

Basic function of I/O system basics and file processing

Syllabus

Stream classes, using formatted & unformatted functions, using manipulator to format I/O, Basics of file system, opening & closing a file, reading & writing character from a file (get, put, getline, write), Command line arguments.

Stream – Standard Input and Output Stream

Q1.What is a Stream? List Standard Streams?

Ans.A stream is an object where a program can either insert or extract characters to or from it. The standard input and output stream objects of C++ are declared in the header file *iostream*.

There are two streams

- 1.Standard Input Stream
- 2.Standard Output Stream

Q2.Describe standard input stream

Ans. Standard Input Stream

Generally, the device used for input is the keyboard. For inputting, the keyword *cin* is used, which is an object. The overloaded operator of extraction, *>>*, is used on the standard input stream, in this case: *cin* stream. Syntax for using the standard input stream is *cin* followed by the operator *>>* followed by the *variable* that stores the data extracted from the stream.

For example

```
int prog;  
cin >> prog;
```

In the example above, the variable *prog* is declared as an integer type variable. The next statement is the *cin* statement. The *cin* statement waits for input from the user's keyboard that is then stored in the integer variable *prog*.

The input stream *cin* wait before proceeding for processing or storing the value. This duration is dependent on the user pressing the RETURN key on the keyboard. The input stream *cin* waits for the user to press the RETURN key then begins to process the command. It is also possible to request input for more than one variable in a single input stream statement. A single *cin* statement is as follows

```
cin >> x >> y;
```

is the same as

```
cin >> x;  
cin >> y;
```

In both of the above cases, two values are input by the user, one value for the variable *x* and another value for the variable *y*.

```
// This is a sample program    This is a comment Statement
```

```
#include <iostream.h>
```

Header File Inclusion Statement

```
void main()
```

```
{
```

```
    int sample, example;
```

```
    cin >> sample;
```

```
    cin >> example;
```

```
}
```

If a programmer wants to write comments in C++ program, the comments should follow after a pair of slashes denoted by `//`. All the characters after the `//` are ignored by C++ compiler and the programmer can choose to comment after the `//`.

In the above example, two integer variables are input with values. The programmer can produce input of any data type. It is also possible to input strings in C++ program using `cin`. But `cin` stops when it encounters a blank space. When using a `cin`, it is possible to produce only one word. If a user wants to input a sentence, then the above approach would be tiresome. For this purpose, there is a function in C++ called *getline*.

Q3.Describe standard output stream

Ans. Standard output Stream

By default, the device used for output is the screen of the computer. For outputting values the keyword `cout` is used, which is an object. The insertion operator `<<` is used on the standard output `cout` stream. The syntax for using the standard output stream is `cout` followed by the operator `<<` followed by the value to be inserted or output by the insertion operator.

For example:

```
int prog; cin
```

```
>> prog;
```

```
cout << prog;
```

In the above example, the variable `prog` is declared as an integer type variable. The next statement is the `cin` statement that waits for input from the user's keyboard. This information is then stored in the integer variable `prog`. The value of `prog` is displayed on the screen by the standard output stream `cout`. It is also possible to display a sentence as follows:

```
cout << "C++ training is given by Prof.Funminiyi A. Olajide";
```

The above gives output as:

```
C++ training is given by Prof.Funminiyi A. Olajide
```

If a programmer chooses to use constant strings of characters, they must be enclosed between double quotes `" "`.

In this situation, it is important to note the difference between the two statements below:

```
cout << "MSKsys";
cout << MSKsys;
```

In the above, the first statement displays on the screen as MSKsys. The second statement outputs the value of the variable MSKsys. As previously explained, the extraction operator `>>` can be used more than once in a single *cin* statement. Similarly, it is possible to use the insertion operator `<<` more than once in a *cout* statement.

For example

```
cout << "C++ training" << "is given by" << "Prof.Funminiyi A. Olajide";
```

This produces output on the screen as

```
C++ trainingis given by Prof.Funminiyi A. Olajide
```

The above concept is mainly used if the programmer chooses to print string constants followed by variables.

In this next example, the programmer chooses to display a combination of string *constants* and *variables*.

For example

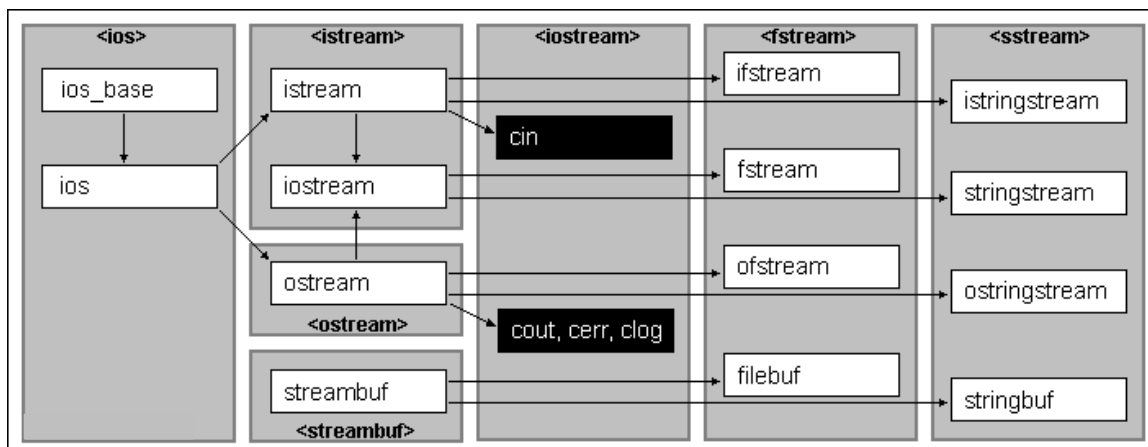
```
int a=5000;
cout << "Funminiyi Olajide has given" << a << "numbers of trainings for Java and C++";
```

This produces the output as:

```
Funminiyi Olajide has given 5000 numbers of trainings for Java and C++
```

Q4.Describe C++ stream classes of C++

Ans. The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files. These classes are called stream classes. Figure shows the hierarchy of the stream classes used for input and output operations with the console unit.



These classes are declared in the header file **iostream**. This file should be included in all the programs that are written to interact with the console unit.

The **ios** is the base class for **istream** (input stream) and **ostream** (output stream) which is base classes for **iostream** (input/output stream). The class **ios** is declared as the virtual base class so that only one copy of its members are inherited by the **iostream**.

The class **ios** provides the basic support for formatted and unformatted I/O operations.

The class **istream** provides the facilities for formatted and unformatted input.

The class **ostream** (through inheritance) provides the facilities for formatted output.

The class **iostream** provides the facilities for handling both input and output streams. Three classes, namely,

1. **istream_withassign**,
2. **ostream_withassign**, and
3. **iostream_withassign**

adds assignment operators to these classes.

Following are the stream classes for console operation

- **ios**
- **istream**
- **ostream**
- **iostream**
- **streambuf**

ios class – General input/output stream class

- Contain basic function which are used by all the other input and output classes
- It contains a pointer to buffer object ie **streambuf** object
- It also declare constant and functions that are necessary that are necessary for handling formatted input and output
- **For example fill character:** character used to pad a field up to the *field width*. It can be obtained or modified by calling member function **fill**.

Istream class - Input Stream

- Properties of **ios** is inherited
- Contain overloaded extraction operator **>>**
- Declare input function such as **get()**, **getline()** and **read()**

istream objects are stream objects used to read and interpret input from sequences of characters. Specific members are provided to perform these input operations, which can be divided in two main groups:

- **Formatted input**

These functions extract data from a sequence of characters that may be interpreted and formatted to a value of a certain type. These type of operation is performed using member and global functions that overload the extraction operator **()**.

- **Unformatted input**

Most of the other member functions of the `istream` class are used to perform unformatted input, i.e. no interpretation is made on the characters got from the input. These member functions can get a determined number of characters from the input character sequence (`get`, `getline`, `peek`, `read`, `readsome`), manipulate the *get pointer* (`ignore`, `seekg`, `tellg`, `unget`) or get information of the last unformatted input operation (`gcount`).

ostream class - output stream

- Properties of `ios` is inherited
- Contain overloaded insertion operator `<<`
- Declare output function such as `put()`, and `write()`

`ostream` objects are stream objects used to write and format output as sequences of characters. Specific members are provided to perform these output operations, which can be divided in two main groups:

- **Formatted output**

These member functions interpret and format the data to be written as sequences of characters. These type of operation is performed using member and global functions that overload the insertion operator (`operator<<`).

- **Unformatted output**

Most of the other member functions of the `ostream` class are used to perform unformatted output operations, i.e. output operations that write the data as it is, with no formatting adaptations. These member functions can write a determined number of characters to the output character sequence (`put`, `write`) and manipulate the *put pointer* (`seekp`, `tellp`).

iostream class - input/output stream

The properties of `ios`, `istream` and `ostream` are inherited using multiple inheritance and hence contains all the input and output functions.

`<iostream>` declares the objects used to communicate through the standard input and output (including `cin` and `cout`).

Streambuf class

- It is used to provide interface for the physical devices via buffers
 - It is also super / base class for `filebuf` class used in `ios` file
- `streambuf` objects are in charge of providing reading and writing functionality to/from certain types of character sequences, such as external files or strings. `streambuf` objects are usually associated with one specific character sequence, from which they read and write data through an internal memory buffer. The buffer is an array in memory which is expected to be synchronized when needed with the physical content of the *associated character sequence*.

This is an abstract base class, therefore no objects can be directly instantiated from it. The standard hierarchy defines two classes derived from this one that can be used to directly instantiate objects: `filebuf` and `stringbuf`.

Formatted and unformatted stream

Q5.Explain the meaning of Formatted and Unforamttd I/O operation

Ans.Unformatted IO Operations

The objects `cin` and `cout` (pre-defined in the `iostream` file) for the input and output of data of various types. This has been made possible by overloading the operators `>>` (Insertion) and `<<` (Extraction) to recognize all the basic C++ types. The `>>` operator is overloaded in the `istream` class and `<<` overloaded in the `ostream` class. **The following is the general -format for reading data from the keyboard**

`cin>>var1>>var2>>var3.....>>varN`

`var1, var2, var3....` are declared and are valid variables.

The above statement will read N data and stop reading the data after that. The input data is separated by space or new line character or tab.

The following is the general -format for write data to the display

`Cout<<var1<<var2<<var3.....<<varN`

`var1, var2, var3....` are declared and are valid variables.

The above statement will display N data and stop writing the data after that.

