Exception is any abnormal behavior that interrupt(stop) the normal flow of control.

Exception are off beat situation in your program where your program should be ready to handle it with appropriate response.

C++ provides a built- in error handling mechanism that is called exception handling.

Using exception handling you can more easily manage and respond to runtime errors.

Program statement that you want to monitor for exceptions are contains in a try block.

If any exception occurs within the try block, it is thrown (using throw keyword).

The exception is caught, using catch, and process.

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.

#include <iostream>

using namespace std;

int main()

{

cout << "Hello world!" << endl;

try{

throw 10;

cout<<"try";

}catch(int e){

cout<<"exception"<<e;

}

cout<<"last line";

return 0;

}

#include<iostream>

using namespace std;

int main(){

try{

throw "exception accrue";

}catch(char const \*x){

cout<<x;

}

cout<<"welcome";

return 0;

}

User define exception:

#include<iostream>

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();

}

return 0;

}

**what()** is a public method provided by exception class and it has been overridden by all the child exception classes. This returns the cause of an exception.

A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope.

Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*. This allows organizing the elements of programs into different logical scopes referred to by names.

* Namespace is a feature added in C++ and not present in C.
* A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it.
* Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

namespace namespace\_name {

// code declarations

}

* Namespace declarations appear only at global scope.
* Namespace declarations can be nested within another namespace.
* Namespace declarations don’t have access specifiers. (Public or private)
* No need to give semicolon after the closing brace of definition of namespace.
* We can split the definition of namespace over several units.