Working with Files

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

- Sometimes we have large volume of data to store. In such situation we use some devices such as floppy disk or hard disk to store the data. The data store in these devices using the concept of files.

- A file is a collection of related data which is stored in a particular area on the disk. In this data stored permanently .

- To read and write operations performed on files using fstream.h

- At lowest level , file is interpreated as a stream of bytes.

What is **Stream :**

A streams acts as an interface between the programs and the files.

The stream that supplies data to the program is known as input stream , and the stream that receives data from the program is known as output stream.

In other words, the input stream extract (reads) data from the file and the putput stream inserts (or writes)data to the file.

Stream classes for file operations

Class	Contents
filebuf	Purpose to set file buffers to read & write. It contains open() and close() methods.
fstreambase	Ot serves as a base for fstream, ifstream, fstream class.
ifstream	Provides input operations. It contains get(), getline(), read() , seekg(), tellg() from istream.
ofstream	Provides outputoperations. It contains put(), write() , seekp(), tellp() from ostream.
fstream	Inherits all the functions from istream & ostream classes through iostream.

Types of Files in C++

There are two types of files in c++

1) Text Files 2) Binary Files

Text Files	Binary Files
Text files stores the data in ASCII information.	Contains information in same format in which information is stored in the memory.
Text files contains lines of text and each of the lines have an end-of-line marker appended automatically.	Contains text but they cannot be broken down into a number of lines.

Note : binary files are faster and easier for program to read & write than text files.

Opening a File :

A file can be opened in two ways:

1) Using the constructor of the stream class.

2) using the member function open() of the class.

- First way is useful when we use only one file in the stream.

- Second way is useful when we want to manage multiple files using one stream.

Closing a File

- When input and output stream objects go out of scope, connections with files are closed automatically.

- A file can be closed explicitly, using member function close() function.

Opening File using Constructor

In this we can open a file by supplying the file name to the constructor of file stream.

Program

```
#include<fstream.h>
Void main()
{
Ofstream fout(“Myfile.txt”);
Char *marks;
Cout<<enter the value\n”;
Cin>>marks;
Fout<<marks;
Cout<<“Data has been stored in the file\n”;
Fout.close();
Getch();
}
```

Output

Enter the value :

23

Data has been stored in the file.

Example

```
int main()
{
char name[30];
ofstream o = (“item.txt”);
cout<<“enter the item\n”;
cin>>name;
o<<name;
o.close();

ifstream i= (“item.txt”);
i>>name;
cout<<“item :”<<name<<“\n”;
i.close();
}
```

Output

Enter the item

Pen

Item:Pen

Multiple files open using open() function :

As we already stated, the function open() can be used to open multiple files that using same stream .

For ex: If we want to process a set of files sequentially then we create a single stream object and use it to open each file in turn.

Syntax: file_stream_class stream_object

stream _object . open (“ file_name ”);

General form of open() function is:

Stream_object . Open (" file_name " , mode);

Where second argument mode called file mode parameter , which specifies the purpose for which the file is opened.

Where ios::in (Open for reading only) and

ios::out (Open for writing only) are default mode.

The other modes describe in following table:

Example:

```
ofstream of;
of.open("Hello");   //    Connect to Hello File
of.close();
…….
…….
of.open("Hi");   //    Connect to Hi File
of.close();
…….
…….
of.open("Bye");   //    Connect to Bye File
of.clsoe();
```

Program

```
#include<fstream.h>
Void main()
{
Ofstream o;
o.open("country.txt");
O<<"Hello\n";
O<<"hi";
o.close();

o.open("capital.txt");
O<<"Hi\n";
O<<"Bye";
o.close();
Char line[n];
Ifstream I;
i.open("country.txt");
Cout<<"Contents are\n");
While(i)
{
i.getline(line, n);
Cout<<line;
}
i.close();
i.open("capital.txt");
Cout<<"Contents are\n");

While(i)
{
i.getline(line, n);
Cout<<line;
}
i.close();
Getch();
}
```

Output

Contents are:
Hello
Hi
Contents are:
Hi
Bye

File Modes

File Mode Parameters

Parameter	Meaning
Ios:app	Append to-end-of-file
Ios::ate	Go to end-of-file on opening
Ios::binary	Binary File
Ios::in	Open file for reading only
Ios::nocreate	Open fails if the file does not exist
Ios:noreplace	Open files if the file already exists
Ios::out	Open file for writing only
Ios:;trunc	Delete the contents of the file if it exists

Detecting end-of-file :

C++ provides a special function, **eof( )**, that returns nonzero (meaning TRUE) when there are no more data to be read from an input file stream, and zero (meaning FALSE) otherwise. eof() is a member function of ios class. Detecting end-of-file condition is necessary for preventing any further attempt to read data from the file.

Syntax: int eof();

It returns non zero when the end of file reached , otherwise it returns zero.