Formatted IO Operations

Creating cleanly formatted output is a common programming requirement--it improves your user interface and makes it easier to read any debugging messages that you might print to the screen. In C++, you can create nicely formatted output to streams such as `cout`.

A set of basic I/O manipulations possible in C++ from the `iomanip` header file. Note that all of the functions in the `iomanip`.

C++ supports many different ways for formatting the output. Those are

1. `ios` class function and flags
2. Manipulator
3. user Defined Functions

Note : for details refer respective topics

get() and put() function

Q6. Describe with example function like `put()` and `get()`

Ans. `get()` function

These member functions perform unformatted input operations. Depending on the type and number of arguments the function behaves in the following way:

`int get(void);`

Returns a character from the stream and returns its value (casted to an integer).

`istream& get (char& c);`

Returns a character from the stream and stores it in `c`.

`istream& get (char* s, streamsize n);`

Returns characters from the stream and stores them as a c-string into the array beginning at `s`. Characters are extracted until either $(n - 1)$ characters have been extracted or the *delimiting character* `'\n'` is found. The extraction also stops if the end of file is reached in the input sequence or if an error occurs during the input operation.

If the *delimiting character* is found, it is not extracted from the input sequence and remains as the next character to be extracted. Use `getline` if you want this character to be extracted (and discarded).

The ending null character that signals the end of a c-string is automatically appended at the end of the content stored in `s`.

istream& get (char* s, streamsize n, char delim);

Same as above, except that the delimiting character is the one specified in *delim* instead of '\n'.

istream& get (streambuf& sb);

Returns characters from the stream and inserts them in the stream buffer *sb* until either the delimiting character '\n' is found or end of file is reached. The extraction also stops if an error occurs either in the input sequence controlled by the stream or in the output sequence controlled by *sb*.

istream& get (streambuf& sb, char delim);

Same as above, except that the delimiting character is the one specified in *delim* instead of '\n'.

The number of characters read by any of the previous input operations can be obtained by calling to the member function `gcount`.

Parameters**c**

A char variable to store the extracted character.

s

A pointer to an array of characters where the string is stored as a c-string

n

Maximum number of characters to store (including the terminating null character). This is an integer value of type `streamsize`.

delim

The delimiting character. The operation of extracting successive characters is stopped when this character is read. This parameter is optional, if not specified the function considers '\n' (a newline character) to be the delimiting character.

sb

An output stream buffer (an object of class `streambuf` or any of its derived classes).

Example

These functions are members of the input/output stream class. These functions are invoked by using appropriate objects.

```
char ch
cin.get(ch);           // read character from keyboard
while(ch!='\n')        // repeat while loop till character read is \n
{
    cout<<"\nEntered character is "<<ch; // display character
    cin.get(ch) ;        // read character from keyboard
}
```

Explanation

This code reads and displays a line of text (terminated by a newline character). Remember, the operator `>>` can also be used to read a character but it will skip the white spaces and newline character. The above while loop will not work properly if the statement

Another way of using

Ch = cin.get();

The value return by get() function will be returned and assign to ch variable.

Put Function

Writes the character *c* to the output buffer at the current *put position* and increases the *put pointer* to point to the next character. This function is member of ostream class and can be used to output a line of text character by character.

ostream& put (char ch);

where ch is the character to be output.

Example

```
cout.put('M');    // display the character X
cout.put(ch) ;    // display the value of character ch
cout.put(69)      // display character equivalent of ASCII 69 ie 'E'
```

Program to Demonstrate put() and get() function

```
#include <iostream>
#include <conio.h>
using namespace std;

void main ()
{
    char ch;
    int vctr , cctr ;
    cout << "\n Start entering the text Here ";
    vctr = cctr = 0;
    do
    {
        ch=cin.get();           // read character from keyboard
        cout.put (ch);          // display character on screen
        if ( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u')
            vctr++              // count for vowel
        else
            cctr++;             // count for consonant

    }while (ch!='\n');
    cout << "\n Number of Vowel =" << vctr;
    cout << "\n Number of consonant =" << cctr;

    getch();
}
```

Explanation

When we type a line of input, the text is sent to the program , as soon as we press the RETURN key. The program then reads one character at a time using the statement cin.get(ch); and displays it using the statement cout.put(ch);.Also it counts

for vowel and consonant. The process is terminated when the newline character is encountered.

write() and getline() function

Q7.Describe the function getline() and write ?Also write program to demonstrate the same

Ans.getline Function

Returns characters from the input sequence and stores them as a c-string into the array beginning at *str*.

`istream& getline (char* s, streamsize n);`

Example`cin.getline(str,size)`

`istream& getline (char* s, streamsize n, char delim);`

Example `cin.getline(str,size,'\n')`

Characters are returned until either $(n - 1)$ characters have been returned or the delimiting character is found (which is *delim* if this parameter is specified, or '\n' otherwise). The returning also stops if the end of file is reached in the input sequence or if an error occurs during the input operation.

If the delimiter is found, it is returned and discarded, i.e. it is not stored and the next input operation will begin after it.

The ending null character that signals the end of a c-string is automatically appended to *str* after the data extracted.

The number of characters read by this function can be obtained by calling to the member function *gcount*.

Parameters

s

A pointer to an array of characters where the string is stored as a c-string.

n

Maximum number of characters to store (including the terminating null character).

This is an integer value of type *streamsize*.

If the function stops reading because this size is reached, the failbit internal flag is set.

delim

The delimiting character. The operation of extracting successive characters is stopped when this character is read. This parameter is optional, if not specified the function considers '\n' (a newline character) to be the delimiting character.

Example

```
char str[40];
```

```
cin.getline(str,40); // it will upto 39 character and input will be terminated
```

```
cin.getline(str,20); // it will read 19 character and input will be terminated
```

Note : blank spaces contained in the string are also taken into account.

Program to Demonstrate getline function

```
// istream getline
#include <iostream>
using namespace std;
void main ()
{
    char name[256], title[256];
    cout << "Enter your name: ";
    cin.getline (name,256);
    cout << "Enter your favourite movie: ";
    cin.getline (title,256);
    cout << name << "'s favourite movie is " << title;
    getch();
}
```

Output

```
Enter your name:
Funminiyi Olajide
Enter your favourite movie:
Disneyland
```

Funminiyi Olajide's favorite movie name is Disneyland

Explanation

The getline() function reads a whole line of text that ends with a newline character (transmitted by the RETURN key). This function can be invoked by using the object cin.

During the run when user enter name and movie name as soon as user press Enter key program control is transferred to next line. Also if enter is not pressed then same statement will read 255 character and control will be transferred to next line.

Write Function

```
ostream& write ( const char* s , streamsize n );
```

Write block of data

Writes the block of data pointed by *str*, with a size of *n* characters, into the output buffer. The characters are written sequentially until *n* have been written.

This is an unformatted output function and what is written is not necessarily a c-string, therefore any null-character found in the array *str* is copied to the destination and does not end the writing process.

Parameters

str

Pointer to a block data with the content to be written.

n

Integer value of type streamsize representing the size in characters of the block of data to write.

Example

```
cout.write(str, size)
```

str – represent the name of the string to be displayed.
size – indicate the number of character to be displayed.

Note: It does not stop displaying character automatically when the null character is encountered. If the size is greater the length of line , then it displays beyond the bounds of line.

Program to Demonstrate write function

```
#include<iostream.h>
#include<conio.h>
using namespace std;
void main()
{
    char *str1 = "Asha Manoj" ;
    char *str2 = " Kavedia / Jain";
    int len1 = strlen(str1);
    int len2 = strlen(str2);

    cout.write(str1,len1);           // write the first string
    cout.write(str2,len2) ;         //write the Second String
    cout<<"\n";
    cout.write(str1,5);              //write the first 5 character of str1
    cout.write(str2,6);              //write the first 8 character of str2

    getch();
}
```

output

Asha Manoj Kavedia / Jain
Asha Kavedia

ios class and Manipulators

Q8.List the different ways in c++ to format the output

Ans.C++ supports many different ways for formatting the output.Those are

- 1.ios class function and flags
- 2.Manipulator
- 3.user Defined Functions

Q9.List and describe with sample code , how ios class function can be used for formatting the output

Ans.ios class consist of many member function which would help us to format the output in number of ways. This member function are inherited from ios_base class.

Following are the format of ios functions

width

Function	Explanation	Syntax	Example
width()	It specify the required field size that will be	width(W)	cout.width(10)

	displayed on the screen		
precision()	It specify the number of digit to be displayed after the decimal point for the float value.	precision(d)	cout.precision(5)
fill()	It specify a character that is used to fill the unused portion of a field	fill(ch)	cout.fill('#');
setf()	It specify format flag that can control the form of output display(like left or right justification)	setf(arg1,arg2)	cout.setf(ios::left,ios::adjusfield)
unsetf()	It reset the flag specified	unsetf()	

Q10.Describe the width and precision function of ios class with example

Ans. Width Function

The *field width* determines the minimum number of characters to be written in some output representations. It is invoked as follow

```
cout.width(w)
```

1. where w is the field width (number of columns). The output will be printed in a field of w characters wide at the right end of the field.
2. The width() function can specify the field width for only one item (the item that follows immediately). After printing one item (as per the specifications) it will revert back to the default.
3. For example, the statements

```
cout.width(6);
```

```
cout << 143 << 17 <<"\n";
```

will give the output as

			1	4	3	1	7
--	--	--	---	---	---	---	---

The value 143 is printed right justified in the first six column.But same setting is not valid for the number 17.

4. Another example

```
cout.width(6);
```

```
cout << 143 ;
```

```
cout.width(6);
```

```
cout << 12 <<"\n";
```

			1	4	3					1	2
--	--	--	---	---	---	--	--	--	--	---	---

In this example width of spaces is set for both the number,

Program to demonstrate width ios function

```
// using width
#include <iostream.h>
#include <conio.h>
void main()
{
    int product[5] = { 100,800,120,150,350};
    int rate[5]    = { 50,75,100,125,150};
    cout.width(6);
    cout<< "Product";
    cout.width(10);
    cout<< "Cost";
    cout.width(15);
    cout<< "\n Amount";
    for (int i = 0; i < 4; i++)
    {
        cout.width(6);
        cout<< product[i];
        cout.width(10);
        cout<< cost[i];
        cout.width(15);
        cout<< product[i] * cost[i];
        cout<< "\n";
    }
    getch();
}
```

output

Product	Cost	Amount
100	50	5000
800	75	60000
120	100	12000
150	125	18750
350	150	52500

Precision Function

Used to control the number of digits to the right of the decimal point. By default, the floating numbers are printed with six digits after the decimal point. However, we can specify the number of digits to be displayed after the decimal point while printing the floating-point numbers. This can be done by using the precision() member function as follows

```
cout.precision(p);
```

Here d is the number of digit to the right of the decimal places.

