Unit 5 - Files and Exception Handling

Content :-Files and Exception Handling: Streams and files, Namespaces, Exception handling.

Overall, studying files and exception handling along with namespaces in programming can lead to a deeper understanding of software development concepts and practices, ultimately resulting in more efficient and robust code implementation.

C++ 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:

Data Type	Description
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.

C++ FileStream example: writing to a file

Let's see the simple example of writing to a text file **testout.txt** using C++ FileStream programming.

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
4. int main () {
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
5.  ofstream filestream("testout.txt");
6.  if (filestream.is_open())
7.  {
8.    filestream << "Welcome to javaTpoint.\n";
9.    filestream << "C++ Tutorial.\n";
10.   filestream.close();
11. }
12. else cout <<"File opening is fail.";
13. return 0;
14. }
```

Output:

```
The content of a text file testout.txt is set with the data:
Welcome to javaTpoint.
C++ Tutorial.
```

C++ FileStream example: reading from a file

Let's see the simple example of reading from a text file **testout.txt** using C++ FileStream programming.

```
1.  #include <iostream>
2.  #include <fstream>
3.  using namespace std;
4.  int main () {
5.   string srg;
6.   ifstream filestream("testout.txt");
7.   if (filestream.is_open())
8.   {
9.    while ( getline (filestream,srg) )
10.   {
11.     cout << srg <<endl;
12.   }
13.   filestream.close();
14. }
15. else {
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
16.      cout << "File opening is fail."<<endl;
17.    }
18.  return 0;
19.}
```

Note: Before running the code a text file named as **"testout.txt"** is need to be created and the content of a text file is given below:
Welcome to javaTpoint.
C++ Tutorial.

Output:

```
Welcome to javaTpoint.
C++ Tutorial.
```

C++ Read and Write Example

Let's see the simple example of writing the data to a text file **testout.txt** and then reading the data from the file using C++ FileStream programming.

```
1.  #include <fstream>
2.  #include <iostream>
3.  using namespace std;
4.  int main () {
5.     char input[75];
6.     ofstream os;
7.     os.open("testout.txt");
8.     cout <<"Writing to a text file:" << endl;
9.     cout << "Please Enter your name: ";
10.    cin.getline(input, 100);
11.    os << input << endl;
12.    cout << "Please Enter your age: ";
13.    cin >> input;
14.    cin.ignore();
15.    os << input << endl;
16.    os.close();
17.    ifstream is;
18.    string line;
```

```
19. is.open("testout.txt");
20. cout << "Reading from a text file:" << endl;
21. while (getline (is,line))
22. {
23. cout << line << endl;
24. }
25. is.close();
26. return 0;
27. }
```

Output:

```
Writing to a text file:
 Please Enter your name: Nakul Jain
Please Enter your age: 22
 Reading from a text file:   Nakul Jain
 22
```

C++ getline()

The cin is an object which is used to take input from the user but does not allow to take the input in multiple lines. To accept the multiple lines, we use the getline() function. It is a pre-defined function defined in a **<string.h>** header file used to accept a line or a string from the input stream until the delimiting character is encountered.

Syntax of getline() function:

There are two ways of representing a function:

- The first way of declaring is to pass three parameters.

1. istream& getline(istream& is, string& str, char delim);

The above syntax contains three parameters, i.e., **is, str**, and **delim**.

Where,

is: It is an object of the istream class that defines from where to read the input stream.

str: It is a string object in which string is stored.

delim: It is the delimiting character.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 5 - Files and Exception Handling

Return value

This function returns the input stream object, which is passed as a parameter to the function.

- The second way of declaring is to pass two parameters.

1. istream& getline(istream& is, string& str);

The above syntax contains two parameters, i.e., **is** and **str**. This syntax is almost similar to the above syntax; the only difference is that it does not have any delimiting character.

Where,

is: It is an object of the istream class that defines from where to read the input stream.

str: It is a string object in which string is stored.

Return value

This function also returns the input stream, which is passed as a parameter to the function.

Let's understand through an example.

First, we will look at an example where we take the user input without using getline() function.

```
1. #include <iostream>
2. #include<string.h>
3. using namespace std;
4. int main()
5. {
6. string name; // variable declaration
7. std::cout << "Enter your name :" << std::endl;
8. cin>>name;
9. cout<<"\nHello "<<name;
10. return 0;
11. }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

In the above code, we take the user input by using the statement **cin>>name,** i.e., we have not used the **getline()** function.

Output

```
Enter your name :
John Miller
Hello John
```

In the above output, we gave the name 'John Miller' as user input, but only 'John' was displayed. Therefore, we conclude that cin does not consider the character when the space character is encountered.

