Working with Input, Output and Files

12

12.1. INTRODUCTION

We have learnt most of the C++ concepts like classes, objects, member functions, concept of inheritance, virtual functions etc. in the previous chapters. In C++, the input and output are not built into the language, i.e., the input and output are not part of the C++ language syntax. The input and output in C++ is built as a separate package (libraries) which are made using C++ concepts. This standard library package is a mandatory component of C++ standard library. This library consists of a set of classes that are defined in "fstream.h" and "iostream.h" header files. Let us now see how to make use of the standard libraries and perform input and output operations in C++ in the following sections.

12.2. STREAMS IN C++

We give input to the executing program and the executing program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing program are called streams. In simple words, streams are nothing but flow of data in a sequence. The following figure helps the reader visualize the concept of streams.

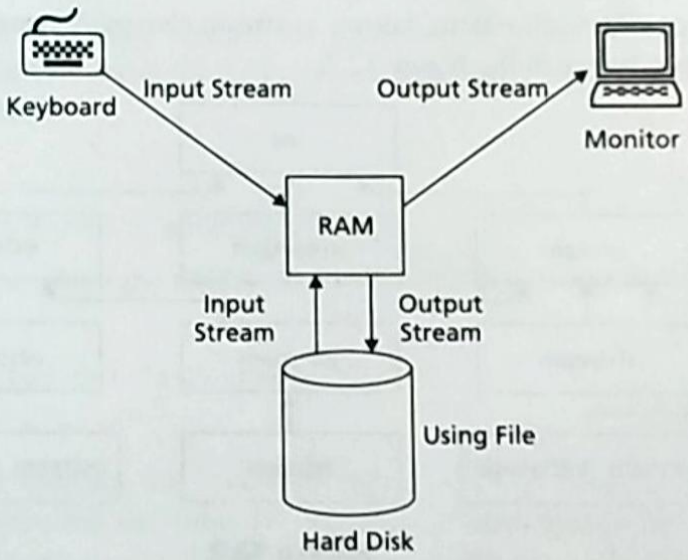


Figure 12.1

As observed from the Figure 12.1, the input to the executing program can either come from a keyboard or from any file stored in the secondary storage device like hard disk drive. Similarly, the output from the executing program can either go to the monitor or to any file stored in the secondary storage device like the hard disk drive. The input and output operation between the executing program and devices like keyboard and monitor are known as *"console i/o operations"*. The input and output operations between executing program and files are known as *"disk i/o operations"*. But it is important to note that, a C++ program handles the streams in the same way for both console i/o operations and disk i/o operations. This is because, almost all operating systems interpret hardware devices as files. For example, in UNIX, the 'stdin' is a file representing the keyboard with a file descriptor value of 0, 'stdout' is a file representing the monitor or screen with a file descriptor value of 1 and 'stderr' is a file representing the monitor or screen with a file descriptor value of 2. Therefore, when a C++ program begins its execution, all these three files are opened automatically by the operating system and is made readily available for the user to perform the input and output operations. In C++, these files are represented by class objects 'cin' for keyboard, 'cout' for monitor or screen and 'cerr' for monitor or screen(for displaying error messages).

12.3. STREAM CLASS MODEL OF C++

Recall, what we learnt in the introduction part of this chapter. That is, we had learnt that, the input and output in C++ are not a built-in language features rather they are made up of several classes packed together in a hierarchical order to make a separate library package. These classes are known as stream classes. A visualized picture of these classes is shown below in the Figure 12.2.

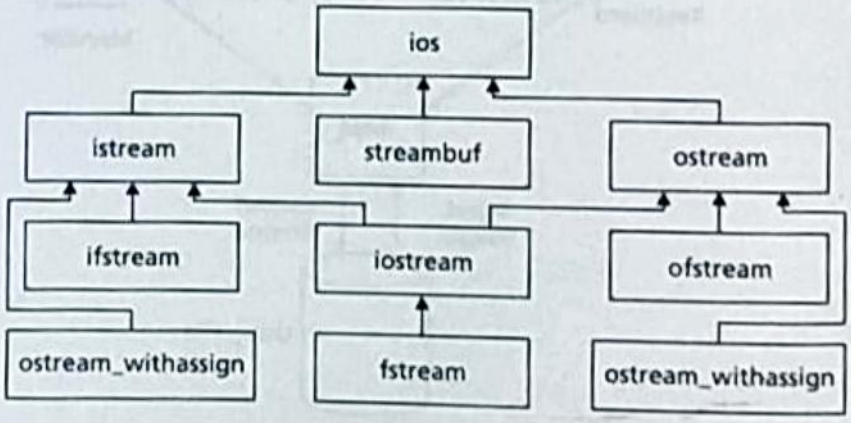


Figure 12.2

Let us discuss about each classes shown in the Figure 12.2 in detail.