For example

Example	Output	Explanation
cout.precision(3);		

cout<<sqrt(3)<<"\n";	1.732	Number is truncated
cout<<4.324712<<"\n";	4.325	Number is rounded to nearest
cout<<5.30002<<"\n";	5.3	Trailing zeros are removed

Precision and width both can be combined

```
cout.precision(4);
cout.width(10);
cout<<143.171734
```

		1	4	3	.	1	7	1	7
--	--	---	---	---	---	---	---	---	---

Program to demonstrate width ios function

```
// using precision
#include <iostream.h>
using namespace std;
#include <iomanip>
// using C++ wrappers to access C function
#include <cmath>

void main(void)
{
    double theroot = sqrt(11.55);

    cout<<"Square root of 11.55 with various"<<endl;
    cout<<"      precisions"<<endl;
    cout<<"-----"<<endl;
    cout<<"Using 'precision':"<<endl;
    for(int poinplace=0; poinplace<=8; poinplace++)
    {
        cout.precision(poinplace);
        cout<<theroot<<endl;
    }
    getch();
}
```

output

```
Square root of 11.55 with various
precisions
-----
Using 'precision':
3
3
3.4
3.4
3.399
3.3985
3.39853
3.398529
3.3985291
```

Q11.Describe with program how filling and padding can be done using c++ ios function

Ans.We have been printing the values using much larger field widths than required by the values. The unused positions of the field are filled with white spaces, by default. However, we can use the fill() function to fill the unused positions by any desired character. It is used in the following form:

```
cout.fill(ch);
```

here ch represents the character which is used for filling the unused portion.

Example

```
cout.fill('#');
cout.width(10);
cout<<1234<<"\n";
```

#	#	#	#	#	#	1	2	3	4
---	---	---	---	---	---	---	---	---	---

Program to demonstrate Filling and Padding

```
// using the fill character
#include <iostream.h>
using namespace std;
```

```
int main ()
{
    char prev;

    cout.width (10);
    cout << 40 << endl;

    prev = cout.fill ('x');
    cout.width (10);
    cout << 40 << endl;

    cout.fill(prev);

    return 0;
}
```

output

```
40
xxxxxxx40
```

Q12.Describe the usage Formatting flags , bit fields and setf() function

Ans. C++ defines some format flags for standard input and output, which can be manipulated with the flags , setf, and unsetf functions.

The setf() a member function of the ios class, can provide answers to these and many other formatting questions. The setf() (setf stands for set flags) function can be used as follows

```
Cout.setf(agrument1,argument2)
```

argument1 – Formatting flag.It specifies action to taken for the output

argument2 – Bit Field .Bit field specifies to which group the formatting flags belongs.

Format Required	Flag (Argument1)	Bit-Field(argument2)
Left-justified output	ios::left	ios::adjustfield
Right-justified output	ios::right	ios::adjustfield
Padding after sign or base indicator(lik	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal Base	ios::dec	ios::basefield
Octal Base	ios::oct	ios::basefield
Hexadecimal	ios::hex	ios::basefield

Example

```
cout.fill('#');
cout.setf(ios::left,ios::adjustfield);
cout.width(17);
cout<< "Manoj Kavedia";
```

output

M	A	N	O	J		K	A	V	E	D	I	A	#	#	#	#
---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	---

Example

```
cout.fill('@');
cout.precision(4);
cout.setf(ios::internal ,ios::adjustfield);
cout.setf(ios::scientific,ios::floatfield);
cout.width(20);
cout<<-24.345678;
```

-	@	@	@	@	@	@	@	@	@	2	.	4	3	5	7	e	+	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Displaying Trailing Zeros and Plus Sign

Generally number are printed with left justification , width of 8 spaces , all trailing zeros are truncated and plus sign is not displayed when the number is positive.

The setf() can be used with the flag to achieve this form of output. For example,

```
cout.setf(ios::showpoint); // display trailing zeros
cout.setf(ios::showpos); // show psitive(+) sign
```

Example

```
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout.precision(3)
cout.setf(ios::fixed, ios::floatfield)
cout.setf(ios::internal, ios::adjustfield);
```



```
cout.width(10);
cout < 143.5 << "\n";
```

+			1	4	3	.	5	0	0
---	--	--	---	---	---	---	---	---	---

The flags such as showpoint and showpos do not have any bit fields and therefore are used as single arguments in setf(). This is possible because the setf() has been declared as an overloaded function in the class ios. Table shown lists the flags that do not possess a named bit field. These flags are not mutually exclusive and therefore can be set or cleared independently.

Flags	Function
ios::showbase	It used as base indicator for output
ios::showpos	It prints positive sign before number
ios::showpoint	It Show trailing decimal point and zeros
ios::uppercase	
ios::skipus	It uses upper case letter for hex output
ios::unitbuf	It flushes all stream after insertion of data
ios::stdio	It flushes stdout and stderr after insertion

Note : Point to be noted for setf() function in ios class

1. The flags set by setf() remain effective until they are reset or unset.
2. A format flag can be reset any number of times in a program.
3. We can apply more than one format controls jointly on an output value.
4. The setf() sets the specified flags and leaves others unchanged.

Manipulators

Q13.What is a Manipulator ? How they can used to format the output?List Different manipulator and their equivalent function ios class?

Ans. Manipulators are operators used in C++ for formatting output. The data is manipulated by the programmer's choice of display.

There are numerous manipulators available in C++. Some of the more commonly used manipulators are provided here below:

endl Manipulator:

This manipulator has the same functionality as the '\n' newline character.

```
cout << "C++ Programming" << endl;
cout << "Training" << endl ;
cout << " By Manoj Kavedia";
```

output

```
C++ Programming
Training
By Manoj Kavedia
```

The header file `iomanip` provides a set of functions called manipulators which can be used to manipulate the output formats. They provide the same features as that of the `ios` member functions and flags. Some manipulators are more convenient to use than their counterparts in the class `ios`. For example, two or more manipulators can be used as a chain in one statement as shown below

```
cout<<manipulator1<< manipulator2<< manipulator3<<product;
cout<<manipulator1<< manipulator2<< manipulator3<<Amount;
```

This concatenation can be used when we want to display multiple column output.

Following table show most commonly used manipulators. To use those manipulator `<iomanip.h>` file is to be included.

Manipulator	Function	ios Eq ui val	Example
<code>Setw(int w)</code>	Set the Field width to w	<code>width()</code>	<code>cout<<set(10)</code>
<code>Setprecision(int d)</code>	Set the floating point precision to d	<code>precision()</code>	<code>cout<<setprecision(4)</code>
<code>Setfill(int c)</code>	Set the fill character to c	<code>fill()</code>	<code>cout<<setfill('&')</code>
<code>Setioflags(long f)</code>	Set the format flag to f	<code>setf()</code>	<code>cout<<setiosflags(ios::scientific)</code>
<code>resetioflags(long f)</code>	reset the format flag specified by f	<code>unsetf()</code>	<code>cout<<resetiosflags(ios::scientific)</code>
<code>endl</code>	Insert new line and	<code>"\n"</code>	<code>cout<<endl;</code>

	flush the stream		
--	---------------------------------	--	--

**Q14.Describe with example setw(),setfill(),setprecision manipulator used in c++?
Demonstrate the same with example**

Ans.setw()

Sets the number of characters to be used as the *field width* for the next insertion operation.

Behaves as if a call to the stream's member `ios_base::width` with *n* as its argument was made.

The *field width* determines the minimum number of characters to be written in some output representations. If the standard width of the representation is shorter than the field width, the representation is padded with *fill characters* at a point determined by the format flag **adjustfield** (**left**, **right** or **internal**).

```
Cout<<setw(10) << 45678;
```

This statement prints the value 12345 right-justified in a field width of 10 characters.

Program to demonstrate setw()

```
// setw example
#include <iostream.h>
#include <iomanip.h>
using namespace std;

int main ()
{
    cout << setw (10);
        cout << 77 << endl;
        return 0;
}
```

This code uses `setw` to set the *field width* to 10 characters.

setprecision()

Sets the *decimal precision* to be used by output operations.

Behaves as if a call to the stream's member `ios_base::precision` with *n* as argument was made.

```
setprecision(p);
```

The *decimal precision*(*p*) determines the maximum number of digits to be written on insertion operations to express floating-point values. How this is interpreted depends on whether the floatfield format flag is set to a specific notation (either fixed or scientific) or it is unset (using the *default notation*, which is neither fixed nor scientific):

- On the default floating-point notation, the precision field specifies the maximum number of meaningful digits to display in total counting both those before and those after the decimal point. Notice that it is not a minimum and therefore it does not pad the displayed number with trailing zeros if the number can be displayed with less digits than the *precision*.
- In both the fixed and scientific notations, the precision field specifies exactly how many digits to display after the decimal point, even if this includes trailing decimal zeros. The number of digits before the decimal point does not matter in this case.

```
// setprecision example
#include <iostream.h>
#include <iomanip.h>
using namespace std;

int main () {
    double f = 3.14159;
    cout << setprecision (5) << f << endl;
    cout << setprecision (9) << f << endl;
    cout << fixed;
    cout << setprecision (5) << f << endl;
    cout << setprecision (9) << f << endl;
    return 0;
}
```

The execution of this example displays something similar to:

```
3.1416
3.14159
3.14159
3.141590000
```

setfill() function

Sets the *fill character* to the value of parameter *ch*.

Setfill(ch)

Behaves as if a call to the stream's member `ios::fill` with *ch* as argument was made. The *fill character* is used in output insertion operations to fill spaces when results have to be padded to the *field width*.

```
// setfill example
#include <iostream.h>
#include <iomanip.h>
using namespace std;

int main () {
    cout << setfill ('x') << setw (10);
    cout << 77 << endl;
    return 0;
}
```

output

xxxxxxxx77

Q15.Describe with example setiosflag and resetioflag manipulator used in c++? Demonstrate the same with example

Ans. Set format flags

Sets the format flags specified by parameter *mask*.

```
setiosflags(ios_base::fmtflags mask)
```

Behaves as if a call to the stream's member `ios_base::setf` with *mask* as argument was made.

// setiosflags example

```
#include <iostream.h>
#include <iomanip.h>
using namespace std;

int main ()
{
    cout << hex << setiosflags (ios_base::showbase | ios_base::uppercase);
    cout << 100 << endl;
    return 0;
}
```

This code uses `setiosflags` manipulator to activate both the `showbase` and `uppercase` flags, with the same effect as if inserting manipulators `showbase` and `uppercase`.

