File Handling Notes

File Handling

File

A file is a stream of bytes stored on some secondary storage devices. • Two types of files Supported by C++

1. **Text file:** A text file stores information in readable and printable form. Each line of text is terminated with an EOL (End of Line) character.

2. **Binary file:** A binary file contains information in the non-readable form i.e. in the same format in which it is held in memory.

File Stream

Stream: A stream is a general term used to name flow of data. Different streams are used to represent different kinds of data flow. •

There are 3 file I/O classes used for file read /write operations.

1. ifstream - can be used for read operations.
2. ofstream - can be used for write operations.
3. fstream - can be used for both read & write operations.

fstream.h: • This header file includes the definitions for the stream classes ifstream, ofstream and fstream.

In C+ file input output facilities implemented through fstream.h header file. •
It contain predefines set of operation for handling file related input and output, fstream class ties a file to the program for input and output operation.

OPENING FILE /CREATING FILE

• A file can be opened using:

1. **By the constructor method**. This will use default streams for file input or output. This method is preferred when file is opened in input or output mode only.

   Example :
   ofstream file(“student.dat”);
   or
   ifstream file(“student.dat”);

2. By the **open() member function** of the stream. It will preferred when file is opened in various modes i.e ios:in, ios:out, ios:ap, ios:ate etc.
   e.g

```
fstream file;
file.open("bok.dat", ios:in |ios:out |ios:binary);
```

FILE MODES:

• **ios:out** It open file in output mode (i.e write mode) and place the file pointer in beginning, if file already exist it will overwrite the file.

• **ios:in** It open file in input mode(read mode) and permit reading from the file.

• **ios:app** It open the file in write mode, and place file pointer at he end of file i.e to add new contents and retains previous contents. If file does not exist it will create a new file.

• **ios:ate** It open the file in write or read mode, and place file pointer at the end of file i.e input/ output operations can performed anywhere in the file.

• **ios:trunc** It truncates the existing file (empties the file).

• **ios:nocreate** If file does not exist his file mode ensures that no file is created and open() fails.

• **ios:noreplace** If file does not exist, a new file gets created but if the file already exists, the open() fails.

• **ios:binary** Opens a file in binary mode.

---

eof(): This function determines the end-of-file by returning true(non-zero) for end of file otherwise returning false(zero).

close(): This function terminates the connection between the file and stream associated with it.

Syntax:
Stream_object.close();
e.g
file.close();

Text File functions:

Read a Character :

get() – read a single character from text file and store in a buffer(temporary area).
e.g file.get(ch);

put() - writing a single character in textfile
e.g. file.put(ch);

Read a line:

getline() - read a line of text from text file store in a buffer.
e.g file.getline(s,80);

We can also use file>>ch for reading and file<<ch writing in text file.
But >> operator does not accept white spaces.

Binary file functions:

read()- read a block of binary data or reads a fixed number of bytes from the specified stream and store in a buffer.

Syntax :
Stream_object.read(char *)& Object, sizeof(Object);
e.g
file.read(char *)&s, sizeof(s);

write() – write a block of binary data or writes fixed number of bytes from a specific memory location to the specified stream.
Syntax :
Stream_object.write(char *)& Object, sizeof(Object);
e.g
file.write(char *)&s, sizeof(s);

Note:

Both read() and write () functions take two arguments.
• The first is the address of variable, and the second is the length of that variable in bytes. The address of variable must be type cast to type char*(pointer to character type)
• The data written to a file using write( )can only be read accurately using read( ).

File Pointer:

The file pointer indicates the position in the file at which the next input/output is to occur. Moving the file pointer in a file for various operations viz modification, deletion ,searching etc. Following functions are used:

seekg(): It places the file pointer to the specified position in input mode of file.
e.g
file.seekg(p,ios:beg);
or
file.seekg(-p,ios:end),
or
file.seekg(p,ios:cur)
i.e to move to p byte position from beginning, end or current position.

seekp(): It places the file pointer to the specified position in output mode of file.
e.g
file.seekp(p,ios:beg);
or
file.seekp(-p,ios:end),
or
file.seekp(p,ios:cur)
i.e to move to p byte position from beginning, end or current position.

