3 CHAPTER Input and Output in C++

CHAPTER OUTLINE

3.1 INTRODUCTION

Computer applications generally involve tremendous amount of data to be read from input devices and sending them to the output devices. Hence, to control such operations, every programming language provides a set of inbuilt functions. C++ supports all input/output (I/O) functions of C. C++ also has library functions. A library is a set of `.obj` files, which is linked to the program and gives additional beneficial support with its functions. The programmer can use the library functions in the programs. The library is also called as `iostream` library. The difference between C and C++ I/O function is that C functions do not support object-oriented platform whereas C++ supports object-oriented platform.

3.2 STREAMS IN C++ AND STREAM CLASSES

The C++ supports a number of I/O operations to perform read and write operations. These C++ I/O functions help the user to work with different types of devices such as keyboard, disk, tape drivers, etc. Stream is an inter-mediator between I/O devices and the user. The standard C++ library contains the I/O stream functions. The I/O functions are part of the standard library that provides portability to the language itself. A library is a set of `.obj` files connected to the user's program.

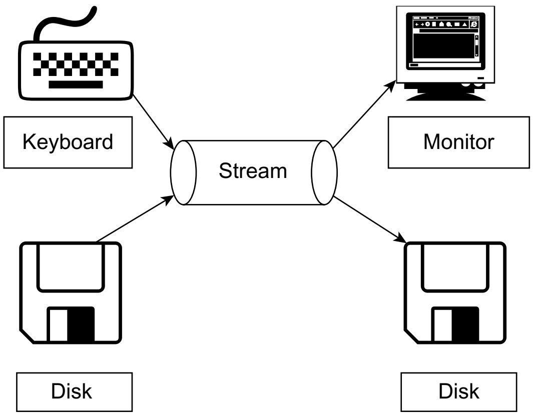


Fig. 3.1 Streams and I/O devices

The stream is a flow of data, measured in bytes, in sequence. If data is received from input devices in sequence, then it is called as source stream, and when the data is passed to output devices, then it is called as destination stream. This process of flow of data is also known as encapsulation through streams. The data is received from the keyboard or disk and can be passed to the monitor or to the disk. Figure 3.1 describes the concept of stream with input and output devices.

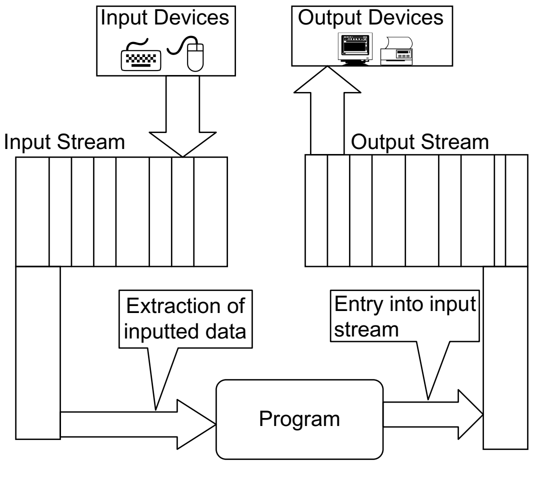


Fig. 3.2 C++ input and output streams

The data in source stream can be used as input data by the program. So, the source stream is also called as input stream. The destination stream that collects output data from the program is known as the output stream. The mechanism of the input and output stream is illustrated in Figure 3.2.

As discussed earlier, the stream is an inter-mediator between I/O devices and the user. The input stream receives data from keyboard or storage devices such as hard disk, floppy disk, etc. The data present in output stream is passed on to the output devices such as monitor or printer according to the user's choice.

3.3 PRE-DEFINED STREAMS

C++ has a number of pre-defined streams. These pre-defined streams are also called as standard I/O objects. These streams are automatically activated when program execution starts. Table 3.1 describes the pre-defined streams.

Table 3.1 Pre-defined C++ Stream or I/O Global Objects

`cin`	Standard input, usually keyboard, corresponding to `stdin` in C. It handles input from input devices usually from keyboard.
`cout`	Standard output, usually screen, corresponding to `stdout` in C. It passes data to output devices such as monitors and printers. Thus, it controls output.
`clog`	A fully buffered version of `cerr` (no C equivalent). It controls error messages that are passed from buffer to the standard error device.
`cerr`	Standard error output, usually screen, corresponding to `stderr` in C. It controls the unbuffered output data. It catches the errors and passes to standard error device monitor.

3.1 Write a program to display a message using pre-defined objects.

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

int main()
{
   clrscr();
   cout<<"\nSTREAMS";
   cerr<<"\nSTREAMS";
   clog<<"\nSTREAMS";
   return 0;
}
```

OUTPUT

```
STREAMS
STREAMS
STREAMS
```

Explanation: In this program the pre-defined objects `cout`, `cerr`, and `clog` are used to display the message "STREAMS" on the screen.

3.4 BUFFERING

It is a block of memory used to hold data temporarily. It is always located between a peripheral device and a faster computer. Buffers are used to pass data between computer and devices. The buffer increases the performance of the computer by allowing read/write operation in larger chunks. For example, if the size of the buffer is N, the buffer can hold N number of bytes of data.

Writing data to the disk is very costly. The read/write operation with disk takes a long time. When these operations are carried out, the execution of the program will be slow for few seconds. If the program involves many read and write operations, the program execution speed will be slower. To avoid this problem, the stream provides buffer. Buffer holds the data temporarily. The input data is passed to the stream buffer. The data is not written to the disk immediately till the buffer fills. When the buffer is completely filled, the data is written to the disk. This is explained with an example in Figure 3.3.

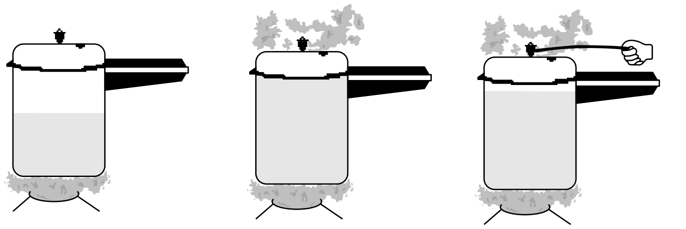


Fig. 3.3 Working of buffer

In Figure 3.3a, the level of the steam in the pressure cooker is increasing. The steam is not full, hence the whistle will not blow, and the steam will not be released from the pressure cooker.

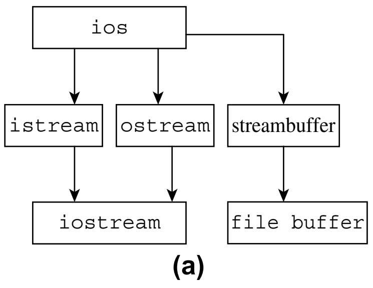
In Figure 3.3b, the pressure cooker is completely filled. The level of the steam reaches to the top. Due to high-pressure the whistle blows automatically and the steam comes out of the pressure cooker.

After the steam comes out the whistle falls down. Even when the cooker is not completely filled with steam, the steam can be released manually by pulling up the whistle. It is just like flushing the buffer even when the buffer is not completely filled.

3.5 STREAM CLASSES

C++ streams are based on class and object theory. C++ has number of classes that work with console and file operations. These classes are known as stream classes. Figure 3.4 shows the stream classes. All these classes are declared in the header file `iostream.h`. The file `iostream.h` must be included in the program, if we are using the functions of these classes.

As described in Figure 3.4a the classes `istream` and `ostream` are derived classes of base class `ios`. The `ios` class contains member variable object streambuf. The streambuf places the buffer. The member function of streambuf class handles the buffer by providing the facilities to flush



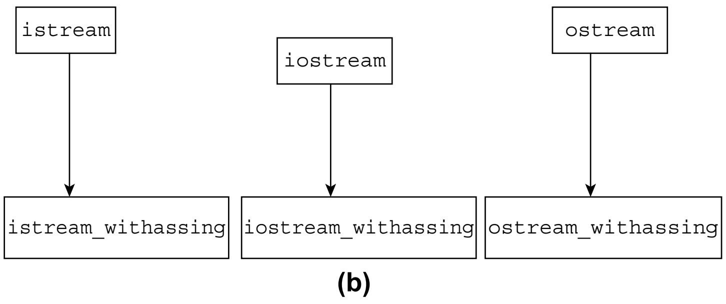


Fig. 3.4 Hierarchy of stream classes

clear and pour the buffer. The class `iostream` is derived from the classes `istream` and `ostream` by using multiple inheritance. The `ios` class is a virtual class and it is present to avoid ambiguity that frequently appears in multiple inheritance. Chapter 11 describes multiple inheritance and virtual classes.

The `ios` class has an ability to handle formatted and unformatted I/O operations. The `istream` class gives support for both formatted and unformatted data. The `ostream` classes handle the formatting of output data. The `iostream` contains functions of both `istream` and `ostream` classes. The classes `istream_withassign`, `ostream_withassign`, and `iostream_withassign` append appropriate assignment operators as shown in Figure 3.4b. Table 3.2 describes functions/contents of C++ stream classes.

Table 3.2 Functions/Contents of C++ Stream Classes

Class	Function/Contents
`ios`	(1) It is an input and output stream class. (2) It is used to implement a buffer, i.e. it is pointer to a buffer streambuf. (3) `ios` maintains the information on the state of streambuf, i.e. good, bad, eof, etc.
`istream`	(1) `istream` provides formatted input. (2) It is used to handle formatted as well as unformatted conversion of character from a streambuf. (3) The properties of `ios` are inherited in `istream` class. (4) The instance of class does not carry out the actual input. (5) istream declares functions such as `peek()`, `tellg()`, `seekg()`, `getline()`, `read()`, etc. (6) `istream` class overloads the `'>>'` operator.
`ostream`	(1) It is used for general-purpose output. (2) It is used to declare the output functions such as `tellp()`, `put()`, `write()`, `seekp()`, etc. (3) It is the parent of all output stream. (4) `ostream` overloads the `'<<'` operator.
`iostream`	(1) It is used to handle both input and output streams.
`istream_withassign`	(1) It is derived from `istream`. (2) It is used for `cin` input.
`iostream_withassign`	(1) It is a bidirectional stream.

3.6 FORMATTED AND UNFORMATTED DATA

Formatting means representation of data with different settings as per the requirement of the user. The various settings that can be done are number format, field width, decimal points, etc.

The data accepted or printed with default setting by the I/O function of the language is known as unformatted data. For example, when the `cin` statement is executed, it asks for a number. The user enters a number in decimal. For entering decimal number or displaying the number in decimal using `cout` statement, the user will not need to apply any external setting. By default, the I/O function represents the number in decimal format. Thus, the data handled in such a way is called as unformatted data.

If the user needs to accept or display data in hexadecimal format, manipulators with I/O functions should be used. The data obtained or represented with these manipulators are known as formatted data. For example, if the user needs to display the data in hexadecimal format, then the manipulator can be used as follows.

