**C++ Streams**

Every input and output in C++ program is managed by **stream**. A stream is flow of data (bytes). An input stream (istream) manages bytes of data flowing into the program and an output stream (ostream) manages bytes of data flowing out of the program. Data is either extracted from a stream (input of data) or data is inserted into a stream (output of data). In a C++ program, keyboard is used to input data (extract data from stream) by using built-in stream class object cin (console input) and screen (monitor) is used to display data on the screen (insert data into a stream output of data) by using built-in stream class cout (console output). Objects cin and cout are already defined in the header file <iostream.h>.

Input and output of data when working with files is not so simple as console input / output. This is because files are stored in backing storage and there are various types of backing storage and each type stores (writes) and reads data in a different way. In C++, a data file is an external stream (a sequence of bytes stored in a backing storage). The stream classes that manage flow of data between **console** and **main** storage along with another specialized stream class are used for managing flow of data between **backing** storage and the **main** storage. Stream classes ofstream, ifstream and fstream are used to manage file input/output.

//Program Name: Console.cpp

#include <iostream.h>

**void** main()

{

**int** roll;

**char** name[20], sex;

**double** mark;

cout<<"Roll? "; cin>>roll;

cout<<"Name? "; cin>>name;

cout<<"Sex ? "; cin>>sex;

cout<<"Mark? "; cin>>mark;

cout<<roll<<','<<name<<',';

cout<<sex<<','<<mark<<endl;

}

//Program Name: TextFile.cpp

#include <fstream.h>

**void** main()

{

**int** roll;

**char** name[20], sex;

**double** mark;

ofstream fout("stu.txt");

cin>>roll>>name>>sex>>mark;

fout<<roll<<','<<name<<',';

fout<<sex<<','<<mark<<endl;

fout.close();

ifstream fin("stu.txt");

fin>>roll>>name;

fin>>sex>>mark;

cout<<roll<<','<<name<<',';

cout<<sex<<','<<mark<<endl;

fin.close();

}

MCj04040530000[1]

MCj03969440000[1]

**Keyboard**

cin

**Monitor**

cout

j0307609

j0307609Diagram is given below showing C++ stream class hierarchy:

**Data File**

**stu.txt**

ofstream fout

**Data File**

**stu.txt**

ifstream fin

**Header File <fstream.h>**

**Header File <iostream.h>**

fstreambase

ios

istream

ostream

ofstream

ifstream

fstream

iostream

Any program working with files in C++ needs the header file <fstream.h>. Brief description of header file <iostream.h> given below:

1. Starting point of the stream class hierarchy is the class ios.

2. Classes istream, ostream and fstreambase are derived from class ios.

3. Class iostream is derived from classes istream and ostream.

4. Class istream\_withassign is derived from class istream. Object cin is an instance of istream\_withassign (not shown in the diagram).

5. Class ostream\_withassign is derived from class ostream. Object cout is an instance of ostream\_withassign (not shown in the diagram).

Brief description of header file <fstream.h> is given below:

1. Header file <fstream.h> includes header file <iostream.h> since class fstreambase is derived from class ios and ios needs header file <iostream.h>.

2. Class ifstream is derived from classes istream and fstreambase. Class ifstream is used to read data from a file.

3. Class ofstream is derived from classes ostream and fstreambase. Class ofstream is used to write (store) into a file.

4. Class fstream is derived from classes ifstream and ofstream or from class iostream and fstreambase. Class fstream can be used either to read data from a file or to write data into a file or both.

As mentioned earlier, file input / output is more complicated compared to console input / output. The most important reason being, file stored in the backing storage is under the control of Operating System (OS). C++ programs neither write data directly into a file nor read data directly from a file. Therefore every transfer of data between main storage and backing storage takes place through buffer. **Buffer** is a temporary intermediate storage inside the main storage (RAM) used during file input / output. Buffer is required for the following reason:

An application program has no knowledge about the physical location of the file in the backing storage. Only Operating System (OS) knows everything about the backing storage and the physical location of the file.

A buffer acts as a temporary intermediary storage to transfer data to and fro between main storage and backing storage. A diagram is given below showing the role of a buffer during file input / output.

