CHAPTER 5

WORKING WITH FILES

Marks 08

Objectives:

- C++ stream classes. Classes for file stream operations.
- Opening files, closing files, reading from and writing to files.
- Detection of end of file, file modes

Concept of Streams

A stream is a source of sequence of bytes. A stream abstracts for input/output devices. It can be tied up with any I/O device and I/O can be performed in a uniform way. The C++ iostream library is an object-oriented implementation of this abstraction. It has a source (producer) of flow of bytes and a sink (consumer) of the bytes. The required classes for the stream I/O are defined in different library header files.
To use the I/O streams in a C++ program, one must include iostream.h header file in the program. This file defines the required classes and provides the buffering. Instead of functions, the library provides operators to carry out the I/O. Two of the Stream Operators are:
<< : Stream insertion for output.
>> : Stream extraction for input.
The following streams are created and opened automatically:
cin : Standard console input (keyboard).
cout : Standard console output (screen).
cprn : Standard printer (LPT1). cerr : Standard error output (screen).
clog : Standard log (screen). caux : Standard auxiliary (screen).

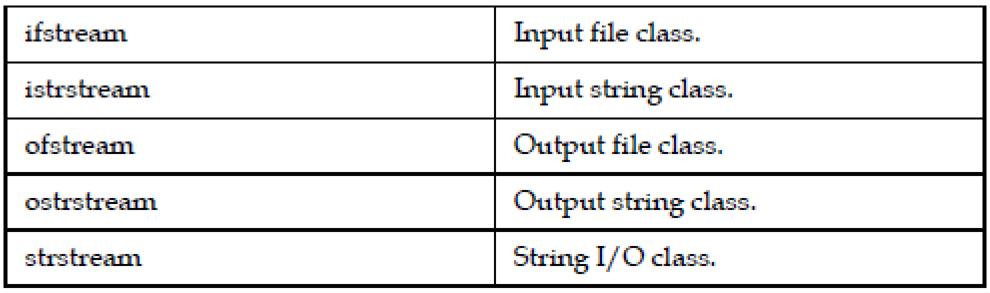
Example: The following program reads an integer and prints the input on the console.

```
#include <iostream> // Header for stream I/O.
int main(void)
{
int p; // variable to hold the input integer
cout << “Enter an integer: “;
cin >> p;
cout << “\n You have entered “ << p;
}
```

