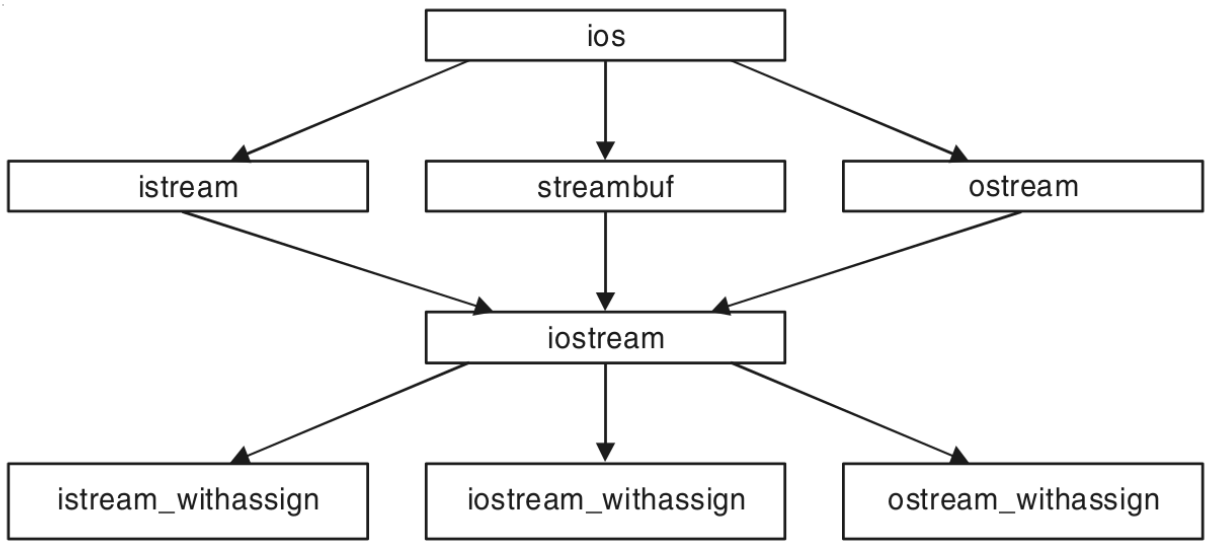
Input-Output and Manipulators

For managing the input and output operations C++ provides the concept of stream classes. We know that cin is treated as standard input stream and cout as standard output stream classes. cin is treated as standard input streams and cout as standard output streams. >> is an extraction operator << is insertion operator. This is so as cin >> extraction data from input stream from input stream and cout << displays inserts data into output stream. C++ provides numbers of stream classes for the efficient handling input and output and which is our next topic of discussion.

C ++ STREAM CLASSES

In C++ there are number of streams classes for defining various streams related with files and for doing input output operations. All these classes are defined in the file iostream. Figure given below shows the hierarchy of these classes :



In the above figure,

1. ios class is the topmost class in the stream classes hierarchy. It is the base class for istream, ostream and streambuf class.

2. istream, ostream serves the base classes for iostream class. The class istream is used for input and ostream for output.

3. Class ios is indirectly to iostream class using istream and ostream. To avoid the duplicity of data and member functions of ios class, it is declared as virtual base class when inheriting in istream and ostream as :

```
class istream : virtual public ios
{
};

class ostream :virtual public ios
{
};
```

4. The _withassign classes are provided with extra functionality for the assignment operations that's why _withassign classes.

After discussing about various stream classes we now understand their purpose and what type of facilities are provided by these stream classes.

1. The ios Class

The ios class is responsible for providing all input and output facilities to all other stream classes as it is the topmost class in the hierarchy of stream classes. As we will be seeing later in the chapter, this class provides number of functions for efficient handling of formatted output for strings and numbers.

2. The istream Class

This class is responsible for handling input stream. It provides number of functions for handling chars, strings and objects, record etc, besides inheriting the properties from ios class. The istream class provides the basic capability for sequential and random—access input. An istream object has a streambuf-derived object attached, and the two classes work together; the istream class does the formatting, and the streambuf class does the low-level buffered input. The class provides number of methods for input handling such as get, getline, read, peek, gcount, ignore, eatwhite, putback etc. This class also contains overloaded extraction operator >> for handling all data types such as int, signed int, char, long, double, float, long double. The extraction

operator is also overloaded for handling streambuf and istream types of objects.

3. istream_withassign Class

The istream_withassign class is a variant istream that allows object assignment. The predefined object cin is an object of this class and thus may be reassigned at run time to a different istream class.

4. ostream_withassign Class

The ostream_withassign class is variant of ostream that allows object assignment. The predefined object cout, cerr and clog are objects of this class and thus may be reassigned at run time to a different ostream object.

UNFORMATTED INPUT/OUTPUT

Until now is all the program with the help of cin and cout we have taking input without making use of any formatting function. Thus, general syntax of using data as input and output is as follows :

```
cin>>data1>>data2>>data3>>.....>>data n;
cout<<data1<<data2<<data3<<.....<<data n;
```

Where data1, data2.... Are variables of int, char, float etc.

Here data is scanned as it is input and similarly it is displayed as it. This is because the operator >> and << overloaded for all the basic types like int, char, unsigned, float etc. In case of string data the input breaks at occurrence of very first white space character like tab or space. Similarly for reading int, float etc., input must match the type of variable in which you are accepting it. Before discussing how to format data for input and output we first see few functions for input and output character, scanning and printing strings.

1. The get Function

This function is used to scan a single character from the keyboard. There are two different syntaxes of using get function.

(a) void cin.get(char);

(b) char cin.get( );

In the first syntax get function takes an argument of type char. This is used as :

```
char x;
cin.get(x);
```

The input character entered from keyboard is taken into the x.

