Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

- The process of converting system error messages into user friendly error message is known as **Exception handling**
- This is one of the powerful feature of C++ to handle runtime error and maintain normal flow of c++ application
- An exception is a problem that arises during the execution of a program
- A C++ exception is a response to an exceptional circumstance that arises while a program is running
- It is an event which occurs during the execution of a program, that disrupts the normal flow of programs instructions
- Ex. attempt to divide by zero.

Mechanism of Error Handling

- 1 Find the problem (hit the exception)
- 2 Inform that an error has occurred (throw the exception)
- 3 Receive an error information (catch the exception)
- 4 Take corrective actions (handle the exception)

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

C++ exception handling is built upon three keywords: try, catch, and throw.

- throw − A program throws an exception when a problem shows up. This is done using a throw keyword.
- catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch as follows −

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

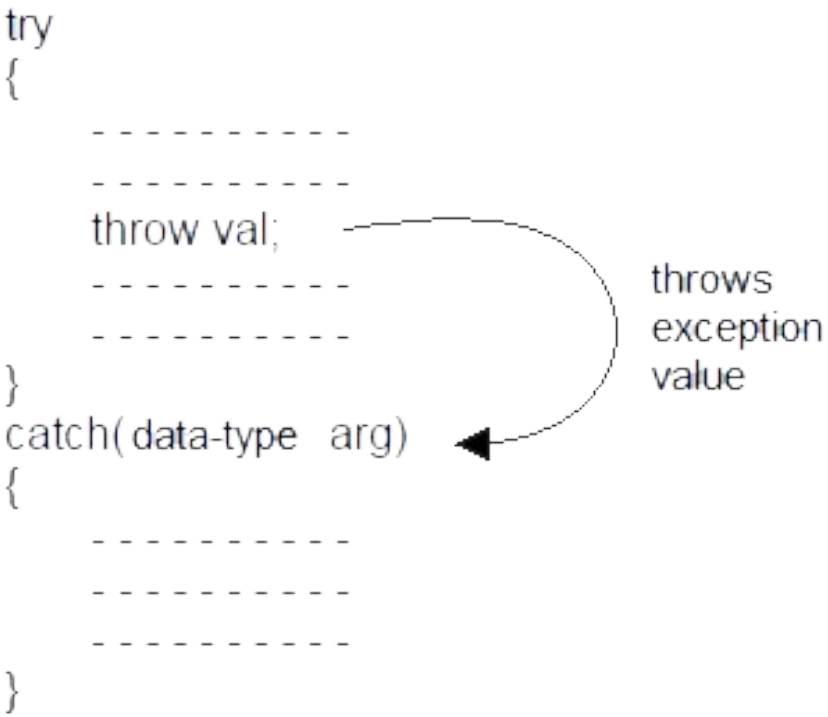
```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
 }
```

You can list down **multiple catch statements** to catch different type of exceptions in case your try block raises more than one exception in different situations.

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates



Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Multiple Catch Statement

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.

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Multiple Catch Statement

```
#include<iostream.h>
    #include<conio.h>
    void main()
    {
       int a=2;

        try
        {

           if(a==1)
              throw a;              //throwing integer
exception

           else if(a==2)
              throw 'A';            //throwing character
exception

           else if(a==3)
              throw 4.5;            //throwing float exception

        }
```

```
catch(int a)
        {
           cout<<"\nInteger exception caught.";
        }
        catch(char ch)
        {
           cout<<"\nCharacter exception caught.";
        }
        catch(double d)
        {
           cout<<"\nDouble exception caught.";
        }

        cout<<"\nEnd of program.";

    }
```

Output :

```
        Character exception caught.
          End of program.
```

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Catch All Exception

The above example will caught only three types of exceptions that are integer, character and double. If an exception occur of long type, no catch block will get execute and abnormal program termination will occur. To avoid this, We can use the catch statement with three dots as parameter (...) so that it can handle all types of exceptions.

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Multiple Catch Statement

```
#include<iostream.h>
      #include<conio.h>
      void main()
      {
          int a=1;

            try
            {

                if(a==1)
                    throw a;          //throwing integer
exception

                else if(a==2)
                    throw 'A';        //throwing character
exception

                else if(a==3)
                    throw 4.5;              //throwing float
exception

            }
            catch(...)
            {
               cout<<"\nException occur.";
            }

            cout<<"\nEnd of program.";

      }
```