**Main Storage (RAM)**

**void** addtext()

{

ofstream fout("Fees.txt");

**int** roll;

**char** name[20];

**double** fees;

**for** (**int** k=0; k<16; k++)

{

cout<<"Roll?"; cin>>roll;

cout<<"Name?"; gets(name);

cout<<"Fees?"; cin>>fees;

fout<<roll<<" "<<name<<" "

<<fees<<endl;

}

fout.close();

}

**void** readtext()

{

ifstream fin("Fees.txt");

**char** str[80];

**while** (fin.getline(str,80))

cout<<str<<endl;

fin.close();

}

Or,

**void** readtext()

{

ifstream fin("Fees.txt");

**char** cv;

**while** (fin.get(cv))

cout<<cv;

fin.close();

}

**Buffer**

**Hard Disk**

**Backing Storage**

**Fees.txt**

**Transfer of data by OS between buffer & backing storage**

**10 ALOK JAIN 6000**

**11 AMITA VERMA 6000**

**12 BIMLA SHAH 6000**

**13 ERIKA JOHN 6000**

**14 KUNAL ROY 6500**

**15 MOHD QADIR 6000**

**16 NEETA GUPTA 5500**

**17 NITIN DEY 6500**

**18 NITU KUMAR 6500**

**19 RITA MATHEW 5500**

**20 RUBI KHAN 5500**

**21 SUMIT DUTT 5500**

**22 SUNILA DEV 6500**

**23 SURBHI JHA 6500**

**24 SUHAS BHAT 5500**

**25 TAPAN DAS 6000**

**Transfer of data by C++ Program between memory variable & buffer**

Transfer of data between **main** storage and the **backing** storage takes place in **two** steps:

1. Transfer of data between **memory** **variable** and the **buffer** is done by C++ program. Statement fout<<roll<<" "<<name<<" "<<fees<<endl; transfers data from **memory** **variables** (roll, name and marks) to the **buffer** through file variable fout. Function fin.getline(str,80) or function fin.get(cv) transfers data from **buffer** to **memory** **variable** (str or cv) through file variable fin.

2. Transfer of data between **buffer** and the **backing** storage is done by OS.

A file stored in backing storage is either stored in text format (Text file) or stored in binary format (Binary file). A file (Text or Binary file) must be opened before any file input / output. Data file stored in human readable format is called a **text** file where as data stored in machine readable form is called a binary file. Opening a binary file requires an extra parameter ios::binary along with the file name.

|  |  |
| --- | --- |
| **Text File** | **Binary File** |
| * Text file is stored in human readable form * Character translation takes place before data is written into a text file or read from a text file * Text file can be created in any Text Editor and read through a C++ program/function and vice-versa * Text file is 100% portable – text file created in Windows OS can be read by other OS and vice-versa | * Binary file is stored in machine readable form (binary format) * No character translation takes place before data is written into a binary file or read from a binary file * Binary file can be created and read through a C++ program/function * Binary file is not at all portable – binary file created in Windows OS cannot be read by other OS and vice-versa |

There are only two basic operations in a file – data is either written into a file or data is read from a file. **Writing** (**storing**) data in a file, the file must be opened in **output** (write) mode. Writing data in a file is **output** mode because data goes **out** of main storage (RAM) to backing storage. **Reading** (**retrieving**) date from a file, the file must be opened in **input** mode. **Reading** data from a file is **input** mode because data is **inputted** from backing storage to main storage (RAM).

* Opening a **text** file in **output** mode using member function open()

**Rule:** FileObject.open(TextFileName, Mode);

FileObject is an instance of either ofstream or fstream

Mode is either output or append

**1. Example**:

ofstream fout;

fout.open("Report.txt");

**2. Example**:

**char** filename[20];

gets(filename);

ofstream fout;

fout.open(filename);

**3. Example**:

fstream fout;

fout.open("Report.txt", ios::out);

**4. Example**:

**char** filename[20];

gets(filename);

fstream fout;

fout.open(filename, ios::out);

If fout is an instance of fstream then ios::out is mandatory.

* Opening a **text** file in **output** mode using **constructor** function

**Rule:** FileObject(TextFileName, Mode);

FileObject is an instance of either ofstream or fstream

Mode is either output or append

**1. Example**:

ofstream fout("Report.txt");

**2. Example**:

**char** filename[20];

gets(filename);

ofstream fout(filename);

**3. Example**:

fstream fout("Report.txt", ios::out);

**4. Example**:

**char** filename[20];

gets(filename);

fstream fout(filename, ios::out);

If fout is an instance of fstream then ios::out is mandatory.