Reset format flags

Unsets the format flags specified by parameter *mask*.

```
resetiosflags (ios_base::fmtflags mask);
```

This manipulator behaves like the stream's member `ios_base::unsetf`.

// resetiosflags example

```
#include <iostream.h>
#include <iomanip.h>
using namespace std;

int main () {
    cout << hex << setiosflags (ios_base::showbase);
    cout << 100 << endl;
    cout << resetiosflags (ios_base::showbase) << 100 << endl;
    return 0;
}
```

This code first sets the `showbase` flag and then resets it using the `resetiosflags` manipulator. The execution of this example displays something similar to:

```
0x64
64
```

Example based on Manipulator

```
cout << setw(5) << setprecision(2) << 1.2345 << setw(10) << setprecision(4) <<
sqrt(2) << setw(15) << setiosflags(ios::scientific) << sqrt(3) << endl;
```

will print all the three values in one line with the field sizes of 5, 10, and 15 respectively.

Example using bot ios flag and manipulator

```
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout << setprecision(4);
cout << setiosflags(ios::scientific);
cout << setw(10) << 123.45678;
```

Q16.State the difference between Manipulator and ios member function

Ans. There is a major difference in the way the manipulators are implemented as compared to the io member functions. The jog member function return the previous format state which can be used later, if necessary. But the manipulator does not return the previous format state. In case, we need to save the old format states, we must use the jog member functions rather than the manipulators.

Example

```
cout.precision(2);          // previous state
int p = cout.precision(4); // current state;
```

When these statements are executed, p will hold the value of 2 (previous state) and the new format state will be 4. We can restore the previous format state as follows:

```
cout.precision(p); // p = 2
```

Q17.Describe stream base class with example

Ans. For stream base we have

Operator/function	Brief description
hex	To set the base to hexadecimal, base 16.
oct	To set the base to octal, base 8.
dec	To reset the stream to decimal.
setbase()	Changing the base of the stream, taking one integer argument of 10, 8 or 16 for decimal, base 8 or base 16 respectively. setbase() is parameterized stream manipulator by taking argument, we have to include iomanip header file.

//Program example:

```
// using hex, oct, dec and setbase stream manipulator
#include <iostream.h>
using namespace std;
#include <iomanip>

void main(void)
{
    int p;

    cout<<"Enter a decimal number:"<<endl;
    cin>>p;
```

```

        cout<<p<< " in hexadecimal is: "<<hex<<p<<'\n'<<dec<<p<<" in octal is:  "
        <<oct<<p<<'\n'<<setbase(10) <<p<<" in decimal is:  "
        <<p<<endl;
        cout<<endl;
    }

```

output

```

Enter a decimal number:
365
365 in hexadecimal is: 16d
365 in octal is: 555
365 in decimal is: 365

```

Q18. Write Program to demonstrate the use of manipulators

```
#include <iostream.h>
```

```
using namespace std;
```

```
int main()
```

```
{
    cout.setf(ios::showpos);
    cout.setf(ios::scientific);
    cout << 123 << " " << 123.23 << endl;

```

```

    cout.precision(2);           // two digits after decimal point
    cout.width(10);              // in a field of 10 characters
    cout << 123 << " ";
    cout.width(10);              // set width to 10
    cout << 123.23 << endl;

```

```

    cout.fill('#');              // fill using #
    cout.width(10);              // in a field of 10 characters
    cout << 123 << " ";
    cout.width(10);              // set width to 10
    cout << 123.23;

```

```

    return 0;
}

```

User defined/own Manipulator

Q20. Describe with example how write user defined / own Manipulator

Ans. User can design its own manipulator if special cases as per the requirement of the output need. The general form for creating own manipulator is as shown

```

Ostream &manipulator(ostream &output)
{

```

```

-----
-----
-----
return output;

```

```

}
```

In above syntax

Manipulator – Name of the manipulator

Example

```

ostream & unit(ostream &output)
{
    output << " cms";
    return output;
}

```

use of manipulator

```

cout<<143<<unit;

```

output

143 cms

Manipulator can be to create some sequence of operation

```

Ostream &display (ostream &output)
{
    output.set(ios::showpoint );
    output.set(ios::showpos);
    output<<setw(10);
    return output;
}

```

This code for display manipulator will do following things when used

- It will turn on the flag showpoint
- It will turn on flag showpos
- Set the field width to 10

Program to demonstrate user defined Manipulator

```

#include<iostream.h>
#include<conio.h>
using namespace std;
//user defined manipulator
ostream &money(ostream &output)
{
    output<<"Rs";
    return output;
}

ostream &form(ostream &output)
{
    output.setf(ios::showpos);
    output.setf(ios::showpoint);
    output.fill('#');
    output.precision(2);
    output << setiosflags(ios::fixed)<<setw(10);
}

```



```

        return output;
    }
    int main()
    {
        cout<<money<<form<<1234.5;
    }

```

output

Rs+###1234.5

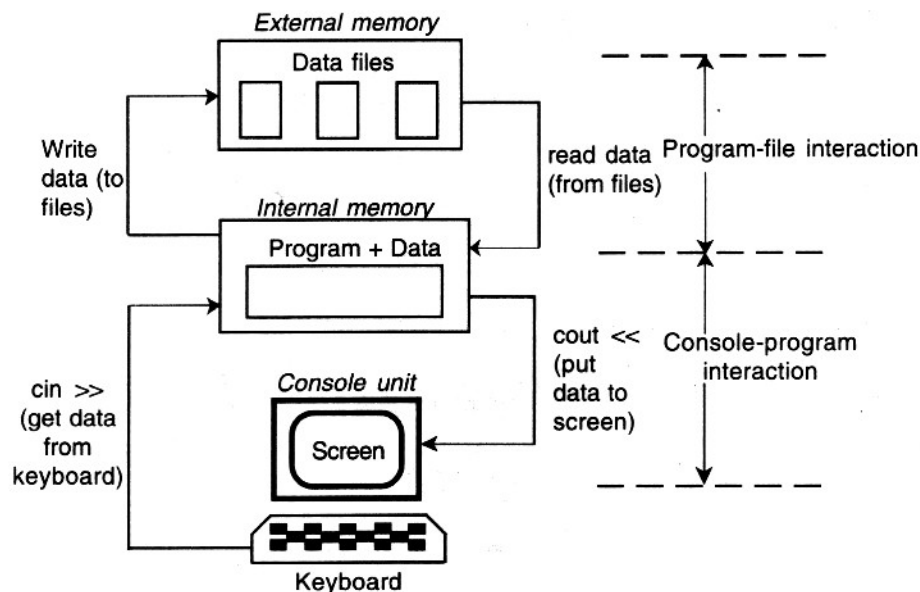
File handling

File System

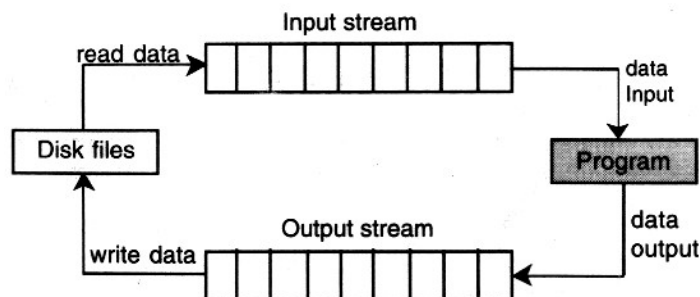
There are condition to handle large volumes of data and, in such situations, we need to use some devices such as floppy disk or hard disk to store the data. The data is stored in these devices using the concept of files. ***A file is a collection of related data stored in a particular area on the disk.*** Programs can be designed to perform the read and write operations on these files.

A program typically involves either or both of the following kinds of data communication

- 1.Data transfer between the console unit(keyboard and Display) and the program.



- 2.Data transfer between the program and a disk file.

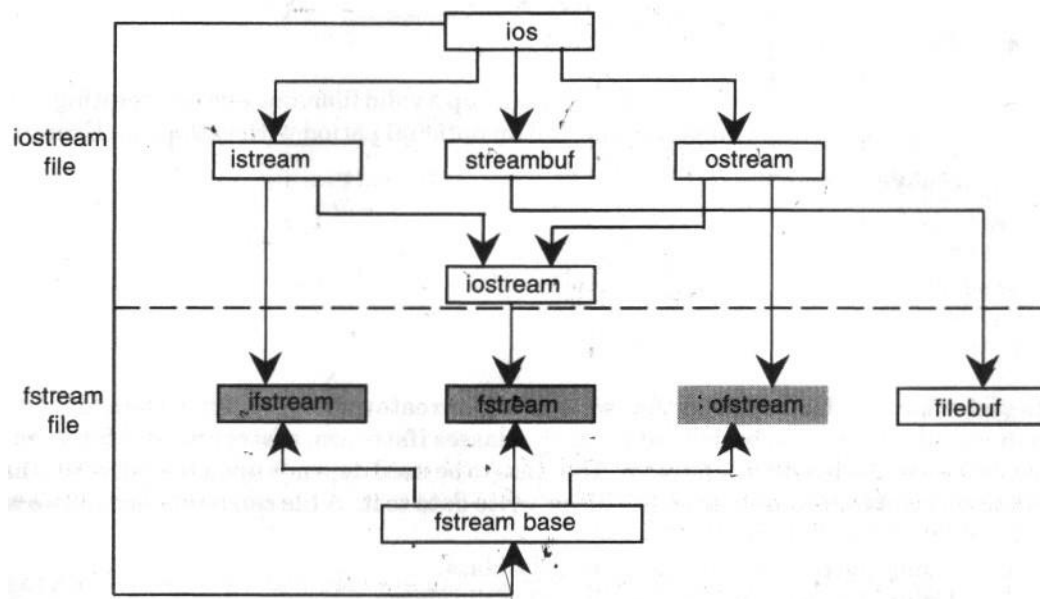


Q21.List and Describe the File stream classes needed for the file operation

Ans. The I/O system of C++ contains a set of classes that define the file handling methods. These method include

- ✓ **ifstream** → ifstream provides an interface to read data from files as input streams.
- ✓ **ofstream** → ofstream provides an interface to write data to files as output streams.
- ✓ **fstream** →fstream provides an interface to read and write data from files as input/output streams.

These classes are derived from fstreambase and from the corresponding iostream class as shown in Fig. These classes, designed to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.