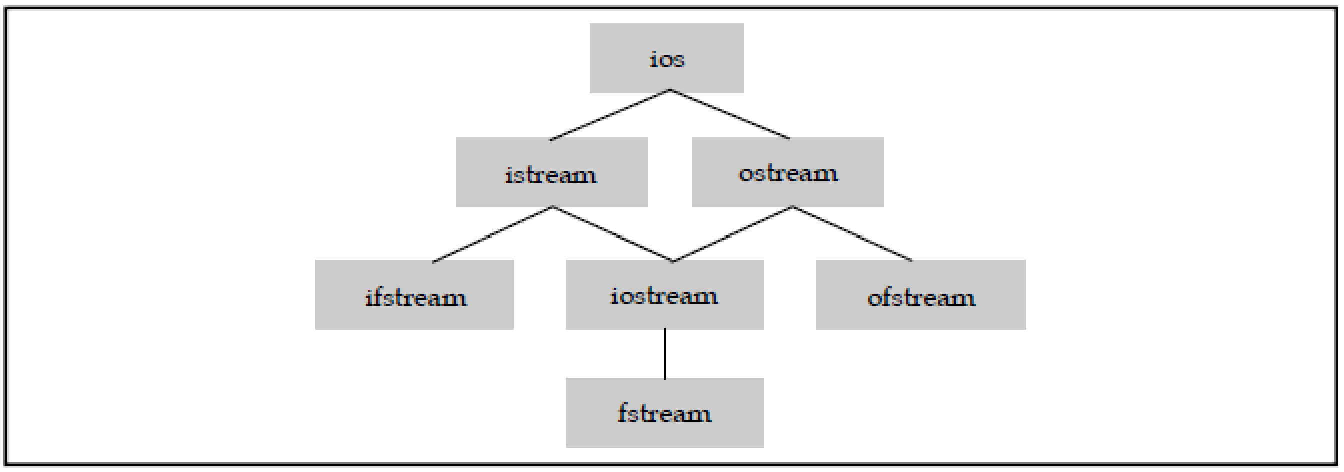
Hierarchy of Console Stream Classes

Streams can also be tied up with data files. The required stream classes for file I/O are defined in fstream.h and/or strstream.h.

fstream : File I/O class.



ifstream	Input file class.
istrstream	Input string class.
ofstream	Output file class.
ostrstream	Output string class.
strstream	String I/O class.



Data Streams

Programs would not be very useful if they cannot input and/or output data from/to users. Some programs that require little or no input for their execution are designed to be interactive through user console - keyboard for input and monitor for output. However, when data volume is large it is generally not convenient to enter the data through console. In such cases data can be stored in a file and then the program can read the data from the data file rather than from the console. The data may be organized in fixed size record or may be in free form text.
The various operations possible on a data file using C++ programs are:

1. Opening a data file
2. Reading data stored in the data file into various variables and objects in the program
3. Writing data from a program into a data file
    a. Removing the previously stored data in the file
    b. Without removing the previously stored data in the file
        i. At the end of the data file
        ii. At any other location in the file
4. Saving the data file onto some secondary storage device
5. Closing the data file once the ensuing operations are over
6. Checking status of file operation

C++ treats each source of input and output uniformly. The abstraction of a data source and data sink is what is termed as stream. A stream is a data abstraction for input/output of data to and from the program

