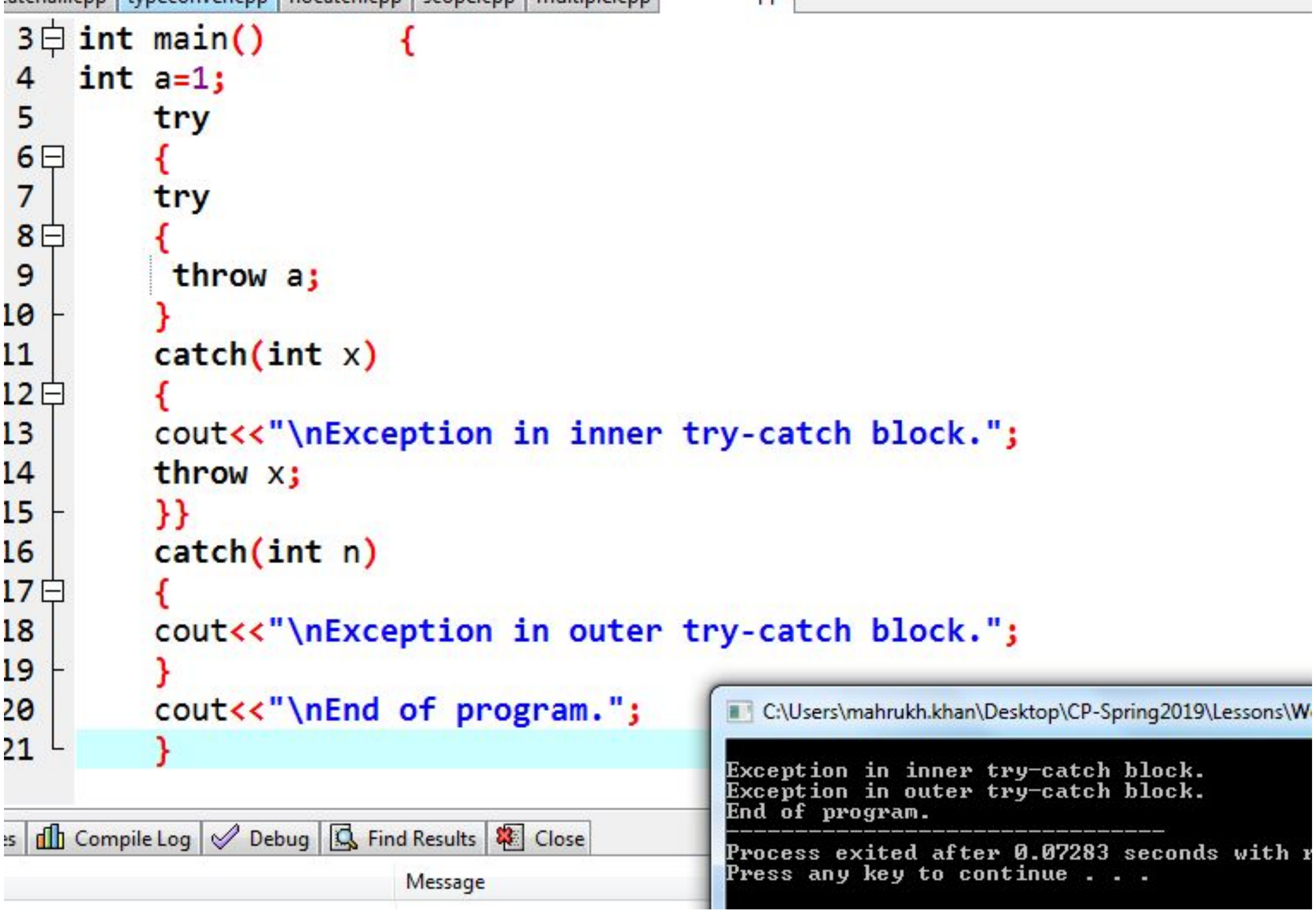
A **single try statement** can have multiple catch statements. Execution of particular catch block depends on the type of exception thrown by the throw keyword. If throw keyword send exception of integer type, catch block with integer parameter will get execute.