Details of the class used for File Handling

Class	Function
filebuf	<p>This class applies the functionality of the streambuf class to read and write from/to files.</p> <p>By calling member open, a physical file is associated to the file buffer as its <i>associated character sequence</i>. Depending on the mode used in this operation, the access to the <i>controlled input sequence</i> or the <i>controlled output sequence</i> may be restricted.</p> <p>The state of the filebuf object -i.e. wether a file is open or not- may be tested by calling member function is_open.</p> <p>Internally, filebuf objects operate as defined in the streambuf class (see streambuf), with the particularity that a joint position pointer is maintained for both the input and the output controlled sequences.</p>

	The class overrides some virtual members inherited from streambuf to provide a specific functionality for files.
fstreambase	Provide operation common to the file streams. Serves as base class for fstream , ofstream , and ifstream class. Contain open() and close()
ifstream	ifstream provides an interface to read data from files as input streams. The objects of this class maintain internally a pointer to a filebuf object that can be obtained by calling member rdbuf. The file to be associated with the stream can be specified either as a parameter in the constructor or by calling member open. After all necessary operations on a file have been performed, it can be closed (or disassociated) by calling member close. Once closed, the same file stream object may be used to open another file. The member function is_open can be used to determine whether the stream object is currently associated with a file. Inherits the function get() , getline(),read().seekg(),and tellg() from istream.
ofstream	ofstream provides an interface to write data to files as output streams. The objects of this class maintain internally a pointer to a filebuf object that can be obtained by calling member rdbuf. The file to be associated with the stream can be specified either as a parameter in the constructor or by calling member open. After all necessary operations on a file have been performed, it can be closed (or disassociated) by calling member close. Once closed, the same file stream object may be used to open another file. The member function is_open can be used to determine whether the stream object is currently associated with a file. It inherits w), seekp(),tellp(),write function from ostream.
fstream	fstream provides an interface to read and write data from files as input/output streams. Inherits all the function from istream and ostream class through iostream.

	<p>The objects of this class maintain internally a pointer to a filebuf object that can be obtained by calling member <code>rdbuf</code>.</p> <p>The file to be associated with the stream can be specified either as a parameter in the constructor or by calling member open.</p> <p>After all necessary operations on a file have been performed, it can be closed (or disassociated) by calling member close. Once closed, the same file stream object may be used to open another file.</p> <p>The member function <code>is_open</code> can be used to determine whether the stream object is currently associated with a file.</p>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Opening and closing a file

Q22. List different ways to open and close the file

Ans. For opening a file following are the steps

1. Create a file stream.
2. Link the stream to file name.

A file stream can be created using `ifstream`, `ofstream`, `fstream` class which is include in header file `<fstream.h>`. Use of class depend on the purpose of file ie for read, write or to perform both operation. A file can be opened in two ways

1. Using Constructor - open single file
2. Using member function `open()` - open Multiple files

Q23. Describe the constructor method to open the file

Ans. A constructor can be used to initialize an object while it is being created. In this a filename is used to initialize the file stream object. This involves the following steps

1. Create a file stream object to manage the stream using the appropriate class. That is to say, the class of stream is used to create the output stream and the class `ifstream` to create the input stream.
2. Initialize the file object with the desired filename.

Example

```
ofstream  outputfile("Student.dat");    // for write operation ie output
ifstream  inputfile("Student.dat");     // for read operation ie input
```

program to demonstrate Writing into file using constructor

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

int main()
{
    ofstream SaveFile("ManojKavedia.txt");
    SaveFile << "Hello World, C++ training from Manoj kavedia";
    SaveFile.close();
}
```

```

        return 0;
    }

```

This program will create the file ManojKavedia.txt in the directory from where you are executing it, and will put “Hello World, C++ training from Manoj kavedia” into it.

Explanation of Program

#include - You need to include this file in order to use C++’s functions for File I/O. In this file, are declared several classes, including ifstream, ofstream and fstream, which are all derived from istream and ostream.

```
ofstream SaveFile("ManojKavedia.txt");
```

- 1) **ofstream** means “output file stream”. It creates a handle for a stream to write in a file.
- 2) **SaveFile** – that’s the name of the handle. It can be anything whatever you want!
- 3) (“ManojKavedia.txt”); - opens the file ManojKavedia.txt, which should be placed in the directory from where you execute the program.

SaveFile << " Hello World, C++ training from Manoj kavedia "; This line put the text in the file. Here SaveFile is a handle to the opened file stream. If we want to pass variables instead of text in inverted commas, just pass it as a regular use in following way

```
SaveFile << variablename;
```

SaveFile.close(); - SaveFile is an object from class ofstream, and this class (ofstream) has a function that closes the stream. That is the close() function. The name of the handle, dot and close(), will close the file stream

Notice: Once you have closed the file, you can’t access it anymore, until you open it again.

program to demonstrate reading file using constructor

```

#include <fstream.h>

void main() //the program starts here
{
    ifstream OpenFile("ManojKavedia.txt");
    char str[80];

    openfile>>str;

    cout<<"\nData in the file is \n";

    cout<<str;

    OpenFile.close();
}

```

ifstream OpenFile("ManojKavedia.txt ") - OpenFile is the object from class ifstream, which will handle the input file stream. And in the inverted commas, is the name of the file to open.

oprnfile>>str – will read the content of the file and store it in the array of character str

cout << str; - Display str which has the read data from the file.

File.close(); - The close() function, is used to close open file.

Output

Data in the file is

Hello World, C++ training from Manoj kavedia

Q24.What happen if the file to be open is already existing

Ans. When a file writing only, a new file is created if there no file of that name.If a file of that name exist,, already, then its contents a deleted and the file is presented as a clean file.

Q25.Describe the process of opening a file for reading and writing process with open() method

Ans. The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a stream object (an instantiation of one of these classes,) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open().open function can be used to open multiple files at a time.open() method is used in the following way

syntax

File-stream-class stream-object;

Stream-object.open("Filename");

Example

Ofstream ofile1,ofile2; //Create object of ofstream

Ofile1.open("Student.txt"); // connect stream ofile1 to students.txt

Ofile2.open("Marks.txt"); // connect stream ofile1 to Marks.txt

```
Ofile1.close();                // Disconnect stream from ofile1.txt
```

```
Ofile2.close();                // Disconnect stream from ofile2.txt
```

Program to read and write file using open method

```
// writing on a text file
#include <iostream.h>
#include <fstream.h>
using namespace std;

int main ()
{
    ofstream myfile ("example.txt");
    if (myfile.is_open())
    {
        myfile << "This is a line.\n";
        myfile << "This is another line.\n";
        myfile.close();
    }
    else cout << "Unable to open file";
    return 0;
}
```

output

```
[file example.txt]
This is a line.
This is another line.
```

```
// reading a text file
#include <iostream.h>
#include <fstream.h>
#include <string.h>
using namespace std;

int main ()
{
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while (! myfile.eof() )
        {
            getline (myfile,line);
            cout << line << endl;
        }
        myfile.close();
    }

    else cout << "Unable to open file";
}
```

```

        return 0;
    }

```

output

This is a line.
This is another line.

Detecting End of File (EOF)

Q26.Describe with example how end of file can be detected in C++

Ans.Detection of end of file is necessary to prevent any attempt to read the data from the file after all data is read.There are two ways of detecting the end of file

- 1.Using object of the ifstream
- 2.Using eof member function from ios class

use of ifstream object

An ifstream object returns a value of 0(zero) if any error in the file operation, or file pointer is actually at the end. Hence while can be used which will terminate when fin returns a value of zero(0) on reaching end of file condition.

```

while(fin)
{
    -----
}

```

use of eof of file member function

This is another approach to detect end of file. eof is the member function of ios class.It returns a non zero value if eof(end of file condition is encountered)and for normal operation it will return zero(0).

```

if ( fin.eof()!=0)
{
    exit(1);
}

```

File Modes

Q27.List and describe with example in different mode a file can be opened.

Ans. The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a stream object (an instantiation of one of these classes, in the previous example this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open().Open method has two arguments, whose syntax is as follows:

```

open (filename, mode);

```

Where

1. filename is a null-terminated character sequence of type `const char *` (the same type that string literals have) representing the name of the file to be opened, and
2. mode is an optional parameter with a combination of the following flags:

<code>ios::in</code>	Open for input operations.
<code>ios::out</code>	Open for output operations.
<code>ios::binary</code>	Open in binary mode.
<code>ios::ate</code>	Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
<code>ios::app</code>	All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
<code>ios::trunc</code>	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.
<code>ios::nocreate</code>	Open fails if the file does not exist
<code>ios::noreplace</code>	Open fails if the file already exists

Examples

All these flags can be combined using the bitwise operator OR (`|`). For example, if we want to open the file `example.bin` in binary mode to add data we could do it by the following call to member function `open()`:

```
ofstream myfile;
```

```
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each one of the `open()` member functions of the classes `ofstream`, `ifstream` and `fstream` has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
<code>ofstream</code>	<code>ios::out</code>
<code>ifstream</code>	<code>ios::in</code>
<code>fstream</code>	<code>ios::in ios::out</code>

To check if a file stream was successful opening a file, you can do it by calling to member `is_open()` with no arguments. This member function returns a `bool` value of `true` in the case that indeed the stream object is associated with an open file, or `false` otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Q28. List the some important point about file mode

Ans.

1. Opening a file in `ios::out` mode also opens it in the `ios::trunc` mode by default.
2. Both `ios::app` and `ios::ate` take us to the end of the file when it is opened.

3. The difference between the two parameters is that the `ios::app` allows us to add data to the end of the file only, while `ios::ate` mode permits us to add data or to modify the existing data anywhere in the file.
4. In both the cases, a file is created by the specified name, if it does not exist.
5. The parameter `ios::app` can be used only with the files capable of output.
6. Creating a stream using `ifstream` implies input and creating a stream using `ofstream` implies output. So in these cases it is not necessary to provide the mode parameters.
7. The `fstream` class does not provide a mode by default and therefore, we must provide the mode explicitly when using an object of `fstream` class.
8. The mode can combine two or more parameters using the bitwise OR operator (symbol `|`) shown as follows:
`foutput.open("data.txt",ios::app|ios::nocreate)`
9. The above syntax opens the file in the append mode but fails to open the file if it does not exist.

File Pointer Manipulation

Q29. Describe the file pointer and manipulators used for file handling

Ans. All i/o streams objects have, at least, one internal stream pointer: `ifstream`, like `istream`, has a pointer known as the *get pointer* that points to the element to be read in the next input operation.

`ofstream`, like `ostream`, has a pointer known as the *put pointer* that points to the location where the next element has to be written.

Finally, `fstream`, inherits both, the *get* and the *put* pointers, from `iostream` (which is itself derived from both `istream` and `ostream`).

