Exception Handling

An **exception** is a problem(error) 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, such as an attempt to divide by zero. Errors disrupt normal execution of a program so we need exception handling to handle these errors.

Exception Handling in C++ is a process to handle runtime errors. We perform exception handling so the normal flow of the application can be maintained even after runtime errors.

Advantage

It maintains the normal flow of the application. In such case, rest of the code is executed even after exception.

Need of Exception Handling:

Here, are the reasons for using Exception Handling in C++:

1) Separation of Error Handling code from Normal Code: In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try catch blocks, the code for error handling becomes separate from the normal flow. **2) Functions/Methods can handle any exceptions they choose:** A function can throw many exceptions, but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by caller of the caller.

In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)

3) 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.

C++ provides three keywords to support exception handling. Try: The try block contain statements which may generate exceptions. **Throw:** When an exception occurs 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. The catch block catching exceptions must immediately follow the try block that throws an exception. A single try statement can have multiple catch statements.

The general form of try-catch block in C++:

```
try
 {
```

```
//code
throw parameter;
}
catch(datatype arg)
 {
//code to handle exception
}
```

Where data-type specifies the type of exception that catch block will handle if the try block throws an exception then program control leaves the block and enters into the catch statement of the catch block. If the type of object thrown matches the argument type in the catch statement, then the catch block is executed for handling the exception.

Example of simple try/catch:

Divided-by-zero is a common form of exception generally occurred in arithmetic based programs

```
#include<iostream.h>
main()
 {
int n1,n2,result;
cout<<"\nEnter 1st number:";
cin>>n1;
cout<<"\nEnter 2nd number:";
cin>>n2;
try
 {
if(n2==0)
throw n2; //Statement1 else
 {
result = n1 / n2;
cout<<"\nThe result is : "<<result; }
}
catch(int x)
 {
cout<<"\nCan't divide by : "<<x; }
cout<<"\nEnd of program."; }
```

Output :

Enter 1st number : 45

Enter 2nd number : 0

Can't divide by :

0

End of program

Multiple catch statements

Catch block will receive value send by throw keyword in try block. 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.

Example:

```
#include<iostream.h>
main()
 {
int a=2;
try
 {
if(a==1)
throw a; //throwing integerexception else if(a==2)
throw 'A'; //throwing characterexception else if(a==3)
throw 4.5; //throwing floatexception
}
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

Catch All Exceptions

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.

Example to catch all exceptions:

```
#include<iostream.h>
void main()
 {
int a=1;
try
 {
if(a==1)
throw a; //throwing integerexception
else if(a==2)
throw 'A'; //throwing characterexception
else if(a==3)
throw 4.5; //throwing floatexception
}
catch(...)
 {
cout<<"\nException occur.";
}
cout<<"\nEnd of program.";
}
```

Output :

Exception occur

End of program

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.

Syntax of rethrowing exceptions

Example of rethrowing exceptions:

```
#include<iostream.h>
```

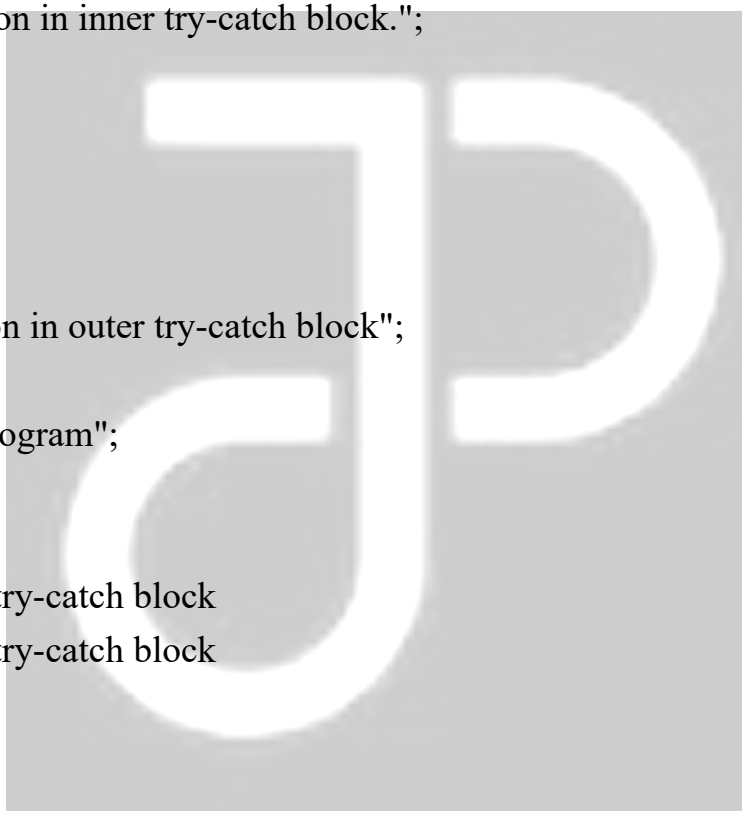
```
void main()
 {
int a=1;
try
 {
try
 {
throw a;
}
catch(int x)
   {
  cout<<"\nException in inner try-catch block.";
  throw x;
  }
  }
 catch(int n)
 {
 cout<<"\nException in outer try-catch block";
 }
cout<<"\nEnd of program";
 }
```

Output:

Exception in inner try-catch block

Exception in outer try-catch block

End of program



Custom Exception

Custom Exception with class in C++:- 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();

Example 1: Program to implement exception handling with single class

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

```
class demo {
};

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

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

C++ User-Defined Exceptions

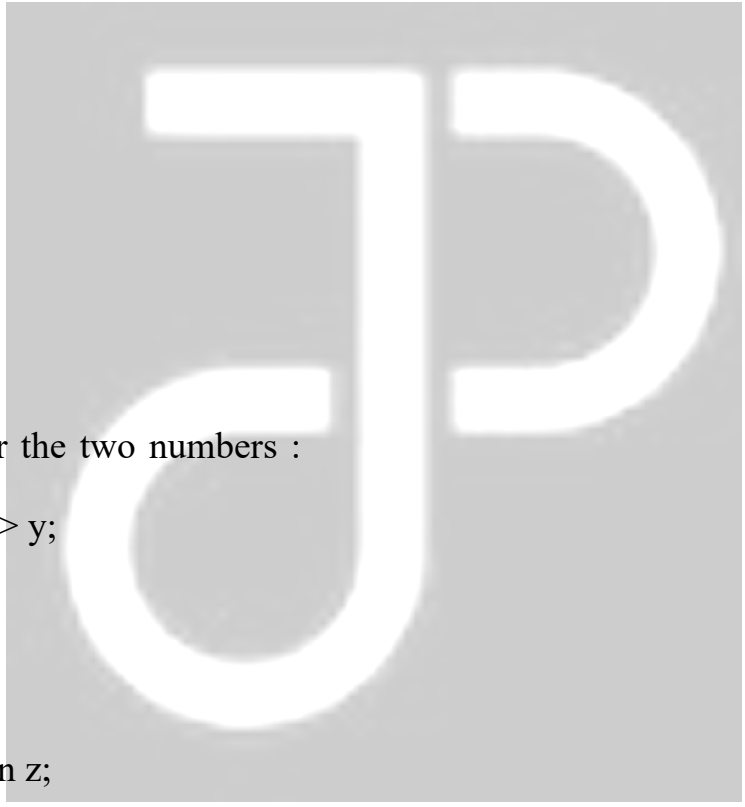
The new exception can be defined by overriding and inheriting **exception** class functionality.

C++ user-defined exception example

Let's see the simple example of a user-defined exception in which the std**::exception** class is used to define the exception.

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

class MyException : public

  exception
```

```
{

public:

    const char * what() const throw()

    {

      return "Attempted to divide by

    zero!\n"; }

};

int main()

{

  try

   {

    int x, y;

    cout << "Enter the two numbers :

    \n"; cin >> x >> y;

    if (y == 0)

    {

      MyException z;

      throw z;

    }

    else

    {

      cout << "x / y = " << x/y << endl;

    }

  }
```



```
    catch(exception& e)
    {
      cout << e.what();
    }
}
```

OUTPUT:-

```
Enter the two numbers :
10
0
Attempted to divide by zero!
```