Files and Streams, Exception Handling in C++

Md. Alamgir Hossain

Senior Lecturer, Dept. of CSE

Prime University

Mail: *alamgir.cse14.just@gmail.com*

Files and Streams

- In C++ programming we are using the **iostream** standard library, it provides **cin** and **cout** methods for reading from input and writing to output respectively.
- To read and write from a file we are using the standard C++ library called **fstream**. Let us see the data types define in fstream library is:

fstream	It is used to create files, write information to files, and read information from files.
ifstream	It is used to read information from files.
ofstream	It is used to create files and write information to the files.

File Stream Example: Writing to a File

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main ()
5  {
6      ofstream filestream("text.txt"); ///"text.txt" is the tex file name
7      if (filestream.is_open())
8      {
9          filestream << "Welcome to Prime University.\n";
10         filestream << "Depart of CSE\n";
11         filestream.close();
12     }
13     else cout <<"File opening is fail.";
14     return 0;
15 }
```

File Stream Example: Reading from a File

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  int main ()
5  {
6      string str;
7      ifstream filestream("text.txt");
8      if (filestream.is_open())
9      {
10         while ( getline (filestream,str) )
11         {
12             cout << str <<endl;
13         }
14         filestream.close();
15     }
16     else
17     {
18         cout << "File opening is fail."<<endl;
19     }
20     return 0;
21 }
22
```

C++ Read and Write Example

```cpp
#include <fstream>
#include <iostream>
using namespace std;
int main ()
{
    char input[75];
    ofstream os;
    os.open("mytext.txt");
    cout <<"Writing to a text file:" << endl;
    cout << "Please Enter your name: ";
    cin.getline(input, 100);
    os << input << endl;
    cout << "Please Enter your age: ";
    cin >> input;
    cin.ignore();
    os << input << endl;
    os.close();
    ifstream is;
    string line;
    is.open("mytext.txt");
    cout << "\nReading from a text file:" << endl;
    while (getline (is,line))
    {
        cout << line << endl;
    }
    is.close();
    return 0;
}
```

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: a) Synchronous, b) Asynchronous(Ex:which are beyond the program's control, Disc failure etc).
- The term synchronous exception means that exceptions can be originated only from throw expressions.
- C++ provides following specialized keywords for this purpose.
  - ✓ *try*: represents a block of code that can throw an exception.
  - ✓ *catch*: represents a block of code that is executed when a particular exception is thrown.
  - ✓ *throw*: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

Advantages of Exception Handling

- ***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.
- ***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.

Exception Handling in C++

- 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;
}
```

Exception Handling in C++

- There is a special catch block called ‘catch all’ catch(…) that can be used to catch all types of exceptions. For example, in the following program, an int is thrown as an exception, but there is no catch block for int, so catch(…) block will be executed.

```
#include <iostream>
using namespace std;
int main()
{
    try
    {
        throw 10;
    }
    catch (char *excp)
    {
        cout << "Caught " << excp;
    }
    catch (...)
    {
        cout << "Default Exception\n";
    }
    return 0;
}
```

Exception Handling in C++

- In C++, try-catch blocks can be nested. Also, an exception can be re-thrown using "throw; "

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

int main()
{
    try {
        try  {
            throw 10;
        }
        catch (int n) {
            cout << "Handle Partially "<<endl;
            throw;   ///Re-throwing an exception
        }
    }
    catch (int n) {
        cout << "Handle remaining "<<endl;
    }
    return 0;
}
```

Handling the Divided by Zero Exception in C++

- We use Exception Handling to overcome exceptions occurred in execution of a program in a systematic manner.
- Dividing a number by Zero is a mathematical error (not defined) and we can use exception handling to gracefully overcome such operations.
- If we write a code without using exception handling then the output of division by zero will be shown as infinity which cannot be further processed.

Handling the Divided by Zero Exception in C++

```
1  /// Program to show division without using Exception Handling
2  #include <iostream>
3  using namespace std;
4  /// Defining function Division
5  float Division(float num, float den)
6  {
7      /// return the result of division
8      return (num / den);
9  } /// end Division
10 int main()
11 {
12     /// storing 12.5 in numerator and 0 in denominator
13     float numerator = 12.5;
14     float denominator = 0;
15     float result;
16     /// calls Division function
17     result = Division(numerator, denominator);
18     /// display the value stored in result
19     cout << "The quotient of 12.5/0 is "<< result << endl;
20
21     return 0;
22 }
23 ///Output: The quotient of 12.5/0 is inf
```

Handling the Divided by Zero Exception in C++

```cpp
#include <iostream>
#include <stdexcept>
using namespace std;
float CheckDenominator(float den)/// defining CheckDenominator
{
    if (den == 0)
        throw "Error";
    else
        return den;
} // end CheckDenominator
int main()
{
    float numerator, denominator, result;
    numerator = 12.5;
    denominator = 0;
    try/// try block
    {
        /// calls the CheckDenominator function by passing a string "Error"
        if (CheckDenominator(denominator))
        {
            result = (numerator / denominator);
            cout << "The quotient is "<< result << endl;
        }
    }
    catch (...)/// catch block capable of catching any type of exception
    {
        /// Display a that exception has occurred
        cout << "Exception occurred" << endl;
    }
    return 0;
}
```

Thank You