Output :

```
            Exception occur.
              End of program.
```

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Rethrowing Exceptions

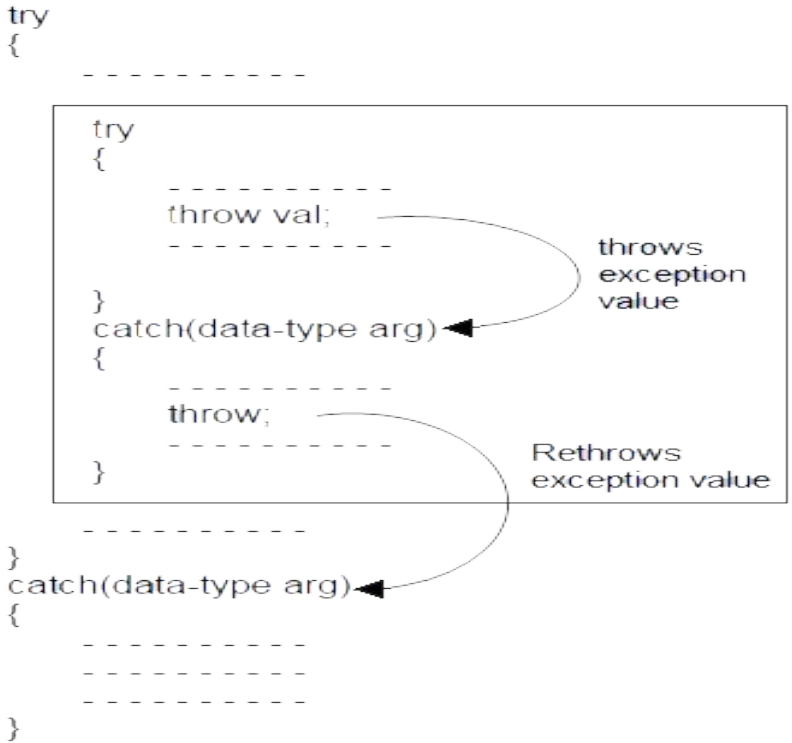
Rethrowing exception is possible, where we have an inner and outer try-catch statements (Nested try-catch). An exception to be thrown from inner catch block to outer catch block is called rethrowing exception.

Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

Rethrowing Exceptions



Subject :- Object Oriented Programming

Unit 5

Exception Handling and Templates

```
#include<iostream.h>
      #include<conio.h>
      void 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.";

      }
```

Output :

```
            Exception in inner try-catch block.
            Exception in outer try-catch block.
             End of program.
```

Subject :- Object Oriented Programming

Unit 5

User Defined Exception

We can use **Exception handling** with class too. Even we can throw an exception of **user defined** class types. For throwing an exception of say **demo** class type within **try** block we may write ; `throw demo();`

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

class demo {
};

int main()
{
	try {
		throw demo();
	}

	catch (demo d) {
		cout << "Caught exception of demo class \n";
	} }
```

Subject :- Object Oriented Programming

Unit 5

Exception and Inheritance

Exception handling can also be implemented with the help of inheritance. In case of inheritance object thrown by derived class is caught by the first catch block.

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

class demo1 {
};

class demo2 : public demo1 {
};

int main()
{
    for (int i = 1; i <= 2; i++) {
        try {
            if (i == 1)
                throw demo1();

            else if (i == 2)
                throw demo2();
        }
        catch (demo1 d1) {
            cout << "Caught exception
of demo1 class \n";
        }
        catch (demo2 d2) {
            cout << "Caught exception
of demo2 class \n";
        }
    }
}
```

Output:-

```
Caught exception of demo1 class
 Caught exception of demo1 class
```

Subject :- Object Oriented Programming

Unit 5

Exception handling with constructor:

Exception handling can also be implemented by using constructor. Though we cannot return any value from the constructor but with the help of **try** and **catch** block we can.

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

class demo {
        int num;

public:
        demo(int x)
        {
                try {

                        if (x == 0)
                                // catch block would be called
                                throw "Zero not allowed ";

                        num = x;
                        show();
                }

                catch (const char* exp) {
                        cout << "Exception caught \n ";
                        cout << exp << endl;
                }
        }
```

```
void show()
        {
                cout << "Num = " << num << endl;
        }
};

int main()
{

        // constructor will be called
        demo(0);
        cout << "Again creating object \n";
        demo(1);
}
```

Subject :- Object Oriented Programming

Unit 5

destructor and exception handling

Destructors in C++ basically called when objects will get destroyed and release memory from the system. When an exception is thrown in the class, the destructor is called automatically before the catch block gets executed.

```
#include <iostream>
using namespace std;
class Sample1 {
   public:
      Sample1() {
         cout << "Construct an Object
of sample1" << endl;
      }
      ~Sample1() {
         cout << "Destruct an Object
of sample1" << endl;
      }