In the second syntax the entered character is returned by the get function. This is used as :

```
char x = cin.get ( );
```

```
/*INPUTTING A CHARACTER USING get( )*/
#include <iostream>
using namespace std;
int main( )
{
    char ch;
    cout<<"Enter a character :=";
    cin.get(ch);
    cout<<"You have entered :="<<ch<<endl;
}
```

OUTPUT :

Enter a character : =S

You have entered : =S

2. The put Function

The function is used to put a single character onto the screen. Its general syntax is given as :

```
void put (char);
```

The character to be displayed is passed as argument to function put. This is used as :

```
char x = 'P';
cout.put(x)
```

We can even pass ASCII values to put function which is converted internally to their character counterpart i.e.,

```
cout.put (97)
```

Will display a.

```
/*INPUTTING A CHARACTER USING get( ) AND OUTPUT USING put( )*/
#include <iostream>
using namespace std;
int main( )
{
    char ch;
    cout<<"Enter a character :=";
    ch=cin.get( );
    cout<<"You entered :=";
    cout.put(ch);
return 0;
}
```

OUTPUT :

Enter a character : =a

You entered : =a

3. The getline Function

The syntax of the function is given as :

```
istream& getline (char *pch, int ncount, char delim='\n');
```

We have seen usage of this function in number of program earlier. The function getline scans character into the array pch till nCount-1 has been scanned or delim is encountered, the default is '\n'.

An example.

```
char str [20];
cin.getline (str, 20,'$');
```

This function scans first 19 character or break at the very first occurrence of '$' character. The function getline scans all white characters like tab, spaces etc. As function returns reference of istream type we can write function getline as :

```
char s1[10], s2 [10];
cin. getline (s1, 10).getline (s2, 10);
```

```
/*ENTER A LINE OF TEXT AND DISPLAY IT BACK USING getline AND cout */
#include <iostream>
using namespace std;
int main( )
{
        char str[50];
        cout<<"Enter line of text"<<endl;
        cin.getline(str,50);
        cout<<"You entered "<<endl;
        cout<<str<<endl;
        return 0;
}
```

OUTPUT :

Enter line of text

I am learning C++

You entered

I am learning C++

4. The write Function

```
ostream & write (const char * pch, int nCount);
```

The function display nCount character from the array pch. For instance.

```
cout.write ("hello", 3);
```

display : hel

As return type of function write is ostream type we can write multiple write statement s to form one as :

```
cout.write ("hello", 5).write ("world", 6);
```

is equal to

```
cout.write ("hello", 5);
cout.write ("world", 6);
```

```
/*DISPLAY PATTERN USING write METHOD */
#include <iostream>
#include <string>
using namespace std;
int main( )
{
    char str [20];
    int i,len;
    cout<<"======================================"<<endl;
    cout<<"Enter a string"<<endl;
    cout<<"======================================"<<endl;
    cin.getline(str,20);
    len=strlen(str);
    cout<<"======================================"<<endl;
    cout<<"Pattern is"<<endl;
    for(i=1;i<=len;i++)
    {
        cout.write(str,i);
        cout<<endl;
    }
    for(i=len;i>=1;i--)
```

```
        {
                cout.write(str,i);
                cout<<endl;
        }
        cout<<endl;
        cout<<"======================================"<<endl;
        return 0;
}
```

OUTPUT :

```
==========================================
Enter a string
==========================================
UNIVERSITY
==========================================
Pattern is
U
UN
UNI
UNIV
UNIVE
UNIVER
UNIVERS
UNIVERSI
UNIVERSIT
UNIVERSITY
UNIVERSITY
UNIVERSIT
UNIVERSI
UNIVERS
UNIVER
UNIVE
UNIV
UNI
UN
U
```

==========

```
/*DISPLAYING FULL NAME USING write METHOD */
#include <iostream>
using namespace std;
int main( )
{
        char fname [20],lname[20];
        cout<<"Enter first name :=";
        cin>>fname;
        cout<<"Enter the last name :=";
        cin>>lname;
        cout<<"Your full name :=";
        ostream & refcout=cout.write(fname, strlen(fname));
        refcout.write(" ",1);
        refcout.write(lname,strlen(lname));
        cout<<endl;
        return 0;
}
```

OUTPUT :

Enter first name : =Vipin

Enter the last name : =S

Your full name : =Vipin S

The whole of the write statements can be written in the shorter form as :

cout.write(fname, strlen (fname))

Write("",1).write (lname,strlen(lname));

5. The gcount Function

The prototype is

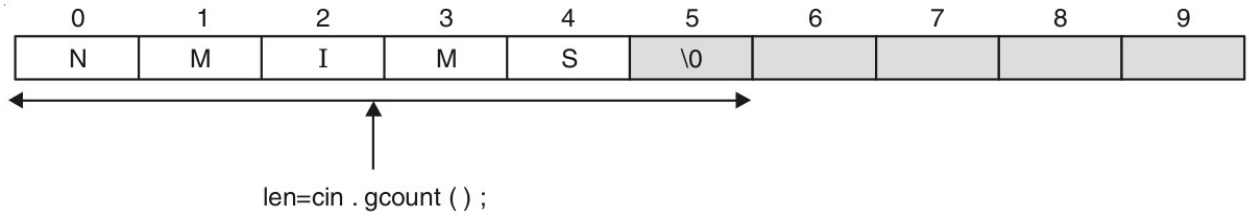
```
int gcount ( ) const;
```

Returns the number of characters extracted by the last unformatted input function. Formatted extraction operators may call unformatted input functions and thus reset this number.

```
#include <iostream>
using namespace std;
#include <ctype.h>
int main( )
{
char str[15];
cout<<————————————————————"<<endl;
cout<<"ENTER A STRING HERE";
cout<<endl<<"————————————————————"<<endl;
cin.getline(str,15);
int len;
len=cin.gcount( );
cout<<endl<<"————————————————————"<<endl;
cout<<"LENGTH OF STRING IS :="<<len-1<<endl;
cout<<"————————————————————"<<endl;
return 0;
}
```

OUTPUT :

```
————————————————————
ENTER A STRING HERE
————————————————————
MPSTME NMIMS
————————————————————
LENGTH OF STRING IS : =12
————————————————————
```



6. The putback Function

The prototype of the function is given as :

istream & putback(char ch);

