Lecture-8

MANIPULATOR FUNCTIONS

Manipulator functions are special stream functions that change certain characteristics of the input and output. The main advantage of using manipulator functions is that they facilitate the formatting of input and output steams.

To use these functions , header file <iomanip.h> must be included in the program. Following is the list of some of the manipulator functions.

endl (this function has already been studied.)
hex, dec, oct
setbase
set
setfill
setprecision
ends
ws
flush
setiosflags
resetiosflags.
We have already studied endl command, hich means end of line.

Below we give to programs to demonstrate hex, dec and oct functions. These functions convert any number to hexadecimal, decimal and octal numbers.

Program: 8.1

```
//using dec,hex,oct manipulator

#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      int value;
      cout<<"Enter number"<<endl;
      cin>>value;
      cout<<"Desimal base="<<dec<<value<<endl;
      cout<<"Hexasimal base="<<hex<<value<<endl;
      cout<<"Octal base="<<oct<<value<<endl;
}
```

OUTPUT OF THE PROGRAM

```
/*Enter number
10
```

```
Desimal base=10
Hexasimal base=a
Octal base=12
Press any key to continue . . .*/
```

Same program can be ritten with setbase(n). Here n is the number to which base is to be converted.

Prog: 8.2

```
//using dec,hex,oct using setbase ()manipulator

#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      int value;
      cout<<"Enter number"<<endl;
      cin>>value;
      cout<<"Desimal base="<<setbase(10)<<value<<endl;
      cout<<"Hexasimal base="<<setbase(16)<<value<<endl;
      cout<<"Octal base="<<setbase(8)<<value<<endl;
}
```

Output is same as in example 8.1. Manipulator setw() sets the width of the output field. Example 8.3 demonstrates the use of setw().

Program 8.3:

```
// Use set width
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      int a=200,b=300;
      cout<<setw(5)<<a<<setw(5)<<b<<endl;
      cout<<setw(6)<<a<<setw(6)<<b<<endl;
      cout<<setw(7)<<a<<setw(7)<<b<<endl;
      cout<<setw(8)<<a<<setw(8)<<b<<endl;
}
/*
200  300
   200   300
    200    300
     200     300
Press any key to continue . . .

*/
```

Use of setfill('*') : Function setfill fills the blank spaces in the output with a simble given in the bracket of setfill. This is demonstrated in the following example.

Program 8.4

```
// Use set width and setfill manipulator
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      int a=200,b=300;
      cout<<setfill('*');
      cout<<setw(5)<<a<<setw(5)<<b<<endl;
      cout<<setw(6)<<a<<setw(6)<<b<<endl;
      cout<<setw(7)<<a<<setw(7)<<b<<endl;
      cout<<setw(8)<<a<<setw(8)<<b<<endl;
}
/*
**200**300
***200***300
****200****300
*****200*****300
Press any key to continue . . .
*/
```

Setprecision: Below we give an example of setprecision(), with the help of which we can define number of decimal required in an output. .

Program 8.5

```
// Use setprecision manipulator
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      float a=5,b=3;float c;
      c=a/b;
      cout<<setprecision(1)<<c<<endl;
      cout<<setprecision(2)<<c<<endl;
      cout<<setprecision(3)<<c<<endl;
      cout<<setprecision(4)<<c<<endl;
      cout<<setprecision(5)<<c<<endl;
}
/*
2
1.7
1.667
1.6667
1.66667
Press any key to continue . . .

*/
```

Ends : The ends is a manipulator, which is used to attach a null terminating character ('\0') at the end of a string. The ends manipulator takes no argument when ever it is invoked. This causes a null character to output. It encloses the output by the character used in '\ '.

Let us see in program 8.6 how it works.

Program 8.6:

```
// Use of ends manipulator
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
      int number =231;

      cout<<'\"'<<"number="<<number<<ends;
      cout<<'\"'<<endl;

}
/*
"number=231" */
```

It is to be noted that hex, dec, oct, endl and ends manipulators are defined in iostream.h. Other manipulators are defined in iomanip.h.

Setiosflags and resetiosflags : Setiosflags manipulator function is used to control different input and output settings. Following program demonstrates the use of setiosflags.

Program 8.9:

```
// Use of setiosflags manipulator
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
     int value;

     cout<<"Enter a number \n"<<endl;
     cin>>value;
     cout<<setiosflags(ios::showbase);
     cout<<setiosflags(ios::dec);
     cout<<"decimal="<<value<<endl;
    cout<<setiosflags(ios::hex);
     cout<<"Hexadecimal="<<value<<endl;
     cout<<setiosflags(ios::oct);
```

```
     cout<<"Octal="<<value<<endl;

}
/*
Enterr a number
10
decimal=10
hexadecimal=0xa
Octal=012

*/
```

cin and strings

We can use cin to get strings with the extraction operator (>>) as we do with fundamental data type variables:

cin >> mystring;

However, as it has been said, cin extraction stops reading as soon as it finds any blank space character, so in this case we will be able to get just one word for each extraction. This behavior may or may not be what we want; for example if we want to get a sentence from the user, this extraction operation would not be useful.

In order to get entire lines, we can use the function `getline`, which is the more recommendable way to get user input with `cin`:

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr <<
".\n";
  cout << "What is your favorite
drink? ";
  getline (cin, mystr);
  cout << "I like " << mystr << "
too!\n";
  return 0;}
```

```
What's your name? Raj
Kamal
Hello Raj Kamal.
What is your favorite
team? coffee
I like coffee too!
```

Notice how in both calls to getline we used the same string identifier (mystr). What the program does in the second call is simply to replace the previous content by the new one that is introduced.

stringstream

The standard header file <sstream> defines a class called stringstream that allows a string-based object to be treated as a stream. This way we can perform extraction or insertion operations from/to strings, which is especially useful to convert strings to numerical values and vice versa. For example, extraction of an integer from a string, we write the folloing program:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

This declares a string object with a value of "1204", and an int object. Then we use stringstream's constructor to construct an object of this type from the string object. Because we can use stringstream objects as if they were streams, we can extract an integer from it as we would have done on cin by applying the extractor operator (>>) on it followed by a variable of type int.

After this piece of code, the variable myint will contain the numerical value 1204.

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;//why storing price in mystr?
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

```
Enter price: 22.25
Enter quantity: 7
Total price: 155.75
```

In this example, we acquire numeric values from the standard input indirectly. Instead of extracting numeric values directly from the standard input, we get lines from the standard input (cin) into a string object (mystr), and then we extract the integer values from this string into a variable of type int (myint).

Using this method, instead of direct extractions of integer values, we have more control over what happens with the input of numeric values from the user, since we are separating the process of obtaining input from the user (we now simply ask for lines) with the interpretation of that input. Therefore, I recommend you to use this method instead of direct extraction in order to get numerical values from the user in all programs that are intensive in user input.