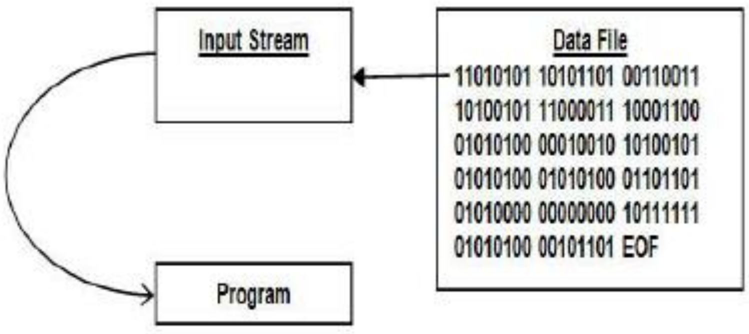


Figure (a): Input Stream Currently Attached to a Data File

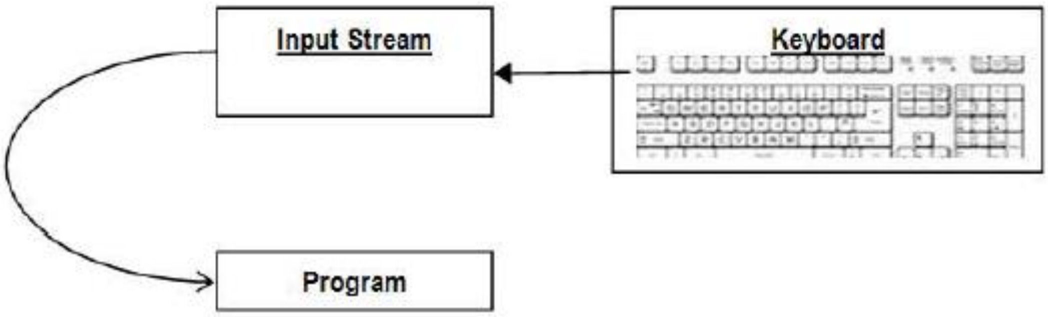


Figure (b): Input Stream Currently attached to Keyboard

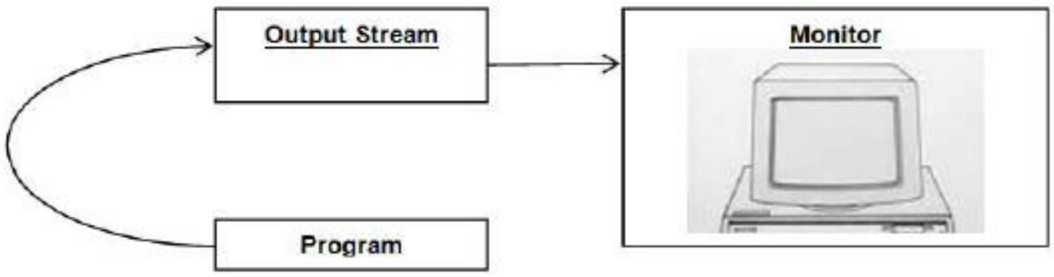


Figure (c): Output Stream Currently Attached to Monitor

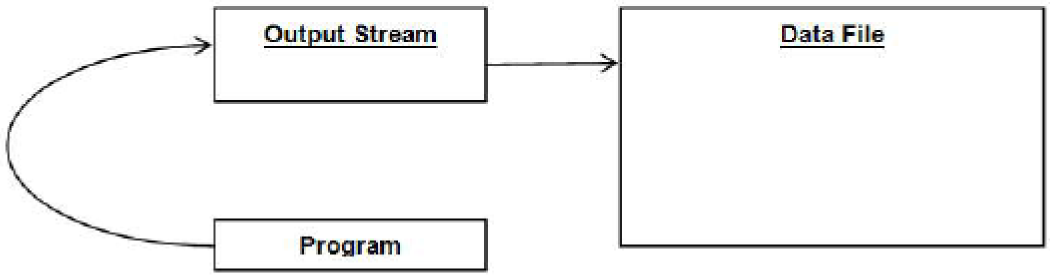


Figure (d): Output Stream Currently Attached to a Data File

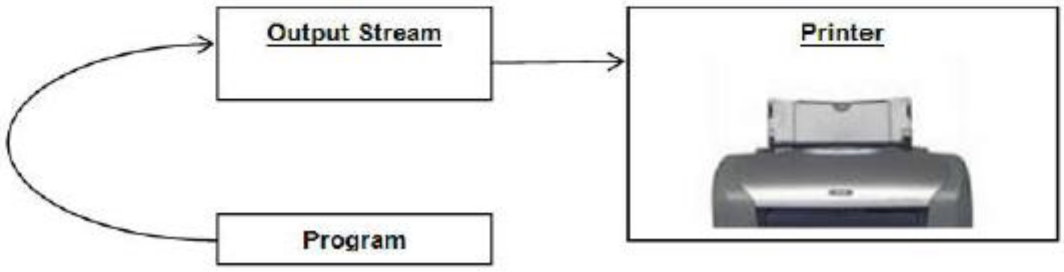


Figure (e): Output Stream Currently Attached to a Printer

C++ library provides prefabricated classes for data streaming activities. In C++, the file stream classes are designed with the idea that a file should simply be viewed as a stream or array or sequence of bytes. Often the array representing a data file is indexed from 0 to len-1, where len is the total number of bytes in the entire file.

A file normally remains in a closed state on a secondary storage device until explicitly opened by a program. A file in open state file has two pointers associated with it:

1. **Read pointer:** This pointer always points to the next character in the file which will be read if a read command is issued next. After execution of each read command the read pointer moves to point to the next character in the file being read.

2. **Write pointer:** The write pointer indicates the position in the opened file where next character being written will go. After execution of each write command the write pointer moves to the next location in the file being written.