* Opening a **binary** file in **output** mode using member function open()

**Rule:** FileObject.open(BinaryFileName, Mode);

FileObject is an instance of either ofstream or fstream

Mode is either output or append

**1. Example**:

ofstream fout;

fout.open("Emp.dat", ios::binary);

**2. Example**:

**char** filename[20];

gets(filename);

ofstream fout;

fout.open(filename, ios::binary);

**3. Example**:

fstream fout;

fout.open("Emp.dat", ios::binary|ios::out);

**4. Example**:

**char** filename[20];

gets(filename); fstream fout;

fout.open(filename, ios::binary|ios::out);

If fout is an instance of fstream then ios::out is mandatory.

* Opening a **binary** file in **output** mode using **constructor** function

**Rule:** FileObject(BinaryFileName, Mode);

FileObject is an instance of either ofstream or fstream

Mode is either output or append

**1. Example**:

ofstream fout("Emp.dat", ios::binary);

**2. Example**:

**char** filename[20];

gets(filename);

ofstream fout(filename, ios::binary);

**3. Example**:

fstream fout("Emp.dat", ios::binary|ios::out);

**4. Example**:

**char** filename[20];

gets(filename);

fstream fout(filename, ios::binary|ios::out);

If fout is an instance of fstream then ios::out is mandatory.

By convention a **text** file is given an extension **txt** while a **binary** file is given an extension **dat**. If ios::binary is omitted when opening a file, C++ program assumes that the file is a **text** file.

When a file is opened in **output** mode:

* If the file does not exist then a new file is created
* If the file exits then the content existing file is replaced by new set of input, that is, there is loss of data

Opening a **text** file in **input** mode using member function open()

**Rule:** FileObject.open(TextFileName, Mode);

FileObject is an instance of either ifstream or fstream

Mode is input

**1. Example**:

ifstream fin;

fin.open("Report.txt");

**2. Example**:

**char** filename[20];

gets(filename);

ifstream fin;

fin.open(filename);

**3. Example**:

fstream fin;

fin.open("Report.txt", ios::in);

**4. Example**:

**char** filename[20];

gets(filename); fstream fin;

fin.open(filename, ios::in);

If fin is an instance of fstream then ios::in is mandatory.

* Opening a **text** file in **input** mode using **constructor** function

**Rule:** FileObject(TextFileName, Mode);

FileObject is an instance of either ofstream or fstream

Mode is input

**1. Example**:

ifstream fin("Report.txt");

**2. Example**:

**char** filename[20];

gets(filename);

ifstream fin(filename);

**3. Example**:

fstream fin("Report.txt", ios::in);

**4. Example**:

**char** filename[20];

gets(filename);

fstream fin(filename, ios::in);

If fin is an instance of fstream then ios::in is mandatory.

* Opening a **binary** file in **input** mode using member function open()

**Rule:** FileObject.open(BinaryFileName, Mode);

FileObject is an instance of either ifstream or fstream

Mode is input

**1. Example**:

ifstream fin;

fin.open("Emp.dat", ios::binary);

**2. Example**:

**char** filename[20];

gets(filename); ifstream fin;

fin.open(filename, ios::binary);

**3. Example**:

fstream fin;

fin.open("Emp.dat", ios::binary|ios::in);

**4. Example**:

**char** filename[20];

gets(filename); fstream fin;

fin.open(filename, ios::binary|ios::in);

If fin is an instance of fstream then ios::in is mandatory.

* Opening a **binary** file in **input** mode using **constructor** function

**Rule:** FileObject(BinaryFileName, Mode);

FileObject is an instance of either ifstream or fstream

Mode is input

**1. Example**:

ifstream fin("Emp.dat", ios::binary);

**2. Example**:

**char** filename[20];

gets(filename);

ifstream fin(filename, ios::binary);

**3. Example**:

fstream fin("Emp.dat", ios::binary|ios::in);

**4. Example**:

**char** filename[20]; gets(filename);

fstream fin(filename, ios::binary|ios::in);

If fin is an instance of fstream then ios::in is mandatory.

open() Stream class member function of fstream, ifstream and ofstream, used to open a file in a particular mode.

**Header File:** fstream.h

**Syntax:** FileObject.open(**char**\* FileName, Mode);

FileObject is an instance of ofstream / ifstream / fstream

**Example 1:** ofstream fout;

fout.open("Emp.dat",ios::binary);

ifstream fin;

fin.open("Emp.dat",ios::binary);

fstream f;

f.open("Emp.dat",ios::binary|ios::out|ios::in);

**Example 2:** ofstream fout;

fout.open("Report.txt");

ifstream fin;

fin.open("Report.txt");

fstream f;

f.open("Report.txt",ios::out|ios::in);

close() Stream class member function of fstream, ifstream and ofstream used to close a file. Every file needs to be closed after being opened.