Puts a character back into the input stream. The putback function may fail and set the error state. If ch does not match the character that was previously extracted, the result is undefined.

```
/* putback FUNCTION */
#include <iostream>
using namespace std;
int main( )
{
cout<<"ENTER A STRING"<<endl;
char ch;
ch=cin.get( );
while(ch!='\0')
{
if(ch=='$')
cin.putback('&');
cout.put(ch);
ch=cin.get( );
}
cout<<endl;
return 0;
}
```

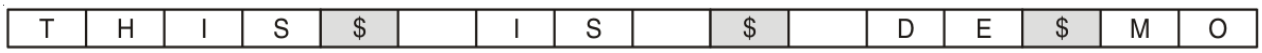
OUTPUT :

ENTER A STRING

This$ is $ de$mo

This$& is $& de$&mo

Before applying putback function

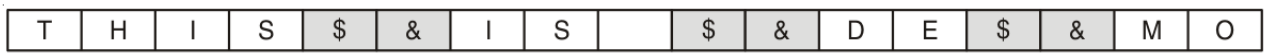


When compiler control finds the statements given below then after '$' symbol put '&'.

if(ch=='$')

cin.putback('&');

After applying putback function



7. The peek Function

The prototype is given as :

```
int peek ( );
```

Returns the next character without extracting it from the stream. Returns EOF if the stream is end of file.

8. The eatwhite Function

The prototype is given as follows :

```
void eatwhite ( );
```

Extract white spaces from the stream by advancing the get pointer past spaces and tabs.

```
#include <iostream>
using namespace std;
int main( )
{
ifstream file("num.txt");
char ch;
file.get(ch);
while(file.eof( )==0)
{
```

```
cout.put(ch);
file.eatwhite( );
file.get( );
}
file.close( );
return 0;
}
```

OUTPUT :

Thisisdemoofeatwhite

9. The ignore Function

istream & ignore (int nCount =1,int delim=EOF);

Extracts and discards up to nCount characters. Extraction stops if the delimiter delim is extracted or the end of file is reached. If delim = EOF (the default), then only the end of file condition causes termination. The delimiter character is extracted.

```
/*PEEK AND IGNORE FUNCTION */
#include <iostream>
using namespace std;
int main( )
{
cout<<"ENTER A STRING"<<endl;
char ch,x;
ch=cin.get( );
while(ch!='\n')
{
cout.put(ch);
x=cin.peek( );
while(x=='$')
{
cin.ignore(1,'$');
x= cin.peek( );
}
```

```
ch=cin.get( );
}
cout<<endl;
return 0;
}
```

OUTPUT :

```
ENTER A STRING
This $ is $$ de$mo$.
This is demo.
```

FORMATTED INPUT/OUTPUT OPERATIONS

In C++ for formatting input and output we have three methods :

1. Use of functions and flags defined by ios class.
2. Use of manipulators (built-in)
3. User defined manipulators.

1. Use of Functions and Flags Defined by ios Class

A. ios Function and Flags

The class ios provides number of formatting functions for input and output. The most frequently used are listed below :

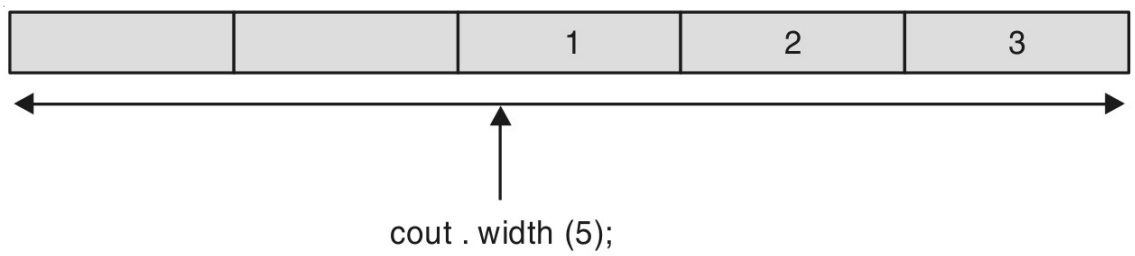
(a) The width ( ) Function

The function is used to set the width i.e., field size for displaying a data value of type numeric or string. The syntax of this is as follows :

```
int width (int);
```

It can be used with cout object as follows :

```
cout.width (5);
cout<<123;
```



```
/*Width Function*/
include <iostream>
using namespace std;
int main( )
{
        int a = 1234, b = 645, c = 28, d = 8;
        cout<<"==================================="<<endl;
        cout<<"THE DEMO OF width IS GIVEN BELOW"<<endl;
        cout<<"==================================="<<endl;
        cout.width(8);
        cout<<a<<endl;
        cout<<b<<endl;
        cout<<c<<endl;
        cout<<d<<endl;
        cout<<"==================================="<<endl;
        return 0;
}
```

OUTPUT :

```
===================================
THE DEMO OF width IS GIVEN BELOW
===================================
1234
645
28
8
```