These two file positions are independent, and either one can point anywhere at all in the file. To elucidate I/O streaming in C++ let us write a small program.

```
#include <fstream.h>
int main()
{
        ofstream filename("c:\cppio.dat");
        filename << "This text will be saved in the file named cppio.dat in
        C:\";
        filename.close();
        return 0;
}
```

When you run this program the text This text will be saved in the file named cppio.dat in C:\ will be saved in a file named cppio.dat in the root directory of the C: drive. Let us run through each line of this program.

```
#include <fstream.h>
```

C++ file stream classes and functions have been defined in this file. Therefore, you must include this header file in the beginning of your C++ program so that these classes may be accessed.

```
ofstream filename ("c:\cppio.dat");
```

This statement creates an object of class ofstream (acronym for output file stream) named filename. This object acts as the output stream to write data in the specified file. Cppio.dat is the name of the data file in which the program will write its output. If this file does not exists in the specified directory, it is created there.

Here the file name c:\cppio.dat is being passed to the constructor of this class. In short we create an object from class ofstream, and we pass the name of the file we want to create, as an argument to the class' constructor. There are other things, too, that can be passed to the constructor.

```
filename << "This text will be saved in the file named cppio.dat in C:\";
```

The << operator is a predefined operator. This line puts the text in the file. As mentioned before, filename is a handle to the opened file stream. So, we write the handle name, << and after it we write the text in inverted commas. If we want to pass variables instead of text in inverted commas, just pass it as a regular use of the cout << as,

```
filename << variablename;
```

```
filename.close();
```

Having finished with writing into the file the stream must be closed. Filename is an object of class ofstream, and this class has a function close() that closes the stream. Just write the name of the stream object, dot and close(), in order to close the file stream. Note that once you close the file, you can't access it anymore, until you reopen it.

Before we take up the subject any further let us quickly go through a program that reads from a data file and presents the same on the monitor. Here is the program listing.

```
#include <fstream.h>
void main() //the program starts here
{
```

```
ifstream filename(“c:\cppio.dat”);
char ch;
while(!filename.eof())
{
filename.get(ch);
cout << ch;
}
filename.close();
}
```

Let us run through this program line by line quickly to catch some salient features.

ifstream filename(“c:\cppio.dat”)

Similar to ofstream, ifstream is the input stream for a file. The term input and output have been used with respect to the program. What goes to the program from outside is the input while processed data coming out of the program is the output. We here create an input file stream by the name - filename - to handle the input file stream. The parameter passed to the constructor is the file we wish to read into the program.

char ch;

This statement declares a variable of type char to hold one character at a time while reading from the file. In this program we intend to read one character at a time.

while(!filename.eof())

The class ifstream has a member function eof() that returns a nonzero value if the end of the file has been reached. This value indicates that there are no more characters in the file to be read further. This function is therefore used in the while loop for stopping condition. The file is read a character at a time till the last character has been successfully read into the program when the while loop terminates.

filename.get(ch);

Another member function of the class ifstream is get() which returns the next character to be read from the stream followed by moving the character pointer to the next character in the stream.

cout << ch;

The character read in the variable ch is streamed to the standard output device designated by cout (for console output, i.e., monitor) in this statement.

filename.close();

Since we reach at this statement when all the characters have been read and processed in the while loop, we are done with the file and hence it is duly closed.

Opening a File

The example programs listed above are indeed very simple. A data file can be opened in a program in many ways. These methods are described below.

ifstream filename(“filename <with path>”); Or ofstream filename(“filename <with path>”);

This is the form of opening an input file stream and attaching the file “filename with path” in one single step. This statement accomplishes a number of actions in one go:

1. Creates an input or output file stream
2. Looks for the specified file in the file system
3. Attaches the file to the stream if the specified file is found otherwise returns a NULL value

In case the file has been successfully attached to the stream the pointer is placed at the first position The file stream created is accessible using the object’s name. The specified file is searched in the specified directory if a path is included otherwise the file is searched only in the current directory. If the file is not found it is created in case it is being opened for output. While opening a file for output if the specified file is found then it is truncated before opening thereby loosing all there was in the file before. Care should be taken to ensure that the program does not overwrite a file unintentionally. In any case if the file is not found then a NULL value is returned which we can check to ensure that we are not reading a file which was not found. This will cause an error in the program if we attempt to read a file which was not found.

```
ifstream filename;

filename.open(“file name <with path>”);
```