**Header File:** fstream.h

**Syntax:** FileObject.close();

FileObject is an instance of ofstream / ifstream / fstream

**Example 1:** ofstream fout;

fout.open("Emp.dat",ios::binary);

fout.close();

ifstream fin;

fin.open("Emp.dat",ios::binary);

fin.close();

fstream f;

f.open("Emp.dat",ios::binary|ios::out|ios::in);

f.close();

**Example 2:** ofstream fout;

fout.open("Report.txt");

fout.close();

ifstream fin;

fin.open("Report.txt");

fin.close();

fstream f;

f.open("Report.txt",ios::out|ios::in);

f.close();

>> A binary operator used to extract data from stream (used for input – cin>>VarName or FileObject>>VarName). Extraction operator (>>) is a member of istream / ifstream / fstream. To use >> with file, file must be opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject>>VarName;

FileObject is an instance of ifstream / fstream

VarName is variable of fundamental data type or string type

**Example 1:** Reads data from text File – Fees.txt

10 ALOK 6000

11 AMITA 6000

12 BIMLA 6000

**int** roll;

**char** name[20];

**double** fees;

ifstream fin("Fees.txt");

**while** (fin>>roll>>name>>fees)

cout<<roll<<','<<name<<','<<fees<<endl;

fin.close();

Running of the program segment produces following output:

10,ALOK,6000

11,AMITA,6000

12,BIMLA,6000

**Example 2:** Reads data from text File – Fees.txt

10 ALOK 6000

11 AMITA 6000

12 BIMLA 6000

ifstream fin("Fees.txt");

**char** ch;

**while** (fin>>ch)

cout<<ch;

fin.close();

Running of the program segment produces following output:

10ALOK600011AMITA600012BIMLA6000

<< A binary operator used to insert data into a stream (used for outputting data – cout<<VarName or FileObject<<VarName). Insertion operator (<<) is a member of ostream / ofstream / fstream. When using with a file, file must be opened in output mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject<<VarName;

FileObject is an instance of ofstream / fstream

VarName is variable of fundamental data type or string type

**Example: int** roll;

**char** name[20];

**double** fees;

ofstream fout("Fees.txt");

**for** (**int** x=0; x<5; x++)

{

cin>>roll>>name>>marks;

fout<<roll<<' '<<name<<' '<<fees<<endl;

}

fout.close();

**sizeof** A unary operator and a keyword (does not need any header file) used in C++, which returns number of bytes allocated to a particular variable / object or a data type.

**Syntax:** **int** **sizeof**(Data);

Data could be a variable / expression / constant or data type

**Example: int** roll;

**char** name[20];

**double** fees;

cout<<**sizeof**(roll)<<endl<<**sizeof**(name)<<endl;

cout<<**sizeof**(fees)<<endl<<**sizeof**(**int**)<<endl;

Running of the program segment produces following output:

4

20

8

4

read() Stream class member function of istream / ifstream / fstream, used to extract a number of bytes or characters from stream (usually from a file buffer). When using with a file, file must be opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.read((**char**\*)Address, **int** Size);

FileObject is an instance of ifstream / fstream

Address is the location of data and Size is the size of data in terms of bytes. Stream function read(), reads Size amount of bytes from file buffer and transfers to variable (Address – address of variable).

**Example 1:** Reads data from binary data file – Fees.dat

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

**struct** student

{

**int** roll;

**char** name[20];

**double** fees;

};

ifstream f("Fees.dat", ios::binary);

student a;

**while** (f.read((**char**\*)&a, **sizeof**(a)))

cout<<a.roll<<' '<<a.name<<' '<<a.fees<<endl;

f.close();

Running of the program segment produces following output:

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

**Example 2:** Reads data from binary data file – Fees.dat

**class** student

{

**int** roll; **char** name[20]; **double** fees;

**public**:

**void** inputdata()

{

cout<<"Roll?";cin>>roll;

cout<<"Name?";gets(name);

cout<<"Fees?";cin>>fees;

}

**void** showdata()

{

cout<<roll<<' '<<name<<' '<<fees<<endl;

}

};

ifstream f("Fees.dat", ios::binary);

student a;

**while** (f.read((**char**\*)&a, **sizeof**(a)))

a.showdata();

f.close();

write() Stream class member function of ostream / ofstream / fstream, used to insert a number of bytes or characters into a stream (usually writing into a file buffer). When using with a file, file must be opened in output mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.write((**char**\*)Address, **int** Size);

FileObject is an instance of ofstream / fstream

Address is the location of data and Size is the size of data in terms of bytes. Stream function write(), transfers (writes) Size amount of bytes from variable (Address – address of variable) into file buffer.