ios :

- This class is the base class for other classes in this class hierarchy.
- This class contains the necessary facilities that are used by all the other derived classes for input and output operations.

istream :

- This class is derived from the class 'ios'.
- This class handles input streams.
- The extraction operator (>>) is overloaded in this class to handle input streams from files to the program in execution.
- Ex: `cin >> a;`
- This class declares input functions such as 'read()', 'getline()' and get().

istream_withassign :

- This class is derived from the class 'istream'.
- '**cin**' is an object of this class which represents the standard input file (keyboard). Recall that even keyboard is treated as a file. Hence, '**cin**' is an object that represents the standard input file.
- Ex: `cin >> a;`

This statement says that, "extract the input stream from the object '**cin**' which represents a standard input file (i.e.keyboard) and store it in the variable 'a' using the extraction operator.

ifstream :

- This class is derived from the class 'istream'.
- This class handles input streams *from the files stored in the hard disk drive to the program variable.*
- Ex:
```
ifstream fin( "subu.txt") ;
    fin >> a ;
```

The statement 'ifstream fin("subu.txt");' in the above example creates an object named 'fin', opens the file 'subu.txt' and associates the opened file 'subu.txt' to the object 'fin'. All these are done by the constructor of the class 'ifstream'. The statement 'fin >> a;' says that, "extract the input stream(data) from the object 'fin', representing

the file 'subut.txt' and store in the variable 'a' using the extraction operator. Note that, the object 'fin' can be of any name like 'f_in', 'ifile' and so on.

ostream :

- This class is derived from the class 'ios'.
- This class handles the output streams.
- The insertion operator (<<) is overloaded in this class to handle output streams to files from the program in execution.
- Ex: `cout << a;`
- This class declares output functions 'put( )' and 'write( )'.

ostream_withassign :

- This class is derived from class 'ostream'
- **'cout'** is an object of this class representing the standard output file(monitor). Recall that monitor is treated as a file. Hence 'cout' is an object that represents the standard output file.
- Ex : `cout << a;`

This statement says that, "insert the output stream from the variable 'a' to the object 'cout' which represents the standard output file (monitor) using the insertion operator (<<).

ofstream :

- This class is derived from the class 'ostream'.
- This class handles output streams *from program variable to the files in the hard disk drive.*
- Ex :
```
ofstream fout( "subu.txt" );
    fout << a;
```

The statement 'ofstream fout("subu.txt");' creates an object 'fout', opens the file 'subu.txt' and associates the file 'subu.txt' with the object 'fout'. All these are done by the constructor of the 'ofstream' class. The statement 'fout << a;' says that "insert the output stream into the object 'fout', representing the file 'subu.txt', from the program variable 'a' using the insertion operator(<<). Note that, the object 'fout' can be of any name like, 'f_out', 'ofile' and so on.

iostream :

- This class is derived from the classes 'ostream' and 'istream' through multiple inheritance.
- This class contains all the input and output functions.

fstream :

- This is derived from the class 'iostream'.
- This class handles input and output streams *from and to files stored in the hard disk drive.*
- Ex:
```
fstream f_in_out( "subu.txt" );
f_in_out >> a;
f_in_out << b;
```

It can be easily understood from the above statements that, 'f_in_out' is the object representing 'subu.txt' for handling both input and output streams. The association of 'f_in_out' to the file 'subu.txt' is done by the constructor of the class 'fstream'.

streambuf :

- This class contains a pointer which points to a buffer which is used to manage the input and output streams.

The header file 'iostream.h' has to be included while performing 'console i/o operations' and the header file 'fstream.h' has to be included while performing 'disk i/o operations.'

