What you see is all you get.

Brian Kernighan

7.1 Introduction

Stream refers to a flow of data. It is intended to encapsulate the problems of moving the data to and from the disk or the screen. If the destination of the stream is a file or the screen, the source is usually some part of the program. If the stream is reversed, the data can come from the keyboard or a disk file and be assigned to data variables. Different streams are used to represent different kinds of data. Each stream is associated with a particular class, which contains member functions and definitions for dealing with that particular kind of data.

Writing data to the disk is very expensive in terms of performance. Writing data to the disk or reading from the disk takes a long time, and execution of the program is generally blocked by disk writes and reads. This problem can be solved using the buffering mechanism. Data is written into the stream, but it is not immediately written back to the disk. Instead, it's written to the stream's buffer, when it's full, the buffer writes to the disk all at once.

The stream classes are arranged in a rather complex hierarchy. Refer to Fig. 7.1.

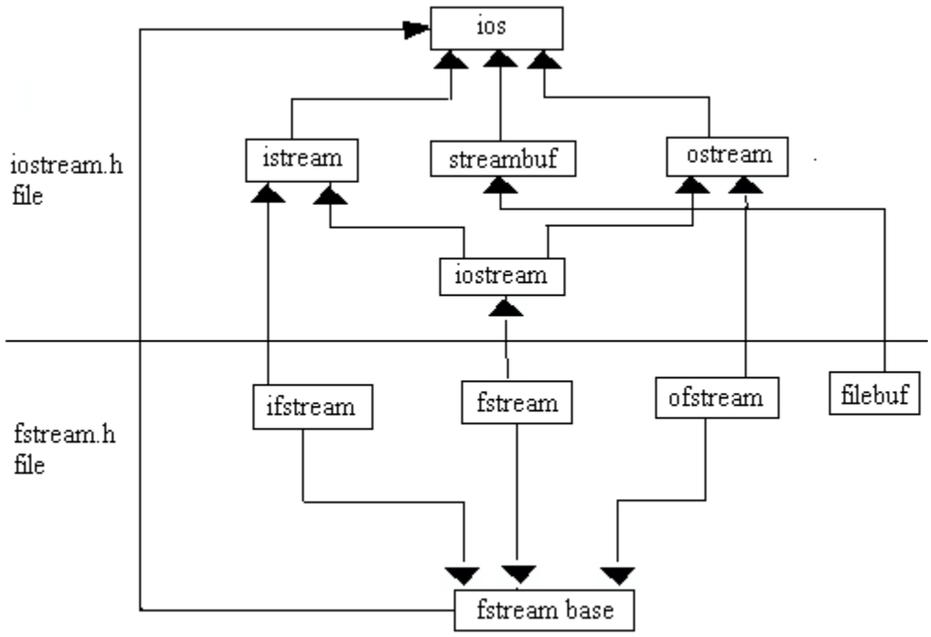

Fig. 7.1 Stream Hierarchy

The streambuf class manages the buffer, and its member functions make it possible to fill, empty, flush, and otherwise manipulate the buffer.

The ios class is the base class to the input and output stream classes. The ios class has a streambuf object as a data member. It also contains many constants and member functions common to input and output operations of all kinds.

The istream and ostream classes are derived from ios class. The istream class contains functions such as get(), getline(), read() and overloaded extraction (>>) operators, while ostream contains put(), write() and overloaded insertion operator (<<). The iostream class is derived from both istream and ostream by multiple inheritance. The fstream classes provide input and output from files.

7.2 File input/output

The ifstream class is used for input files, ofstream class for output files, and fstream class for both input and output.