When we open a file in read-only mode, the input pointer is automatically set at the beginning so that we can read the file 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. This enables us to write to the file from the start. In case, we want to open an existing file to add more data, the file is opened in append mode. This moves the output pointer to the end of the file.

Function of Manipulator of File pointer

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

tellg() and tellp()

These two member functions have no parameters and return a value of the member type `pos_type`, which is an integer data type representing the current position of the *get* stream pointer (in the case of `tellg`) or the *put* stream pointer (in the case of `tellp`).

seekg() and seekp()

These functions allow us to change the position of the *get* and *put* stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

seekg (position);
seekp (position);

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos_type, which is an integer value.

The other prototype for these functions is:

seekg (offset, direction);
seekp (offset, direction);

Using this prototype, the position of the get or put pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from, and that can take any of the following values:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position of the stream pointer
ios::end	offset counted from the end of the stream

The following example uses the member functions we have just seen to obtain the size of a file:

```
// obtaining file size
#include <iostream.h>
#include <fstream.h>
using namespace std;

int main ()
{
    long begin,end;
    ifstream myfile ("example.txt");
    begin = myfile.tellg();
    myfile.seekg (0, ios::end);
    end = myfile.tellg();
    myfile.close();
    cout << "size is: " << (end-begin) << " bytes.\n";
    return 0;
}
```

Output

size is: 40 bytes.

Reading and Writing in Binary File

Q30.Describe how to use read() and write function to perform operation on file/Binary Files

Ans. In binary files, to input and output data with the extraction and insertion operators (<< and >>) and functions like getline is not efficient, since we do not need to format any data, and data may not use the separation codes used by text files to separate elements (like space, newline, etc...).

File streams include two member functions specifically designed to input and output binary data sequentially: write and read. The first one (write) is a member function of ostream inherited by ofstream. And read is a member function of istream that is inherited by ifstream. Objects of class fstream have both members. Their prototypes are:

```
write ( memory_block, size );  
read  ( memory_block, size );
```

Where memory_block is of type "pointer to char" (char*), and represents the address of an array of bytes where the read data elements are stored or from where the data elements to be written are taken. The size parameter is an integer value that specifies the number of characters to be read or written from/to the memory block.

The binary format is more accurate for storing the number as they are stored in the exact internal representation. There are no conversion needed while saving the data and therefore saving is much more faster.

Example

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

Program to demonstrate Reading of Binary File

```
// reading a complete binary file  
#include <iostream.h>  
#include <fstream.h>  
using namespace std;  
  
ifstream::pos_type size;  
char * memblock;  
  
int main ()  
{  
    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 complete file content is in  memory";  
  
        delete[] memblock;  
    }  
}
```

```

        else cout << "Unable to open  file";
        return 0;
    }

```

Output

the complete file content is in memory

Explanation of program

First, the file is open with the `ios::ate` flag, which means that the get pointer will be positioned at the end of the file. This way, when we call to member `tellg()`, we will directly obtain the size of the file. Notice the type we have used to declare variable `size`:

```

    ifstream::pos_type size;

```

`ifstream::pos_type` is a specific type used for buffer and file positioning and is the type returned by `file.tellg()`. This type is defined as an integer type, therefore we can conduct on it the same operations we conduct on any other integer value, and can safely be converted to another integer type large enough to contain the size of the file. For a file with a size under 2GB we could use `int`:

```

    int size;
    size = (int) file.tellg();

```

Once we have obtained the size of the file, we request the allocation of a memory block large enough to hold the entire file:

```

    memblock = new char[size];

```

Now proceed to set the get pointer at the beginning of the file (remember that we opened the file with this pointer at the end), then read the entire file, and finally close it:

```

    file.seekg (0, ios::beg);
    file.read (memblock, size);
    file.close();

```

At this point we could operate with the data obtained from the file. Our program simply announces that the content of the file is in memory and then terminates.

Character by character reading/writing in a file

Q31.Describe how `get()` and `put()` can be used to perform read and write operation on the file

Ans.The `get()` and `put()` function are used handle single character in a file at a time. The function `put()` writes a single character to the associated stream. Similarly, the function `get()` reads a single character from the associated stream.

Program to demonstrate is of `get` and `put` function

```

// use of get and put function
#include <iostream.h>
#include <fstream.h>
using namespace std;

int main ()
{
    char ch;

```

```

ofstream outfile ("test.txt");

do {
    ch=cin.get();           // get character from a keyboard.
    outfile.put (ch);       // write character to file
} while (ch!='. ');

return 0;
}

```

Q32. Give the meaning for the following

Ans.

```

fseek.seekg(0,ios::beg) go to start
fseek.seekg(0,ios::cur) stay at the current position
fseek.seekg(0,ios::end) go to end of file
fseek.seekg(m,ios::beg) move to (m+1)th byte in the file
fseek.seekg(m,ios::cur) go forward by m byte from current position
fseek.seekg(-m,ios::cur) go backward by m byte from current position
fseek.seekg(-m,ios::end) go backward by m byte from end

```

Command Line Argument

Q33. Describe command line Argument in C++

Ans. In C++ it is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system. In fact, main can actually accept two arguments:

- one argument is number of command line arguments, and
- the other argument is a full list of all of the command line arguments.

Syntax

```
int main ( int argc, char *argv[] )
```

The integer, argc is the argument Count (hence argc). It is the number of arguments passed into the program from the command line, including the name of the program.

The array of character pointers is the listing of all the arguments. argv[0] is the name of the program, or an empty string if the name is not available. After that, every element number less than argc are command line arguments. We can use each argv element just like a string, or use argv as a two dimensional array. argv[argc] is a null pointer.

Program to demonstrate command Line Argument in C++

```

#include <fstream.h>
#include <iostream.h>

```

```
using namespace std;
```

```

int main ( int argc, char *argv[] )
{
    if ( argc != 2 ) // argc should be 2 for correct execution
        // argv[0] program name
        cout<<"usage: "<< argv[0] <<" <filename>\n";
}

```

```

else {
    // argv[1] is a filename to open -source
    ifstream the_file ( argv[1] );
    // check to for file opening succeeded
    if ( !the_file.is_open() )
        cout<<"Could not open file\n";
    else {
        char x;
        // the_file.get ( x ) returns false if the end of the file
        // is reached or an error occurs
        while ( the_file.get ( x ) )
            cout<< x;
    }
    // the_file is closed implicitly here
}
}

```

Explanation

it first checks to ensure the user added the second argument, theoretically a file name. The program then checks to see if the file is valid by trying to open it. This is a standard operation that is effective and easy. If the file is valid, it gets opened in the process.

Q34.What is the difference between ios::app and ios::ate?

Ans. : Both ios::app and ios::ate take us to the end of file, when it is opened. But the difference between the two parameters is that the ios::app allows us to add data to the end of file only, while ios::ate mode permits us to add data or to modify the existing data anywhere in the file. In both the cases, a file is created by the specified name, if it does not exist.

Q35.Describe the built-in function for reading a line of text from a file.

Ans. : The get and get line functions are used for reading a line of text from a file. The get member function taken three arguments a character array, a size limit and delimiter with default value '\n'. This version reads character and terminates or terminates as soon as the delimiter read.

The getline member function is also used. It inserts a null character after the line in a character away. The getline function removes the delimiter from the stream but does not store it in the character array.

Q36. What is the use of following function.

i. fputs () ii. fgets () iii. fputc () iv. Fgetc ()

Ans. : i. fputs () : It writes strings to an opened file.

Syntax :

int fputs (s, Fp)

S is the string we want to write into a file pointed by file. pointer Fp.

ii. **fgets ()** : It reads the contents of an opened file.

Syntax

char fgets (S, length of a string, Fp)

First argument to this functions is a string, which we want to write in a file.

Second argument is maximum length of string.

Third argument is file pointer, pointing to the opened file.

iii. for reading and writing a character from a file there are, fputc () and fgetc () functions.

Syntax

fputc (ch, fp)

ch is a character we want to store in a file and fp is a pointer, which is opened file

Syntax

fgetc (fp)

fp is a pointer which points to opened file This function is used for reading the contents i.e. a character from an opened file.

Q38. Write error handling functions and their meaning.

Ans. : The available error handling functions related to files are as follows :

- i) **eof ()** : It returns TRUE if end-of-file is encountered while reading.
- ii) **fail ()** : returns TRUE when input or output operations failed.
- iii) **bad()** : returns TRUE if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is FALSE, it may be possible to recover from any other error reported and continue the operation.
- iv) **good ()** : It returns TRUE if no error has occurred. This means, all above functions are false. If it returns FALSE, no further operation associated stream and read () is another function which is able to read data stored in binary form.

Program based on ios member function , Manipulator , File handling

Program-1

Program to demonstrate user of showpoint, controlling the trailing zeroes and floating points

```
// using showpoint, controlling the trailing zeroes and floating points
#include <iostream.h>
#include <iomanip.h>
using namespace std;

void main(void)
{
    cout<<"Before using the ios::showpoint flag\n"
```



```

        <<"-----"<<endl;
        cout<<"cout prints 88.88000 as: "<<88.88000
        <<"\ncout prints 88.80000 as: "<<88.80000
        <<"\ncout prints 88.00000 as: "<<88.00000
        <<"\n\nAfter using the ios::showpoint flag\n"
        <<"-----"<<endl;
        cout.setf(ios::showpoint);
        cout<<"cout prints 88.88000 as: "<<88.88000
        <<"\ncout prints 88.80000 as: "<<88.80000
        <<"\ncout prints 88.00000 as: "<<88.00000<<endl;
    }

```

output

```

Before using the ios::showpoint flag
cout prints 88.88000 as: 88.88
cout prints 88.80000 as: 88.8
cout prints 88.00000 as: 88

After using the ios::showpoint flag
cout prints 88.88000 as: 88.8800
cout prints 88.80000 as: 88.8000
cout prints 88.00000 as: 88.0000

```

Program-2

Write program to demonstrate use of setw(), setiosflags(), resetiosflags() manipulators and setf and unsetf member functions

Ans.// use of setw(), setiosflags(), resetiosflags() manipulators and setf and unsetf member functions

```

#include <iostream.h>
#include <iomanip.h>
using namespace std;

void main(void)
{
    long p = 123456789L;
    // L - literal data type qualifier for long...
    // F - float, UL unsigned integer...
    cout<<"The default for 10 fields is right justified:\n"
        <<setw(10)<<p
        <<"\n\nUsing member function\n"
        <<"-----\n"
        <<"\nUsing setf() to set ios::left:\n"<<setw(10);
    cout.setf(ios::left,ios::adjustfield);
    cout<<p<<"\nUsing unsetf() to restore the default:\n";
    cout.unsetf(ios::left);
    cout<<setw(10)<<p
        <<"\n\nUsing parameterized stream manipulators\n"
        <<"-----\n"
        <<"\nUse setiosflags() to set the ios::left:\n"
        <<setw(10)<<setiosflags(ios::left)<<p
        <<"\nUsing resetiosflags() to restore the default:\n"