12.4. TEXT FILES AND BINARY FILES

When we store data in a file, it can be stored in either text mode or binary form. Before getting to know the difference between the storage of data in text form and binary form in a file, we should know, how exactly the data is stored in the memory during execution time. We know that, there are two forms of data. They are

- Character data
- Numerical data

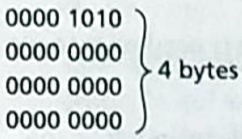
Consider the following program 12.1,

Program 12.1

```
/* 12.1.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char c = 'A';
   int a = 10;
   return 0;
}
```

When the program 12.1 is getting executed, the character 'A' in the variable 'c' is represented as the binary equivalent (base 2) of its ASCII equivalent in the memory. It occupies one byte of memory location. The ASCII value of 'A' is 65, and the binary equivalent of 65 is 01000001.

Now, let us see how the integer data is stored in the memory. During the program execution, the integer type value 10 is represented in its binary format. It occupies four bytes of memory location. The binary equivalent of the number 10 in the memory during execution looks as shown below.



Similarly, if a double value was there in the program, it would have taken 8 bytes of memory.

Now, let us see what is a Text file and a Binary file.

Binary Files :

A file in which the data is being input/output in binary mode is called Binary file. This means that, the form of data present in the binary file is equal to the form of the same data present in the memory during execution. For example, if an integer value 22222, takes 4 byes in the memory, then the same value takes 4 bytes in the binary file also. The representation of value in both the file and memory is in base 2 format or binary format.

Text files :

A file in which the data is being input/output in text mode is called a Text file. This means that, if an integer value 22222 takes 4 bytes in memory (in binary format), then, while writing the same value to a text file, the base 2 format or binary format is converted

to base 10 format. Therefore in a text file, the value 22222 occupies 5 bytes. As another example, the integer value 22 occupies 4 bytes in memory as it is an integer, but when it is written into a text file, it occupies only 2 bytes.

It is to be noted that, in case of character data , there is no difference between binary files and text files. This is because, the character data occupies only one byte both in the file and in the memory during program execution. Let us understand the above discussion on text and binary mode input and output pictorially.

(i) Binary mode I/O :

- Integer value = 25

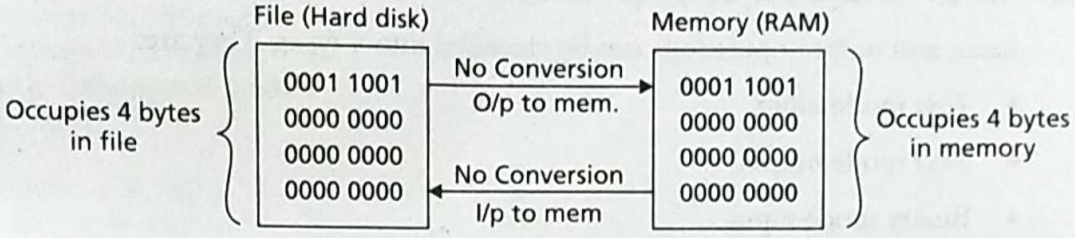


Figure 12.3

- Character value = 'A'

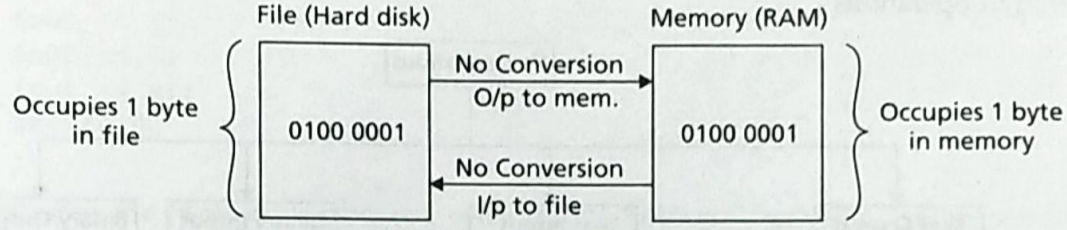


Figure 12.4

(ii) Text mode I/O:

- Integer value = 25

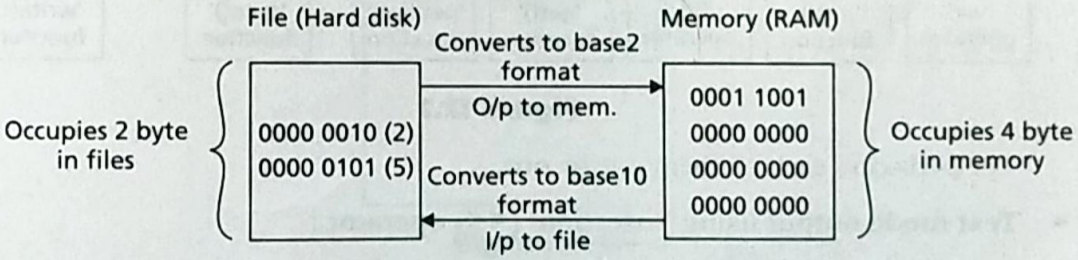


Figure 12.5

- Character value = 'A'

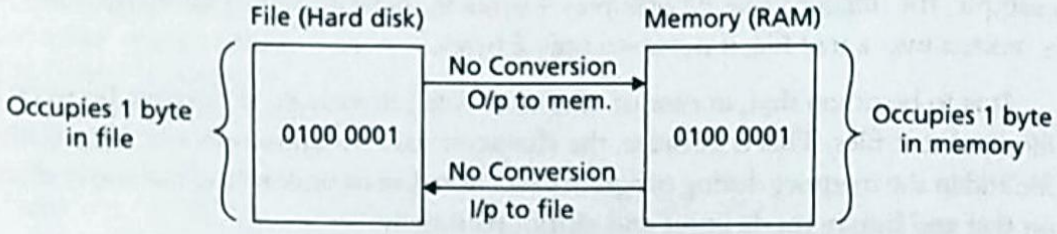


Figure 12.6

12.5. INPUT AND OUTPUT OPERATIONS

Input and output operations can be classified into 4 types. They are;

- Text mode input
- Text mode output
- Binary mode input
- Binary mode output

Look at the Figure 12.7 below to understand about the classification of input and output operations.

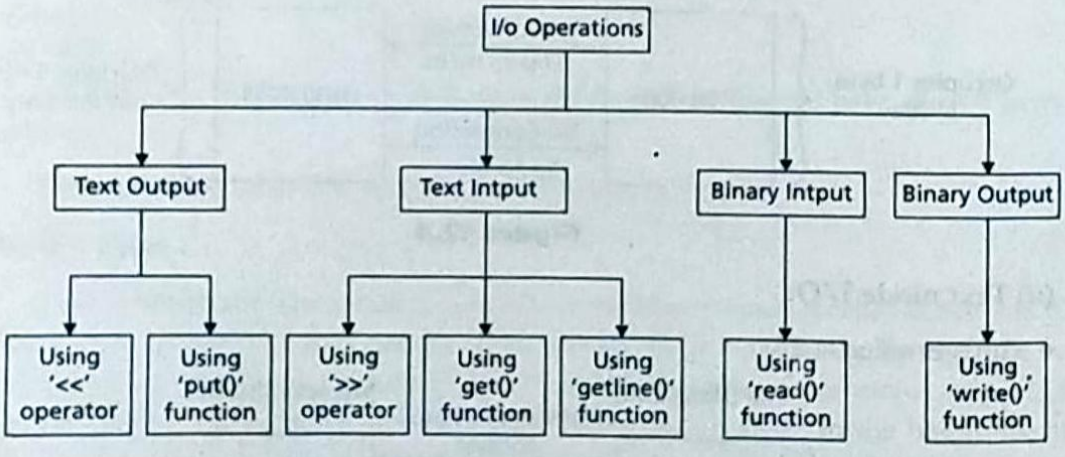


Figure 12.7

Let us discuss each of them one by one.

- **Text mode output using 'insertion' (<<) operator :**

'<<' operator outputs data into the file in text mode from a program variable i.e. from memory during execution. Using '<<' operator, we can output values of all built-in

data types to a specific file from the memory, where the program is executing. To output the objects of some user-defined class, we have to overload and define '<<' operator as done in the chapter 'operator overloading'. It is suggested to the reader to go through the chapter 'Operator Overloading' once to recall what we studied. When the data(except character data) is copied from memory to the file, the data is converted from base 2 format to decimal format. Let us understand these with the help of an example program 12.1. Consider a file 'subu.txt' being present in the hard disk drive. If not present, the below program will create it automatically.