7.1 Code Listing

// Program to write data to a file

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

void main()
{
   ofstream outfile("test.txt");
   outfile << "Testing";
   outfile << "writing data to a file in C++";
}
```

An object, outfile of class ofstream is created, which initializes the file name to test.txt. This initialization sets aside various resources for the file, and accesses the file of that name on the disk. We output the text of the file as the insertion operator << is overloaded in ostream. When the program terminates, the output object goes out of scope. This calls the destructor, which closes the file, so we don't need to close the file explicitly.

7.2 Code Listing

// Program to read data from a file

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

void main()
{
   const int MAX=80;
   char buffer[MAX];
   ifstream infile("test.txt");

   while(!infile.eof())
   {
      infile.getline(buffer, MAX);
      cout << buffer;
   }
}
```

An object, infile of class ifstream is created, which initializes the file name to test.txt. This initialization sets aside various resources for the file, and accesses the file of that name on the disk.

We read the text from the file one line at a time using the getline() function, which is a member function of istream. This function reads characters until it encounters the '\n' character, and places the resulting string in the buffer supplied as an argument. The maximum size of the buffer is given as the second argument. The contents of the buffer are displayed on the screen a line at a time.
An ifstream object, such as infile in this program, has a value that can be tested for various error conditions. If a condition is true, the object returns a zero value, otherwise it's nonzero. One of these conditions is the end of file (EOF). The EOF is a signal sent to the program from the hardware when a read or write operation has reached the end of file.

The above while condition could have also been written as
while(infile), instead of while(!infile.eof())

An ifstream object, such as infile, returns a value of 0 if any error occurs in the file operation including the end of file condition.

Note

Modify the above program to accept file name as command line argument.

7.3 Code Listing

/* This program writes a single character at a time using put(). An ofstream object is created. The length of the string is found using strlen(), and the characters are output using put() in a loop */

```
# include <string.h>
# include <fstream>
using namespace std;

void main()
{
   char str[] = "writing character at a time";

   ofstream outfile("test.txt");

   for(int j = 0; j < strlen(str); j++)
      outfile.put(str[j]);
}
```

7.4 Code Listing

/* This program reads a single character at a time using get() using an ifstream object. The program reads a single character using get(), and continues reading until the EOF is reached. Each character read is displayed using cout */

```
# include <fstream>
```

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

void main()
{
   char ch;
   ifstream infile("test.txt");

   while(!infile.eof())
   {
      infile.get(ch);
      cout << ch;
   }
}
```

7.5 Code Listing

/* This program opens a file and writes as many objects as the user wants. Then it reads and displays the entire contents of the file. The open() call includes several mode bits to specify certain aspects of the file object we open. The app (APPend) bit is used to preserve whatever was in the file before, in and out is used to perform both input and output on the file. Using the write() call one person object is written at a time to the file and is read back using the read() call. Before the reading of the file is done seekg() call is used to set the file get pointer (input) to the beginning. The seekg() is used in two ways. In the first case the single argument represents the position. If the two arguments are specified the first argument represents an offset from a particular location in the file and the second specifies the location from which the offset is measured. There are three possibilities for the second argument: beg is the beginning of the file (seekg(0, ios::beg)), cur is the current position (seekg(0, ios::cur)), and end is the end of the file seekg(0, ios::end)). The tellg() function returns the current position of the get pointer. This function can be used to return the pointer at the end of the file, i.e. the length of the file in bytes.

The other functions that manipulate File pointers are

seekp()	Moves put pointer (output) to a specified location.
tellp()	Gives the current position of the put pointer.

The different file modes are

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 fails if the file already exists
ios :: out	Open file for writing only
ios :: trunc	Delete contents of the file if it already exists */

```
# include <fstream>
# include <iostream>
# include <iomanip>     // setw, setiosflags functions

class person
{
   protected:
      char name[20];
```

```
      int age;
   public:
      void getdata(void)
      {
         cout << "\n enter name:";
         cin >> name;
         cout << "\n enter age:";
         cin >> age;
      }

      void showdata(void)
      {
         cout << "name:" << setw(20) << name << endl;
         cout << "age:" << setw(4) << age << endl;
      }
};

void main()
{
   char ch;
   person pers;
   fstream file;
   file.open("person.dat", ios::app | ios::out | ios::in);

   do
   {
      cout << "\n enter person's data:";
      pers.getdata();
      file.write((char *)&pers, sizeof(pers));
      cout << "Enter another person(y/n)?";
      cin >> ch;
   } while(ch == 'y');