};
```

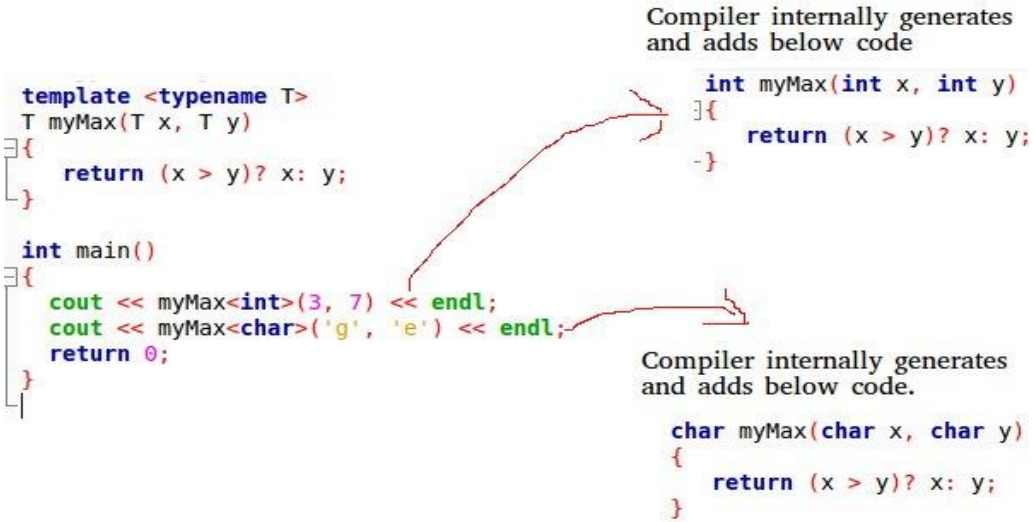
```
class Sample2 {
   public:
      Sample2() {
         int i =7;
         cout << "Construct an Object of
sample2" << endl;
         throw i;
      }
      ~Sample2() {
         cout << "Destruct an Object of sample2"
<< endl;
      }
};
int main() {
   try {
      Sample1 s1;
      Sample2 s2;
   } catch(int i) {
      cout << "Caught " << i << endl;
   }

}
```

Subject :- Object Oriented Programming

Unit 5

Templates

A template is a simple and yet very powerful tool in C++.
The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.
For example, a software company may need sort() for different data types.
Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.



Subject :- Object Oriented Programming

Unit 5

Template Function

Function Templates We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray().

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

// One function works for all data types.
This would work
// even for user defined types if operator
'>' is overloaded

template <typename T>
T myMax(T x, T y)
{
return (x > y)? x: y;
}
```

```
int main()
{
cout << myMax<int>(3, 7) <<
endl; // Call myMax for int
cout << myMax<double>(3.0, 7.0)
<< endl; // call myMax for
double
cout << myMax<char>('g', 'e')
<< endl; // call myMax for char

return 0;
}

}
```

Subject :- Object Oriented Programming

Unit 5

Template Function

Like function templates, you can also create class templates for generic class operations.

Sometimes, you need a class implementation that is same for all classes, only the data types used are different.

Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class.

This will unnecessarily bloat your code base and will be hard to maintain, as a change is one class/function should be performed on all classes/functions.

However, class templates make it easy to reuse the same code for all data types.

Subject :- Object Oriented Programming

Unit 5

Template Function

How to declare a class template?

```
template <class T>
class className
{
   ... .. ...
public:
   T var;
   T someOperation(T arg);
   ... .. ...
    };
```

In the above declaration, `T` is the template argument which is a placeholder for the data type used.

Inside the class body, a member variable `var` and a member function `someOperation()` are both of type `T`.

Subject :- Object Oriented Programming

Unit 5

Template Function

How to create a class template object?

To create a class template object, you need to define the data type inside a `< >` when creation.

```
className<dataType> classObject;
```

For example:

```
className<int> classObject;
className<float> classObject;
className<string> classObject;
```

Subject :- Object Oriented Programming

Unit 5

Template Function

You may overload a function template either by a non-template function or by another function template.

Overloading Function template

```
#include <iostream>
using namespace std;
template<class T>
void f(T x, T y)
{
   cout << "Template" << endl;
}
void f(int w, int z)
{
   cout << "Non-template" << endl;

}
```

```
int main() {
   f( 1 ,  2 );
   f('a', 'b');
  return 0;
}
```

Subject :- Object Oriented Programming

Unit 5

Template Function

The type name and export keywords.

- In a template declaratio ss to declare type template parameters and template parameters
- Inside a declaration or a definition of a template, typename can be used to declare that a dependent qualified name is a type.
- Inside a declaration or a definition of a template, (until C++11) typename can be used before a non-dependent qualified type name. It has no effect in this case.

```
template <class T> class Demonstration
{
 public: void method()
 {
T::A *aObj;
};
```

`typename T::A* a6;` it instructs the compiler to treat the subsequent statement as a declaration.

Exception Handling in C++

- An exception is an unexpected problem that arises during the execution of a program.
- Exception handling mechanism provide a way to transfer control from one part of a program to another. This makes it easy to separate the error handling code from the code written to handle the actual functionality of the program.
- C++ exception handling is built upon three keywords: **try**, **catch**, & **throw**.

Exception Handling in C++

- **try** : A block of code which may cause an exception is typically placed inside the try block. It's followed by one or more catch blocks. If an exception occurs, it is thrown from the try block.
- **catch** : this block catches the exception thrown from the try block. Code to handle the exception is written inside this catch block.
- **throw** : A program throws an exception when a problem shows up. This is done using a throw keyword.
- Every try catch should have a corresponding catch block. A single try block