Program 12.2

```
/* 12.2.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char c = 'A';
   int i = 25;
   double d = 5.5;
   char arr[50] = "Subhash";
   ofstream fout( "subu.txt");
   fout << c;
   fout << i;
   fout << d;
   fout << arr;
   return 0;
}
```

Now, after the program is executed, open the file 'subu.txt' manually, and you can find the values of the variables copied into that file. The pictorial representation of the file is as shown in the figure 12.8 below.

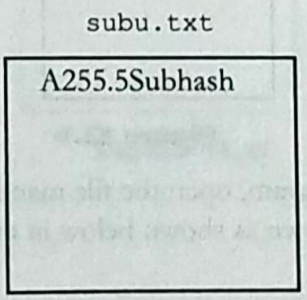


Figure 12.8

- **Text mode output using 'put( )' function :**

'put( )' is a member function of class 'ostream'. The prototype of 'put( )' function is as follows.

```
ostream & ostream :: put( char c );
```

The 'put( )' function has to be invoked by an object of class 'ostream' or object of its derived class like 'cout'. 'cout' is an object of class 'ostream_withassign'. 'ostream_withassign' is derived from ostream. The 'put( )' function is used to copy the character data passed to it as a parameter to the output file. The following program illustrates this.

Program 12.3

```
/* 12.3.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   cout.put( 'A' ); // copies 'A' to monitor
   ofstream ofile( "subu.txt" );
   ofile.put( 'A' ); //copies 'A' to 'subu.txt'
   return 0;
}
```

Assuming the file "subu.txt" was empty before executing the program 12.3, the file looks as shown in the Figure 12.9 below.

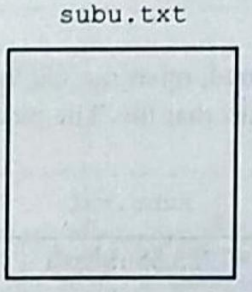


Figure 12.9

After executing the above program, open the file manually and we can see the character 'A' written in it. It is pictorially shown as shown below in the Figure 12.10.

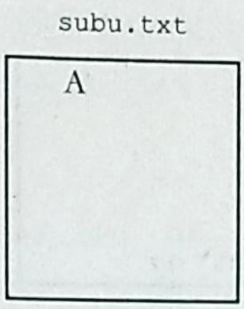


Figure 12.10

- **Text mode input using 'extraction' (>>) operator :**

'>>' operator inputs data into a program variable (into memory location) in text mode from a specific file present in the hard disk drive. Using '>>' operator, we can input values of all built-in data types to a specific location in the memory(program variable), where the program is executing. To input the objects of some user-defined class, we have to overload and define '>>' operator as done in the chapter 'Operator Overloading'. It is suggested to the reader to go through the chapter 'Operator Overloading' once to recall what we studied. When the data(except character data) is copied from file to the memory, the data is converted from base 10 format to base 2 format. Let us understand these with the help of an example program 12.1. Consider a file 'subu.txt' being present in the hard disk containing some data in it.

Before executing the below program 12.4, assume the file "subu.txt" to present in the hard disk drive as shown in the Figure 12.11.

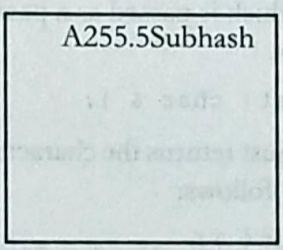


Figure 12.11

Program 12.4

```
/* 12.4.cpp */
#include <iostream>
#include <fstream>
```

```
using namespace std;
int main( )
   {
      char c;
      int i;
      double d;
      char arr[10];
      ifstream fin( "subu.txt" );
      fin >> c;
      fin >> i;
      fin >> d;
      fin >> arr;
      cout << c << endl;
      cout << i << endl;
      cout << d << endl;
      cout << arr << endl;
      return 0;
   }
```

Output:

```
A
25
5.5
Subhash
```

- **Text mode input using 'get( )' function :**

  The 'get()' function is a member function of class 'istream'. There are two versions of 'get( )' functions. Among them one is used to extract one character from a specified file to a character variable which is passed as a parameter to it. The prototype of this 'get( )' function is as follows:

  ```
  istream &istream :: get( char & );
  ```

  The other version of 'get()' just returns the character read from the file. The prototype of this 'get( )' function is as follows:

  ```
  istream &istream :: get( );
  ```

  Illustration of both the versions of 'get()' function is shown below. Before executing the programs 12.5 and 12.6, lets have a file named 'subu.txt' with a single character data 'A' written in it as shown in the below Figure 12.12.