Rethrowing Exceptions

Rethrowing exception is possible, where we have an inner and outer try-catch statements (Nested try-catch). An exception to be thrown from inner catch block to outer catch block is called rethrowing exception



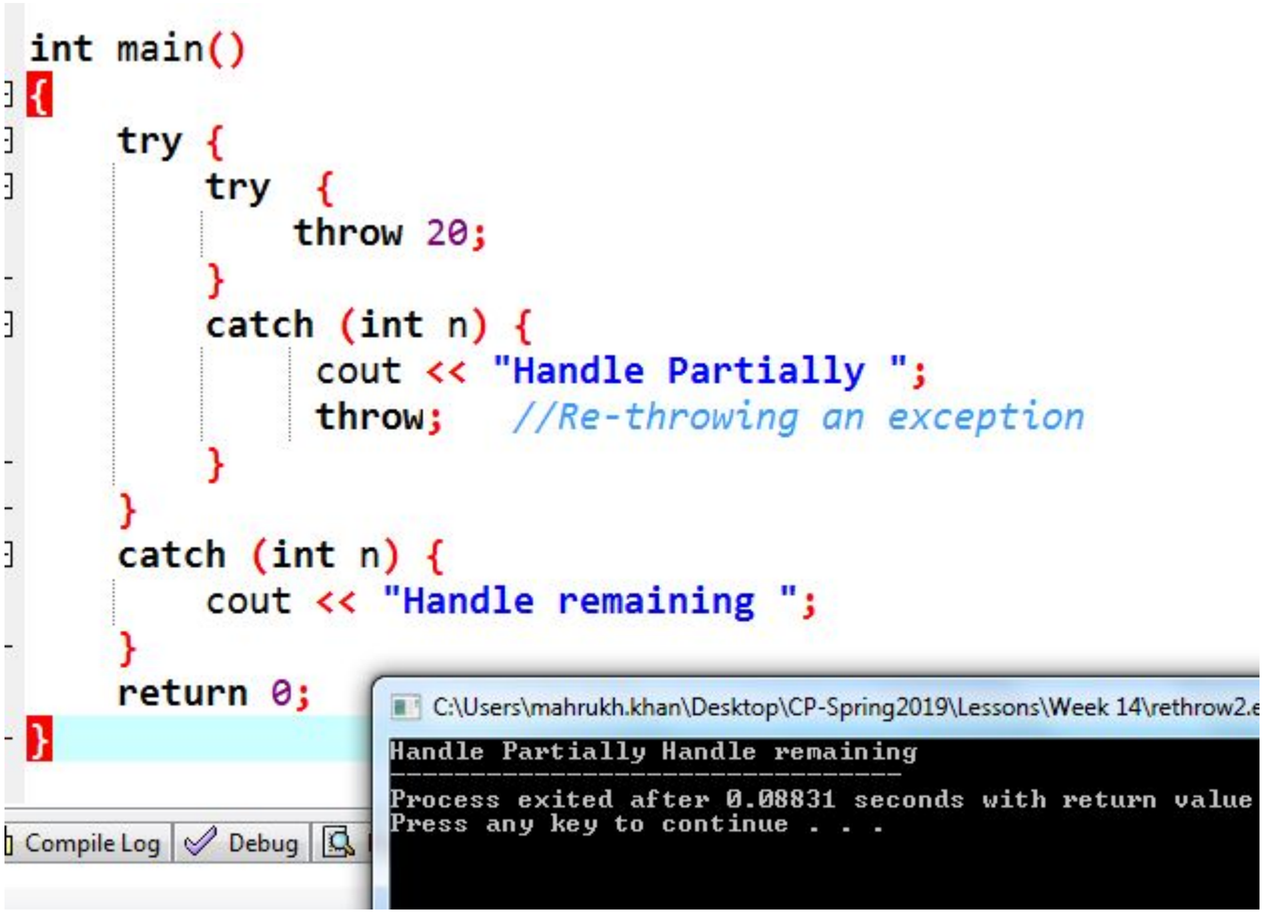
```
try
{
      ----------

   try
   {
         ----------
         throw val;
         ----------

   }
   catch(data-type arg)
   {
         ----------
         throw;
         ----------
   }

      ----------
}
catch(data-type arg)
{
      ----------
      ----------
      ----------
}
```



```
int main()      {
int a=1;
    try
    {
    try
    {
     throw a;
    }
    catch(int x)
    {
    cout<<"\nException in inner try-catch block.";
    throw x;
    }}
    catch(int n)
    {
    cout<<"\nException in outer try-catch block.";
    }
    cout<<"\nEnd of program.";
    }
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\W