```
cout<<hex<<15;
```

The above statement converts decimal 15 to hexadecimal F.

3.7 UNFORMATTED CONSOLE I/O OPERATIONS

Input and Output Streams

The input stream uses `cin` object to read data and the output stream uses `cout` object to display the data on the screen. The `cin` and `cout` are pre-defined streams for input and output data. The data type is identified by these functions using operator overloading of the operators << (insertion operator) and >> (extraction operator). The operator << is overloaded in `ostream class` and the operator >> is overloaded in `istream class`. Figure 3.5 shows the flow of input and output stream.

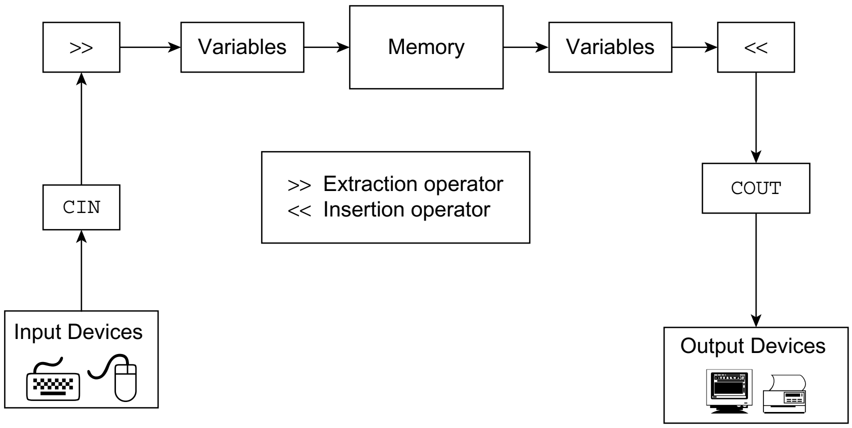


Fig. 3.5 Working of `cin` and `cout` statements

Input Stream

The input stream reads operation through keyboard. It uses `cin` as object. The `cin` statement uses >> (extraction operator) before a variable name. The `cin` statement is used to read data through the input device. Its syntax and example are as follows.

Syntax:

```
cin>>variable;
```

Example:

```
int v1;
float v2;
char v3;
cin>> v1>> v2>> v3…>> vn;
```

where `v1`, `v2`, and `v3` are variable names. The response of the user to this statement would be as shown below.

```
2 5.4 A // input data
```

If the user enters data in the manner 2 5.4 A, then the operator will assign 2 to `v1`, 5.4 to `v2` and A to `v3`. If the entered data is greater than the variable, it remains in the input stream. While entering string, blank spaces are not allowed. More than one variable can be used in `cin` statement to input data. Such operations are known as cascaded input operations.

For example,

```
cin>>v1>>v3;
```

where `v1` and `v2` are variables.

The operator >> accepts the data and assigns it to the memory location of the variables. Each variable requires >> operator. Both these statements should not be included in the bracket. The enter data is separated by space, tab, or enter. Similar to `scanf()` statement, `cin` does not require control strings such as %d for integer, %f for `float`, etc.

More examples

```
int weight;
cin>>weight // Reads integer value
float height;
cin>>height; // Reads float value
double volume;
cin>>volume; // Reads double value
char result[10];
cin>>result; // Reads char string
```

Output Streams

The output streams manage output of the stream, that is, display contents of variables on the screen. It uses << insertion operator before the variable name. It uses the `cout` object to perform console write operation. The syntax and example of `cout` statement are as follows.

Syntax:

```
cout<<variable
```

Example:

```
cout<<v1 <<v2 <<v3 … <<vn;
```

where, `v1`, `v2`, and `v3` are variables. The above statement displays the contents of these variables on the screen. The syntax rules are similar to `cin`. Here, the `cout` statement uses the insertion operator <<. The `cout` statement does not use the format specification such as %d, %f as used in C, etc. The `cout` statement allows us to use all ‘C’ escape sequences such as ‘\n’, ‘\t’, etc.

More examples

```
int weight;
cout<<weight // Displays integer value
float height;
```

```
   cout<<height; // Displays float value
   double volume;
   cout<<volume; // Displays double value
   char result[10];
   cout<<result; // Reads char string
```

The table given below illustrates comparative programs on `cout` statements.

PROGRAM 3.2

```
#include
<iostream.h>

int main()
{
   cout<<" C PLUS
          PLUS";
   return 0;
}
```

OUTPUT

C PLUS PLUS

In this program, the `cout` statement displays the message "C PLUS PLUS" on the screen.

PROGRAM 3.3

```
#include
<iostream.h>

int main()
{
   char *n="Hello";
   cout<<n;
   return 0;
}
```

OUTPUT

Hello

In this program, the string is first assigned to character pointer n. The `cout` statement displays the contents of variable n.

PROGRAM 3.4

```
#include
<iostream.h>

int main()
{
   int x=10;
   float f=3.14;
   cout<<x;
   cout<<"\t";

   cout<<f;
   return 0;
}
```

OUTPUT

10 3.14

In this program, integer and `float` values are assigned to variables x and f. The `cout` statement displays the values of x and f.

3.5 Write a program to accept a string through the keyboard and display it on the screen. Use `cin` and `cout` statements.

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

int main()
{
   char name[15];
   clrscr();
   cout<<"Enter Your Name :";
   cin>>name;
   cout<<"Your name is"<<name;
```

```
    return 0;
}
```

OUTPUT

```
Enter Your Name :Amit
Your name is Amit
```

Explanation: In the above program, `cout` statement displays a given string on the screen. It is similar to `printf()` statement. The `cin` statement reads data through the keyboard. It is similar to `scanf()` statement.

3.6 Write a program to read two integers and display them. Use `cin` and `cout` statements.

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

int main()
{
    int num,num1;
    clrscr();
    cout<<"Enter Two numbers :";
    cin>>num>>num1;
    cout<<"Entered Numbers are :";
    cout<<num<<"\t"<<num1;
    return 0;
}
```

OUTPUT

```
Enter Two numbers : 8 9
Entered Numbers are : 8 9
```

Explanation: In the above program, the `cin` statement reads two integers in variables `num` and `num1`. The `cout` statement displays the read numbers. The escape sequence '\t' is used to insert tab between two numbers.

3.7 Write a program to display data using `cout` statements.

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

int main()
{
   clrscr();
   cout<<"================";
   cout<<endl;
   cout<<"Hello"; // prints hello
   cout<<"\n";
   cout<<123; // prints 123
   cout<<endl<<3.145; // prints 3.145
   cout<<endl<<"NUMBER:"<<"\t"<<452; // prints string and value
   cout<<"\n==== The end =====";
   return 0;
}
```

OUTPUT

```
================
Hello
123
3.145
NUMBER : 452
==== The end =====
```

Explanation: Explanation of each of the `cout` statement is as follows.

(a) `cout<<"================"` – Displays line of on the screen.
(b) `cout<<endl` – Breaks a line.
(c) `cout<<"Hello"` – Display string `"Hello"` on the screen.
(d) `cout<<"\n"` – Breaks a line.
(e) `cout<<123;` – Displays integer 123.
(f) `cout<<endl<<3.145` – Breaks a line and displays `float` number 3.145.
(g) `cout<<endl<<"NUMBER: "<<"\t"<<452` – Breaks a line, displays the string "number", inserts a tab and integer number 452.
(h) `cout<<"\n==== The end ====="` – Displays the line and string.

3.8 Write a program to display `int`, `float`, `char`, and `string` using `cout` statement.

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

int main()
{
    clrscr();
    int x=5;
    float y=5.2;
    char z='z';
```

```
   char city[]="NANDED";
   cout<<"x = "<<x <<" y ="<<y <<" z = "<<z <<endl;
   cout<<"City = "<<city;
   return 0;
}
```

OUTPUT

```
X = 5 y =5.2 z = z
City = NANDED
```

Explanation: In the above program, the statement `cout<<"x = "<<x <<" y ="<<y << " z = "<<z <<endl` displays values of `x`, `y`, and `z`. The statement `cout<<"City = "<<city` displays the contents of character array city.

3.9 Write a program to input `int`, `float`, `char`, and string using `cin` statement and display using `cout` statement.

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

int main()
{
   clrscr();
   int x;
   float y;
   char z;
   char city[15];
   cout<<"\n Enter integer, float and char";
   cin>>x>>y>>z;
   cout<<"\n Enter a string :";
   cin>>city;
   cout<<"x = "<<x <<" y ="<<y <<" z = "<<z <<endl;
   cout<<"City = "<<city;
   return 0;
}
```

OUTPUT

```
Enter integer, float and char 12 1.2 H
Enter a string : NAGPUR
X = 12 y =1.2 z = H
City = NAGPUR
```

Explanation: Explanation of the program is as follows.

(a) `cout<<"\n Enter integer, float, and char"` – Prompts message "Enter integer, `float`, and `char`"
(b) `cin>>x>>y>>z` – Accepts integer, `float`, and `char` and stores in x, y, and z.
(c) `cout<<"\n Enter a string: "` – Displays message "Enter a string: ".
(d) `cin>>city` – Reads string through the keyboard and stores in the array `city[15]`.
(e) `cout<<"City = "<<city;` – Displays contents of the array city.

3.8 TYPE CASTING WITH THE `cout` STATEMENT

Type casting refers to conversion data of one basic type to another by applying external use of data type keywords. The description of type casting is explained in Chapter 3. Programs on type casting are as follows.

3.10 Write a program to use different formats of type casting and display the converted values.

{coderipe}

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

int main()
{
   int a=66;
   float f=2.5;
   double d=85.22;
   char c='K';
   clrscr();
   cout<<"\n int in char format :"<<(char)a;
   cout<<"\n float in int format :"<<(int)f;
   cout<<"\n double in char format :"<<(char)d;
   cout<<"\n char in int format :"<<(int)c;
   return 0;
}
```

OUTPUT

```
int in char format : B
float in int format : 2
double in char format : U
char in int format : 75
```

Explanation: In the above program, the variables of `int float`, `double`, and `char` type are declared and initialized. The variable `a` is initialized with 66, `f` with 2.5, `d` with 85.22, and `c` with character 'K'.

The first `cout` statement converts integer value to the corresponding character according to the ASCII character set and the character B is displayed.

The second `cout` statement converts `float` value to integer. The value displayed is 2 and not 2.5. When type cast format `(int)` is used, the decimal portion of `float` value is removed and only the integer part is considered.

In the third statement, the `double` value is converted to character. The number 85.22 is converted to integer and then to character. The `char` data type is nothing else than `int` data type. The only difference is that the `char` data type has range from –128 to 127, which requires one byte in the memory.

The last statement converts character to `int`. The value of 'K' is 75 when printed as an integer. The format `(int)` converts `char` to `int`.
In Figure 3.6, 65 is an integer and it is converted to character A by using type casting format `(char)`. Table 3.3 describes various type casting formats and their output results.

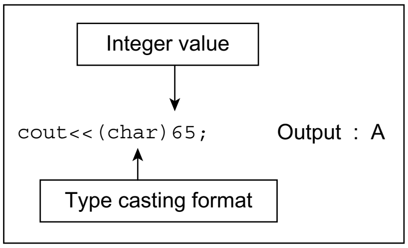


Fig. 3.6 Typecasting integer

Table 3.3 Type Casting Formats

Type Casting Formats	Outputs	Conversion
`cout<<(char)65;`	A	`int to char`
`cout<<(int)'A';`	65	`char to int`
`cout<<(int)5.22;`	5	`float to int`
`cout<<(char)78.33;`	N	`float to char`
`cout<<(double)123445338.33;`	1.234453e+08	`float to double`
`cout<<(unsigned)-1;`	65535	`signed to unsigned`

3.11 Write a program to display data using type casting.

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

int main()
{
   clrscr();
   int x=77;
   float y=5.1252;
   char z='A';
   char city[15];
   cout<<" x = "<<(char)x<<endl;
   cout<<" y = "<<(int)y <<endl;
   cout<<" z = "<<(int)z;
   return 0;
}
```

OUTPUT

```
x = M
y = 5
z = 65
```

Explanation: Consider the following statements.