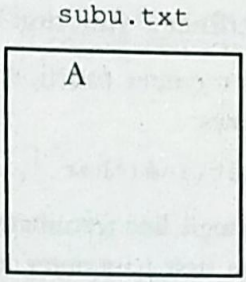


Figure 12.12

Program 12.5

```
/* 12.5.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char c;
   ifstream fin( "subu.txt" );
   fin.get( c ); // copies from file 'subu.txt' to variable 'c'
   cout << c;
   return 0;
}
```

Output:

```
A
```

Program 12.6

```
/* 12.6.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char c;
   ifstream fin( "subu.txt" );
   c = fin.get(  ); // copies from file 'subu.txt' to variable 'c'
   cout << c;
   return 0;
}
```

Output:

```
A
```

- **Text mode input using 'getline( )' function :**

  The 'getline( )' function is a member function of 'istream' class. The prototype of 'getline( )' function is as follows.

```
istream & istream :: getline(char *, int , char = '\n' );
```

The 'getline( )' function reads a single line terminated by a newline character ('\n') from a file into a string. The string is the first parameter to the 'getline( )' function. If certain characters of a line from a specified file have to be read then, no need of specifying the last parameter. The second parameter to the 'getline( )' function indicates the number of characters to be read. Let us understand this with an example.

Consider a file, 'subu.txt' with a line written in it terminated by a newline character as shown below.

```
Subhash loves  $ C++
```

The file 'subu.txt' pictorially looks like this.

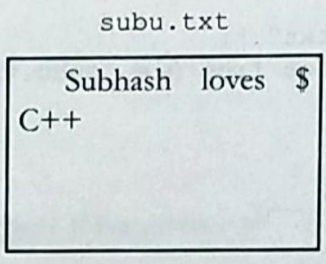


Figure 12.13

Now, consider the following program 12.7 and analyze the output.

Program 12.7

```
/* 12.7.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char s[50];
   ifstream fin( "subu.txt" );
   fin.getline( s, 25 );
   cout << s;
   return 0;
}
```

Output:

```
Subhash loves $ C++
```

We can also supply our own de-limiter instead of newline character as shown below.

Program 12.8

```
/* 12.8 */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   char s[50];
   ifstream fin ("subu.txt");
   fin.getline (s,50,'$');
   cout << s;
   return 0;
}
```

delimiter to quit.

Output:

```
Subhash loves
```

- **Binary output using 'write( )' function :**

  'write( )' function is the member function of 'ostream' class. The prototype of 'write( )' function is as shown below.

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

'write( )' function copies the value of the variable from an executing program into a specified file in binary mode. While copying the value from memory to a file, there is no conversion taking place. In other words, the data in binary format or base 2 format in the memory is copied exactly as it is on to the file in the hard disk drive. The first parameter of the 'write( )' function is the address of the variable whose value needs to be outputted. The second parameter is the size of the variable. Let us write a simple program to understand the working of 'write( )' function. Let us consider an empty file 'subu.dat' present in the hard disk drive. If it is not present, it will be automatically created by the below program 12.9.

Program 12.9

```
/* 12.9.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   ofstream fout( "subu.dat" );
```

```
    char c = 'A';
    int i = 25;
    double d = 5.56;

    fout.write( &c, sizeof(c ) ); // copies 'A' to subu.dat
    fout.write( (char *)&i, sizeof(i)); // copies 25 to subu.dat
    fout.write( (char *)&d, sizeof(d)); // copies 5.56 to subu.dat
    return 0;
}
```

Now, open the file 'subu.dat' and try reading. You can read only the character value, but ot the integer value and double value as it is represented in binary format. Please do not delete this file 'subu.dat' as it is required for the next program in the next section.

- **Binary input using 'read( )' function :**

  'read( )' function is a member function of 'istream' class. The prototype of 'read( )' function is as shown below.

  ```
  istream & istream :: read( char *, int );
  ```

The 'read( )' function reads the data from a specified file and stores it in the variable of an executing program. The first parameter of the 'read( )' function is the address of the variable into which the data has to be copied from the file. The following program 12.10 helps us to understand the concept better. The file 'subu.dat' obtained from the program 12.9 is opened and read in this program.