```
Exception in inner try-catch block.
Exception in outer try-catch block.
End of program.
--------------------------------
Process exited after 0.07283 seconds with r
Press any key to continue . . .
```



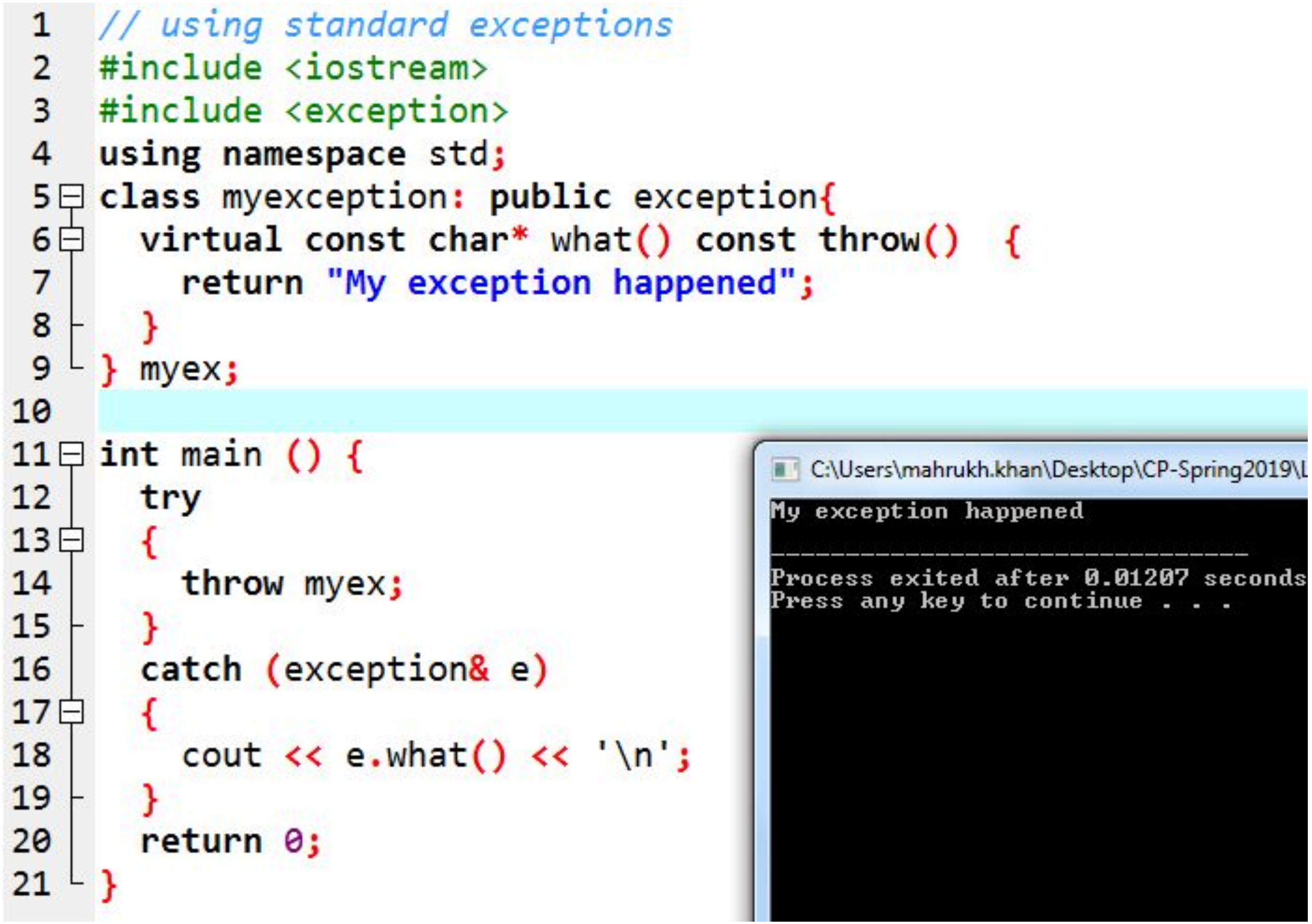
```cpp
int main()
{
    try {
        try  {
            throw 20;
        }
        catch (int n) {
            cout << "Handle Partially ";
            throw;   //Re-throwing an exception
        }
    }
    catch (int n) {
        cout << "Handle remaining ";
    }
    return 0;
}
```