**Example 1:** Writes data into binary data file – Fees.dat

**struct** student

{

**int** roll; **char** name[20]; **double** fees;

};

student a;

ofstream f("Fees.dat", ios::binary);

**for**(**int** x=0; x<5; x++)

{

cout<<"Roll? "; cin>>a.roll;

cout<<"Name? "; gets(a.name);

cout<<"Fees? "; cin>>a.fees;

f.write((**char**\*)&a, **sizeof**(a));

}

f.close();

**Example 2:** Writes data into binary data file – Fees.dat

**class** student

{

**int** roll;

**char** name[20];

**double** fees;

**public**:

**void** inputdata()

{

cout<<"Roll? "; cin>>roll;

cout<<"Name? "; gets(name);

cout<<"Fees? "; cin>>fees;

}

**void** showdata()

{

cout<<roll<<' '<<name<<' '<<fees<<endl;

}

} a;

ofstream f("Fees.dat", ios::binary);

**for**(**int** x=0; x<5; x++)

{

a.inputdata();

f.write((**char**\*)&a, **sizeof**(a));

}

f.close();

eof() Stream class member function of istream / ifstream / fstream, returns a non-zero (true) value if **end** **of** **the** **file** (**EOF**) is encountered while reading from file; otherwise returns false (zero). When using with a file, file must be opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.eof();

FileObject is an instance of ifstream / fstream

**Example 1:** Reading data from binary file using stream class function eof()

**struct** student

{

**int** roll;

**char** name[20];

**double** fees;

};

ifstream f("Fees.dat", ios::binary);

student a;

f.read((**char**\*)&a, **sizeof**(a));

**while** (!f.eof())

{

cout<<a.roll<<' '<<a.name<<' '<<a.fees<<endl;

f.read((**char**\*)&a, **sizeof**(a));

}

f.close();

**Example 2:** Reading data from binary file using stream class function eof()

**class** student

{

**int** roll;

**char** name[20];

**double** fees;

**public**:

**void** inputdata()

{

cout<<"Roll? "; cin>>roll;

cout<<"Name? "; gets(name);

cout<<"Fees? "; cin>>fees;

}

**void** showdata()

{

cout<<roll<<' '<<name<<' '<<fees<<endl;

}

};

ifstream f("Fees.dat", ios::binary);

student a;

f.read((**char**\*)&a, **sizeof**(a));

**while** (!f.eof())

{

a.showdata();

f.read((**char**\*)&a, **sizeof**(a));

}

f.close();

**Example 3:** Reading data from text file using stream class function eof()

ifstream f("Fees.txt"); **char** ch;

f.get(ch);

**while** (!f.eof())

{

cout<<ch; f.get(ch);

}

f.close();

get() Stream class member function of istream / ifstream / fstream, used to extract a character from a file. Usually used with text file to read character by character (including white space characters like space, tab and new line character). When using with a file, file must be opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** **char** FileObject.get();

FileObject.get(**char**);

FileObject is an instance of ifstream / fstream

**Example 1:** ifstream f("Fees.txt");

**char** ch=f.get();

**while** (f)

{

cout<<ch; ch=f.get();

}

f.close();

**Example 2:** ifstream f("Fees.txt");

**char** ch;

**while**(f.get(ch))

cout<<ch;

f.close();

put() Stream class member function of ostream / ofstream / fstream, used to insert a character into a stream. Usually used with text file to write one character at a time in a text file. When using with a file, file must be opened in output mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.put(**char**);

FileObject is an instance of ofstream / fstream

**Example:** Storing 10 names in a text file, one character at a time

ifstream f("Names.txt");

**char** name[20];

**for**(int k=0; k<10; k++)

{

cout<<"Name? "; gets(name);

**for**(**int** x=0; name[x]; x++)

f.put(name[x]);

f.put('\n');

}

f.close();

getline() Stream class member function of istream / ifstream / fstream, used to extract characters up to the size given or up to the delimiter whichever is encountered first. Usually used with text file to read one line of text delimited by new line character. When using with a file, file must be opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.getline(**char**\* String, **int** Size);

FileObject is an instance of ofstream / fstream

Function getline() has three parameters – String (char\*), Size – number of characters (length of the string) and default parameter – **new line character** ('\n').