`int x=77` – Declares integer variable `x` and initializes it with `77`.
`float y=5.1252` – Declares `float` variable `y` and initializes it with 5.1252.
`char z='A';` – Declares character variable `z` and initializes it with character `'A'`.

The following statements are used to display the contents on the screen.

`cout<<" x = "<<(char) x<<endl` – Variable `x` is an integer but value displayed will be `'M'` because the statement `(char)` converts integer to the corresponding *ASCII* character.

`cout<<" y = "<<(int)y <<endl` – Variable `y` is a `float` but before printing the value 5.1252, it is converted to integer and the output will be 5.

`cout<<" z = "<<(int)z` – Variable `z` is of character type. The character value is converted to integer and the output displayed will be 65.

3.12 Write a program to display A to Z alphabets using ASCII values.

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

int main()
{
   clrscr();
   int j;
   for (j=65;j<91;j++)
   cout<<(char)j<<" ";
   cout<<endl;
   for (j=65;j<91;j++)
   printf ("%c ",j);
   return 0;
}
```

OUTPUT

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Explanation: In the above program, `A` to `Z` alphabets are displayed using `cout()` and `printf()` statements. In `cout()` statement before printing, type casting is done. The integer is converted to corresponding `char` type symbol and displayed. In the `printf()` statement, the control string %c performs this task. The quotation mark (“ ”) inserts space between two successive characters.

3.13 Write a program to display addresses of variables in hexadecimal and unsigned integer formats.

{coderipe}

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

int main()
{
   clrscr();
   int x=77;
   float y=5.1252;
   int z=78;
   cout<<" Address of x = "<<&x<<endl;
   cout<<" Address of y = "<<&y <<endl;
   cout<<" Address of z = "<<(unsigned)&z <<endl;
   return 0;
}
```

OUTPUT

```
Address of x = 0x887ffff4
Address of y = 0x887ffff0
Address of z = 65518
```

Explanation: The `cout` statement displays the address of a variable in hexadecimal format. Using type casting, syntax (unsigned) converts hexadecimal to unsigned integer (decimal). The output shows addresses in both hexadecimal and unsigned integer (decimal) formats.

The `&` (ampersand) operator is used to display the address of the variable. The `address` operator is preceded by the variable name. The address is always represented as an unsigned integer. The `cout` statement displays the address in hexadecimal format. To convert the hexadecimal address to an unsigned integer, type casting is used.

3.14 Write a program to display string using & and * operators with `cout` statements.

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

int main()
{
   clrscr();
   char *name="c plus plus";
   cout<<name<<"\n";
```

```
    cout<<&name[0]<<"\n";
    cout<<*(&name);
    return 0;
 }
```

OUTPUT

C plus plus
C plus plus
C plus plus

Explanation: In the above program, the character pointer `name` is assigned to the string "c plus plus". The first `cout` statement displays the string using variable name. The statement `cout<<&name[0]<<"\n"` displays the string. Here, & operator is used and the 0 (zero) points to the base address of the string. In the second statement, if the base address is not specified, then it will display the address. The third statement uses the pointer notation to display the string.

3.15 Write a program to display a string using different syntaxes using the operators * and & with `cout` statement.

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

int main()
{
   clrscr();
   char *name="c plus plus";
   cout<<*&(name)<<"\n";
   cout<<&(*name)<<"\n";
   cout<<*&name<<"\n";
   return 0;
 }
```

OUTPUT

C plus plus
C plus plus
C plus plus

Explanation: In the first statement, the operators * and & are used one after another and the variable name is inside the parentheses. In the second statement, operator & is outside and the operator * and variable name are inside the parentheses. In the third statement, parentheses are not used. All the three statements display the output `"c plus plus."`

3.8.1 Difference of Using C and C++ I/O Functions

The `printf()` and `scanf()` of C language needs format string. For example, to read and display an integer, the `scanf` and `printf()` statement can be written as follows.

```
int x;
scanf ("%d",&x)
printf ("%d",x);
```

In the above statements, %d is used to tell the I/O functions to treat the data as integer.

If the integer x is changed to long integer, the programmer needs to change every occurrence of %d in the program with %ld.

The C++ statement reads and displays the same data as follows.

```
int x;
cin>>x;
cout<<x;
```

Here, the `cin` and `cout` statements do not require any format string. If the type of x is changed to long integer, the user need not specify the type of data or any correction in the statement. The `cin` and `cout` statement identifies the data type. The format of `cin` and `cout` statement is the same for all types of variables.

`get()` and `put()` functions

`get()` function

The single character input and output operations in C++ can be done using `put()` and `get()` functions. The classes `istream` and `ostream` provide the two member functions `put()` and `get()`. The `get()` is used to read a character and `put()` is used to display the character on the screen.

The `get()` function has two syntaxes:

(a) `get(char*);`
(b) `get(void);`

If syntax (a) is used, the `get()` function assigns the read data to its argument, whereas when the statement (b) is used, the `get()` function returns the data read. The data is assigned to the variable present on the left-hand side of the assignment operator. These functions are members of I/O stream classes and can be called using object.

`put()` function

The `put()` function is used to display the string on the screen. It is a member of `ostream` class. The syntax of `put()` is as follows:

(a) `cout.put ('A');`
(b) `cout.put (x);`

The statement (a) displays the character `'A'` on the screen and the statement (b) displays the contents of variable `x` on the screen. If an integer is used as an argument, its corresponding ***ASCII*** value is displayed. Few examples are illustrated below.

3.16 Write a program to display the character on the screen using `put()` function.

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

int main()
{
   clrscr();
   cout.put('C');
   cout.put ('+');
   cout.put ('+');
   return 0;
}
```

OUTPUT

```
C++
```

Explanation: The `cout.put()` statement displays one character at a time on the screen. In this program, three characters are displayed on the screen using `cout.put()` statement.

3.17 Write program to use escape sequence with `cout.put()` statement.

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

int main()
{
   clrscr();
   cout.put('1');
   cout.put ('\n');
   cout.put ('2');
   return 0;
}
```

OUTPUT

```
1
2
```

Explanation: The escape sequences such as '\n', '\t', etc., can be used with `cout.put()` statement. The escape sequences are combinations of two characters. The output of the program displays the number 1 and 2 in two separate lines. The statement `cout.put('\n')` splits a line.

3.18 Write a program to use multiple `put()` statements with single `cout` object and display the characters.

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

int main()
{
   clrscr();
   cout.put('C').put('+').put('+');
   return 0;
}
```

OUTPUT

C++

Explanation: In the above program, the single object `cout` is used followed by sequence of `put()` statement. The `put()` statements are separated by dot operators. In this way, multiple statements can be combined.

3.19 Write a program to read character using `get()` and display it using `put()`.

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

int main()
{
   char ch;
   clrscr();
   cout<<"\n Enter a character :";
   cin.get(ch);
   cout<<"\n Entered character was :";
   cout.put(ch);
   return 0;
}
```

OUTPUT

Enter a character : C
Entered character was : C

Explanation: In the above program, character variable ch is declared. The first `cout` statement displays message `"Enter a character:"` on the screen. The `cin.get()` function activates input stream and character entered by the user is stored in the variable ch. The `cout.put()` statement displays the character on the screen.

3.20 Write a program to read characters using a sequence of `get()` statements and display the characters read using a sequence of `put()` statements.

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

int main()
{
   char ch[3];
   clrscr();
   cout<<“\n Enter characters :”;
   cin.get(ch[0]).get(ch[1]).get(ch[2]);
   cout<<“\n Characters Entered :”;
   cout.put(ch[0]).put(ch[1]).put(ch[2]);
   return 0;
}
```

OUTPUT

```
Enter characters : cpp
Characters entered : cpp
```

Explanation: In the above program, a character array ch[3] is declared. The sequence of `get()` and `put()` functions are used to read and display the characters. The `get()` function reads characters and stores in array ch[3]. The `put()` function displays the same on the screen.

3.21 Write a program to read and display the string. Use `get()` and `put()` functions.

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

int main()
{
   clrscr();
   char j=0;
   char x[20];
   cout<<“\n Enter a string :”;
   while (x[j++]!=‘\n’)
   cin.get(x[j]);
   j=0;
   cout<<“\n Entered string :”;
   while (x[j++]!=‘\n’)
   cout.put(x[j]);
   return 0 ;
}
```

OUTPUT

```
Enter a string : Programming
Entered string : Programming
```

Explanation: In the above program, the character array `x[]` is declared. The first `while` loop reads characters using `cin.get()` function through the keyboard. When a user presses the enter key, the `while` loop terminates. The second `while` loop displays the characters read using the function `cout.put()`. The output of the program is as shown above.

`getline()` and `write()` functions

`getline()` function

The `getline()` and `write()` functions are useful in string input and output. The `getline()` functions read the string including white space. The `cin()` function does not allow to enter the string with blank spaces. The input reading is terminated when a user presses the enter key. The new line character is accepted but not saved and replaced with the null character. The object `cin` calls the function as follows.

```
cin.getline (variable, size);
```

where the variable name may be any character array name and the size is the size of the array.

`write()` function

The `write()` function is used to display the string on the screen. Its format is similar to `getline()` function, but the function is exactly the opposite. The syntax is as follows.

```
cout.write (variable, size);
```

where the variable name may be a character type and size is the size of the character arrays. The `cout.write()` statement displays only a specified number of characters given in the second argument, though the actual string may be more in length. If the size of the array is larger than the actual string length, then the size argument contains the actual size of the array. In this case, the `getline()` displays blank spaces for the remaining unfilled elements. The following program illustrates both the functions.

3.22 Write a program to display a string using `cout.write()` statements.

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

int main()
{
   clrscr();
   cout.write ("INDIA",6);
   cout.write ("IS",3);
   cout.write ("GREAT",5);
   return 0;
}
```

OUTPUT

INDIA IS GREAT

Explanation: In this program, the first statement displays "INDIA" followed by one blank space. This is because the argument value is greater by one than the actual string length.

Similarly, the second statement displays "IS" followed by one blank space. The reason is same.

The last statement displays "GREAT." Here, the argument value and the string lengths are same. Hence, no blank spaces are displayed.

In case the argument value is less than the actual string length, the complete string will not be displayed. The number of characters of the string displayed depends on the value of the given argument.

3.23 Write a program to show the effect if less argument is given than the actual string length in the `cout.write()` statement.

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

int main()
{
   clrscr();
   cout.write ("SUNDAY",3);
   return 0;
}
```

OUTPUT

SUN

Explanation: In the above program, the `cout.write()` will not display the complete string. The statement displays the characters according to the value of the second argument. The value of the second argument is three. Hence, instead of six characters only three characters are displayed and the remaining characters are skipped.

3.24 Write a program to read a string using `getline()` function and display it using the `write()` statement.

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

```
int main()
{
   clrscr();
   char x[30];
   cout<<"\n Enter any string :";
   cin.getline(x,30);
   cout<<"\n Entered string :" <<x;
   cout<<"\n Entered string :";
   cout.write(x,30);
   cout<<"\n Entered string :";
   cout.write(x,strlen(x));
   return 0 ;
}
```

OUTPUT

```
Enter any string : C++ is advanced C
Entered string : C++ is advanced C
Entered string : C++ is advanced C h >& )
Entered string : C++ is advanced C
```

Explanation: In the above program, the `x[]` is a character array. The `getline()` function reads the string through the keyboard. The `getline()` function accepts the string including spaces. The `cout()` statement displays the string including white spaces. The first `write()` statement displays the string with garbage values. This is because the string length is less than the actual size of the array. In the second `write()` statement, `strlen()` function calculates the length of the string and that length is used as an argument in the `write()` statement. This statement displays the entered string without any garbage collection.