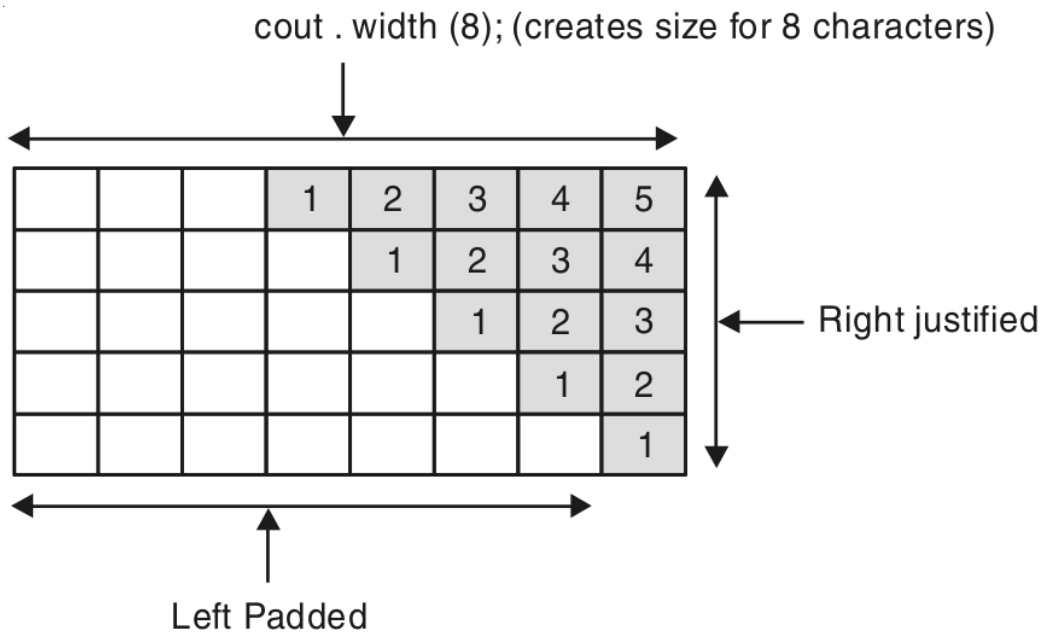
Another example

```
#include <iostream>
using namespace std;
int main( )
{
        int a= 12345, b = 1234, c =123, d = 12, e=1;
        cout<<"==================================="<<endl;
```

```
        cout<<"\tTHE PATTERN IS GIVEN AS"<<endl;
        cout<<"======================================"<<end;
        cout.width(8);
        cout<<a<<endl;
        cout.width(8);
        cout<<b<<endl;
        cout.width(8);
        cout<<c<<endl;
        cout.width(8);
        cout<<d<<endl;
        cout.width(8);
        cout<<e<<endl;
        cout<<"======================================"<<endl;
        return 0;
}
```

OUTPUT :

```
==========================================
THE PATTERN IS GIVEN AS
==========================================
12345
1234
123
12
1
```



(b) The precision Function

The function is used for setting the number of digits to be displayed after a floating point number.

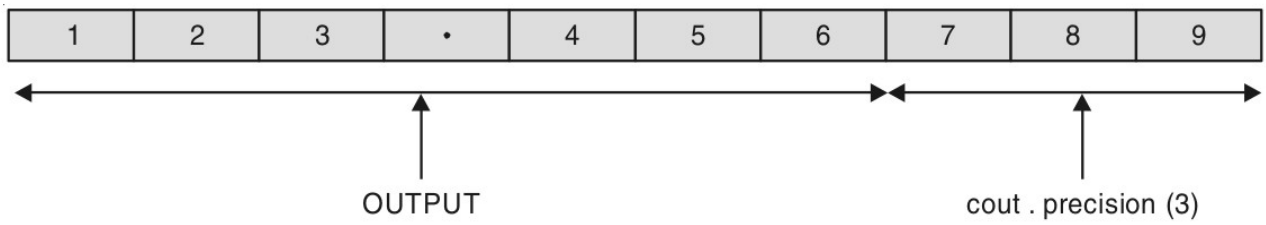
Its syntax is

```
int precision (int);
```

It can be used as :

```
cout.precision (3);
cout<<123.456789;
```

Output displayed is given as :




```
/*PRECISION FUNCTION */
#include <iostream>
using namespace std;
int main( )
{
cout<<"#####################################"<<endl;
cout<<" DEMO OF PRECISION FUNCTION"<<endl;
cout<<"#####################################"<<endl;
cout.precision (3);
cout<<"\t"<<123.12345<<endl;
cout.precision(4);
cout<<"\t"<<345.656767<<endl;
cout.precision(10);
cout<<"\t"<<22/7.0<<endl;
cout<<endl;
cout<<"#####################################"<<endl;
return 0;
```

```
}
```

OUTPUT :

```
####################################
DEMO OF PRECISION FUNCTION
####################################
123.123
345.6568
3.1428571429
####################################
```

(c) The fill Function

The fill function is used to fill the empty spaces in the given field size set by width function.

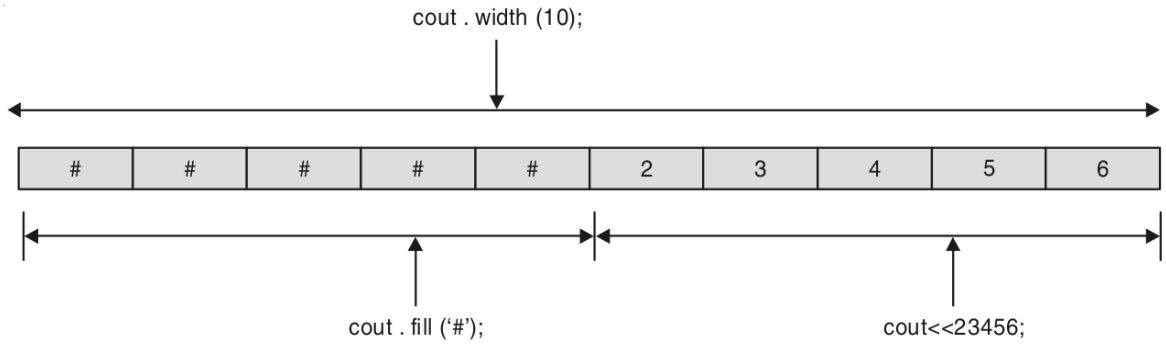
Its syntax is given as :

```
char fill (char);
```

It can be used as :

```
#include <iostream>
using namespace std;
int main( )
{
int a = 1234, b = 645, c = 28, d = 8;
cout.width (10);
cout.fill ('#');
cout<<23456;
return 0;
}
```

The output for the above given code snippet is given as :