**Example 1:** Reading text file – Fees.txt

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

ifstream f("Fees.txt"); **char** str[20];

**while** (f.getline(str,20))

cout<<str<<endl;

f.close();

Each line of the text file Fees.txt is delimited by new line character. Function getline() reads the entire line of text and transfers the data in a string (**char** str[20]). The string is then displayed on the screen.

**Example 2:** Reading text file – Fees.txt

1 Anit Suri 6000#2 Anika Rao 6500#3 Bipul Jha 6000

ifstream f("Fees.txt");

**char** str[20];

**while** (f.getline(str,20,'#'))

cout<<str<<endl;

f.close();

Each line of the text file Fees.txt is delimited by hash (#). Function getline() reads the entire characters from text file till it encounters hash (#), getline() transfers the data in a string (**char** str[20]). The string is then displayed on the screen.

tellg() Stream class member function of istream / ifstream / fstream, used to return the current byte position of a file pointer in a file when file is opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** **int** FileObject.tellg();

FileObject is an instance of ifstream / fstream

**Example 1:** Reading data from Binary file – Fees.dat

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

**struct** student

{

**int** roll;

**char** name[20];

**double** fees;

};

ifstream f("Fees.dat", ios::binary);

student stu;

cout<<f.tellg()<<endl;

**while** (f.read((**char**\*)&stu, **sizeof**(stu)))

{

cout<<stu.roll<<' '<<stu.name<<' '

<<stu.fees<<' '<<f.tellg()<<endl;

}

f.close();

Running of the program segment produces following output:

0

1 Anit Suri 6000 32

2 Anika Rao 6500 64

3 Bipul Jha 6000 96

4 Tarun Dey 5500 128

First output displays initial position of file pointer. When a file is opened in input mode, initial position of file pointer is 0 (zero). Last column displays the position of file pointer after reading a record. Every time a record is read from a file, file pointer is updated by 32 bytes since variable stu is allocated 32 bytes of memory.

**Example 2:** Reading data from text file – Fees.txt

1 Anita 6000

2 Akhileshwar 6500

3 Chandan 6000

4 Tarun 5500

**int** roll;

**char** name[20];

**double** fees;

ifstream f("Fees.txt");

cout<<f.tellg()<<endl;

**while** (f>>roll>>name>>fees)

cout<<roll<<' '<<name<<' '<<fees

<<' '<<f.tellg()<<endl;

f.close();

Running of the program segment produces following output:

0

1 Anita 6000 12

2 Akhileshwar 6500 32

3 Chandana 6000 49

4 Tarun 5500 63

First output displays initial position of file pointer. When a file is opened in input mode, initial position of file pointer is 0 (zero). Last column displays the position of file pointer after reading a record. Every time a record is written into a file, file pointer is updated. Kindly note, number of characters varies one for line to line.

tellp() Stream class member function of ostream / ofstream / fstream, used to return the current byte position of a file pointer in a file when file is opened in output mode.

**Header File:** iostream.h, fstream.h

**Syntax:** **int** FileObject.tellp();

FileObject is an instance of ofstream / fstream

**Example 1:** Writing data into Binary file – Fees.dat

**struct** student

{

**int** roll;

**char** name[20];

**double** fees;

};

ofstream f("Fees.dat", ios::binary);

student stu;

cout<<f.tellp()<<endl;

**for** (int x=0; x<4; x++)

{

cout<<"Roll? "; cin>>stu.roll;

cout<<"Name? "; gets(stu.name);

cout<<"Fees? "; cin>>stu.fees;

f.write((**char**\*)&stu, **sizeof**(stu))

cout<<f.tellp()<<endl;

}

f.close();

Running of the program segment produces following output:

0

Roll? 1

Name? Anit Suri

Fees? 6000

32

Roll? 2

Name? Anika Rao

Fees? 6500

64

Roll? 3

Name? Bipul Jha

Fees? 6000

96

Roll? 4

Name? Tarun Dey

Fees? 5500

128

First output displays initial position of file pointer. When a file is opened in output mode, initial position of file pointer is 0 (zero). Numbers after each input represent position of file pointer. Every time a record is written into a file, file pointer is updated by 32 bytes.