   file.seekg(0);
   file.read((char *)&pers, sizeof(pers));

   while(!file.eof())
   {
      cout << "\nperson:";
      pers.showdata();
      file.read((char *)&pers, sizeof(pers));
   }
}
```

7.3 Manipulators

The flush manipulator flushes the buffer and the endl manipulator flushes the buffer and inserts a new line. Manipulators allow us to control the number base used to display integers. By using the ios member functions we can control the field width, and the number of places to the right of the decimal. We can also use the dec, hex and oct manipulators to display the integer data in base ten, base sixteen and base eight. To set a number to display in hex format we use cout << hex.

The width member function is used to place differently sized numbers in fields having equal width.

int width();

int width(int i);

The first form returns the current setting for field width and the second form sets the field width to i spaces and returns the previous field width space. The width method affects only the next item displayed. We use the width function as follows

cout.width(8);

The fill() function is used to fill unused parts of a field with spaces. The new fill character stays in effect until you change it. We use the fill function as follows

cout.fill('*');

Precision means the maximum number of digits displayed to the right of the decimal point. The default for C++ is four. The precision() function allows you to select other values. The precision setting stays in effect until reset.

cout.precision(2);

7.6 Code Listing

```
# include <fstream.h>

void main()
{
   int num = 123;

   cout << hex << num << endl;
   cout << oct << num << endl;
   cout << dec << num << endl;

   float f = 23.456789;
   cout << f << endl;

   double d = 12.9876543;
   cout << d << endl;

   cout.precision(10);
   cout << f << endl;
   cout << d << endl;

   int i = 20, j = 30;

   cout.width(15);
   cout.fill('#');
   cout << i;

   cout.width(8);
   cout.fill('*');
   cout << j ;
}
```

7.4 Exercise

Theory

1. What role does the iostream.h play in C++ I/O?

2. How is the file handling of C++ different than C?
3. What's the difference between standard output and the standard error?
4. How are manipulators used in data output formats?
5. How is multiple inheritance implemented in stream class hierarchy?
6. How is the data written in binary format? How do you read it back?
7. What does seekg & tellg mean in C++ file I/O?
8. How do you know when to use the insertion and extraction operators and when to use the other function members of the stream classes?
9. What is the difference between cerr and clog?
10. Why is the advantage of streams package over stdio package of C.
11. When would you use ignore()?
12. What is the insertion operator and what does it do?
13. What is extraction operator and what does it do?
14. What are the three forms of cin.get() and cin.getline()?
15. What is the return value of insertion operator?
16. What does ios :: ate argument do?
17. Describe the various classes available for file operations?
18. What is a file mode? Describe the various file mode options available?
19. What does the "current position" mean when applied to files?
20. What are the advantages of saving the data in binary form?

Problems

1. Write a program to copy data from one file to another using command line argument.

2. Write a program that displays its command line arguments in reverse order (do not display the program name).

3. Write a program that counts the numbers of lines, words and characters in the input file (similar to Unix word count program).

4. Write a program that reads data from 2 files and appends it to the 3rd file when the ID's match. The 1st file has name and ID as data and the 2nd file has ID and marks as data.

File 1		File 2	
Ramesh	50	150	90
Ram	100	50	92
Nagaraj	150	100	93

Data of 3rd file should be as follows

Ramesh	50	92
Ram	100	93
Nagaraj	150	90

5. Write a program that reads a text file and creates another file that is identical except that every sequence of consecutive blank spaces is replaced by a single space

6. A file contains a list of telephone numbers in the following form

   Ravi 1234567 Hyderabad
   Kavita 7654321 Bombay

   Write a program that creates a datafile containing a list of telephone numbers as given above. The name field contains only one word. The field separator is white space. Write a program to read the file and output the data in the order of name, address and phone number, sorted by name in the ascending order.

7. Write an interactive menu driven program that reads the data from file created in problem 6 to implement the following tasks.

   List the telephone numbers, given a name
   Determine the name if a telephone number is known
   Update the telephone number, whenever there is a change