```
C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\Week 14\rethrow2.e
```

```
Handle Partially Handle remaining
--------------------------------
Process exited after 0.08831 seconds with return value
Press any key to continue . . .
```

C++ Standard Exceptions

- C++ library has a standard exception class which is base class for all standard exceptions. All objects thrown by components of the standard library are derived from this class. Therefore, all standard exceptions can be caught by catching this type
- C++ Standard library provides a base class specifically designed to declare objects to be thrown as exceptions. It is called std::exception and is defined in the <exception> header. This class has a virtual member function called what that returns a null-terminated character sequence (of type char *) and that can be overwritten in derived classes to contain some sort of description of the exception.



```cpp
// using standard exceptions
#include <iostream>
#include <exception>
using namespace std;
class myexception: public exception{
  virtual const char* what() const throw()  {
    return "My exception happened";
  }
} myex;

int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << '\n';
  }
  return 0;
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\L

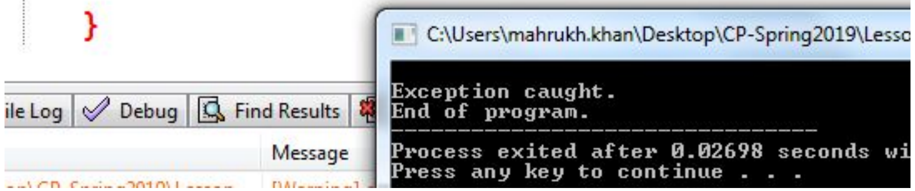
```
My exception happened

--------------------------------

Process exited after 0.01207 seconds
Press any key to continue . . .
```

Restricting Exceptions

- Older code may contain dynamic exception specifications. They are now deprecated in C++, but still supported. A dynamic exception specification follows the declaration of a function, appending a throw specifier to it.
- We can restrict the type of exception to be thrown, from a function to its calling statement, by adding throw keyword to a function definition.

```
void Demo() throw(int)
    {
        int a=2;
        if(a==1)
        throw a;
        else if(a==2)
       throw 1;
        else if(a==3)
        throw 4.5;

    }
```

```
int main()
{

        try {
            Demo();
        }
        catch(int n){
            cout<<"\nException caught.";
        }
      cout<<"\nEnd of program.";

}
```



```
C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lesso

Exception caught.
End of program.
--------------------------------
Process exited after 0.02698 seconds wi
Press any key to continue . . .
```