**Example 2:** Writing data into text file – Fees.txt

**int** roll;

**char** name[20]; **double** fees;

ofstream f("fees.txt");

cout<<f.tellp()<<endl;

**for** (int x=0; x<3; x++)

{

cout<<"Roll? "; cin>>roll;

cout<<"Name? "; gets(name);

cout<<"Fees? "; cin>>fees;

f<<roll<<' '<<name<<' '<<fees<<endl;

cout<<f.tellp()<<endl;

}

f.close();

Running of the program segment produces following output:

0

Roll? 1

Name? Anit Suri

Fees? 6000

16

Roll? 2

Name? Anikita Singh Bedi

Fees? 6500

41

Roll? 3

Name? Bipul Kumar Jha

Fees? 6000

63

First output displays initial position of file pointer. When a file is opened in output mode, initial position of file pointer is 0 (zero). Numbers after each input represent position of file pointer. Every time a record is written into a file, file pointer is updated.

seekg() Stream class member function of istream / ifstream / fstream used to move the file pointer to a specified position in a file when the file is opened in input mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.seekg(**int** Bytes, Offset);

FileObject.seekg(**int** Bytes);

FileObject is an instance of ifstream / fstream

Bytes represent number of bytes and it is calculated from the beginning of the file when seekg() has only **one** parameter. If seekg() has **two** parameters, number of bytes is calculated either from beginning (ios::beg) or current location (ios::cur) or end (ios::end).

**Example:** Reads data from binary data file – Fees.dat

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

5 Rony John 6500

**struct** student

{

**int** roll;

**char** name[20];

**double** fees;

};

ifstream f("fees.dat", ios::binary);

student a;

f.read((**char**\*)&a, **sizeof**(a));

cout<<a.roll<<' '<<a.name<<' '<<a.fees<<endl;

f.seekg(3\***sizeof**(a));

f.read((**char**\*)&a, **sizeof**(a));

cout<<a.roll<<' '<<a.name<<' '<<a.fees<<endl;

f.seekg(-2\***sizeof**(a), ios::cur);

f.read((**char**\*)&a, **sizeof**(a));

cout<<a.roll<<' '<<a.name<<' '<<a.fees<<endl;

f.close();

Running of the program segment produces following output:

1 Anit Suri 6000

4 Tarun Dey 5500

3 Bipul Jha 6000

First record is read and displayed on the screen. Then f.seekg(3\***sizeof**(a)) takes the file pointer to the end of 3rd record (beginning of 4th record). 4th record is read and displayed. Next f.seekg(-2\***sizeof**(a), ios::cur) takes the pointer two records back from the current position, that is, beginning of 3rd record. 3rd record is read and displayed on the screen.

seekp() Stream class member function of ostream / ofstream / fstream, used to move the file pointer to a specified position in a file when the file is opened in output mode.

**Header File:** iostream.h, fstream.h

**Syntax:** FileObject.seekp(**int** Bytes, Offset);

FileObject.seekp(**int** Bytes);

FileObject is an instance of ofstream / fstream

Bytes represent number of bytes and it is calculated from the beginning of the file when seekp() has only **one** parameter. If seekp() has **two** parameters, number of bytes is calculated either from beginning (ios::beg) or current location (ios::cur) or end (ios::end).

**Example:** Updates the names to uppercase in a text file – Fees.txt

1 Anit Suri 6000

2 Anika Rao 6500

3 Bipul Jha 6000

4 Tarun Dey 5500

fstream f("fees.txt", ios::in|ios::out);

**char** ch;

**while**(f.get(ch))

**if** (ch>='a' && ch<='z')

{

ch=**char**(ch-32);

f.seekp(-1, ios::cur);

f.put(ch);

}

f.close();

f.open("Fees.txt", ios::in);

**while**(f.get(ch))

cout<<ch;

f.close();

Running of the program segment produces following output:

1 ANIT SURI 6000

2 ANIKA RAO 6500

3 BIPUL JHA 6000

4 TARUN DEY 5500

Text file Fees.txt is opened in input/output mode. Every character is read from the file. If a character is lowercase then the character is converted to uppercase, file pointer moved 1 byte back and lowercase character is replaced by equivalent uppercase character. Next file is opened in input mode and edited file is displayed on the screen.

**Append Mode**: If a file is opened in **output** mode and if the file already exists then the new set of data replaces existing data in the file. Therefore **append** mode is used to add data to an existing file without losing previously stored data in the file. To open a file in append mode we need an instance of ofstream or fstream plus ios::app after the file name. In append mode if a file does not exist then new file is created.