Let's resolve the above problem by using getline() function.

```
1. #include <iostream>
2. #include<string.h>
3. using namespace std;
4. int main()
5. {
6. string name; // variable declaration.
7. std::cout << "Enter your name :" << std::endl;
8. getline(cin,name); // implementing a getline() function
9. cout<<"\nHello "<<name;
10. return 0;}
```

In the above code, we have used the **getline()** function to accept the character even when the space character is encountered.

Output

```
Enter your name :
John Miller
Hello John Miller
```

In the above output, we can observe that both the words, i.e., John and Miller, are displayed, which means that the getline() function considers the character after the space character also.

When we do not want to read the character after space then we use the following code:

```
1. #include <iostream>
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
2. #include<string.h>
3. using namespace std;
4. int main()
5. {
6. string profile; // variable declaration
7. std::cout << "Enter your profile :" << std::endl;
8. getline(cin,profile,' '); // implementing getline() function with a delimiting character.
9. cout<<"\nProfile is :"<<profile;
10. }
```

In the above code, we take the user input by using getline() function, but this time we also add the delimiting character('') in a third parameter. Here, delimiting character is a space character, means the character that appears after space will not be considered.

Output

```
Enter your profile :
Software Developer
Profile is: Software
```

Getline Character Array

We can also define the getline() function for character array, but its syntax is different from the previous one.

Syntax

```
1. istream& getline(char* , int size);
```

In the above syntax, there are two parameters; one is **char***, and the other is **size**.

Where,

char*: It is a character pointer that points to the array.

Size: It acts as a delimiter that defines the size of the array means input cannot cross this size.

Let's understand through an example.

```
1. #include <iostream>
```

```
2. #include<string.h>
3. using namespace std;
4. int main()
5. {
6. char fruits[50]; // array declaration
7. cout<< "Enter your favorite fruit: ";
8. cin.getline(fruits, 50); // implementing getline() function
9. std::cout << "\nYour favorite fruit is :"<<fruits << std::endl;
10. return 0;
11. }
```

Output

```
Enter your favorite fruit: Watermelon
Your favorite fruit is: Watermelon
```

C++ Namespaces

Namespaces in C++ are used to organize too many classes so that it can be easy to handle the application.

For accessing the class of a namespace, we need to use namespacename::classname. We can use **using** keyword so that we don't have to use complete name all the time.

In C++, global namespace is the root namespace. The global::std will always refer to the namespace "std" of C++ Framework.

C++ namespace Example

Let's see the simple example of namespace which include variable and functions.

```
1. #include <iostream>
2. using namespace std;
3. namespace First {
4.     void sayHello() {
5.         cout<<"Hello First Namespace"<<endl;
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
    }
}
namespace Second  {
        void sayHello() {
            cout<<"Hello Second Namespace"<<endl;
        }
}
int main()
{
 First::sayHello();
 Second::sayHello();
return 0;
}
```

Output:

```
Hello First Namespace
Hello Second Namespace
```

C++ namespace example: by using keyword

Let's see another example of namespace where we are using "using" keyword so that we don't have to use complete name for accessing a namespace program.

```
#include <iostream>
using namespace std;
namespace First{
    void sayHello(){
        cout << "Hello First Namespace" << endl;
    }
}
namespace Second{
    void sayHello(){
        cout << "Hello Second Namespace" << endl;
    }
}
using namespace First;
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
14. int main () {
15.    sayHello();
16.    return 0;
17. }
```