tellg(): This function returns the current working position of the file pointer in the input mode.
e.g
long/int p=file.tellg();

tellp(): This function returns the current working position of the file pointer in the output mode.
e.g.
long/ int p=file.tellp();

Q Suggest the situation where write() and read() are preferred over get() and put() for file I/O operations. Support your answer with examples.
Ans.
The get() and put() functions perform I/O byte by byte. On the other hand, read() and write() functions let you read
and write structures and objects in one go without creating need for I/O for individual constituent fields.
Example:
file.get(ch);
file.put(ch);
file.read((char *)&obj, sizeof(obj));
file.write((char *)&obj, sizeof(obj));

Q Discuss the working of good() and bad() functions in file I/O error handling.
Ans.
good(): Returns nonzero (true) if no error has occurred. For instance, if fin.good() is true, everything is okay with the stream named as fi and we can proceed to perform I/O operations. When it returns zero, o further operations can be carried out.
bad(): Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

Q What are the similarities and differences between bad() and fail() functions.
Ans. Similarities: bad() and fail() both are error handling functions and return true if a reading or writing operation fails.
Differences: Both bad() and fail() return true if a reading or writing operation fails but fail() also returns true in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

Q How is the working of file I/O error handling functions associated with error-status flags?
Ans The error-status flags store the information on the status of a file that is being currently used. The current state of the I/O system is held in an integer, in which the following flags are encoded:
Name Meaning
eofbit 1 when end-of-file is encountered, 0 otherwise.
failbit 1 when non-fatal I/O error has occurred, 0 otherwise.
badbit 1 when fatal I/O error has occurred, 0 otherwise.
goodbit 0 value

Q. What are input and output streams? What is the significance of fstream.h file?
Ans.
Input stream: The stream that supplies data to the program is known as input stream.
Output stream: The stream that receives data from the program is known as output stream.
fstream.h file includes the definitions for the stream classes ifstream, ofstream and fstream. In C++ file input output
facilities implemented through fstream.h header file.

Q. Discuss the files stream classes defined inside fstream.h header file.
Ans. ifstream: can be used for read operations.
ofstream: can be used for write operations.
fstream: can be used for both read & write operations.

Q. What are the steps involved in using a file in a C++ program?
Ans. In order to process files, follow these steps:
(i) Determine the type of link.

(ii) Declare a stream accordingly.
(iii) Link file with the stream
(iv) Process as required, and
(v) De-link the file with the stream.

Q. Describe the various classes available for file operations.
Ans. Class Functions
filebuf It sets the file buffers to read and write.
fstreambase This is the base class for fstream, ifstream and ofstream classes.
ifstream It provides input operations for file.
ofstream It provides output operations.
fstream It provides support for simultaneous input and output operations.

Q. Discuss the two methods of opening a file within a C++ program. When is one method preferred over the other?
Ans. A file can be opened in two ways :-
a) Using the constructor of the stream class – This method is useful when only one file is used in the stream.
Constructors of the stream classes ifstream, ofstream and fstream are used to initialize the file stream object with the file name. For example,
ifstream read_file("Names.Dat");

b) Using the function open() – This method is useful when we want to use different files in the stream. If two or more files are to be processed simultaneously, separate streams must be declared for each. For example,
ifstream ifl; //input stream ifl created
ifl.open("Names.Dat"); // file Names.Dat linked with ifl

Second method is preferred over first method when there is a situation to open more than one file.

Q. How many file objects would you need to create to manage the following situations? Explain
(i) to process three files sequentially
(ii) to merge two sorted files into a third file
Ans. i) 3
ii) 3

Q. Both ios::ate and ios::app place the file pointer at the end of the file when it is opened. What then, is the difference between them?
Ans. Both ios::ate and ios::app place the file pointer at the end of the file when it is opened. The difference between the two is that ios::app lets you add data to the end of the file only, while the ios::ate mode when opened with ofstream allows you to write data anywhere in the file, even over old data.

Q When a file is opened for output what happens when
(i) the mentioned file does not exist.
(ii) the mentioned file does exist.
Ans. (i) Creates a new file.
(ii) the act of opening a file for output scraps it off so that output starts with a fresh file.
Q When do you think text files should be preferred over binary files?
Ans. When file does need to be read by people or need to be ported to a different type of system, text files should be preferred over binary files.