Chapter 8 : Exception Handling in C++

Basis of Exception Handling

One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution.

There are two types of exceptions :

1. Synchronous

2. Asynchronous

(Ex:which are beyond the program's control, Disc failure etc).

Following main advantages of exception handling over traditional error handling :

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)

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++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (throw an error).

try and catch

- Exception handling in C++ consists of three keywords: try, throw and catch:
- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The throw keyword throws an exception when a problem is detected, which lets us create a custom error.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs :

- We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

- In the catch block, we catch the error and do something about it. The catch statement takes a parameter: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.
- If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped

Following is a simple example to show exception handling in C++.

The output of program explains flow of execution of try/catch blocks :

```
#include <iostream>

using namespace std;

int main()

{  int x = -1;

   // Some code

   cout << "Before try \n";

   try {

      cout << "Inside try \n";
```

```
 if (x < 0)  {
        throw x;
        cout << "After throw (Never executed) \n";      }
}
   catch (int x ) {
      cout << "Exception Caught \n";  }
   cout << "After catch (Will be executed) \n";
   return 0;
}
```

Output :

Before try

Inside try

Exception Caught

After catch (Will be executed)

Throwing, Catching Mechanism :

throw– when a program encounters a problem, it throws an exception. The throw keyword helps the program perform the throw.

catch– a program uses an exception handler to catch an exception. It is added to the section of a program where you need to handle the problem. It’s done using the catch keyword.

Throwing Exceptions

Exceptions can be thrown anywhere within a code block using throw statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Following is an example of throwing an exception when dividing by zero condition occurs :

```
double division(int a, int b) {
    if( b == 0 )
    {
    Throw
```

```
return (a/b);
}
throw "Division by zero condition!";
}
return (a/b) ;
}
```

Catching Exceptions

The catch block following the try block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try {
  // protected code
} catch( ExceptionName e ) {
  // code to handle ExceptionName exception
}
```

The following is an example, which throws a division by zero exception and we catch it in catch block :

```
#include <iostream>

using namespace std;

double division(int a, int b) {
   if( b == 0 ) {
      throw "Division by zero condition!";
   }
   return (a/b);
}
```

```
int main () {   int x = 50;
   int y = 0;
   double z = 0;
   try {
      z = division(x, y);
      cout << z << endl;  }
catch (const char* msg) {
     cerr << msg << endl;    }
   return 0;
}
```