```

```

        <<setw(10)<<resetiosflags(ios::left)
        <<p<<endl;
    }

```

Output:

```

The default for 10 fields is right justified:
123456789

Using member function
-----

Using setf() to set ios::left:
123456789
Using unsetf() to restore the default:
123456789

Using parameterized stream manipulators
-----

Use setiosflags() to set the ios::left:
123456789
Using resetiosflags() to restore the default:
123456789

```

Program-3

Program to demonstrate setw(), setiosflags(), showpos and internal

Ans. //use of setw(), setiosflags(), showpos and internal

```

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

void main(void)
{
    cout<<setiosflags(ios::internal
ios::showpos)<<setw(12)<<12345<<endl;
}

```

Output:

```

+      12345

```

Program-4

Program to demonstrate use of fill() member function and setfill() manipulator

Ans.

```

// using fill() member function and setfill() manipulator
#include <iostream.h>
#include <iomanip.h>
using namespace std;

void main(void)
{
    long p = 30000;

    cout<<p

```

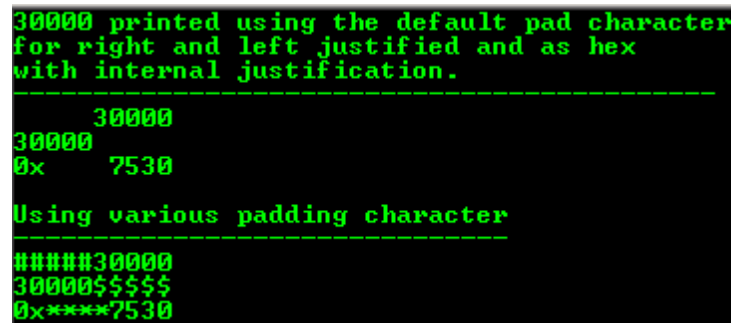
```

        <<" printed using the default pad character\n"
        <<"for right and left justified and as hex\n"
        <<"with internal justification.\n"
        <<"-----\n";
    cout.setf(ios::showbase);
    cout<<setw(10)<<p<<endl;
    cout.setf(ios::left,ios::adjustfield);
    cout<<setw(10)<<p<<endl;
    cout.setf(ios::internal,ios::adjustfield);
    cout<<setw(10)<<hex<<p<<"\n\n";

    cout<<"Using various padding character"<<endl;
    cout<<"-----"<<endl;
    cout.setf(ios::right,ios::adjustfield);
    cout.fill('#');
    cout<<setw(10)<<dec<<p<<'\n';
    cout.setf(ios::left,ios::adjustfield);
    cout<<setw(10)<<setfill('$')<<p<<'\n';
    cout.setf(ios::internal,ios::adjustfield);
    cout<<setw(10)<<setfill('*')<<hex<<p<<endl;
}

```

Output:



```

30000 printed using the default pad character
for right and left justified and as hex
with internal justification.
-----
      30000
30000
0x    7530

Using various padding character
-----
#####30000
30000$$$$$
0x*****7530

```

Program-5

Program to demonstrate use of displaying floating number in system default, scientific and fixed format

Ans.

```

// displaying floating number in system default, scientific and fixed format
#include <iostream.h>
using namespace std;

void main(void)
{
    double p = 0.000654321, q = 9.8765e3;
    cout<<"Declared variables\n" <<"-----\n"
        <<"0.000654321"<<'\n'<<"9.8765e3"<<"\n\n";
    cout<<"Default format:\n" <<"-----\n" <<p<<'\t'<<q<<'\n'<<endl;
    cout.setf(ios::scientific,ios::floatfield);
    cout<<"Scientific format:\n" <<"-----\n" <<p<<'\t'<<q<<'\n';
    cout.unsetf(ios::scientific);
    cout<<"\nDefault format after unsetf:\n" <<"-----\n"

```

```

        <<p<<'t'<<q<<endl;
    cout.setf(ios::fixed,ios::floatfield);
    cout<<"\nIn fixed format:\n" <<"-----\n" <<p<<'t'<<q<<endl;
}

```

Output:

```

Declared variables
-----
0.000654321
9.8765e3

Default format:
-----
0.000654321      9876.5

Scientific format:
-----
6.543210e-04      9.876500e+03

Default format after unsetf:
-----
0.000654321      9876.5

In fixed format:
-----
0.000654      9876.500000

```

Program-6

Program to demonstrate use of ios::uppercase flag

Ans.

```

// using ios::uppercase flag
#include <iostream.h>
#include<iomanip.h>
using namespace std;

void main(void)
{
    long p = 12345678;

    cout<<setiosflags(ios::uppercase)
        <<"Uppercase letters in scientific\n"
        <<"notation-exponents and hexadecimal values:\n"
        <<"-----\n"
        <<5.7654e12<<'n'
        <<hex<<p<<endl;
}

```

Output:

```

Uppercase letters in scientific
notation-exponents and hexadecimal values:
-----
5.7654e+12
BC614E

```

Program-7

Program to demonstrate for displaying floating number in system default, scientific and fixed format

Ans.

```
// displaying floating number in system
// default, scientific and fixed format
#include <iostream.h>
using namespace std;

void main(void)
{
    double p = 0.000654321, q = 9.8765e3;

    cout<<"Declared variables\n" <<"-----\n"
    <<"0.000654321"<<"\n"<<"9.8765e3"<<"\n\n";

    cout<<"Default format:\n"
        <<"-----\n"
        <<p<<"\t"<<q<<"\n"<<endl;

    cout.setf(ios::scientific,ios::floatfield);
    cout<<"Scientific format:\n"
        <<"-----\n"
        <<p<<"\t"<<q<<"\n";

    cout.unsetf(ios::scientific);
    cout<<"\nDefault format after unsetf:\n"
        <<"-----\n"
        <<p<<"\t"<<q<<endl;
    cout.setf(ios::fixed,ios::floatfield);
    cout<<"\nIn fixed format:\n"
        <<"-----\n"
        <<p<<"\t"<<q<<endl;
}
```

Output:

```
Declared variables
-----
0.000654321
9.8765e3

Default format:
-----
0.000654321      9876.5

Scientific format:
-----
6.543210e-004    9.876500e+003

Default format after unsetf:
-----
0.000654321      9876.5

In fixed format:
-----
0.000654          9876.500000
```

Program-8

Program: Writing and reading data into file, using constructors

```
# include <iostream.h>
#include <conio.h>
#include <fstream.h>
void main()
{
    ofstream  outfile("sample.txt");                // create file for output
    char ch = 'a';
    int i = 12;
    float f = 4356.15;
    char arr[ ] = "hello";
    outfile << ch << endl <<< endl << f << endl << arr;    //send the data to file
    outfile.close();

    ifstream infile("sample.txt");
        infile >> ch >> i >> f >> arr;                // read data from file
        cout << ch << i << f << arr;                    // send data to screen
}
```

To write data into the file, character by character.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
    char str[]="C++ is superset of C. It is an object-oriented /
programming language.";

    ofstream outfile("sample2.txt"); // Open the file in write mode

    for(int i = 0; i < strlen(str); i++)
```

```

outfile.put(str[i]); // write data into the file, character by character.
}

```

Program-9

Program to counts the number of objects already written into the file “Person.txt”. Then is reads the second object and displays the values of its data members

```

#include<iostream.h>
#include<conio.h>
#include<string.h>
class person
{
    private:
        char name[40];
        int age;
    public:
        void showData()
        {
            cout << "\n Name = " << name;
            cout << "\n Age = " << age;
        }
};

void main()
{
    person pers; // create person object
    ifstream infile; // create input file
    infile.open("Person.txt"); // open the file
    infile.seekg(0, ios::end); // go to end from 0 byte
    int endposition = infile.tellg(); // find where we are
    int n = endposition/sizeof(person); // number of persons
    cout << "\n There are " << n << " persons in file: ";
    cout << "\n Enter person number: ";
    cin >> n;
    int position = (n-1) * sizeof(person); // number times size
    infile.seekg(position);
    infile.read( (char*)&pers, sizeof(pers) );
    pers.showData(); // display the person
}

```

Program-10

Program to Binary file operation on file

```

#include<iostream.h>
#include<conio.h>
#include<string.h>
const char *filename = "Binary.dat";

void main()

```

```

{
    float height[4] = {101.01 , 202.02 , 303.03, 404.04};
    ofstream ofile;
    ofile.open(filename);           // open file for writing
    ofile.write( (char *) &height , sizeof(height)); //write array to file
    ofile.close();                  //close the file

    for(int i=0;i<len;i++)          // clear the array
    {
        height[i] = 0;
    }

    ifstream infile;
    infile.open(filename);          // open file for reading
    infile.read((char *) & height , sizeof(height)); //read file content

    for(i=0;i<4;i++)               // display content on screen
    {
        cout.setf(ios::showpoint);
        cout<<setw(10)<<setprecision(2)<<height[i];
    }
    infile.close();
    getch();
}

```

Program-11

Program to read and write character in a file

//Character by character read /write operation on file

```

#include<iostream.h>
#include<conio.h>
#include<string.h>
int main()
{
    char str[80];
    char ch;
    fstream fileio;                // create fstream object
    cout<<"\n Enter a String \n";
    cin>>str;                      // read the string to be written
    int len = strlen(str);
    fileop.open("Data.txt", ios::in|ios::out); // open file in IO mode

    for (int i=0;i<len;i++)
    {
        fileio.put(str[i]);        //write character in file
    }
    file.seekg(0);                 //reset point to start of file
    cout<<"\n Data in the file is ";
}

```



```

        while(fileio)
        {
            fileio.get(ch);           // read character from file
            cout<<ch;                 // display on screen
        }

        return 0;
    }

```

program-12

Data input from a file can also be performed in the same way that we did with cin:

```

// reading a text file
#include <iostream.h>
#include <fstream.h>
#include <string.h>
using namespace std;

int main ()
{
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while (! myfile.eof() )
        {
            getline (myfile,line);
            cout << line << endl;
        }
        myfile.close();
    }

    else cout << "Unable to open  file";

    return 0;
}

```

This is a line.
This is another line.

Program-13

Program to copy file in c++

```

#include <fstream.h>
int copyFile (const char SRC[], const char  DEST[])
{
    ifstream  src;           // the source file
    ofstream  dest;          // the destination file

```

```

src.open (SRC, ios::binary); // open in binary to prevent jargon at the end of
                                //the buffer
dest.open (DEST, ios::binary); // same again, binary
if (!src.is_open() || !dest.is_open())
    return 0; // could not be copied

dest << src.rdbuf (); // copy the content
dest.close (); // close destination file
src.close (); // close source file

return 1; // file copied successfully
}

```

/ Example Usage **/**

```

#include <iostream.h>

int main ()
{
    if (!copyFile ("C:\\Manoj.txt", "C:\\kavedia.txt"))
        cout << "File could not be copied successfully";

    else
        cout << "File copied successfully!";

    cin.get (); // pause for input
    return 0 // program was executed successfully
}

```

program-14

Write a program for creating a text file and then count the number of lines in that file.