We use following statement in program to detect end-of-file condition

while(fin)

Example

```
int main()
{
  FILE *fp = fopen("test.txt", "r");
  int ch = getc(fp);
  while (ch != EOF)
  {
     putchar(ch);                          /* display contents of file on screen */
      ch = getc(fp);
  }
  if (ch.eof(fp))
    printf("\n End of file reached.");
  else
    printf("\n Something went wrong.");
  fclose(fp);
  getchar();
  return 0;
}
```

File Pointers in C++

Each file has two associated pointers known as the file pointers.

1) Input pointer (get pointer)
2) Output pointer (put pointer)

We can use these pointers to move through the files while reading or writing.

The input pointer is used for reading the contents of the given file location.

The output pointer is used for writing to a given fiel location.

Default Actions:

When we open a file in read-only mode, the input pointer is automatically set at the beginning so that user can read from the start.

Similarly when we open a file in write-only mode, the existing contents are deleted and the output pointer is set at the beginning. So that user can write from the beginning.

If user want to add more data to an existing file , the file is opened in append mode. This moves the output pointer to the end of the file

Functions for manipulate File Pointer:

The function for random access the data in the file are :

Function	Description
Seekg()	Moves get pointer (input) to a specified location.
Seekp()	Moves put pointer (output) to a specified location.
Tellg()	Gives the current position of the get pointer.
Tellp()	Gives the current position of the put pointer.

Seekg() & Seekp() :

As we know seekg() is used to move a file pointer to desired location.

seekg() and seekp() can also be used with two arguments as follows:

Seekg(offset , refposition);

Seekp(offset , refposition);

Here offset represent the no of bytes the file pointer to be moved from the location specified by the parameter refposition.

The refposition takes one of the three constants defined in the ios class (ios :: beg , ios :: cur , ios :: end).

Pointer offset calls :

The function for random access the data in the file are :

Seek call	Action
Seekg(0, ios :: beg)	Go to start.
Seekg(0, ios :: cur)	Stay at the current position.
Seekg(0, ios :: end)	Go to the end of Line.
Seekg(m, ios :: beg)	Move to (m+1)th byte in the file.
Seekg(m, ios :: cur)	Go forward by m byte from the current position.
Seekg(-m, ios :: cur)	Go backward by m byte from the current position.
Seekg(-m, ios :: end)	Go backward by m byte from the end.

Example:

- fin.seekg(30); // will move the get_pointer (in ifstream) to byte number 30 in the file

- fout.seekp(30); // will move the put_pointer (in ofstream) to byte number 30 in the file

When seekg() or seekp() function is used according to Form 2, then it moves the get_pointer or put_pointer to a position relative to the current position, following the definition of seek_dir. Since, seek_dir is an enumeration defined in the header file iostream.h, that has the following values:

- ios::beg, // refers to the beginning of the file ios::cur, // refers to the current position in the file
- ios::end} // refers to the end of the file

fin.seekg(30, ios::beg); // go to byte no. 30 from beginning of file linked with fin
fin.seekg(-2, ios::cur); // back up 2 bytes from the current position of get pointer
fin.seekg(0, ios::end); // go to the end of the file
fin.seekg(-4, ios::end); // backup 4 bytes from the end of the file

Sequential Input and Output Operations

The file stream classes support a number of member function to perform Input and Output operations on files.

1) put() and get() functions handle single character at a ime.
2) write() and read() function read and write blocks of binary data.

Write() and read () Functions :

Write () and Read() functions handle the data in binary form. This means that the values are stored in the disk file in the same format in which they are stored in the internal memory.

Following figure shows how an int value 2594 is stored in the binary and character formats.

Dig on C++ book

An int takes 2 bytes to store its value in the binary form, irrespective of its size. But a 4digit int will take 4 bytes to store it in character form

Advantage of Binary Files :

The binary format is more accurate for storing the numbers as they are stored in the exact internal representation. There are no conversions while saving the data and therefore saving is much fater.

The binary input & output functions takes following form:

```
Infile.read ((char * ) & V , sizeof(V));
Infile.write ((char * ) & V , sizeof(V));
```

Here first argument is the address of Variable V & second argument is the length of that variable in bytes.

The address of that variable must be cast to type char* (i.e.pointr to charactre type)

Writing to Binary File

```
struct Person
 {
   char name[50];
   int age;
 char phone[24];
 };

int main()
 {
Person me = {"Robert", 28, "364-2534"};
Person book[30];
 int x = 123;
 double fx = 34.54;

ofstream outfile;   outfile.open("junk.dat", ios::binary | ios::out);
  outfile.write(&x, sizeof(int)); // sizeof can take a type
outfile.write(&fx, sizeof(fx)); // or it can take a variable name
outfile.write(&me, sizeof(me));
outfile.write(book, 30*sizeof(Person))   ;
outfile.close();
}
```

Reading from Binary File

```
int main ()
 {
Int size;
char * memblock;
ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
if (file.is_open())
{
 size = file.tellg();
 memblock = new char [size];
 file.seekg (0, ios::beg);
 file.read (memblock, size);
file.close();
cout << "the entire file content is in memory";
delete[] memblock;
}
 else
 cout << "Unable to open file";
return 0;
}
```

OutPut:

"the entire file content is in memory