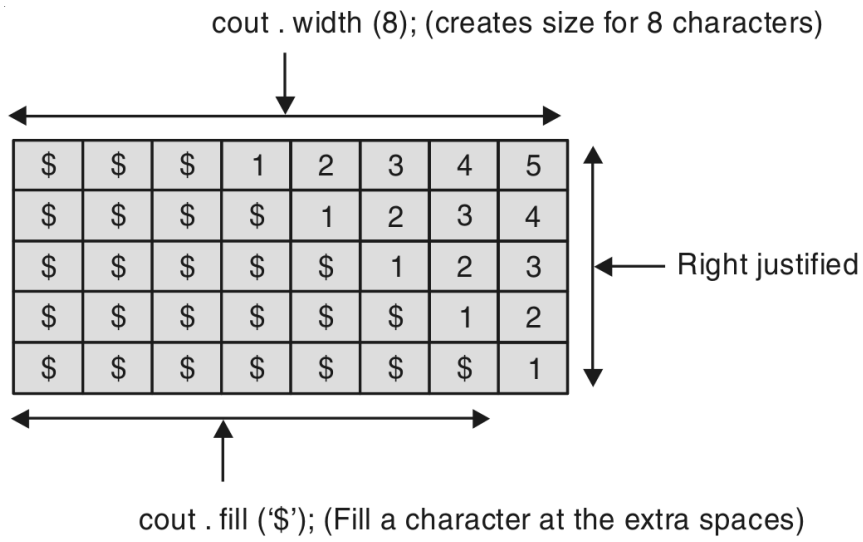


```
/*WIDTH AND FILL FUNCTION TOGETHER */
#include <iostream>
using namespace std;
int main( )
{
int a = 12345, b = 1234, c= 123, d = 12, e=1;

cout<<"+*+*+*+**+*+*+*+*+*+*+**+*+*+*+*+*+*+*"<<endl;
cout<<"DEMO OF fill AND width FUNCTION"<<endl;
cout<<"+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*"<<endl<<endl;
cout.fill('$');
cout.width(8);
cout<<a<<endl;
cout.width(8);
cout<<b<<endl;
cout.width(8);
cout<<c<<endl;
cout.width(8);
cout<<d<<endl;
cout.width(8);
cout<<e<<endl;
cout<<endl<<"+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*"<<endl;
return 0;
}
```

OUTPUT :

```
+*+*+*+*+*+*+*+*+*+*+*+**+*+*+*+*+*+*+**+*+*+*+*+*+*+*
DEMO OF fill AND width FUNCTION
+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
$$$12345
$$$$1234
$$$$$123
$$$$$$12
$$$$$$$1
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*
```



(d) The setf Function

This function is used for setting the various flags for controlling the output such as displaying such as displaying output left justified or right justified, displaying numbers in decimal, hex, octal, scientific notation etc. The setf function is very important function for printing output in a verity of fashion. Whether it is numeric data or string data. The function stands for setting the flag. There are two different variant of setf.

(i) The first syntax of setf is given below as :

```
long setf (long Flags, long Mask);
```

The first statement Flags is number of flags defined in the class ios. Each flag is equivalent of 1 if set. For combining many flag, OR (|) can be used. The second parameter Mask specifies the group to which first parameter belongs. The Mask parameter is also known as bit-field. The table given below shows all types of formatting flags and their corresponding Mask.

Formatted Flags Used in C++

Sl. No.	Flags	Mask	Output
1.	ios : :left	ios : :adjustfield	Left justified
2.	ios : :right	ios : :adjustfield	Right justified
3.	ios : :internal	ios : :adjustfield	Add fill characters after any leading sign or base indication
4.	ios : :scientific	ios : :floatfield	Scientific notation
5.	ios : :fixed	ios : :floatified	Fixed point notation
6.	ios : :doc	ios : :basefield	Output in decimal
7.	ios : :hex	ios : :basefield	Output in hex
8.	ios : :oct	ios : :basefield	Output in octal

(g) The Second Syntax of setf is :

long setf (long flags);

Function turns on only those format bits that are specified b 1s in flags. It returns a long that contains the previous values of all the flags. These are the flags which are used independently without the use of Mask parameter. The table given below lists all the flags :

Sl. No.	Flags	Purpose
1.	ios : :showpoint	Show decimal point and trailing zero for floating-point values.
2.	ios : :showbase	Display numeric constants in a format that can be read by the C++ compiler.
3.	ios : :showpos	Show plus sign (+) for positive values.
4.	ios : :uppercase	Display uppercase A through F for hexadecimal values and E for scientific values.
5.	ios : :skipws	Skip white space on input
6.	ios : :unitbuf	Flush the stream after each insertion.
7.	ios : :stdio	Flush stdout and stderr after each insertion

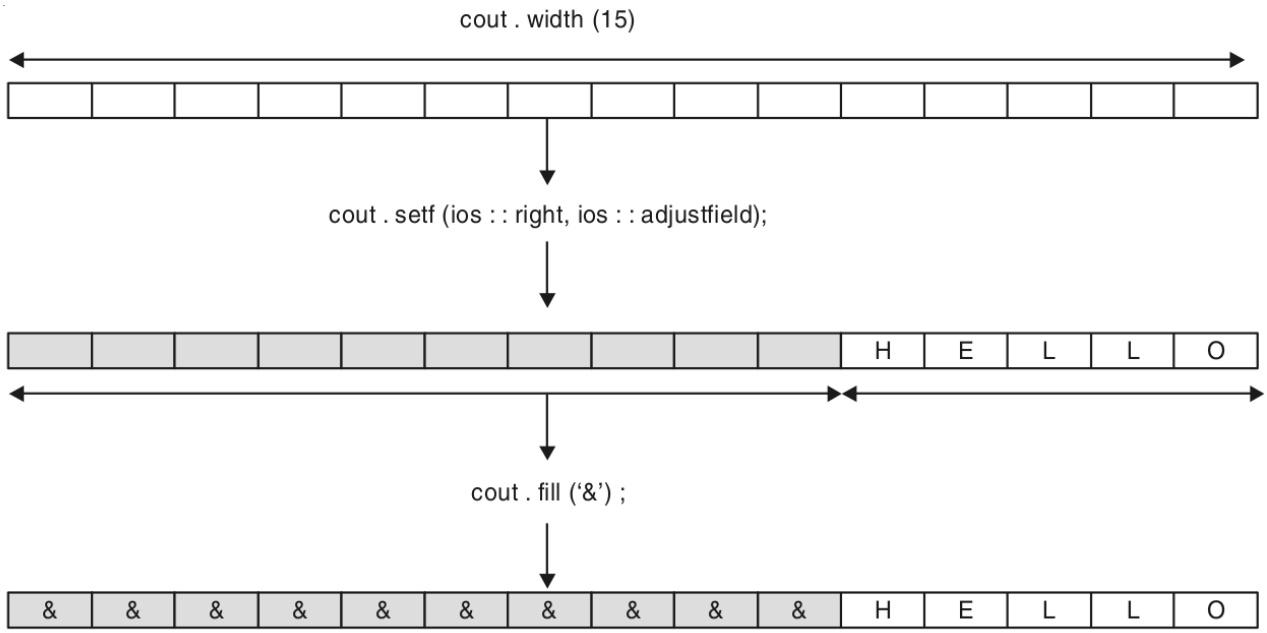
```
/* FORMATTING FLAGS */
#include <iostream>
using namespace std;
int main( )
{
char s[ ]="Hello";

cout<<"————————————————————————"<<endl;
cout<<"STRING WITHOUT SETTING WIDTH\n";
cout<<"————————————————————————"<<endl;
cout<<s<<endl;
cout<<endl<<"————————————————————"<<endl;
cout<<"STRING AFTER SETTING WIDTH AND PADDING"<<endl;
cout<<"————————————————————————"<<endl;
cout<<"RIGHT JUSTIFIED"<<endl;
cout<<"————————————————————————"<<endl;
cout.fill('&');
cout.setf(ios : :right, ios : :adjustfield);
cout.width(15);
```