Output:

```
Hello First Namespace
```

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

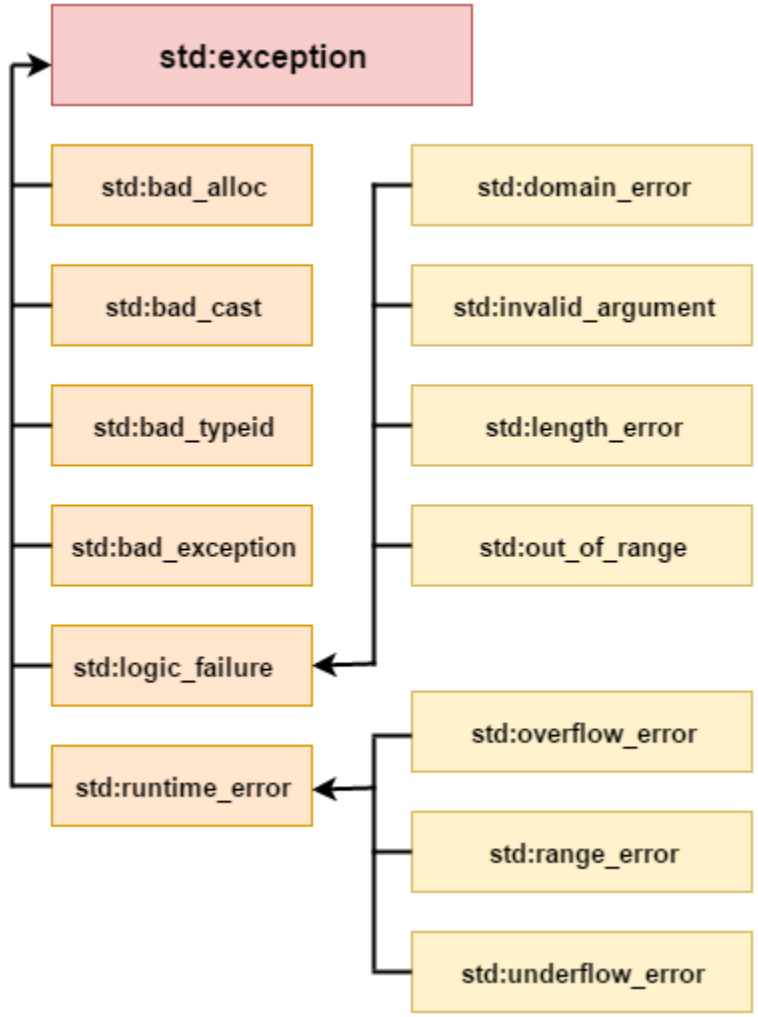
It maintains the normal flow of the application. In such case, rest of the code is executed even after exception.

C++ Exception Classes

In C++ standard exceptions are defined in <exception> class that we can use inside our programs. The arrangement of parent-child class hierarchy is shown below:

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit 5 - Files and Exception Handling



All the exception classes in C++ are derived from std::exception class. Let's see the list of C++ common exception classes.

Exception	Description
std::exception	It is an exception and parent class of all standard C++ exceptions.
std::logic_failure	It is an exception that can be detected by reading a code.
std::runtime_error	It is an exception that cannot be detected by reading a code.
std::bad_exception	It is used to handle the unexpected exceptions in a c++ program.
std::bad_cast	This exception is generally be thrown by **dynamic_cast.**

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

std::bad_typeid	This exception is generally be thrown by **typeid.**
std::bad_alloc	This exception is generally be thrown by **new.**

C++ Exception Handling Keywords

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

- try
- catch, and
- throw

Moreover, we can create user-defined exception which we will learn in next chapters.

C++ try/catch

In C++ programming, exception handling is performed using try/catch statement. The C++ **try block** is used to place the code that may occur exception. The **catch block** is used to handle the exception.

C++ example without try/catch

```
1. #include <iostream>
2. using namespace std;
3. float division(int x, int y) {
4.    return (x/y);
5. }
6. int main () {
7.    int i = 50;
8.    int j = 0;
9.    float k = 0;
10.     k = division(i, j);
11.     cout << k << endl;
12.   return 0;
13.}
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Output:

```
Floating point exception (core dumped)
```

C++ try/catch example

```
1.  #include <iostream>
2.  using namespace std;
3.  float division(int x, int y) {
4.     if( y == 0 ) {
5.        throw "Attempted to divide by zero!";
6.     }
7.     return (x/y);
8.  }
9.  int main () {
10.    int i = 25;
11.    int j = 0;
12.    float k = 0;
13.    try {
14.       k = division(i, j);
15.       cout << k << endl;
16.    }catch (const char* e) {
17.       cerr << e << endl;
18.    }
19.    return 0;
20. }
```

Output:

```
Attempted to divide by zero!
```

C++ User-Defined Exceptions

The new exception can be defined by overriding and inheriting **exception** class functionality.

C++ user-defined exception example

Unit 5 - Files and Exception Handling

Let's see the simple example of user-defined exception in which **std::exception** class is used to define the exception.

```
1.  #include <iostream>
2.  #include <exception>
3.  using namespace std;
4.  class MyException : public exception{
5.     public:
6.        const char * what() const throw()
7.        {
8.           return "Attempted to divide by zero!\n";
9.        }
10. };
11. int main()
12. {
13.    try
14.    {
15.       int x, y;
16.       cout << "Enter the two numbers : \n";
17.       cin >> x >> y;
18.       if (y == 0)
19.       {
20.          MyException z;
21.          throw z;
22.       }
23.       else
24.       {
25.          cout << "x / y = " << x/y << endl;
26.       }
27.    }
28.    catch(exception& e)
29.    {
30.       cout << e.what();
31.    }
32. }
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Output:

```
Enter the two numbers :
10
2
x / y = 5
```

Output:

```
Enter the two numbers :
10
0
Attempted to divide by zero!
```

-->

Note: In above example what() is a public method provided by the exception class. It is used to return the cause of an exception.