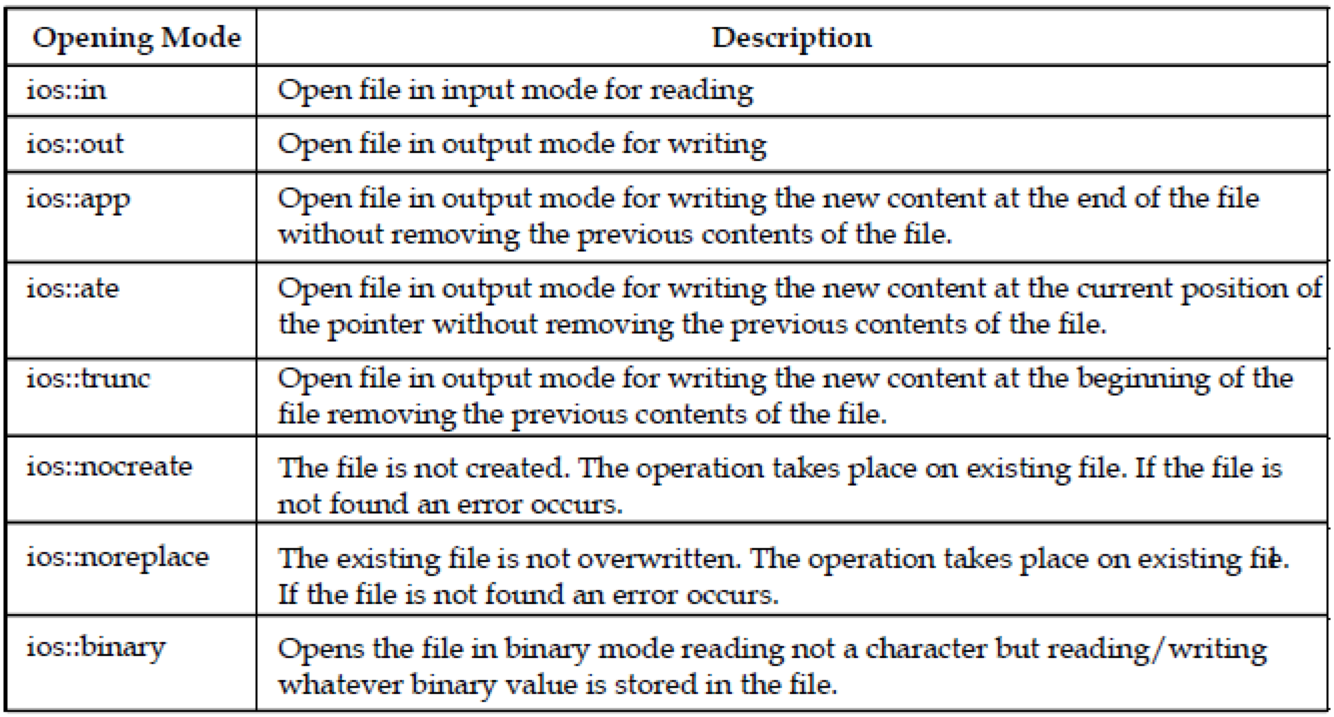
In this approach the input stream - filename - is created but no specific file is attached to the stream just created. Once the stream has been created a file can be attached to the stream using open() member function of the class ifstream or ofstream as is exemplified by the following program snippet which defines a function to read an input file.

```
#include <fstream.h>
void read(ifstream &ifstr) // file streams can be passed to functions
{
char ch;
while(!ifstr.eof())
{
ifstr.get(ch);
cout << ch;
}
cout << endl << “————” << endl;
}
void main()
{
ifstream filename(“data1.dat”);
read(filename);
filename.close();
filename.open(“data2.dat”);
read(filename);
filename.close();
}
```

```
ifstream filename(char *fname, int open_mode);
```

In this form the ifstream constructor takes two parameters - a filename and another the mode in which the input file would be read. C++ offers a host of different opening modes for the input file each offering different types of reading control over the opened file. The file opening modes have been implemented in C++ as enumerated type called ios.