```

// reading a text file
#include <iostream.h>
#include <fstream.h>
#include <string.h>
using namespace std;

int main ()
{
    int line;
    char ch;
    if (argc <= 1)
    {
        cout << "\n File name no specified";
        exit(0);
    }
    line =0;
}

```

```

ifstream myfile (argv[2]);
if (myfile.is_open())
{
    while (! myfile.eof() )
    {
        if (myfile.get(ch)=='\n')
        {
            line++;
        }
    }
    myfile.close();
}
else cout << "Unable to open  file";
cout << "\n Number of Lines in file = "<< line  ;

return 0;
}

```

Program-15

Write a program to copy contents of one file to other

Program to copy file in c++

```

#include <iostream.h>
void main ()
{
    char ch;

    ifstream  src;           // the source file
    ofstream  dest;          // the destination file

    if (argc <3 )
    {
        cout << "\n Either source of destination file name is missing";
    }
    else
    {
        src.open (argv[1], ios::binary); // open in binary to prevent jargon at the end of
        //the buffer
        dest.open (argv[2], ios::binary); // same again, binary
        if (!src.is_open() || !dest.is_open())
            return 0; // could not be copied

        while(src.eof ==0) //copy content of one file to  another
        {
            src.get(ch);
            destput(ch);
        }
        cout<<"\n File copied Sucessfully";
    }
}

```

```

        dest.close ();                // close destination file
        src.close ();                // close source file
    }                                // end of if -else

    return 0                          // program was executed successfully
}

```

Program-16

```

#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <sstream.h>

using namespace std;

int main()
{
    string fileName;
    char c;
    cout << endl;
    cout << "Enter the name of the file: ";
    cin >> fileName;
    cout << endl << endl;

                                                                    //declares filename
    ifstream d_file;
    d_file.open(fileName.c_str());    //attempts to open file

    if (!d_file.is_open())
    {
                                                                    //if file doesn't exist; don't create a new one
        cout << "File" << fileName << " does not exist in the client's current directory"
        << endl << "Press any key to continue";
        getch();
        exit(1);
    }
    else
    {
        d_file.get(c);
        d_file.seekg(0, ios::end);
        int charCount = d_file.tellg();
        d_file.seekg (0, ios::beg);
                                                                    //prints the number of characters in a file;
        cout << "the number of characters in the " << fileName;
        cout << " is " << charCount << endl;

                                                                    //prints the number of lines in a file
        string t;

```

```

int lineCount=0;
while(getline(d_file, t, '\n'))
    ++lineCount;

cout << "The number of lines in the file is " <<
lineCount << endl;

//reads the number of lines in a file, then print
int wordCount=0;
string word;

for (;;)
{
    d_file >> word;
    if ( d_file.eof() )
    {
        break;
    }
    wordCount++;
}
cout << wordCount << endl;
}
d_file.close();
cout<<"\Press any key to continue ";
return 0;
}

```

program-17

**Write program to read number of character , tabs , spaces , lines in a file
Accept file name as command line parameter**

Ans.

```

// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main ()
{
    int line , tab , cha , space;
    char ch;
    if (argc <= 1)
    {
        cout << "\n File name no specified";
        exit(0);
    }
    line= space=tab=cha=0;
    ifstream myfile (argv[2]);
    if (myfile.is_open())
    {

```

```

while (! myfile.eof() )
{
    switch (myfile.get(ch))
    {
        case '\n' :
            line++;break;
        case '\t' :
            tab++; break;

        case ' ' :
            space++;break;
        default:
            cha++;
    }
    // end of switch

}
// end of while
myfile.close();
//close file
//end of if
}

else cout << "Unable to open file";
cout<<"\n content Analysis of the file";
cout <<"\n Line = "<<line ;
cout<<"\n Spaces ="<<space;
cout<<"\n Tabs  ="<<tab;
cout<<"\n Characters ="<<cha;

return 0;
}

```

program - 18

Write a program to read and display a record for employee_id = 1529 from a file containing record with field employee_id,name and basic - salary.

Ans. : Suppose a file emp.dat contains the record of the employee. Following program will search and display the a record of employee_id = 1529.

```

# include <fstream.h>
# include <conio.h>
class emp
{

    int employee_id ;
    char name [20] ;
    float bs ;
public:
    void showdata ()
    {

```

```

        cout << "employee_id" << employee_id << endl ;
        cout << "Name :" << name << endl  ;
        cout << "basic salary :" << bs << endl ;
    }
};

void main ()
{
    fstream fp;
    char choice ;
    fp.open ("emp.dat" ; ios :: in)
    emp e ;
    while (fp)
    {
        fp. read ((char*) & e, size of (emp)) ;
        if (e. employee_id == 1529)
            e. showdata () ;
    }
    f. close () ;
}

```

program-19

Write a program to accept a string from user and write it into file. Display the string by reading it from the same file.

Ans. : # include <iostream.h>

#include <conio.h>

void main ()

```

{
    ofstream outf ("string") ;           // open file in write mode
    cout << "Enter your name"  ;
    char name [30] ;                     //read the name
    cin >> name ;
    outf << name << " \n" ;             // write name in file
    outf.close () ;                      //close the file
    ifstream inf ("string") ;
    inf >> name ;                         // read name from file
    cout << "your Name =" << name ; //display the name
    inf.close () ;
    getch();
}

```

program-20

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

Ans. : # include <iostream.h>

```

void main ( )
{
    ifstream fin ;
    fin.open ("filename");
    int size = 200 ;
    int nlines = 0 ;
    char line [200] ;
    while (fin)
    {
        fin.getline(line,size)
        no lines ++ ;
    }
    cout << "no of lines" << no  lines;
}

```

Program-20

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

Ans. : `_# include <iostream.h>`

`# include <fstream.h>`

`# include <string.h>`

`# include <conio.h>`

`void main ()`

```

{
    fstream file ;
    file . open ("TEXT", ios :: in) ;    //open the file
    char ch ;
    int lcount = 0 ;
    file . seekg (0) ;
    while (file)                        // repeat till end of file is  reached
    {
        file.get(ch) ;                  //get character form file
        if (ch == '\n')
            l count + +                // count line number
    }
    cout << "No. of lines in a file = " << lcount ;
    file . close () ;
}

```

Program -21

Write a C++ program to read a set of numbers form the standard input device and sort them in ascending order.


```

Ans. : #include<iostream.h>
#include<conio.h>
main()
{
    int a[100];
    int i, j, n, temp;
    cout << "How many numbers ?" << endl;
    cin >> n; cout << "Enter the elements value" << endl;
    for (i = 0; i <= n - 1; ++i)
    {
        cin >> a [i];
    }
    for (i = 0; i <= n - 1; ++j)
    {
        for (j = 0; j <= n - 1, ++j)
        {
            if (a [i] < a [j])
            {
                temp = a [i ];
                a [i] = a [j];
                a [j] = temp;
            }
        }
    }
    cout << "Ascending order" << endl;
    for (i = 0; i <= n - 1; ++i)
    {
        cout << a [i] << '\t';
    }
    cout << endl;
    getch();
}

```

Write a program in C++, to copy contents of first file to other using concept of command Line Arguments.

Ans. : Command Line Arguments :

```
# include <iostream.h>
# include <process.h>           // for exit
# include <fstream.h>          // for ifstream & ofstream
main (int argc, char* argv [ ] )
{
    char ch;
    if (argc != 3)
    {
        cout << "Syntax is : copyf file1 file2 \n" ;
        exit(1);
    }
    ifstream srcfile;
    ofstream dstnfile;
    srcfile.open (argv [1], ios :: nocreate);
    if (!srcfile)
    {
        cout << "cannot open source file" << argv [1] << "for input \n";
        exit(1);
    }
    dstnfile.open (argv [2]);
    if (!dstnfile)
    {
        cout<< "cannot open destination file"<<argv [2]<< "for output \n";
        exit(1);
    }
    while (srcfile)
    {
        src.get (ch);
        dest.put (ch);
    }
    cout << "copyf completed \n" ;
    srcfile.close ();
```

```

        dstnfile.close ();
    getch();
}

```

Practice Questions and Answers

Question1

- Write a program copies contents of one file into another file
- Explain use of following function with suitable example
 - setw()
 - setfill()
- Explain use of following function and ios::noreplace
- Describe built in function for reading a line of text from file. Give example

Question2

- What will be the output of following program

```

int main()
{
    cout << fill('<')
    cout << precision(3)
    for(int n=1;n<=6;n++)
    {
        cout << width(5);
        cout << n;
        cout << width(10);
        cout << 1.01 << float(n) << "\n";
        if ( n==3)
            cout << fill('>');
    }
    cout << fill('=')
    cout << width(15);
    cout << 12.345678;
    return 0;
}

```

- Give syntax and use of following with respect to file
- How we open file using constructor?
- Explain different file opening mode in details

Question 3

- List any two input and two output manipulators.
- Explain ios::in and ios::out with example. Write any four ios format functions.
- Write a program for creating a text file and then count the number of lines in that file.**
- State the syntax of seek () and tell () function related to file.
- Give syntax and use of
 - Open () function
 - Close () function

Question 4

- a. Explain the uses of following
 - i) substr ()
 - ii) find ()
 - iii) insert()
 - iv) empty()
- b. Mention and explain any four file manipulator function.
- c. Write the syntax for following
 - i) Opening file using open()
 - ii) Closing a file using close()
- d. Explain the following
 - (i) ifstream()
 - (ii) ofstream ()
 - (iii) write ()
 - (iv) read ()

Question 5

- a. Write a program to copy contents of one file to other.
- b. What is file mode? List various file mode operation available.
- c. Write built in function for reading a line of text from file.

Question 6

- a. Write a program to count the number of lines in file.
- b. Define manipulators. List any four manipulators.
- c. Give syntax and use of following
 - (i) open
 - (ii) close
- d. Explain ios::in & ios::out with suitable example.