The `cin()` statement cannot accept a string including spaces. It accepts only a single word. The `cout()` statement displays the string read through `cin() and getline()` functions, that is, it can display the string with or without blank spaces. The `write()` statement displays the string according to the specified size. It displays the string with or without blank spaces. The `write()` does not support any escape sequence.

3.25 Write a program to display the string using different arguments in `write()` **statement.**

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

int main()
{
   clrscr();
   char a,x[30];
   cout<<"\n Enter any string :";
   cin.getline(x,30);
   cout<<"\n Entered string : \n";
   for (a=0;a<=strlen(x);a++)
   {
   cout<<"\n";
   cout.write(x,a);
   }
   return 0 ;
}
```

OUTPUT

```
Enter any string : C Plus Plus
Entered string :
C
C
C P
C Pl
C Plu
C Plus
C Plus
C Plus P
C Plus Pl
C Plus Plu
C Plus Plus
```

Explanation: In the above program, two strings are entered in character arrays `x[]` and `a[]`. The successive `write()` statement in one line displays the string one after another. Thus, we can use multiple `write()` statements followed by a single `cout` object.

3.9 MEMBER FUNCTIONS OF THE `istream` CLASS

The `istream` contains the following functions that can be called using `cin` object.

peek(): It returns the succeeding character without extraction. For example,

```
cin.peek()=='#';
```

where `cin` is an object and '#' is the symbol that is to be caught in the stream.

ignore(): The member function `ignore()` has two arguments, maximum number of characters to avoid and the termination character. For example,

```
cin.ignore (1,'#');
```

The statement ignores character 1 till character `'#'` is found.

3.26 Write a program to demonstrate the use of `peek()` and `ignore()` functions.

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

int main()
{
   char c;
   clrscr();
   cout<<"enter text (press F6 to end :";
   while (cin.get(c))
   {

cout<<c;
   while (cin.peek()=='#')
   {
   cin.ignore(1,'#');
   }
   }
   return 0;
}
```

OUTPUT

```
enter text (press F6 to end : ABCDEFG###HIJK
ABCDEFGHIJK
```

Explanation: In the above program, the `cin.get()` function continuously reads characters through the keyboard till the user presses F6. The `cout` statement inside the loop displays the contents of variable c on the console. The `cin.peek()` statement checks the variable c. If the variable c contains '#', it is ignored from the stream and not displayed on the screen.

putback() : The `putback()` replaces the given character into the input stream. For example,

```
cin.putback ('*');
```

where `cin` is an object and '*' is the symbol which is replaced in the stream.

3.27 Write a program to demonstrate the use of `putback()` function.

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

int main()
{
   char c;
   clrscr();
```

```
    cout<<"enter text (press F6 to end :";
    while (cin.get(c))
    {
    if (c=='s')
    cin.putback('S');
    cout.put(c);
    }
    return 0;
}
```

OUTPUT

```
enter text (press F6 to end :
c plus plus^Z
c plusS plusS
```

Explanation: In the above program, the `cin.get()` function continuously reads characters through the keyboard till the user presses F6. The `cout` statement inside the loop displays the contents of the variable c on the console. The `if` statement checks the contents of the variable c. If variable c contains a small letter 's', the `putback()` statement sends capital 'S' in the stream. The small 's' is replaced with capital 'S'. The contents displayed will be with capital 'S'.

Gcount(): This function returns the number of unformatted characters extracted from the input stream. The last statement should be `get()`, `getline()`, or `read()`.

3.28 Write a program to demonstrate the use of `gcount()` function.

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

int main()
{
    char text[20];
    int len;
    clrscr();
    cout<<"Enter text :";
    cin.getline(text,20);
    len=cin.gcount();
    cout<<"The numbers of characters extracted are:"<<len;
    return 0;
}
```

OUTPUT

```
Enter text : Virtual
The numbers of characters extracted are : 8
```

Explanation: In the above program, the `getline()` function reads text through the keyboard. The `gcount()` function returns the number of characters extracted from stream to variable len. It also counts the null character. The `cout` statement displays the value of variable len on the screen.

3.29 Write a program to perform the operation with `peek()` and `putback()`.

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

int main()
{
   char c;
   clrscr();
   cout<<"enter text (press F6 to end :";
   while (cin.get(c))
   {
   if (c==' ')
   cin.putback ('.');
   else
   cout<<c;
   while (cin.peek()=='#') cin.ignore(1,'#');
   }
   return 0;
}
```

OUTPUT

```
Enter text (press F6 to end: One! Two! Three! Four!
One..Two..Three..Four.
```

Explanation: In the above program, the `cin.get()` function reads data through the keyboard using first `while loop.` The `if` condition checks for blank space in the entered text. If space is found, the `putback()` statement replaces space dot (.). The `putback()` sends the given character in the input stream. The `peek()` also checks the '#' symbol in the entered text, if it is found, the `ignore()` statement ignores the character and the character will not be displayed on the screen. The program is terminated when the key `F6` or `ctrl+z` is pressed.

3.10 FORMATTED CONSOLE I/O OPERATIONS

C++ provides various formatted console I/O functions for formatting the output. They are of three types.

(1) Ios class function and flags
(2) Manipulators
(3) User-defined output functions

The `ios` grants operations common to both input and output. The classes (`istream, ostream, and iostream`) derived from `ios` are special I/O with high-level formatting operations: The `iostream class` is automatically loaded in the program by the compiler. Figure 3.4 describes the hierarchy of stream classes.

(a) `istream` performs formatted input.
(b) `ostream` performs formatted output.
(c) `iostream` performs formatted input and output.

`streambuf` allows an abstraction for connecting to a physical device. Classes derived from it work with files, memory, etc. The `ios` communicates to a `streambuf`. It keeps information on the state of the `streambuf` (good, bad, eof, etc.) and saves flags for use by `istream` and `ostream`.

- The `streambuf` class controls the buffer and its related member functions. It allows the ability to fill, flush, and empty the buffer.
- The `streambuf` is an object of `ios` class. The base class of input and output stream classes is `ios` class.
- The `istream` and `ostream` classes are derived classes of `ios` class and control input and output streams.
- The `iostream` class is a derived class of `istream` and `ostream`. It provides input and output functions to control console operations.

Table 3.4 describes the functions of `ios` class in brief.

Table 3.4 `ios` Class Function

Function	Working
`width()`	To set the required field width. The output will be displayed in given width.
`precision()`	To set number of decimal point to a `float` value.
`fill()`	To set a character to fill in the blank space of a field.
`setf()`	To set various flags for formatting output.
`unsetf()`	To remove the flag setting.

(1) `ios::width` (member functions)
The `width()` function can be declared in two ways.

(a) `int width();`
(b) `int width (int);`

`int width();`
If this function is declared as given above, it returns the present width setting.
`int width (int);`
If this function is declared as given above, it sets the width as per the given integer and returns the previous width setting. The setting should be reset for each input or output value if a width other than the default is desired.

3.30 Write a program to set column width and display the characters at specified position.

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

int main()
{
   clrscr();
   cout . width(5);
   cout<<"A";
   cout . width(15);
   cout<<"B";
   return 0;
}
```

OUTPUT

```
A B
```

Explanation: The first `cout. width()` statement sets the width 5. The `cout` statement sets the column width position at 5 and the `cout` statement displays the character "A" at column 5. Similarly, the column width is set 15 and the character 'B' is displayed at column 15.

3.31 Write a program to use both the formats of `width()` function and display the result.

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

int main()
{
   clrscr();
   cout.width(50);
   int x=cout.width(1);
   cout<<x;
   return 0;
}
```

OUTPUT

```
50
```

Explanation: In the above program, the `width()` function call sets the width at column `50`. The second `width()` function call sets the width at column 1 and returns the previous setting, that is `50`. The integer `x` collects this value. The `cout()` statement displays the number `50` at column `1`.

(2) `ios::precision`

This function can be declared in two ways.

(a) `int precision (int);`

(b) `int precision();`

`int precision (int);`

If the function is declared as given above, it sets the floating-point precision and returns the previous setting. The precision should be reset for every value being output if we want a precision result other than the default.

`int precision();`

If the function is used as given above, it returns the current setting of floating-point precision.

3.32 Write a program to set precision to two and display the `float` number

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

int main()
{
   clrscr();
   cout.precision(2);
   cout<<3.1452;
   return 0;
}
```

OUTPUT

```
3.14
```

Explanation: In the above program, the `cout.precision()` statement sets `float` point precision to 2. The `cout` statement displays 3.14 instead of 3.1452.

3.33 Write a program to set number of precision points. Display the results of 22/7 in different precision settings.

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

int main()
{
   clrscr();
   int x=0;
   float pi;
```

```
    for (x=10;x>=1;x--)
    {
    pi=(float)22/7;
    cout.precision(x);
    cout<<"\n"<<pi;
    }
    x=cout.precision(1);
    cout<<"\n\n Previous Setting :"<<x;
    return 0;
}
```

OUTPUT

```
3.1428570747
3.142857075
3.14285707
3.1428571
3.142857
3.14286
3.1429
3.143
3.14
3.1
Previous Setting : 1
```

Explanation: In the above program, the `for loop` executes from 10 to 1 in reverse order. Each time the equation `22/7` is calculated and the result is stored in the variable `pi`. The precision is set according to the value of variable `x`. The output of the program is as given above.

(3) `ios::fill`
This function can be declared in two ways.
(a) `char fill();`
(b) `char fill (char);`

`char fill();`
If the function is used as given above, it returns the current setting of fill character.

`char fill (char);`
If the function is used as given above, it resets the fill character and returns the previous setting.

3.34 Write a program to fill empty spaces in a line with a specific symbol (*).

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

int main()
```

```
{
   clrscr();
   cout.fill('*');
   cout.width(10);
   cout<<"H";
   return 0;
}
```

OUTPUT

```
*********H
```

Explanation: In the above program, the statement `cout.fill ('*');` fills the blank spaces with * symbols. The `cout. width()` sets the width to 10 and the character 'H' is displayed at column 10. The column 1 to 9 remains blank. The symbol '*' is displayed instead of blank spaces. This effect is due to `cout.fill()` statement as shown below.

*	*	*	*	*	*	*	*	*	**H**

3.35 Write a program to fill blank spaces with different symbols.

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

int main()
{
   clrscr();
   cout.fill('/');
   cout.width(20);
   cout<<"WEL"<<endl;
   cout.fill('-');
   cout.width(10);
   cout<<"DONE";
   return 0;
}
```

OUTPUT

```
/////////////////WEL
------DONE
```

Explanation: This program is similar to the last one. Here, two different characters are used to fill the blank spaces in the specified width. The width is specified using `cout.width()` statement. The output can be observed as follows.

/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	**W**	**E**	**L**

-	-	-	-	-	-	**D**	**O**	**N**	**E**

TIP

The character '\' (back slash) is not allowed in `cout.fill()` statement to fill the blanks. This is because it is used with escape sequences.

3.36 Write a program to show the effect `cout.fill()`.

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

int main()
{
   clrscr();
   cout<<"Begin :";
   cout.width(15);
   cout<<"ABC";
   cout<<"\n";
   cout<<"Begin :";
   cout.width(15);
   cout.fill('#');
   cout<<"ABC";
   return 0;
}
```

OUTPUT

```
Begin :            ABC
Begin : ############ABC
```

Explanation: The first line displays ABC following blank spaces because the `fill();` is not used. In the second statement, the symbol '#' is displayed because the fill character is set to '#'.