Program 12.10

```
/* 12.10.cpp */
int main( )
{
    char c;
    int i;
    ifstream fin( "subu.dat" );
    fin.read( &c, sizeof(c) );
    fin.read( (char *)&i , sizeof(i));
    fin.read( (char *)&d, sizeof(d));
    cout << c << "\n" << i << "\n" << d << "\n" << endl;
    return 0;
}
```

Output:

```
A
25
5.56
```

12.6. OPENING AND CLOSING FILES

Opening a disk file can be done in two ways. They are,

- Using constructors of the class 'ifstream' or 'ofstream' or 'fstream' as done in the previous example programs.
- Using the member functions 'open( )' of the class 'ifstream' or 'ofstream' or 'fstream' which will be discussed here.

Closing a disk file can be done

- Using 'close( )' member function of the classes mentioned above.

Using constructors :

Let us directly jump into an example program to understand this.

Program 12.11

```
/* 12.11.cpp */
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
   int price, cost;
   char product[30], name[30];
   cout << "Enter the product name" << endl;
   cin >> product;
   cout << "Enter the price" << endl;
   cin >> price;

   ofstream fout("product.txt" ); //using constructor opening a file
   fout << product;//copies the value of product to file 'product.txt'
   fout << endl;
   fout << price;//copies the value of price to file 'product.txt'
   fout.close( );

   ifstream fin( "product.txt" ); // using constructor opening a file
   fin >> name; //copies the first data in file "product.txt" to name
   fin >> cost; //copies the second data in file "product.txt" to cost
   cout << name << endl;
   cout << cost << endl;
   fin.close( );
   return 0;
}
```

Output:

```
Enter the product name
C++ Book ↵
Enter the price
350 ↵
C++ Book
350
```

In the above program, the statement,

```
ofstream fout( "product.txt" );
```

opens a file, 'product.txt' for writing data(writing only) into the file, using the constructor of class 'ofstream' and associates the opened file with the 'ofstream' class object 'fout'. After writing into the file, the file is closed using the member function, 'close( )' of the 'ofstream' class. Now, using the statement,

```
ifstream fin( "product.txt" );
```

the file is again opened for extracting the data from the file using the constructor of the class 'ifstream' and associates the opened file with the 'ifstream' class object 'fin'. After the extraction operation is over the file is closed using the member function 'close( )' of the 'ifstream' class.

Using 'open( )' function :

The general form of using the 'open( )' function is ,

```
stream-class object;
object.open( filename) ;
```

- Opening a file for inserting data into it can be done using the following statements:

```
ofstream  fout;
fout.open( "product.txt" );
```

- Opening a file for extracting data from it can be done using the following statements:

```
ifstream fin;
fin.open( "product.txt" );
```

Let us look into a program to understand this.

Program 12.12

```
/* 12.12.cpp */
#include <iostream>
#include <fstream>
```

```
using namespace std;
int main( )
{
   int price, cost;
   char product[30], name[30];
   cout << "Enter the product name" << endl;
   cin >> product;
   cout << "Enter the price" << endl;
   cin >> price;

   ofstream fout;
   fout.open( "product.txt" ); // using 'open( )' opening a file
   fout << product;//copies the value of product to file 'product.txt'
   fout << endl;
   fout << price;//copies the value of price to file 'product.txt'
   fout.close( );

   ifstream fin;
   fin.open( "product.txt" ); //using 'open( )' opening a file
   fin >> name; //copies the first data in file "product.txt" to name
   fin >> cost; //copies the second data in file "product.txt" to cost
   cout << name << endl;
   cout << cost << endl;
   fin.close( );
   return 0;
}
```

Output:

```
Enter the product name
C++ Book ↵
Enter the price
250 ↵
C++ Book
250
```

'open( )' Extended :

We learnt a simple version of 'open()' function in the previous section. The syntax of the 'open()' function can be further extended. In other words, to the 'open()' function a second parameter can be passed. The general form will look like this.

```
object.open( filename, open_mode );
```

The second parameter 'open_mode' is an integer type value which informs the program how to open the file. That is, whether to open a file for writing , or whether to open a file for reading etc. The second parameter is the integer type constant defined in the 'ios' class. Let us see few examples.