**Example 1**: Appending records in a text file – Fees.txt

**int** roll;

**char** name[20];

**double** fees;

ofstream f("fees.txt", ios::app);

**for** (int x=0; x<5; x++)

{

cout<<"Roll? "; cin>>roll;

cout<<"Name? "; gets(name);

cout<<"Fees? "; cin>>fees;

f<<roll<<' '<<name<<' '<<fees<<endl;

}

f.close();

**Example 2**: Appending records in a binary file – Fees.dat

**struct** student

{

**int** roll; **char** name[20]; **double** fees;

};

ofstream f("fees.dat", ios::binary | ios::app);

student stu;

**for** (int x=0; x<5; x++)

{

cout<<"Roll? "; cin>>stu.roll;

cout<<"Name? "; gets(stu.name);

cout<<"Fees? "; cin>>stu.fees;

f.write((**char**\*)&stu, **sizeof**(stu))

}

f.close();

**Reading a text file**: A text file can be read in various ways since C++ stream class has various functions and operator to read data stored in a text file. Text file Marks.txt will be used as a sample text file (this is a formatted text file – data appear in tabular/column format).

1 ANITA 70

2 ANIKA 65

3 BIPUL 80

4 RONY 85

5 TARUN 75

6 VIKASH 60

* Reading a text file character by character using extraction operator (>>).

ifstream f("marks.txt");

**char** ch;

**while** (f>>ch)

cout<<ch;

f.close();

Running of the program segment produces following output:

1ANIT702ANIKA653BIPUL804RONY855TARUN756VIKASH60

Extraction operator (>>) ignores all white space characters (space, tab and new line character). All the non-white space characters are displayed on the screen but the format of the file is completely lost.

* Reading a text file character by character using stream class member function get().

ifstream f("marks.txt"); **char** ch;

**while** (f.get(ch))

cout<<ch;

f.close();

Running of the program segment produces following output:

1 ANITA 70

2 ANIKA 65

3 BIPUL 80

4 RONY 85

5 TARUN 75

6 VIKASH 60

Stream class member function get() reads every character from text file (including white space characters). All characters are displayed on the screen and the format of the file is retained. When the format of text file is not known, reading a text file using get() is the best option.

* Reading a text file line by line using stream class member function getline().

ifstream f("marks.txt");

**char** str[20];

**while** (f.getline(str, 20))

cout<<str<<endl;

f.close();

Running of the program segment produces following output:

1 ANITA 70

2 ANIKA 65

3 BIPUL 80

4 RONY 85

5 TARUN 75

6 VIKASH 60

Stream class member function getline() reads a line of text delimited / terminated by new line character. The entire line of text is transferred to a string. The string is displayed on the screen and format of the file is retained. Function getline() is best suitable when line of text in a text file is delimited / terminated by new line character.

* Reading a text file using extraction operator (>>) to read data from a formatted text file so that the data stored in different columns can be processed. For example, from text file Marks.txt, Highest marks and Average marks are to be calculated.

ifstream f("marks.txt");

**int** roll; **char** name[10]; **double** mark, hi=0, sum=0, count=0;

**while** (f>>roll>>name>>mark)

{

cout<<roll<<'\t'<<name<<'\t'<<mark<<endl;

sum+=mark; count++;

**if** (mark>hi)

hi=mark;

}

f.close();

**double** avg=sum/count;

cout<<"Highest Mark="<<hi<<endl<<"Average Mark="<<avg<<endl;

Running of the program segment produces following output:

1 ANITA 70

2 ANIKA 65

3 BIPUL 80

4 RONY 85

5 TARUN 75

6 VIKASH 60

Highest Mark=85

Average Mark=72.5

* Reading a text file word by word using extraction operator (>>) to read data from unformatted text file DPS.txt. Text file DPS.txt is given below:

FAIPS conducted its Annual Scholar Badge Function felicitating the students who achieved academic excellence over the year. Students were felicitated with Blue Tie, Green Tie, Scholar Badge or Merit Certificate for brilliant and consistent academics performance.

ifstream f("sbadge.txt");

**char** word[20];

**while** (f>>word)

cout<<word<<' ';

f.close();

Running of the program segment produces following output:

FAIPS conducted its Annual Scholar Badge Function felicitating the students who achieved academic excellence over the year. Students were felicitated with Blue Tie, Green Tie, Scholar Badge or Merit Certificate for brilliant and consistent academics performance.

* Reading a text file word by word using stream class member function getline() (using space as delimiter) to read data from unformatted text file DPS.txt.

ifstream f("sbadge.txt");

**char** word[20];

**while** (f.getline(word, 20, ' '))

cout<<word<<' ';

f.close();

Running of the program segment produces following output:

FAIPS conducted its Annual Scholar Badge Function felicitating the students who achieved academic excellence over the year. Students were felicitated with Blue Tie, Green Tie, Scholar Badge or Merit Certificate for brilliant and consistent academics performance.