Begin							**A**	**B**	**C**
Begin	#	#	#	#	#	#	A	B	C

3.37 Write a program to fill the unused filled area with '*'

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

int main()
```

```
{
   clrscr();
   int x=0,j=0;
   float pi;
   cout.fill('*');
   for (x=10;x>=1;x--)
   {
   pi=(float)22/7;
   cout.width(j++);
   cout.precision(x);
   cout<<"\n"<<pi;
   }
   cout.width(j++);
   cout.precision(x);
   cout<<"\n"<<pi;
   x=cout.fill();
   cout<<"\n\n Fill setting :"<<(char)x;
   return 0;
}
```

OUTPUT

```
3.1428570747
3.142857075*
3.14285707**
3.1428571***
3.142857****
3.14286*****
3.1429******
3.143*******
3.14********
3.1*********
3.142857
Fill setting :*
```

Explanation: In the above program, the precision setting and display of floating point number using precision is the same as in the previous example. In the `for` loop, each time `width()` is set to the value of variable `j`. The value of `j` increases at each iteration. The floating precision number decreases from top to bottom, and blank spaces remain in the specified field area by the statement `width()`. The unused area is filled with `'*'`. The setting of fill character is done before the `for` loop statement. The fill character may be set to any character. The `fill()` statement after `for` loop body displays the character used for filling.

(4) `ios::setf`

This function can be declared in two ways.

(a) `long setf (long sb, long f);`

(b) `long setf (long);`

```
long setf (long sb, long f);
```

The bits according to those marked in variable `f` are removed in the data member `x_flags`, and then reset to be those marked in variable `sb`. Using the constants in the formatting flag enumeration of class `ios` can specify the value of variable `sb`.

```
long setf (long);
```

If the above declaration is used in the program, it sets the flags according to those marked in the given long. The flags are set in the data member `x_flags` of `class ios.` Using the constants in the formatting flags can specify the long enumeration of class `ios`. It returns the previous settings.

3.11 BIT FIELDS

The `ios` class contains the `setf()` member function. The flag indicates the format design. The syntax of the `unsetf()` function is used to clear the flags. The syntaxes of the function are as follows.

```
Syntax: cout.setf (v1, v2);
Syntax: cout.unsetf(v1);
```

where variable `v1` and `v2` are two flags. Table 3.5 describes the flag, and bit-field setting can be used with this function. The `unsetf()` accepts setting of `v1` and `v2` arguments.

Table 3.5 Flags and Bits

Format	Flag (V1)	Bit Field (V2)
Left justification	`ios::left`	`ios:adjustfield`
Right justification	`ios::right`	`ios:adjustfield`
Padding after sign and base	`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 base	`ios::hex`	`ios::basefield`

(a) `ios::adjustfield`

It is a data member and used with `setf()` function to arrange padding to the left or right, or for internal fill. It can be declared as follows.

```
static const long adjustfield;
```

(b) `ios::floatfield`

It is a data member and used with `setf()` function to set the `float` point notation to scientific or fixed. It is declared as follows.

```
static const long floatfield;
```

(c) `ios::basefield`
It is a data member and used with `setf()` function and used with `setf()` function to set the notation to a decimal, octal, or hexadecimal base. It is declared as follows.

```
static const long basefield;
```

3.38 Write a program to display the message left and right justified.

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

int main()
{
   clrscr();
   cout.fill('=');
   cout.setf(ios::right, ios::adjustfield);
   cout.width(20);
   cout<<"Figure" << "\n";
   cout.setf(ios::left, ios::adjustfield);
   cout.width(20);
   cout<<"Figure" << "\n";
   return 0;
}
```

OUTPUT

```
==============Figure
Figure ==============
```

Explanation: In the above program, the fill character is set to sign `"="`. The `setf()` is set to right justified. The column width is set to `20`. The message `"Figure"` appears at the `20th` column. Before the message the blank space is filled with the `sign "="`. Again the justification property is set to `left` in `setf()` function. The text appears left justified. The output can be observed as follows.

=	=	=	=	=	=	=	=	=	=	=	=	=	=	F	i	g	u	r	e
F	i	g	u	r	e	=	=	=	=	=	=	=	=	=	=	=	=	=	=

3.39 Write a program to display the number in scientific format with sign.

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

int main()
{
   clrscr();
   cout.fill('=');
```

```
    cout.setf(ios::internal, ios::adjustfield);
    cout.setf(ios::scientific, ios::floatfield);
    cout.width(15);
    cout<<- 3.121;
    return 0;
}
```

OUTPUT

```
- ===== 3.121e+00
```

Explanation: In the above program, the fill character is set to sign "=". The `setf()` properties are set to `internal` and `scientific`. The scientific properties display the number in scientific (e) format. The internal properties display the sign before blank spaces. If the setting were removed, the output would be as given below.

`Effect of statement`

```
   cout.setf (ios::internal, ios::adjustfield);
```

-	**=**	**=**	**=**	**=**	**=**	**3**	**.**	**1**	**2**	**1**	**e**	**+**	**0**	**0**

The output without this statement would be as follows.

=	**=**	**=**	**=**	**=**	**-**	**3**	**.**	**1**	**2**	**1**	**e**	**+**	**0**	**0**

The property `ios::fixed` displays the `float` number without scientific format, though the number is big. If the floating-point number is more than 6, then the extras are ignored. This format displays floating point up to 6 only.

3.40 Write a program to convert a decimal number to hexadecimal and octal format.

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

int main()
{
    clrscr();
    cout.setf(ios::hex, ios::basefield);
    cout<<"\n Hexadecimal of 184 is :"<<184;
    cout.setf(ios::oct, ios::basefield);
    cout<<"\n Octal of 15 is :"<<15;
    cout.setf(ios::dec, ios::basefield);
    cout<<"\n Decimal of 0xfe is :"<<0xfe;
    return 0;

}
```

OUTPUT

```
Hexadecimal of 184 is : b8
Octal of 15 is : 17
Decimal of 0xfe is : 254
```

Explanation: In the above program, the properties of `setf()` function are set to `ios::hex`, `ios:: oct`, and `ios::dec.` After this setting, the decimal number given in `cout()` statement will convert to its hexadecimal and octal equivalent, respectively. The hexadecimal number 0xfe is converted to its decimal equivalent.

3.12 FLAGS WITHOUT BIT FIELDS

The flags described in Table 3.6 have no corresponding bit fields. The programs on these functions are illustrated as follows.

Table 3.6 Flags without Bit Fields

Flag	Working
ios :: showbase	Uses base indicator on output
ios :: showpos	Displays + preceding positive number
ios :: showpoint	Shows trailing decimal point and zeros
ios :: uppercase	Uses capital case for hex output
ios :: skipws	Skips white space on input
ios :: unitbuf	Flushes all streams after insertion
ios :: stdio	Flushes `stdout` and `stderr` later insertion
ios :: uppercase	Uses capital characters for scientific and hexadecimal values
Ios::stdio	Adjusts the stream with C standard input and output
Ios::boolalpha	Converts Boolean values to text (“true” or “false”)

3.41 Write a program to use the various settings given in the table.

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

int main()
{
   clrscr();
   cout.setf(ios::showpos);
   cout<<1254;
   cout.setf(ios::showpoint);
   cout<<"\n"<<1254.524;
   cout.setf(ios::hex,ios::basefield);
   cout<<"\n"<<184;
   cout.setf(ios::uppercase);
   cout<<"\n"<<184;
   return 0;
}
```

OUTPUT

```
+1254
+1254.524000
0xb8
0XB8
```

Explanation: In the above program, the setting `ios::showpos` displays the + sign before the number `1254`. The setting `ios::showpoint` displays the trailing zeros after the number 1254.524. The setting `ios::hex` converts the decimal number to hexadecimal. It uses small letters. The setting `ios::uppercase` displays the hexadecimal number in uppercase.

3.13 MANIPULATORS

The output formats can be controlled using manipulators. The header file `iomanip.h` has a set of functions. The effect of these manipulators is similar to `ios` class member functions. Every `ios` member function has two formats. The first format is used for setting and the second format is used to understand the previous setting. But the manipulator does not return the previous setting. The manipulator can be used with `cout()` statement as follows.

```
   cout<<m1 <<m2 <<v1;
```

Here, `m1` and `m2` are two manipulators and `v1` is any valid C++ variable.

Table 3.7 describes the most useful manipulators. The manipulators hex, dec, oct, ws, endl, and flush are defined in `iostream.h`. The manipulators setbase, `width()`, `fill()`, etc., that require an argument are defined in iomanip.h.

Table 3.7 Pre-defined Manipulators

Manipulator	Function
`setw (int n)`	The field width is fixed to n
`setbase`	Sets the base of the number system
`setprecision(int p)`	The precision point is fixed to p
`setfill (char f)`	The fill character is set to the character stored in variable f
`setiosglags(long l)`	Format flag is set to l
`resetiosflags(long l)`	Removes the flags indicated by l
`endl`	Splits a new line
`skipws`	Omits white space on input
`noskipws`	Does not omit white space on input
`ends`	Adds null character to close an output string
`flush`	Flushes the buffer stream
`lock`	Locks the file associated with the file handle
`ws`	Omits the leading white spaces present before the first field
`hex, oct ,dec`	Displays the number in hexadecimal, octal, and decimal format

3.42 Write a program to display formatted output using manipulators.

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

int main()
{
   clrscr();
   cout<<setw(10) <<"Hello"<<endl;
   cout<<setw(15) <<setprecision(2) <<2.5555;
   cout<<setiosflags(ios::hex);
   cout<<endl<<"Hexadecimal of 84 is :"<<84;
   return 0;
}
```

OUTPUT

```
Hello
      2.56
Hexadecimal of 84 is : 54
```

Explanation: In the above program, the manipulator `setw (10)` sets the field width to `10`. The message `"hello"` is displayed at 10th column. The `endl` inserts a new line. In the second `cout()` statement, the `setprecision (2)` manipulator sets the decimal point to 2. The number 2.5555 will be displayed as 2.56 at column 15. The manipulator `setiosflag` sets the hexadecimal setting for the display of number. The last `cout()` statement displays the equivalent hexadecimal of 84, that is, 54.

3.43 Write a program to display the given decimal number in hexadecimal and octal format.

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

int main()
{
   clrscr();
   int x=84;
   cout<<"\n Hexadecimal Number :"<<hex<<x;
   cout<<"\n Octal Number :"<<oct<<x;
   return 0;
}
```

OUTPUT

```
Hexadecimal Number : 54
Octal Number : 124
```

Explanation: In the above program, the integer variable x is declared and initialized with 84. The first `cout` statement displays the decimal number to its equivalent hexadecimal number. The manipulator `hex` converts a decimal number to its hexadecimal equivalent. The second `cout` statement converts a decimal number to its equivalent octal number.

3.44 Write a program to read a number in hexadecimal format using `cin` statement. Display the number in decimal format.

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

int main()

{
   clrscr();
   int x;
   cout<<"\n Enter Hexadecimal Number :";
   cin>>hex>>x;
   cout<<"\n Decimal Number :"<<dec<<x;
   return 0;
}
```

OUTPUT

```
Enter Hexadecimal Number : 31
Decimal Number : 25
```

Explanation: In the above program, the `cin` statement reads a number in `hex` format. The `cout` statement displays its equivalent decimal number with the use of dec manipulator.

