File Handling

Today’s Agenda

Today we are going to cover -

- File Handling
- Opening and Closing of files
- Modes of file
- Reading and Writing of files
- Sequential access processing
- Random access file processing
- Binary file operations
- Classes and file operations
- Structures and file operations

Let’s Get Started-

File Handling and Stream

Files store data permanently in a storage device. With file handling, the output from a program can be stored in a file. Various operations can be performed on the data while in the file.

A stream is an abstraction of a device where input/output operations are performed. You can represent a stream as either a destination or a source of characters of indefinite length. This will be determined by their usage. C++ provides you with a library that comes with methods for file handling.

fstream file

The fstream library provides C++ programmers with three classes for working with files. These classes include:

- **ofstream**- This class represents an output stream. It's used for creating files and writing information to files.

- **ifstream**- This class represents an input stream. It's used for reading information from data files.

- **fstream**- This class generally represents a file stream. It comes with ofstream/ifstream capabilities. This means it's capable of creating files, writing to files, reading from data files.

Opening of a file

Before performing any operation on a file, you must first open it. If you need to write to the file, open it using fstream or ofstream objects. If you only need to read from the file, open it using the ifstream object.

The three objects, that is, fstream, ofstream, and ifstream, have the open() function defined in them. The function takes this syntax:

Syntax:-   open (file_name, mode);

- The file_name parameter denotes the name of the file to open.
- The mode parameter is optional. It can take any of the following values.

Opening of a file

ios:: app	The Append mode. The output sent to the file is appended to it.
ios::ate	It opens the file for the output then moves the read and write control to file's end.
ios::in	It opens the file for a read.
ios::out	It opens the file for a write.
ios::trunc	If a file exists, the file elements should be truncated prior to its opening.

Closing of a file

Once a C++ program terminates, it automatically

- flushes the streams
- releases the allocated memory
- closes opened files.

Closing of a file

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
        fstream my_file;
        my_file.open("my_file", ios::out);
        if (!my_file) {
                cout << "File not created!";
        }
        else {
                cout << "File created successfully!";
                my_file.close();
        }
        return 0;
}
```

Writing of a file

We use stream insertion operator (<<) for writing on a file. The text to be written to the file should be enclosed within double-quotes.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
        fstream my_file;
        my_file.open("my_file.txt", ios::out);
        if (!my_file) {
                cout << "File not created!";
        }
        else {
                cout << "File created successfully!";
                my_file << "LPU";
                my_file.close();
        }   }
```

Read from file

We can read from a file using stream extraction operator (>>). We use the operator in the same way you use it to read user input from the keyboard. However, instead of using the cin object, you use the ifstream/ fstream object.

Read from file

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

int main() {
	fstream my_file;
	my_file.open("my_file.txt", ios::in);
	if (!my_file) {
		cout << "No such file";
	}
	else {
		char ch;
```

Read from file

```
while (1) {
my_file >> ch;
if (my_file.eof())
break;

cout << ch;
}

}
my_file.close();
return 0;
}
```

MCQ 1

1. Which header file is required to use file I/O operations?

a) <ifstream>

b) <ostream>

c) <fstream>

d) <iostream>

MCQ 1

1. Which header file is required to use file I/O operations?

a) <ifstream>

b) <ostream>

c) <fstream>

d) <iostream>

MCQ 2

Which of the following statements are correct?

1) It is not possible to combine two or more file opening mode in open() method.

2) It is possible to combine two or more file opening mode in open() method.

3) ios::in and ios::out are input and output file opening mode respectively.

a) 1, 3

b) 2, 3

c) 3 only

d) 1, 2

MCQ 2

Which of the following statements are correct?

1) It is not possible to combine two or more file opening mode in open() method.

2) It is possible to combine two or more file opening mode in open() method.

3) ios::in and ios::out are input and output file opening mode respectively.

a) 1, 3

b) 2, 3

c) 3 only

d) 1, 2

MCQ 3

3. Which of the following is the default mode of the opening using the ifstream class?

a) ios::in

b) ios::out

c) ios::app

d) ios::trunc

MCQ 3

3. Which of the following is the default mode of the opening using the ifstream class?

a) ios::in

b) ios::out

c) ios::app

d) ios::trunc

MCQ 4

4.Which of the following is the default mode of the opening using the fstream class?

a) ios::in

b) ios::out

c) ios::in|ios::out

d) ios::trunc

MCQ 4

4.Which of the following is the default mode of the opening using the fstream class?

a) ios::in

b) ios::out

c) ios::in|ios::out

d) ios::trunc

MCQ 5

5. Which operator is used to insert the data into file?

a) >>

b) <<

c) <

d) >

MCQ 5

5. Which operator is used to insert the data into file?

a) >>

b) <<

c) <

d) >

Sequential access processing

When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

Sequential Access

It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation *-read next-* read the next position of the file and automatically advance a file pointer, which keeps track I/O location. Similarly, for the write*write next* append to the end of the file and advance to the newly written material.

Sequential Access

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

Random Access

Rather than reading all of the records until you get to the one you want, you can skip directly to the record you wish to retrieve. **Random file access** is done by manipulating the **file** pointer using either seekg() function (for input) and seekp() function (for output).

Seekg()

seekg() is a function in the iostream library (part of the standard library) that allows you to seek to an arbitrary position in a file. It is used in file handling to sets the position of the next character to be extracted from the input stream from a given file. For example

Input : "Hello World"
Output : World

Seekg()

Syntax – There are two syntax for seekg() in file handling :

- seekg(streampos position);
- seekg(streamoff offset, ios_base::seekdir dir);

Description –

- **position :** is the new position in the stream buffer.
- **offset :** is an integer value of type streamoff representing the offset in the stream's buffer. It is relative to the dir parameter.
- **dir :** is the seeking direction. It is an object of type ios_base::seekdir that can take any of the following constant values.

Seekg()

There are 3 direction we use for offset value :

- ios_base::beg (offset from the beginning of the stream’s buffer).
- ios_base::cur (offset from the current position in the stream’s buffer).
- ios_base::end (offset from the end of the stream’s buffer).

Seekg()