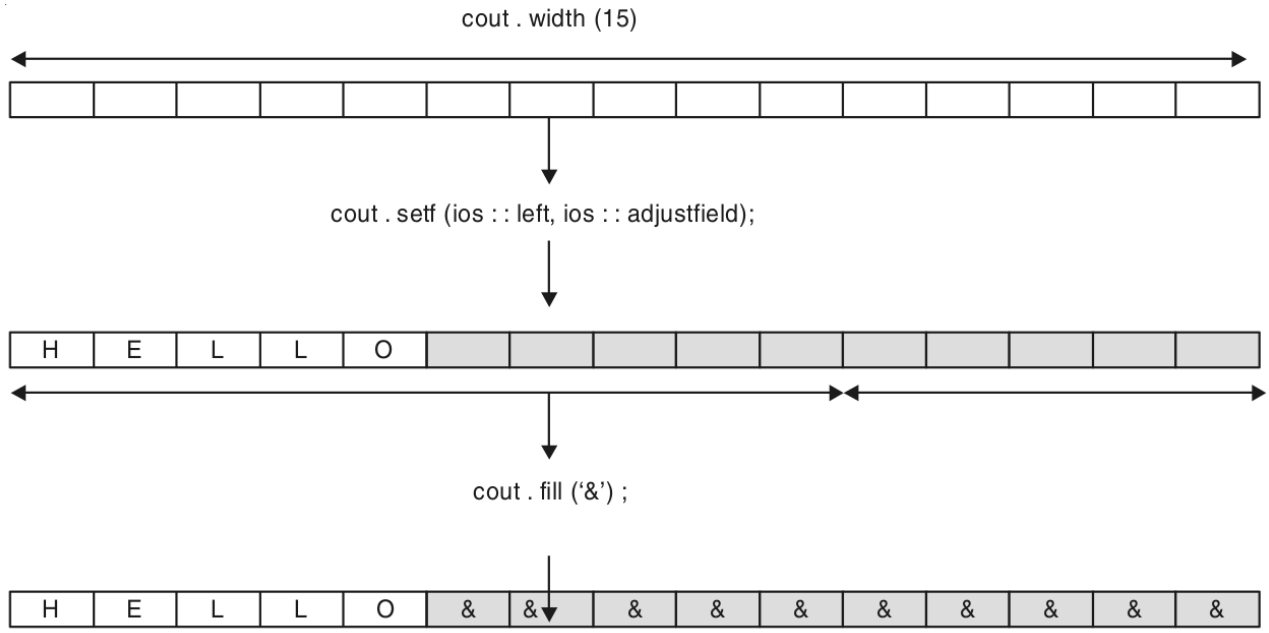
```
cout<<s<<endl;
cout<<"————————————————————————"<<endl;
cout<<"LEFT JUSTIFIED"<<endl;
cout<<"————————————————————————"<<endl;
cout.fill('&');
cout.setf(ios : : left,ios : : adjustfield);
cout.width(15);
cout<<s<<endl;
cout<<"————————————————————————"<<endl;
return 0;
}
```

OUTPUT :

```
————————————————————————-
STRING WITHOUT SETTING WIDTH
————————————————————————-
Hello
————————————————————————-
STRING AFTER SETTING WIDTH AND PADDING
————————————————————————-
RIGHT JUSTIFIED
————————————————————————-
&&&&&&&&&&Hello
————————————————————————-
LEFT JUSTIFIED
————————————————————————-
Hello&&&&&&&&&&
————————————————————————-
```



Now, by changing ios :: right to ios :: left we get the output as :



Another example

```
#include <iostream>
using namespace std;
int main( )
{

cout<<"====================================="<<endl;
cout<<"THE FIRST FORMATTED FLAG"<<endl;
cout<<"====================================="<<endl;
cout.fill('+');
cout.width(10);
cout.setf(ios : :internal, ios : :adjustfield);
cout<<-7.89<<endl;
cout<<"====================================="<<endl;
cout<<"SECOND FORMATTED FLAG"<<endl;
cout<<"====================================="<<endl;
cout.fill('X');
cout.precision(3);
```

```
cout.width(15);
cout.setf(ioClass ios is indirectly to iostream class using istream and ostream.s
: :internal,ios : :adjustfield);
cout.setf(ios : :scientific,ios : :floatfield);
cout<<−32.43567<<endl;
cout<<endl<<"================================="<<endl;
return 0;
}
```

OUTPUT :

```
=================================
THE FIRST FORMATTED FLAG
=================================
-+++++7.89
=================================
SECOND FORMATTED FLAG
=================================
-XXXXX3.244e+01
=================================

}
```

```
/* SHOW POS AND SHOWPOINT FLAGS */
#include <iostream>
using namespace std;
int main( )
{
cout.setf(ios : :showpos);
cout.setf(ios : :showpoint);
cout.width(8);
cout.precision(4);
cout<<125<<endl;
cout.width(8);
cout<<23.0<<endl;
cout.width(8);
cout<<34.5<<endl;
cout.setf(ios : :hex,ios : :basefield);
cout.setf(ios : :uppercase);
cout<<0x34f<<endl;
return 0;
}
```

OUTPUT :

```
+125
+23.0000
+34.5000
34F
```

For the first three cout width of 8 is set. So output appears as :

				+	1	2	5
+	2	3	.	0	0	0	0
+	3	4	.	5	0	0	0

In last cout output appear in hex in uppercase due to flag ios : :uppercase.