3.45 Write a program to demonstrate the use of endl manipulator.

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

int main()
{
   clrscr();
   cout<<" Demo of endl";
   endl(cout);
   cout<<" It splits a line" ;
   return 0;
}
```

OUTPUT

```
Demo of endl
It splits a line
```

Explanation: In the above program, `endl` manipulator is used to split the line. The two strings are displayed on two separate lines.

3.46 Write a program to demonstrate the use of `flush()` statement.

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

int main()
{
   char text[20];
   clrscr();
   cout<<"Enter text :";
   cin.getline(text,20);
   cout<<"Text entered :"<<text;
   flush(cout);
   return 0;
}
```

OUTPUT

```
Enter text : Buffer
Text entered : Buffer
```

Explanation: In the above program, the statement flush (`cout`) flushes the buffer. This statement can be used as `cout`<<flush. For more information on buffer, read Section 3.4.

3.14 USER-DEFINED MANIPULATORS

The programmer can also define his or her own manipulator according to the requirement of the program. The syntax for defining manipulator is as follows.

```
ostream & m_name ( ostream & o )
{
   statement1;
   statement2;
   return o;
}
```

The `m_name` is the name of the manipulator.

3.47 Write a program to create manipulator equivalent to '\t'. Use it in the program and format the output.

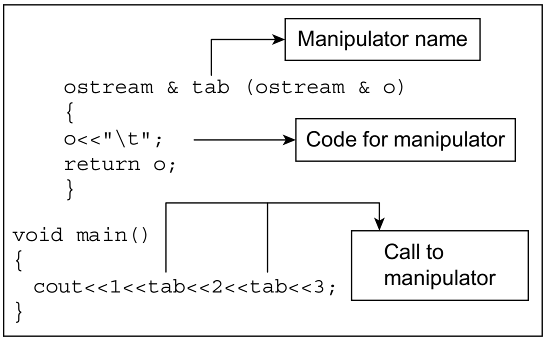
```
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
```

```
ostream & tab (ostream & o)
{
   o <<"\t";
   return o;
}
void main()
{
   clrscr();
   cout<<1<<tab<<2 <<tab<<3;
}
```

OUTPUT

1 2 3

Explanation: Figure 3.7 illustrates the working of the program.



```
ostream & tab (ostream & o)
{
o<<"\t";
return o;
}
void main()
{
  cout<<1<<tab<<2<<tab<<3;
}
```

Fig. 3.7 Working of tab manipulator

In the above program, `tab` named manipulator is defined. The definition of tab manipulator contains the escape sequence `'\t'`. Whenever we call the `tab` manipulator, the `'\t'` is executed and we get the effect of tab.

3.48 Write a program to display a message using manipulator.

```
#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
ostream & N (ostream & o)
{
   o <<"Negative Number";
   return o;
}
ostream & P (ostream & o)
{
   o <<"Positive Number";
   return o;
}
void main()
{
   int x;
   clrscr();
   cout<<"\n Enter a Number :";
   cin>>x;
   if (x<0)
   cout<<x <<" is" <<N;
   else
   cout<<x <<" is" <<P;
}
```

OUTPUT

```
Enter a Number : –15
–15 is Negative Number
```

Explanation: In the above program, two manipulators `N` and `P` are created. When called, `N` displays the message `"Negative number"` and `P` displays the message `"Positive number"`.

3.15 MANIPULATOR WITH ONE PARAMETER

The prototype declared in the `iomanip.h` as described earlier allows us to define individual set of macros. The manipulator can be created without the use of `int` or `long` argument.

3.49 Write a program with one parameter to display the string in the middle of the line.

```
#include<iostream.h>
#include<iomanip.h>
#include<string.h>
#include<conio.h>
ostream& fc (ostream& ost, int iw)
{
   for (int k=0;k<((75-iw)/2);k++)
   ost <<" ";
   return (ost);
}
OMANIP (int) middle (int iw)
{
   return OMANIP (int) (fc,iw);
}
```

```
int main()
{
   clrscr();
   char *m=" * Well come *";
   cout<<middle (strlen(m)) <<m;
   return 0;
}
```

OUTPUT

```
** WEL COME **
```

Explanation: In the above program, `middle` is a user-defined parameterized manipulator. It receives a value `strlen (m)`, that is, string length. The header file `IOMANIO.H` defines a macro, `OMANIO (int)`. It is expanded to `class_OMANIP_int`. Due to this, the definition consists of a constructor and an overloaded `ostream` insertion operator. When `middle()` function is added into the stream, it invokes the constructor which generates and returns an `OMANIP_int` object. The `fc()` function is called by the object constructor.

3.16 MANIPULATORS WITH MULTIPLE PARAMETERS

In this type of manipulators, multiple arguments are passed to the manipulator. The following program explains it.

3.50 Write a program to create a manipulator with two arguments.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<math.h>

struct w_p
{
   int w;
   int p;
};
IOMANIPdeclare (w_p);
static ostream& ff(ostream& os, w_p w_p)
{
   os.width (w_p.w);
   os.precision(w_p.p);
   os.setf(ios::fixed);
   return os;
}
OMANIP (w_p) column (int w, int p)
{
   w_p w_p;
   w_p.w=w;
   w_p.p=p;
   return OMANIP (w_p) (ff,w_p);
}
int main()
{
   clrscr();
   double n,sq,sqr;
   cout<<"number\t" <<"square\t" <<"\tsquare root\n";
   cout<<"====================================\n";
   n=1.0;
   for (int j=1;j<16;j++)
   {
   sq=n*n;
   sqr=sqrt(n);
   cout.fill('0');
   cout<<column(3,0)<<n <<"\t";
   cout<<column(7,1) <<sq <<"\t\t";
   cout<<column(7,6) <<sqr <<endl;
   n=n+1.0;
   }
   return 0;
}
```

OUTPUT

```
number  square      square root
===========================
001     0000001     0000001
002     0000004     1.414214
003     0000009     1.732051
004     0000016     0000002
005     0000025     2.236068
006     0000036     2.44949
007     0000049     2.645751
008     0000064     2.828427
009     0000081     0000003
010     0000100     3.162278
011     0000121     3.316625
012     0000144     3.464102
013     0000169     3.605551
014     0000196     3.741657
015     0000225     3.872983
```

Explanation: The above program is similar to the previous one. The only difference is that here two parameters are used. The user-defined manipulator is assigned two integer values. The first argument decides the number of spaces and the second argument decides the number of decimal places. After initializing the w_p structure, the constructor is called. The constructor creates and returns a _OMANIP object. The output of the program is shown above.

3.17 MORE PROGRAMS

3.51 Write a program to calculate the simple interest and total amount, input principal amount, period, and rate of interest.

{coderipe}

```
#include<iostream.h>
#include<conio.h>
int main()
{
   clrscr();
   int p_amount; // principle amount
   int period; // period in years
   int i_rate; // interest rate
   int interest; // interest
   int t_amount; // total amount
   cout<<endl<<"Enter Principle amount :";
   cin>>p_amount;
   cout<<"Enter period (years) :";
   cin>>period;
   cout<<"Enter interest rate :";
   cin>>i_rate;
   interest=(p_amount*period*i_rate) / 100;
   cout<<"Interest :"<<interest;
   t_amount=p_amount+interest;
   cout<<endl<<"Total Amount :" <<t_amount;
   return 0;
}
```

OUTPUT

```
Enter Principle amount : 10000
Enter period (years) : 3
Enter interest rate : 5
Interest : 189
Total Amount : 10189
```

Explanation: In the above program, integer variables `p_amount`, `period`, `i_rate`, `interest`, and `t_amount` are declared. The `principal amount`, `period` in `years`, and `rate of interest` are read through the keyboard using respective variable with `cin` statement.

The interest is calculated by the statement `interest=(p_amount*period*i_rate)/100` and the interest calculated is assigned to variable interest. The total amount is calculated by adding interest in the principle amount.

3.52 Write a program to set width and display the integer number.

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

int main()
{
   clrscr();
   cout.width(7);
   cout<<20123<<endl;
   cout.width(10);
   cout<<12541;
   return 0;
}
```

Explanation: In the above program, the `cout.width(7)` statement sets width to 7. The `cout` statement displays the five digits. The total number of digits are less than the total width. Hence, two blank spaces are displayed at the beginning as shown below.

		2	0	2	1	3

					2	0	2	1	3

3.53 Write a program to demonstrate use of showpos and showpoint flags.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
int main()

{

   clrscr();
   cout.setf(ios::showpos);
   cout.setf(ios::showpoint);
   cout.setf(ios::internal,ios::adjustfield);
   cout.precision(2);
   cout.width(9);
   cout<<587.4;
   return 0;
}
```

Explanation: The output of the program can be understood from the following table.

+			5	8	7	.	4	0

The showpos flag displays the positive sign at the beginning. The showpoint option displays decimal point and trailing zeros. The internal and adjustfield flags add blank spaces between the sign and the number.

3.54 Write a program to display results in right justification.

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

int main()
{
   clrscr();
   int k=0;
   long double n=1.00,f=1.00;
   cout.precision(0);
   cout.setf(ios::fixed);
   while (k<15)
   {
   f*=n++;
   cout.width(15);
   cout<<f<<"\n";
   k++;
   }
   return 0;
}
```

OUTPUT

```
            1
            2
            6
           24
          120
          720
         5040
        40320
       362880
      3628800
     39916800
    479001600
   6227020800
  87178291200
1307674368000
```

Explanation: In the above program, the integer variable `k` is initialized to 0 and long `double` variables `n` and `f` are initialized to 1.00. The precision is set to 0. The `setf()` function sets the precision to fixed position. The statement `cout.width (15)` fixes the field width. Thus, using `while loop` repetitive operations are performed. The output of the program is given above.

3.55 Write a program to display square and cube of numbers from 1.5 to 16.5 in table format.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<math.h>

int main()
{
   clrscr();
   int k=0;
   float num=1.5,sq,cb;
   cout<< "Number\t\t" <<"Square\t\t" <<"Cube\n";
   cout<<" ======================================\n";
   cout.setf(ios::fixed);
   while (k<16)
   {
   sq=pow(num,2);
   cb=pow(num,3);
   cout.fill('*');
   cout.width(4);
   cout.precision(0);
   cout<<num <<"\t\t";
   cout.width(6);
   cout.precision(2);
   cout<<sq<<"\t\t";
   cout.width(8);
   cout.precision(2);
   cout<<cb<<endl;
   num+=1;
   k++;
   }
   return 0;
}
```

OUTPUT

```
Number              Square              Cube
========            ========            =========
*1.5                **2.25              ****3.38
```

```
*2.5          **6.25         ***15.63
*3.5          *12.25         ***42.87
*4.5          *20.25         ***91.12
*5.5          *30.25         **166.38
*6.5          *42.25         **274.62
*7.5          *56.25         **421.88
*8.5          *72.25         **614.13
*9.5          *90.25         **857.37
10.5          110.25         *1157.63
11.5          132.25         *1520.88
12.5          156.25         *1953.12
13.5          182.25         *2460.37
14.5          210.25         *3048.63
15.5          240.25         *3723.88
16.5          272.25         *4492.12
```

Explanation: In the above program the variable `num` is initialized to 1.5. The `while loop` is executed repetitively and square and cube of the number are calculated using the `pow()` function. The functions `cout.width()` and `cout.precision()` set the field width and the number of floating points. The output of the program is shown above.

3.56 Write a program to display octal and hexadecimal equivalents of decimal numbers 100 to 200 with a difference of 10.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<math.h>

int main()
{
   clrscr();
   int k=100;
   cout<< "Number\t" <<"Octal\t" <<"Hexadecimal\n";
   cout<<"==========================\n";
   while (k<201)
   {
   cout.setf(ios::dec, ios::basefield);
   cout<<k <<"\t";
   cout.setf(ios::oct, ios::basefield);
   cout<<k<<"\t";
   cout.setf(ios::hex, ios::basefield);
   cout<<k<<endl;
   k+=10;
   }
   return 0;
}
```

OUTPUT

```
Number        Octal       Hexadecimal
=======       =======      ========
  100           144           64
  110           156           6e
  120           170           78
  130           202           82
  140           214           8c
  150           226           96
  160           240           a0
  170           252           aa
  180           264           b4
  190           276           be
  200           310           c8
```

Explanation: In the above program, `ios` base fields such as `oct`, `hex`, and `dec` are set using `setf()` function. After this setting the decimal number given in `cout()` before printing is converted to the above formats. The `while loop` repetitively performs this task and the result is displayed as above.