The various file opening modes are listed below.



Opening Mode	Description
ios::in	Open file in input mode for reading
ios::out	Open file in output mode for writing
ios::app	Open file in output mode for writing the new content at the end of the file without removing the previous contents of the file.
ios::ate	Open file in output mode for writing the new content at the current position of the pointer without removing the previous contents of the file.
ios::trunc	Open file in output mode for writing the new content at the beginning of the file removing the previous contents of the file.
ios::nocreate	The file is not created. The operation takes place on existing file. If the file is not found an error occurs.
ios::noreplace	The existing file is not overwritten. The operation takes place on existing file. If the file is not found an error occurs.
ios::binary	Opens the file in binary mode reading not a character but reading/writing whatever binary value is stored in the file.

Note that all these values are int constants from an enumerated type ios. The following program demonstrates the usages of a file opening mode.

```
#include <fstream.h>
void main()
{
ofstream myfile("data1.dat", ios::ate);
myfile << "Save this text to the file";
myfile.close();
}
```

This program will write Save this text to the file at the end of the file data1.dat without removing the previous content of the file.

A file can also be opened in mixed mode using OR (|) operator to combine the modes. For example,

ios::ate | ios::binary

opens the file in binary mode for output at the current pointer position without removing the previous contents of the file. Another example of mixed mode is given below.

fstream myfile("data.dat",ios::in | ios::out);

Input file stream (ifstream) allows only reading from the file and output file stream (ofstream) only for writing into the file. If you wish to open a file both for writing and reading at the same time you should use file stream (fstream) with specific opening mode as the following program demonstrates.

```
#include <fstream.h>
void main()
{
fstream myfile ("data.dat",ios::in | ios::out);
myfile << "Write this text"; //Writing into the file
static char TextRead[50]; //Create an array to hold what is
read from the file
myfile.seekg(ios::beg); //Get back to the beginning of
the file explained later
myfile >> TextRead;
cout << TeaxtRaed << endl;
myfile.close();
}
```

In this program the statement fstream myfile ("data.dat",ios::in | ios::out); creates an object from class fstream. At the time of execution, the program opens the file data.dat in read/write mode (ios::in | ios::out) which allows to read from the file, and put data into it, at the same time.

This is done by allowing the program to move the character pointer in the file to a desired location before reading or writing.

The read and write operation takes place on the current position of the pointer in the file.

Therefore you might need to shift this position to a desired location. The member function seekg() of the class fstream allows you to do just that as shown below.

```
myfile.seekg(ios::beg) //Take the pointer to the beginning of the file
myfile.seekg(ios::end) //Take the pointer to the end of the file

myfile.seekg(10) //Take the pointer to 10 characters after the current location Notes
myfile.seekg(-10) //Take the pointer to 10 characters before the current location
```

Moreover, the function seekg()is overloaded to take two parameters. Thus, another form of the function is,

myfile.seekg(-4, ios::end) //Take the pointer 4 characters before the end of file Most of the time it is necessary to check whether a specified file exists or not while opening it.

Opening of file may also fail due to other reasons. Remember that if the file opening fails the open() function return a non-zero value. By checking the return type of the function one can ensure whether the file was opened successfully or not as demonstrated in the following program snippet.

```
fstream myfile("data.dat");

if (!myfile)
{
cout << "Error : File could not be opened\n";
exit(1);
}
```

The ofstream class also provides a fail() function which return true if the opening of file operation failed as shown in the following code snippet

```
ofstream myfile("data.dat", ios::nocreate);

if(myfile.fail())
{
cout << "Error : File could not be opened\n";
exit(1);
}
```

binary files are unformatted and uninterpreted file of binary digits. It simply contains binary numbers whose meaning is provided by the program that manipulates the file contents. The functions that give you the possibility to write/read unformatted files are get() and put(). To read a byte, you can use get() and to write a byte, use put(). Both get() and put() functions take one parameter - a char variable or character.

