Exception Handling

Chapter 9

Contents

Introduction

- C++ provides a built in error handling mechanism that is called exception handling.
  - Using exception handling we can more easily manage and response synchronous runtime errors, which occur when a statement execute.
  - Common examples of these errors are:
    - Out-of-range-array-subscripts
    - arithmetic overflow
    - division by zero
    - invalid functionparameters and
    - unsuccessful memory allocation (due to lack of memory).
- In C++, exception handling is built upon 3 new keywords:
  - try
  - catch
  - throw

Basics of Exception Handling

- Program statement that we want to monitor for exception are contained in try block.
- If an exception (i.e. an error) occurs within in the try block, it is thrown (using throw).
- The exception is caught, using catch and process.
- Any statement that throws an exception must have been executed from within a try block.
- Any exception must be caught by a catch statement, which immediately follows the try statement that throws the exception.

The general form of try and catch are shown as below:

```
try
{
        .........
}
catch (type 1 arg)
{
        .........
}
catch (type 2 arg)
{
        .........
}
catch (type n arg)
{
        .........
}
```

Basics of Exception Handling [contd..]

- The try block must contain the portion of our program that we want to monitor for errors
- when an exception is thrown; it is cut by its corresponding catch statement, which process the exception.
- There can be more than one catch statement associated with try where the catch statement that is used is determined by the type of the exception.
  - That is if the data type specified by a catch matches that of the exception, that catch statement is executed (all others are bypassed)
  - When an exception is caught, argument will receive its value

Exception Handling Mechanism:

Sequence of events for exception handling:

- Code is executed normally outside a block.
- Control enters the try block.
- A statement in the try block causes an error in a member function.
- The member function throws an exception.
- Control transfers to the exception handler (i.e. catch block).
- Catch block takes corrective action to handle the exception.

Discussion on Exception Handling 1

- Any type of data can be caught (and thrown), including classes that will create.
- Exception is a value thrown.
- If we throw an exception for which there is no applicable statements, an abnormal program termination might occur, throwing an unhandled exception causes the standard library function terminate ( ) to invoke. By default, terminate function calls abort ( ) function to stop our program.
- Once an exception has been thrown control passes to the catch block and the try block is terminated. After catch block executes program control continues with the statements following catch.
- The type of exception must match the type specified in catch statement, if the type does not match the exception will not be caught and abnormal termination will occur.

Discussion on Exception Handling 2

- Although we can have more than one catch associated with a try. Each catch must catch different type of exception.
- An exception handler can catch all exception instead of just of certain type by using following form of catch.

```
catch ( )
{
………
}
```

- We can restrict the type of exceptions that a function can throwback to its caller by using following form

```
return_type function_name (argument list) throw (type list)
{
        ………….
}

int myfunc (int i) throw (int i) throw (int a, float y,double d)
```

- Only these data types obtained in comm. Separated type list. Throwing any other type of exception will occur abnormal program termination by calling standard library function unexpected ( ) which causes the terminate function to be called.
- Following statement prevents X handler to throw an exception

```
void Xhandler (int test) throw ( )
{
try
{
                ……….
}
catch (int b)
{
                ………
}
```

Throwing and Catching Exception

Throwing an Exception:

- When an exception that is desired to be handled is detected. It is thrown using one of the following throw statements
  - throw (exception)
  - throw exception
  - throw; //used for re-throwing exception without using argument.

Catching an Exception:

- Catch block include code for handling exception. When an exception whose type matches with the type of catch statement the exception is caught and the code in the catch is executed.

```
catch (type arg)
{
        //statements for managing exception
}
```

Re- Throwing an Exception:

- A handler may decide not to process an exception caught by it. In such cases we can re-throw an exception. The most likely reason for doing so is to allow multiple handler access to the exception.
  - e.g. perhaps one exception handler manages one aspect of an exception and a second handler copes with another exception.
  - An exception can only be re-thrown from within a catch block (or from any function call from within that block). When we re-throw an exception, it will not be re-caught by the same catch block (statement). It will propagate to an outer (next) catch statement (block).

Sample Program 1

```
//sample program for exception handling
 #include<iostream>
using namespace std;
int main ( )
{
   cout<<"start\n";
   try //start a try block
   {
      cout<<"Inside try block \n";
      throw 10; //throw an error
      cout<<"This will not execute";
   }
   catch (int i) //catch an error
   {
      cout<<"caught one! Number is:";
      cout<<i<<"\n";
   }
   cout<<"end\n";
   return 0;
}
```

<u>Output</u>:

```
Start
Inside a try block
caught one! Number is:10
end
```

Sample Program 2

```
#include <iostream>
using namespace std;
int main()
{
    int StudentAge;
    try {
        cout << "Student Age: ";
        cin >> StudentAge;

        if(StudentAge < 0)
            throw "Positive Number Required";

        cout << "\nStudent Age: " << StudentAge << "\n\n";
    }
    catch(const char* Message)
    {
        cout << "Error: " << Message;
    }
    cout << "\n";
    return 0;
}
```

Sample Program 3

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

int main()
{
    double Operand1, Operand2, Result;

// Request two numbers from the user
cout << "This program allows you to perform
    a division of two numbers\n";
cout << "To proceed, enter two numbers: ";

try {
    cout << "First Number: ";
    cin >> Operand1;
    cout << "Second Number: ";
    cin >> Operand2;

// Find out if the denominator is 0
    if( Operand2 == 0 )
    throw "Division by zero not allowed";

// Perform a division and display the result
Result = Operand1 / Operand2;

cout << "\n" << Operand1 << " / " <<
Operand2 << " = " << Result << "\n\n";
}

catch(const char* Str) // Catch an exception
{
// Display a string message accordingly
cout << "\nBad Operator: " << Str;
}

            return 0;
}
```

Catching Multiple Exception: Program 4

```
#include<iostream>
#include <string>
using namespace std;
int main ()
{
    int num;
    string str_bad = "wrong number used";
    cout << "Input 1 or 2: ";
    cin >> num;
    try
    {
        if ( num == 1 )
        {
            throw 5;
        }
        if ( num == 2 )
        {
            throw 1.1f;
        }
        if ( num != 1 || num != 2 )
        {
            throw str_bad;
        }
    }
```

Catching Multiple Exception: Program 4

```
catch (int a)
   {
   cout << "An exception occurred!" << endl;
   cout << "Exception number is: " << a << endl;
   }
   catch (float b)
   {
   cout << "An exception occurred!" << endl;
   cout << "Exception number is: " << b << endl;
   }
   catch (...)
   {
   cout << "A default exception occurred!" << endl;
  cout << "Why? : " << str_bad << endl;
   }
   return 0;
}
```