3.57 Write a program to create manipulator. Use `setf()` function.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
ostream & rs ( ostream & o)
{
   o <<"Rs.";
   return o;
}
ostream & arg (ostream & o)
{
   o.setf(ios::showpos);
   o.setf(ios::showpoint);
   o.fill(' ');
   o.precision(2);
   o<<setiosflags (ios::fixed) <<setw(7);
   return o;
}
```

```
int main()
{
   clrscr();
   cout<<rs<<arg<<132.58;
   return 0;
}
```

OUTPUT

```
Rs.+132.58
```

Explanation: In the above program, `rs` and `arg` are two user-defined manipulators. The arg manipulator contains various member functions of ios class that are used to manage the screen.

3.58 Write a program to define macro for %d and <<. Use them in `cout` and `printf` statements.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#define d <<j
#define i "\n%d"
int main()
{
   clrscr();
   int j=1;
   cout d ;
   printf (i,j);
   return 0;
}
```

OUTPUT

```
1
1
```

Explanation: The operator `<<` and variable `j` together are assigned to macro `d` and the format string `%d` is assigned to macro `i`. Both are used in `cout` and `printf` statements, respectively.

3.59 Write a program to display trailing decimal zeros using formatted functions.

```
#include<iostream.h>
#include<conio.h>
int main()
{
   clrscr();
   cout.setf(ios::showpoint);
   cout.precision(2);
   cout<<44.74<<endl;
   cout<<23.30<<endl;
   cout<<20.00;
   return 0;
}
```

Explanation: By default, the `cout` statement omits the trailing decimal zeros. The trailing zeros can be displayed using a setting as shown in the above program. The result of the above program is shown as follows in tabular format for the sake of understanding.

4	4	.	7	4
2	3	.	3	0
2	0	.	0	0

3.60 Write a program to display contents of an array using width() function. Fill the blank space with '*'.

```
#include<iostream.h>
#include<conio.h>
int main()
{
   clrscr();
   int i,j;
   float amount;
   float item[3][3]= {111,45,75.25,112,50,100.15,113,45,101.75};
   cout.precision(2);
   cout<<" Code No."<<" Qty" <<"\t Rate"<<endl;
   cout.fill('*');
   for ( i=0;i<3;i++)
   {
   cout<<endl;
   for (j=0;j<3;j++)
   {
   cout.width(8);
   cout<<item[i][j];
   }
   }
   return 0;
}
```

Explanation: In the above program, an array item[3][3] is declared and initialized with `float` numbers. The following formatting statements are used.

`cout.precision(2)` – Sets decimal point limit to two digits.
`cout.fill('*')` – Fills '*' in blank space of field.
`cout.width(8)` – Sets field width to 8.

The first and second `for` loops are used to read and display the contents of the array on the screen. The output of the program is as follows.

CODE NO.								QTY								RATE							
*	*	*	*	*	1	1	1	*	*	*	*	*	*	4	5	*	*	*	7	5	.	2	5
*	*	*	*	*	1	1	2	*	*	*	*	*	*	5	0	*	*	1	0	0	.	1	5
*	*	*	*	*	1	1	3	*	*	*	*	*	*	4	5	*	*	1	0	1	.	7	5

Here, each field has width 8. The numbers are displayed with right justification. The blank spaces are filled with '*' symbol.

SUMMARY

(1) The input/output function of C++ works with different physical devices. It also acts as an interface between the user and the device.

(2) A stream is a series of bytes and acts as source and destination for data. The source stream is called as `input stream` and the destination stream is called as `output stream.`

(3) The `cin`, `cout`, `cerr`, and `clog` are pre-defined streams.

(4) The header file `iostream.h` files must be `#include` when we use `cin` and `cout` functions.

(5) The `istream` and `ostream` are derived class from `ios` base class. Figure 3.4 displays all the derived classes.

(6) The formatting of output can be effectively done with member function of `ios` class. The member function `width()`, `precision()`, `fill()`, and `setf()` allows the user to design and display the output in formatted form.

(7) Table 3.6 describes the list of functions without bit fields. These functions are also used for formatting the output

(8) The `putback()` replaces the given character into the input stream. The member function `ignore` ignores the number of given character till it finds the termination character.

(9) Manipulators also help the user in formatting the output. The programmer can also create his or her manipulators.

(10) The header file `iomanip.h` contains pre-defined manipulators. Table 3.7 describes these manipulators.

EXERCISES

(A) Answer the following questions

(1) List the names of `pre-defined streams` with their `'C'` equivalents?

(2) What are formatted and unformatted I/O functions?

(3) Distinguish between

- (a) `cin()` and `scanf()`
- (b) `cout()` and `printf()`
- (c) `ios::fixed` and `cout.precision()`

(4) What are the uses of `put()` and `get()` functions?

(5) What is the use of `getline()` function? Which two arguments does it require?

(6) Describe the bit fields required in `setf()` function.

(7) List the flags without bit fields with their working.

(8) What is the role of `iostream.h` and `iomanip.h` header files?

(9) Write the statement for concatenation of two strings using `cout.write()` statement.

(10) Describe the procedure for designing manipulator.

(11) What is the function of `peek()` and `ignore()` functions?

(12) What are single and multiple parameter manipulators?

(13) In which format does the out statement display the address of a variable? How can it be converted to unsigned?

(14) In which situation is the `putback()` function useful?

(15) What is the use of `ignore()` function?

(16) What do you mean by formatted and unformatted data?

(B) Answer the following by selecting the appropriate option

(1) The `cin` and `cout` functions require the header file to include

- **(a) `iostream.h`**
- (b) `stdio.h`
- (c) `iomanop.h`
- (d) none of the above

(2) The `set.precision()` is used to set

- (a) number of digits
- **(b) decimal places**
- (b) field width
- (d) none of the above

(3) To fill unused sections of the field, the character is set by the function

- **(a) `fill()`**
- (b) `width()`
- (c) `precision()`
- (d) none of the above

(4) The manipulator <<endl is equivalent to

- **(a) '\n'**
- (b) '\t'
- (c) '\b'
- (d) none of the above

(5) This function accepts the string with blank spaces

- (a) `cin`
- (b) `scanf()`
- **(c) `getline();`**
- (d) none of the above

(6) The streams is a

- (a) flow of integers
- **(b) flow of data**
- (c) flow of statements
- (d) none of the above

(7) The statement `cout`<<hex<<15; gives the data in

- **(a) hexadecimal format**
- (b) octal format
- (c) binary format
- (d) decimal format

(8) The `gcount()` function counts the

- (a) inserted character

(b) unformatted extracted character
(c) both (a) and (b)
(d) none of the above

(9) The buffer is used to move data between
(a) input and output devices
(b) i/O devices and computer
(c) input devices to storage devices
(d) none of the above

(10) The buffer is a
(a) block of memory
(b) part of ram
(c) part of hard disk
(d) none of the above

(C) Attempt the following programs

(1) Write a program to read five `float` numbers with six decimal places.

(2) Write a program to display the hexadecimal and octal equivalent of 85, 25, 152, 251, and 458 numbers.

(3) Write a program to display decimal equivalent of hexadecimal numbers 0x52, 0x98, 0x101, 0x524, and 0x421.

(4) Write a program to read ten records of a student with the information Name, Age, and Date of Birth. Arrange the information (output) in the following format. Fill the unused field with dots.

```
Sr. no. Name Age  Date of Birth
====    ========  =========
01      Ajay       20      01/01/1991
```

(5) Write a program to accept string using `get()` function. Display the string using `write()` and `put()` function. Mention the difference between `write()` and `put()` functions.

(6) Write a program to design the following manipulators.

Manipulator Name	Function (Escape sequence)
`<<bkp (backspace)`	`'\b' or '\r'`
`<<newl`	`'\n'`
`<<bell`	`'\a'`
`<<plus`	`'+' (displays + sign)`
`<<dollar`	`$ (displays $ sign)`

(7) Write a program to read item code, quantity, and price and calculate the amount. Display the data in the following format.
Note: (a) Sr.no. and Item code are left justified.
(b) The price and amount are right justified.
(c) Three-digit precision for amount field.

```
Sr.no.     Item code     Quantity     Price     Amount
===        ======        =====        ====      =====
 01          0101          10         55.15     551.501
```

(8) Write a program to enter text up to 100 characters through the keyboard. Ignore? symbol and replace it with *. Display dot between two words.

(9) Given x = 5, y = 8, and z = 12, what will be the value of the variables on the left side of the following equations?
(a) a = x/y * z
(b) b = y/z * x
(c) c = z * y/z
(d) d = x * z/y
(e) d = (x/y) * z

(10) If p = 0.5, q = –2.0, r = 7.3, s = 10.4, m = 2, n = –3, what will be the value of the variables on the left side of the following equations.
(a) x = 5.0 * p + q – (r + s*3.0)
(b) y = 3.0 * p + q + s*2

(c) z = p * s + q*r – s * q/5.0
(d) a = sqrt(4.0 – r + 9.0 * q)

(11) Solve the following algebraic equations.
(a) $k = 4.5 \log_{10} x + 2xy + x^5$
(b) j = (bx + x)/ (bx – b)
(c) z = (x)2 + (x + 1)2 + (x + 2)2
(d) u = a * b/(c + d * g + k) + e

(12) Write a program to generate the following outputs.

-	#	#	#	7	8	4	.	5	0

*	4	5	·	4	8	$	$	$	2	8	.	3	4	*	*	5

	7	8	·	4	9			7	9	.	4						9

-	%	%	5	8	9	.	4	8	0

(13) Write a program to generate the following output.

```
         C
        C L
       C L A
      C L A S
     C L A S S
 C P L U S P L U S
   C P L U S P L
     C P L U S
       C P L
         C
```

(D) What will be the output of the following programs?