(e) The unsetf Function

The function is used for clearing the flags previously set by setf function.

```
/* RETURN TYPE OF WIDTH AND PRECISION FUNCTION */
#include <iostream>
using namespace std;
int main( )
{
cout.width(10);
int pw= cout.width(5);
cout<<"previous width="<<pw<<endl;
cout.precision(3);
int pp=cout.precision(4);
cout<<"previous precision="<<pp<<endl;
return 0;
}
```

OUTPUT :

```
previous width=10
previous precision=3
```

```
//Giving new name to cin and cout

#include <iostream>
using namespace std;
int main( )
{
ostream print(1);
istream_withassign scan;
scan = cin;
print<<"————————————————"<<endl;
print<<"ENTER YOUR NAME"<<endl;
print<<"————————————————"<<endl;
char str[15];
```

```
scan.getline(str,15);
print<<endl<<"————————————————"<<endl;
print<<"HELLO "<<str<<endl;
print<<endl<<"————————————————"<<endl;
return 0;
}
```

OUTPUT :

```
————————————————
ENTER YOUR NAME
————————————————
VIPIN
————————————————
HELLO VIPIN
————————————————
```

MANIPULATORS

Manipulators are functions which are used to manipulate the input/output formats. Manipulators are of two types :

(a) One which take arguments.
(b) Second which does not take any argument.

One type of manipulator which we have seen is endl which is used to insert new line into stream. All the manipulators which are built-in and which is used to insert new line are defined in the header file iomanip.h. The file iostream.h provides some manipulators which does not take any argument.

Most of the built-in manipulators are similar to their setf counterpart like width, precision, fill etc. The advantage here is that manipulators can directly be put into the stream as :

```
cout<<setw(5)<<123;
cout<<x<<endl;
cout<<hex<<x;
```

All manipulators which do not take any argument are having the following syntax :

```
ostream & mani_name(ostream &)
{
}
```

For example the most commonly used manipulator endl is defined as :

ostream endl(ostream &);

When we write cout<<endl; It is interpreted internally as cout.operator<<(endl); Again as endl takes a reference of ostream as argument writing endl(cout); is equivalent to writing cout<<endl;

Sl. No.	Manipulator	Meaning
1.	setw(int n)	Sets the output width of n characters
2.	setfill(char x)	The manipulator sets the stream's fill character
3.	setprecision(int n)	Sets the precision for floating points number
4.	hex	Converts to hexadecimal
5.	oct	Convert to octal
6.	dec	Convert to decimal
7.	left	Left justify, right padding
8.	right	Right justify, left padding
9.	endl	Insert a new line and flush output stream
10.	uppercase	Displays A-F for hex and E for scientific
11.	showpos	To insert a plus sign in a non-negative generated numeric field
12.	scientific	To insert floating-point values in scientific format

13.	fixed	To insert floating-point values in fixed-point format
14.	setiosflags(flags f)	Sets the formatting flags specified by f. setting remains in effect until changed.
15.	resetiosflags(flags f)	Clear the formatting flags specified by f. setting remains in effect until changed.

```
#include <iostream>
#include <iomanip.h>
using namespace std;
int main( )
{

cout<<setw(5)<<345<<endl;
cout<<setw(5)<<45<<endl;
cout<<5<<endl;
cout<<setfill('$')<<setw(10)<<"hello"<<setw(10)<<345.678<<endl;
return 0;
}
```

OUTPUT :

```
345
45
5
$$$$$hello$$$345.678
```

Creating Your Own Manipulators

Apart from using the built-in manipulators of C++, you can create your own manipulator for whatever purpose you want. For creating your own manipulator which does not take any argument the syntax is as shown earlier :

```
ostream & manip_name (ostream & mycout)
{
————————————;
————————————;
————————————;
return mycout;
}
```

All user defined manipulator must take an argument by reference of ostream type and return ostream type by reference. As we pass reference of cout when use this manipulator, actual changes done by manipulator occur as if we are working with cout.

For example consider a manipulator which emulates double space i.e., when we use it leaves two spaces in the stream and continues.

```
ostream & DS(ostream & mycout)
{
mycout<<" ";
return mycout;
}
```

To use this we can write as cout<<DS<<"hello"<<DS;

This first leaves two spaces, prints "hello" and then again leaves two spaces. As mentioned earlier the main advantage of manipulators whether user defined or built-in that they can be put into the cin and cout stream easily using >> and <<. This advantage of manipulator makes them popular.

```
#include <iostream>
using namespace std;
ostream & DS(ostream & mycout)
{
mycout<<" ";
return mycout;
}
int main( )
{
cout<<"Before Applying manipulator";
endl(cout);
cout<<"HELLO";
endl(cout);
cout<<"After applying own manipulator";
endl(cout);
cout<<DS<<"HELLO"<<DS;
return 0;
}
```