If you want to read/write whole blocks of data, then you can use the read() and write() functions.

Their prototypes are:

```
istream &read(char *buf, streamsize num);
```

```
ostream &write(const char *buf, streamsize num);
```

For the read() function, but should be an array of chars, where the read block of data will be put.
For the write() function, buf is an array of chars, where is the data you want to save in the file. For the both functions, num is a number, that defines the amount of data (in symbols) to be read/written.

Another function that provides the number of symbols read so far - gcount(). It simply function returns the number of read symbols for the last unformatted input operation. You can specify that the file is going to be operated on in binary mode in the open() function. The required mode
is ios::binary.

Let us write a program that uses get() and put() functions for binary files.

```
#include <fstream.h>
void main()
{
fstream myfile(“data.dat”,ios::out | ios::in | ios::binary);
//open file in binary mode
char ch;
ch=’A’;
myfile.put(ch); //put the content of ch to the file
myfile.seekg(ios::beg); //go to the beginning of the file
myfile.get(ch); //read one character
cout << ch << endl; //display it on console
myfile.close();
}
```

The following program uses read() and write() functions for binary file manipulations.

```
#include <fstream.h>
#include <string.h>
void main()
{
fstream myfile(“data.dat”,ios::out | ios::in | ios::binary);
char Carray[13];
strcpy(Carray,”C++ Programming”); //put the string into the
character array
myfile.write(Carray,5); //put the first 5 characters “C++ P”
into the file
myfile.seekg(ios::beg); //go to the beginning of the file
static char Rarray[10]; //To store read data
myfile.read(Rarray,3); //read the first 3 characters- “C++”
cout << Rarray << endl; //display them on console
myfile.close(); //close file
}
```

PROGRAMS

Write a C++ program to count number of spaces present in contents of file.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
ifstream file;
charch;
int s=0;
clrscr();
file.open("abc.txt");
while(file)
{
file.get(ch);
if(ch==' ')
 {
s++;
 }
}
cout<<"\nNumber of spaces present in the content of the given file
are:"<<s;
getch();
}
```

Write a C++ program to write 'Welcome to poly' in a file. Then read the data from file and display it on screen.

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
 {
char str[25] = "Welcome to poly",ch;
clrscr();
ofstream fout;
fout.open("output.txt");
fout<<str;
fout.close();
ifstream fin;
fin.open("output.txt");
while (!fin.eof())
 {
 fin.getline(str, 25);
 cout<<str<<endl;
 }
fin.close();
getch();
 }
```

Write a C++ program to append data from abc.txt to xyz.txt file.

Assuming input file as abc.txt with contents "World" and output file named as xyz.txt with contents "Hello" have been already created.

```
#include <iostream.h>
#include<fstream.h>
int main()
{
fstream f;
ifstream fin;
fin.open("abc.txt",ios::in);
ofstream fout;
fout.open("xyz.txt", ios::app);
if (!fin)
 {
 cout<< "file not found";
 }
 else
 {
```

```
 fout<<fin.rdbuf();
 }
 char ch;
 f.seekg(0);
 while (f)
 {
 f.get(ch);
 cout<< ch;
 }
 f.close();
 return 0;
}
```

Output:

```
Hello World
```

Write a program to count the number of lines in file.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
ifstream file;
char ch;
int n=0;
clrscr();
file.open("abc.txt");
while(file)
{
file.get(ch);
if(ch=='\n')
n++;
}
cout<<"\n Number of lines in a file are:"<<n;
file.close();
getch();
}
```