(1)
```
cout.setf(ios::left,
ios::adjustfield);
cout.fill('*');
cout.precision(2);
cout.width(7);
cout<<345.54;
cout.width(8);
cout<<78;
```

(2)
```
cout.setf(ios::internal,
ios::adjustfield);
cout.fill('$');
cout.precision(4);
cout.width(11);
cout<<-543.453;
```

(3)
```
cout.setf(ios::showpos);
cout.setf(ios::showpoint);
cout.setf(ios::internal,
ios::adjustfield);
cout.precision(4);
cout.width(12);
cout<<2342.34;
```

(4)
```
cout.precision(2);
cout<<4.57<<endl;
cout<<7.1453<<endl;
cout<<4.5132<<endl;
cout<<8.004<<endl;
```

(5)
```
cout.fill('$');
cout.precision(4);
cout.setf(ios::internal,
ios:: adjustfield);
cout.setf(ios::scientific,
ios:: floatfield);
cout.width(13);
cout<<-78.47854;
```

(6)
```
cout.fill('=');
cout.setf(ios::internal,
ios::adjustfield);
cout.setf(ios::scientific,
ios::floatfield);
cout.unsetf
(ios::scientific);
cout.width(15);
```

```
    cout<<- 3.121;
(7) cout.width (7);
    cout<<123<56;
```

(E) Additional programs

(1) Program to read integer, character, `float`, and `double` integer and display them.

```
#include<iostream.h>
#include<conio.h>
int main()
{
clrscr();
int a=10;
float b=10.4;
char c='b';
double d=12345.4567;
cout<<"\nValue of a is
:"<<a;
cout<<"\nValue of b is
:"<<b;
cout<<"\nCharacter assigned
to c is :"<<c;
cout<<"\n The double value d
is: :"<<d;
return(0);
}
```

OUTPUT

```
Value of a is :10
Value of b is :10.4
Character assigned to c is :b
The double value d is: :12345.4567
```

(2) Program to use different formats of typecasting and display the converted values.

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

int main()
{
      clrscr();
      int a=97;
      float b=100.4;
      char c='z';
      double d=12345.4567;
      cout<<"\n Integer
in character format is
:"<<char(a);
      cout<<"\n Float in
      integer format is
      :"<<int(b);
      cout<<"\n Character
      in integer format is
      :"<<int(c);
      cout<<"\n Double in
      integer format is
      :"<<int(d);
      return(0);
}
```

OUTPUT

```
Integer in character format is :a
Float in integer format is :100
Character in integer format is :122
Double in integer format is :12345
```

(3) Program to display HAI! using, C++ using the `put()` statement.

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

int main()
{
      clrscr();
      cout.put('H').
      put('A').put('I').
      put('!');
      cout<<endl;
      cout.put('C').
      put('+').put('+');
      return (0);
}
```

OUTPUT

```
HAI!
C++
```

(4) Program to display lower- and uppercase characters from a to z.

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

int main()
{
clrscr();
int j;
cout<<"Lowercase charac-
ters:";
for(j=122;j>96;j--)
cout<<(char)j<<" ";
cout<<endl;
cout<<"Uppercase charac-
ters:";
for(j=90;j>64;j--)
printf("% c",j);
return (0);
}
```

OUTPUT

Lowercase characters: z y x w v u t s r q p o n m l k j i h g f e d c b a
Uppercase characters: Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

(5) Program to read characters using a sequence of `get()` statements and display the characters read using a sequence of `put()` statements.

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

int main()
{
clrscr();
int age;
float num;
char sex;
char*msg="C++ is an object
          oriented lan-
          guage";
sex='F';
age=25;
num=1004.5;
cout<<"Sex:"<<sex<<endl;
cout<<"Age:"<<age<<endl;
cout<<"Entered float number
       is:"<<num<<endl;
cout<<"Message entered
is:"<<msg<<endl;
cout<<10<<" "<<20<<"
"<<30<<endl;
cout<<"Modified float number
is:";
cout<<num+5;
return(0);
}
```

OUTPUT

Sex:F
Age:25
Entered `float` **number is:1004.5**
Message entered is: C++ is an object oriented language
10 20 30
Modified `float` **number is:1009.5**

(6) Program to display the addresses of variables in hex and unsigned integer format.

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

int main()
{
clrscr();
int a=77,c=78;
float b=5.1252;
cout<<"Address of
a="<<&a<<endl;
cout<<"Address of
b="<<&b<<endl;
cout<<"Address of
c="<<(unsigned)&c<<endl;
return(0);
}
```

OUTPUT

Address of a=0x8f69fff4
Address of b=0x8f69ffee
Address of c=65522

(7) Program to read characters using read() and display it by write() function.

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

int main()
{
clrscr();
char name[6];
cout<<"\n Enter a string:";
cin.read(name,5);
cout<<endl;
cout<<"Entered string is:";
cout.write(name,5);
return(0);
}
```

OUTPUT

```
Enter a string:Amit
Entered string is:Amit
```

(8) Program to read a string using getline() function and display it using write() statement.

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

int main()
{
      char a[7];
      clrscr();
      cout<<"\n Enter any
      string:";
      cin.getline(a,7);
      cout<<"\n Entered
      string:"<<a;
      cout<<"\n Entered
      string:";
      cout.write(a,7);
      cout<<"\n Entered
      string;";
      cout.
      write(a,strlen(a));
      return(0);
}
```

OUTPUT

```
Enter any string:Mumbai
Entered string: Mumbai
Entered string: Mumbai
Entered string: Mumbai
```

(9) Program to generate the following output.

```
C
C+
C++
C++e
C++ea
C++eas
C++easy
C++easy
C++easy
```

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

int main()
{
      clrscr();
      char p,b[8];
      cout<<"\n Enter any
      string:";
      cin.getline(b,8);
      cout<<"\n Entered
      string:";

for(p=0;p<=strlen(b);p++)
      {
      cout.write(b,p);
      cout<<"\n";
      }

for(p=8;p>=strlen(b);p--)
      {
      cout.write(b,p);
      cout<<"\n";
}
      return(0);
}
```

OUTPUT

```
Enter any string:C++easy
Entered string:
C
C+
C++
C++e
C++ea
C++eas
C++easy
C++easy
C++easy
```

(10) Program to generate the following output.

```
Decimal.................64
Hexadecimal..........40
Octal.....................100
```

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

int main()
{
      clrscr();
      int a=64;
      cout<<setfill('.');
      cout<<setiosflags(ios:
      :left);
      cout<<setw(16)
      <<"Decimal";
      cout<<resetiosflags
      (ios::left);
      cout<<setw(6)
      <<dec<<a<< endl;
      cout<<setiosflags
      (ios::left);
      cout<<setw(15)
      <<"Hexadecimal";
      cout<<resetiosflags
      (ios::left);
      cout<<setw(6)
      <<hex<<a<<endl;
      cout<<setiosflags
      (ios::left);
      cout<<setw (15)<<"Oc-
      tal";
      cout<<resetiosflags
      (ios::left);
      cout<<setw(6)
      <<oct<<a<<endl;
      return(0);
}
```

OUTPUT

```
Decimal.................64
Hexadecimal..........40
Octal.......................100
```

(11) Program to generate the following output.

```
c
c+
c++
c++e
c++ea
c++eas
c++easy
c++easy
c++eas
c++ea
c++e
c++
c+
c
```

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

int main()
{
clrscr();
      char p,b[9];
      cout<<"\n Enter any
      string";
      cin.getline(b,9);
      cout<<"\n Entered
      string:";

for(p=0;p<=strlen(b);p++)
   {
      cout.write(b,p);
      cout<<"\n";
   }

for(p=strlen(b);p>=0;p--)
 {
      cout.write(b,p);
      cout<<"\n";
   }
      return(0);
}
```

OUTPUT

```
Enter any stringc++easy
Entered string:
c
c+
c++
c++e
c++ea
c++eas
c++easy
c++easy
c++eas
c++ea
c++e
```

```
c++
c+
c
```

(12) Program to generate output using flag bits such as showpos, showpoint, hex, and uppercase of given number.

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

int main()
{
       clrscr();
       cout.setf
       (ios::showpos);
       cout<<1254;
       cout.setf
       (ios::showpoint);
       cout<<"\n"<<1254.524;
       cout.setf(ios::hex,
       ios::basefield);
       cout<<"\n"<<184;
       cout.setf
       (ios::uppercase);
       cout<<"\n"<<184;
       return(0);
}
}
```

OUTPUT

```
+1254
+1234.524000
b8
B8
```

(13) Program to fill the trailing digits of number by '$' symbol.

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

int main()
{
       clrscr();
       int s=0,j=0;
       float pi;
       cout.fill('$');
       for(s=4;s>=1;s--)
       {
        pi=(float)22/7;
        cout.width(j++);
        cout.precision(s);
        cout<<"\n"<<pi;
       }
       cout.width(j++);
       cout.precision(s);
       cout<<"\n"<<pi;
       s=cout.fill();
       cout<<"\n    previous
       setting :"<<(char)s;
       return(0);
}
```

OUTPUT

```
3.1429
3.143$
3.14$$
3.1$$$
3.142857
previous setting :
```

(14) Program to print "*******ABC*******".

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

int main()
{
clrscr();
cout.fill('*');
cout.width(10);
cout<<"ABC";
cout.fill('*');
cout.width(7);
cout<<"*";
return(0);
}
```

OUTPUT

```
*******ABC*******
```

(15) Program to use different formats of width() and display the results.

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

int main()
{
       clrscr();
       cout.width(20);
       int x=cout.width(20);
```

```
      cout<<x;
      return(0);
}
```

OUTPUT

20

(16) Program to use skipws and stdio flags.

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

int main()
{
      clrscr();
      char x[5];
      cout<<"Enter array of
      characters:";
      cin>>x;
      cout.setf
      (ios::skipws);
      cout<<"Entered array
      of characters:"<<x;
      cin>>x;
      cout.
      setf(ios::skipws);
      cout<<"\nEn-
      tered array of
      characters:"<<x;
      cin>>x;
      cout.setf
      (ios::stdio);
      cout<<"\nEn-
      tered array of
      characters:"<<x;
      return(0);
}
```

OUTPUT

```
Enter array of characters: as h ok
Entered array of characters: as
Entered array of characters: h
Entered array of characters: ok
```

(17) Program to convert decimal to hexadecimal and octal number.

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

int main()
{
      clrscr();
      int a;
      cout<<"Enter the
      number :";
      cin>>a;
      cout<<"\n Hexadecimal
      number :"<<hex<<a;
      cout<<"\n Octal number
      :"<<oct<<a;
      return(0);
}
```

OUTPUT

```
Enter the number : 156
Hexadecimal number : 9c
Octal number : 234
```

(18) Program for converting hexadecimal number to decimal number.

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

int main()
{
      clrscr();
      int a;
      cout<<"\n Enter hexa-
      decimal number:";
      cin>>hex>>a;
      cout<<"\n Equiva-
      lent decimal number
      is:"<<dec<<a;
      return(0);
}
```

OUTPUT

```
Enter hexadecimal number is : f
Equivalent decimal number is : 15
```

(19) Program to convert octal number to hexadecimal and decimal number.

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

int main()
{
      clrscr();
```

```
        int a;
        cout<<"\n Enter octal
        number :";
        cin>>oct>>a;
        cout<<"\n  Equivalent
        hexadecimal number is
        :"<<hex<<a;
        cout<<"\n  Equivalent
        decimal   number   is
        :"<<dec<<a;
        return(0);
}
```

OUTPUT

Enter octal number : 124
Equivalent hexadecimal number is : 54
Equivalent decimal number is : 84

(20) Program to generate the following output.

****1000.2
*100.45

```
#include<iostream.h>
#include<conio.h>
int main()
{
      clrscr();
      cout.fill('*');
      cout.precision(3);
      cout.width(10);
      cout<<1000.2<<endl;
      cout.fill('*');
      cout.precision(2);
      cout.width(7);
      cout<<100.4532;
      return(0);
}
```

OUTPUT

```
****1000.2
*100.45
```

This page is intentionally left blank.