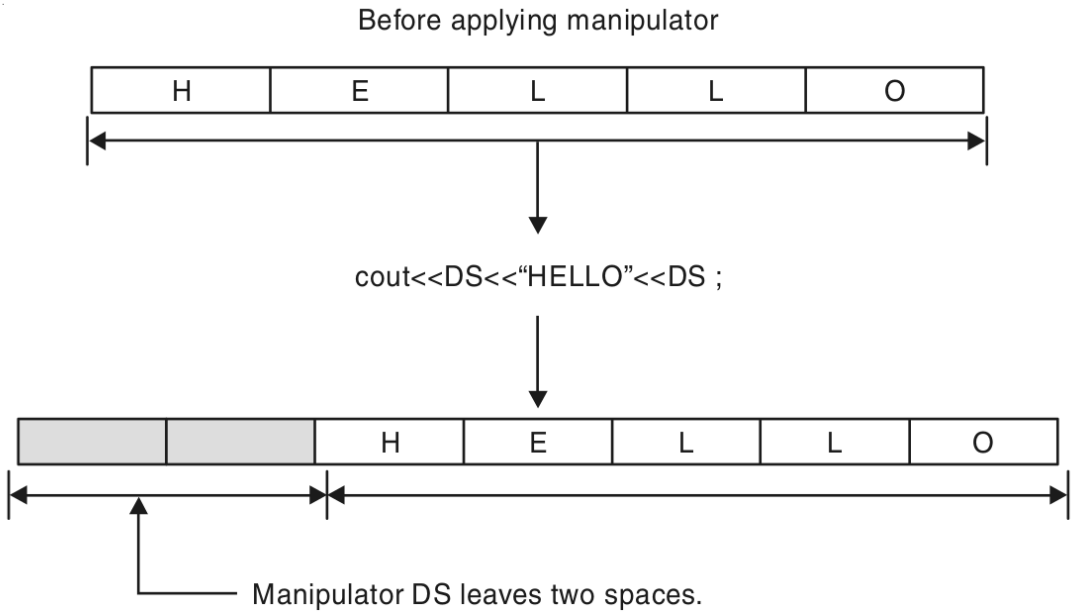
OUTPUT :

Before Applying manipulator

Hello

After applying own manipulator

HELLO



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

/* Manipulator for two spaces */
ostream & DSD(ostream & mycout)
{
mycout<<"";
return mycout;
}

/*Manipulator for three spaces */
ostream & DST(ostream & mycout)
{
mycout<<"";
return mycout;
}

int main( )
{
cout<<"Before Applying manipulator";
endl(cout);
cout<<"HELLO";
endl(cout);
cout<<"After applying own manipulator DSD";
endl(cout);
cout<<DSD<<"HELLO"<<DSD;
endl(cout);
cout<<"After applying own manipulator DST";
endl(cout);
cout<<DST<<"HELLO"<<DST;
return 0;
}
```

OUTPUT

Before Applying manipulator

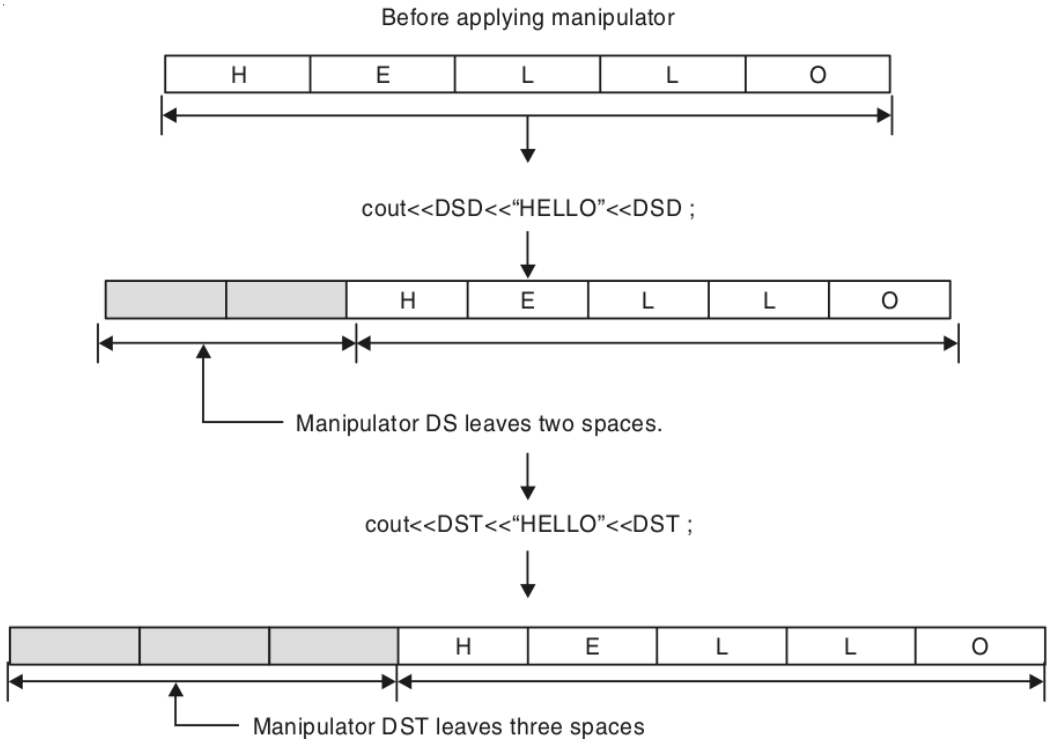
HELLO

After applying own manipulator DSD

HELLO

After applying own manipulator DST

HELLO



```
ostream & DS(ostream & mycout)
{
mycout<<" ";
return mycout;
}
#include <iostream>
using namespace std;

/* CREATION OF MANIPULATOR */
ostream & Rup(ostream & mycout)
{
mycout<<"Rs ";
return mycout;
}
int main( )
{
int money = 8000;
cout<<"==============================";
endl(cout);
cout<<"AMOUNT IS SHOWN HERE";
endl(cout);
cout<<"==============================";
endl(cout);
cout<<Rup<<money<<"/-"<<endl;
cout<<"==============================";
endl(cout);
return 0;
}
```

OUTPUT

```
==============================
AMOUNT IS SHOWN HERE
==============================
Rs 8000/-
==============================
```

Question

Create an input manipulator say skiptoletter that reads and discards all characters which are not letters. When the first letter is found, the manipulator puts it back into the input stream and returns. Write a suitable main function which illustrates the use of this manipulator.