```
// Code to demonstrate the seekg function in file handling
#include <fstream>
#include <iostream>

using namespace std;

int main (int argc, char** argv)
{
   // Open a new file for input/output operations
   // discarding any current in the file (assumes
   // a length of zero on opening)
   fstream myFile("test.txt", ios::in | ios::out | ios::trunc);

   // Add the characters "Hello World" to the file
   myFile << "Hello World";
```

Seekg()

```
  // Seek to 6 characters from the beginning of the file
   myFile.seekg(6, ios::beg);

   // Read the next 5 characters from the file into a buffer
   char A[6];
   myFile.read(A, 5);

   // End the buffer with a null terminating character
   A[5] = 0;

   // Output the contents read from the file and close it
   cout << A << endl;

   myFile.close();
}
```

Binary File Operation

Writing:-

To write a binary file in C++ use write method. It is used to write a given number of bytes on the given stream, starting at the position of the "put" pointer. The file is extended if the put pointer is currently at the end of the file. If this pointer points into the middle of the file, characters in the file are overwritten with the new data.
If any error has occurred during writing in the file, the stream is placed in an error state

Syntax of write method

```
ostream:: write(const char*, int);
```

Binary File Operation

Reading:-

To read a binary file in C++ use read method. It extracts a given number of bytes from the given stream and place them into the memory, pointed to by the first parameter. If any error is occurred during reading in the file, the stream is placed in an error state, all future read operation will be failed then.

gcount() can be used to count the number of characters has already read. Then clear() can be used to reset the stream to a usable state.

Syntax of read method

```
istream:: write(const char*, int);
```

Algorithm

```
Begin
  Create a structure Student to declare variables.
  Open binary file to write.
  Check if any error occurs in file opening.
  Initialize the variables with data.
  If file open successfully, write the binary data using write method.
     Close the file for writing.
  Open the binary file to read.
  Check if any error occurs in file opening.
  If file open successfully, read the binary data file using read method.
     Close the file for reading.
  Check if any error occurs.
  Print the data.
End.
```

Code

```
 #include<iostream>
#include<fstream>
using namespace std;
struct Student {
   int roll_no;
   string name;
};
int main() {
   ofstream wf("student.dat", ios::out | ios::binary);
   if(!wf) {
      cout << "Cannot open file!" << endl;
      return 1;
   }
```

Code

```
Student wstu[3];
   wstu[0].roll_no = 1;
   wstu[0].name = "Ram";
   wstu[1].roll_no = 2;
   wstu[1].name = "Shyam";
   wstu[2].roll_no = 3;
   wstu[2].name = "Madhu";

for(int i = 0; i < 3; i++)
      wf.write((char *) &wstu[i], sizeof(Student));
   wf.close();
   if(!wf.good()) {
      cout << "Error occurred at writing time!" << endl;
      return 1;
   }
```

Code

```
ifstream rf("student.dat", ios::out | ios::binary);
  if(!rf) {
    cout << "Cannot open file!" << endl;
    return 1;
  }
  Student rstu[3];
  for(int i = 0; i < 3; i++)
    rf.read((char *) &rstu[i], sizeof(Student));
  rf.close();
  if(!rf.good()) {
    cout << "Error occurred at reading time!" << endl;
    return 1;
  }
```

Code

```
cout<<"Student's Details:"<<endl;
   for(int i=0; i < 3; i++) {
      cout << "Roll No: " << wstu[i].roll_no << endl;
      cout << "Name: " << wstu[i].name << endl;
      cout << endl;
   }
   return 0;
}
```

Output

```
Student’s Details:
Roll No: 1
Name: Ram
Roll No: 2
Name: Shyam
Roll No: 3
Name: Madhu
```

QNA Time

Any Questions??

See you guys in next class.