C++ Exception Handling

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.

In C++, exception is an event or object which is thrown at runtime. All exceptions are derived from std::exception class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

Advantage

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

C++ Exception Handling Keywords

In C++, we use 3 keywords to perform exception handling:

- try
- catch, and
- throw
- 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.
- 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.
- throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

Why do we need Exception Handling in C++?

The following are the main advantages of exception handling over traditional error handling:

Separation of Error Handling Code from Normal Code: There are always if-else conditions to handle errors in traditional error handling codes. 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.

Functions/Methods can handle only the 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 the caller. If the caller chooses not to catch them, then the exceptions are handled by the 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).

Grouping of Error Types: In C++, both basic types and objects can be thrown as exceptions. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, and categorize them according to their types.

C++ example without try/catch

```
#include <iostream>
using namespace std;
float division(int x, int y)
{
  return (x/y);
}
int main ()
{
```

```
  int i = 50;
  int j = 0;
  float k = 0;
    k = division(i, j);
    cout << k << endl;
  return 0;
}
```

C++ try/catch example

```
#include <iostream>
using namespace std;
float division(int x, int y)
{
  if( y == 0 )
 {
    throw "Attempted to divide by zero!";
  }
  return (x/y);
}
int main ()
{
  int i = 25;
  int j = 0;
  float k = 0;
  try {
    k = division(i, j);
```

```
    cout << k << endl;
  }
catch (const char* e)
{
    cerr << e << endl;
  }
  return 0;
}
```

// C++ program to demonstate the use of try,catch and throw

// in exception handling.

```
#include <iostream>
using namespace std;
int main()
{
        try
{
                int numerator = 10;
                int denominator = 0;
                int res;
if (denominator == 0)
{
   throw runtime_error("Division by zero not allowed!");
}
```

```
            res = numerator / denominator;
            cout << "Result after division: " << res << endl;
        }
          catch (const exception& e)
         {
          cerr << "Exception " << e.what() << endl;
        }
        }
```

```
#include<iostream>
using namespace std;
int main()
{
        int x=10;
        try
        {
            if(x>=20)
            {
                    cout<<"try excuted"<<endl;
            }

        else
        {
                    throw(x);
```

```
            }
}
            catch( int b)
            {
             cout<<"error occured"<<endl;
             cout<<"x is "<<x<<endl;
            }
}
```

```
#include <iostream>
using namespace std;
int main() {
 try {
   int age = 20;
   if (age >= 18) {
     cout << "Access granted - you are old enough.";
   } else {
     throw (age);
   }
 }
 catch (int myNum) {
   cout << "Access denied - You must be at least 18 years old.\n";
   cout << "Age is: " << myNum;
 }

}
```