Standard I/O Streams and Manipulators

Sukhbir Tatla
sukhbir.tatla@senecacollege.ca

Practice Example

```
#include <iostream>
using namespace std;
  void main()
  {
          char *string1 = "C++";
          char *string2 = "Program";
          int m = strlen(string1);
          int n = strlen(string2);
          for (int i = 1; i<n; i++)
          {
                  cout.write(string2, i);
                  cout << endl;
          }
```

Practice Example Cont'd

```
        for (int i = n; i>0; i--)
        {
                cout.write(string2, i);
                cout << endl;
        }
        cout.write(string1, m).write(string2, n);
        cout << endl;
        //crossing the boundary
        cout.write(string1, 10);
        cout << endl;
}
```

Casting

cout.write(string1,m).write(string2,n);

- Is equivalent to the following two statements:

cout.write(string1,m);

cout.write(string2,n);

Formatted Console I/O Operations

- C++ supports a number of features that could be used for formatting the output.
- These features include:
  - ios class functions and flags.
  - Manipulators.
  - User-defined output functions.
- The ios class contains a large number of member functions that would help us to format the output in a number of ways.
- The most important ones among them are listed in Table.

Formatted I/O Operations Table

Function	Task
Width()	To specify the required field size for displaying an output value.
precision()	To specify the number of digits to be displayed after the decimal point of a float value.
fill()	To specify a character that is used to fill the unused portion of a field.
setf()	To specify format flags that can control the form of output display(such as left-justification and right-justification)
unsetf()	To clear the flags specified.

Formatted Console I/O Manipulators

- Manipulators are special functions that can be included in the I/O statements to alter the format parameters of a stream.

- To access manipulators, the file <iomanip> should be included in the program.

Manipulators Table

Manipulators	Equivalent ios function
setw()	width()
setprecision()	precision()
setfill()	fill()
setiosflags()	setf()
resetiosflags()	unsetf()

Defining Field Width: width ()

- We can use the width() function to define the width of a field necessary for the output of an item.

  cout.width (w);

- **Where w is the field width(number of columns).**
- **The field width should be specified for each item separately.**

Defining Field Width: width ()

- For example, the statements

```
cout.width(5);
cout<<543<<12<<"\n";
```

- Will produce the following output:

SETTING PRECISION: PRECISION()

- By default ,the floating numbers are printed with six digits after the decimal point.
- However ,we can specify the number of digits to be displayed after the decimal point while printing the floating-point numbers.
- This can be done by using the precision() member function as follows:
- **cout.precision(d);**
- Where d is the number of digits to the right of the decimal point.
- Default precision is 6 digits.

Setting Precision: precision()

- For example ,the statements

```
cout.precision(3);
cout<<sqrt(2)<<"\n";
cout<<3.14159<<"\n";
cout<<2.50032<<"\n";
```

- Will produce the following output:

  1.141(truncated)

  3.142(rounded to the nearest cent)

  2.5(no trailing zeros)

Setting Precision: precision()

- We can set different values to different precision as follows:

```
cout.precision(3);
cout<<sqrt(2)<<"\n";
cout.precision(5);//reset the precision
cout<<3.14159<<"\n";
```

- We can also combine the field specification with the precision setting. Example:

```
cout.precision(2); cout.width(5); cout<<1.2345;
```

- The output will be:

Filling and Padding :fill()

- We can use the fill() function to fill the unused positions by any desired character.
- It is used in the following form:

  cout.fill(ch);

- Where ch represents the character which is used for filling the unused positions.
- Example:

  cout.fill(‘*’); cout.width(10);
  cout<<5250<<”\n”;

- The output would be:

Formatting Flags, Bit-fields and setf()

- The setf() function can be used as follows:

  cout.setf (arg1,arg2);

- The arg1 is one of the formatting flags defined in the class ios. The formatting flag specifies the format action required for the output.
- Another ios constant, arg2, known as bit field specifies the group to which the formatting flag belongs.

Formatting Flags, Bit-fields and setf()

- Table shows the bit fields, flags and their format actions. There are three bit fields and each has a group of format flags which are mutually exclusive.

Format required	Flag(arg1)	Bit-Field(arg2)
Left-justified output	ios::left	ios::adjustfield
Right-justified output	ios::right	ios::adjustfield
Padding after sign or base Indicater(like +##20)	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::doc	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

EXAMPLES:

- Examples:

  cout.setf(ios::left,ios::adjustfied);

  cout.setf(ios::scientific,ios::floatfield);

- Consider the following segment of code:

  cout.filll(“*”);

  cout.setf(ios::left,ios::adjustfield);

  cout.width(15);

  cout<<”12345”<<”\n”;

- This will produce the following output:

Displaying Trailing Zeros And Plus Sign

- Certain situations, such as a list of prices of items or the salary statement of employees, require trailing zeros to be shown.
- The above output would look better if they are printed as follows:
- 10.75
- 25.00
- 15.50
- The setf() can be used with the flag ios::showpoint as a single argument to achieve this form of output. For example,

cout.setf(ios::showpoint); //display trailing zeros

Displaying Trailing Zeros And Plus Sign

- Similarly, a plus sign can be printed before a positive number using the following statement:

```
cout.setf(ios::showpos);          //show + sign
```

- For example:

```
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout.precision(3);
cout.setf(ios::fixed,ios::floatfield);
cout.setf(ios::internal,ios::adjustfield);
cout.width(10);
cout<<275.5<<"\n";
```

Table lists the flags that do not possess a named bit field.

Flag	Meaning
ios::showbase ios::showpos ios::showpoint ios::uppercase	Use base indicator on output Print + before positive numbers Show trailing decimal point and zeroes Use uppercase letters for hex output
ios::skipus	skip white space on input
ios::unitbuf ios::stdio	Flush all streams after insertion Flush stdout and stderr after insertion

MANAGING OUTPUT WITH MANIPULATORS

- The header file <iomanip> provides a set of functions called manipulators which can be used to manipulate the output formats.

- Two or more manipulators can be used as a chain in one statement as shown below:

```
cout<<manip1<<manip2<<manip3<<item;
cout<<manip1<<item1<<manip2<<item2;
```

The most commonly used manipulators are shown in table.

Manipulator	Meaning	Equivalent
setw(int w) setprecision(int d)	Set the field width to w Set the floating point precision to d.	width() precision()
setfill(int c)	Set the fill character to c	fill()
setiosflags(long f)	Set the format flag f	setf()
resetiosflags(long f)	Clear the flag specified by f	unsetf()
Endif	Insert new line and flush stream	"\n"

EXAMPLES:

- One statement can be used to format output for two or more values.
- For example,the statement

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

- Will print all the three values in one line with the field size of 5,10,and 15 respectively.

Difference Between IOS Functions and manipulators

- There is a major difference in the way the manipulators are implemented as compared to the ios member functions.
- The ios member function return the previous format state which can be used later.
- In case, we need to save the old format states, we must use the ios member function rather than the manipulators.

EXAMPLE:

```
cout.precision(2);          //previous state

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

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

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

Designing Our Own Manipulators

- We can design our own manipulators for certain special purpose.
- The general form for creating a manipulator without any arguments is:

```
ostream & manipulator (ostream & output)
    {
            .....(code)
            .....
            return output
    }
```

- Here the manipulator is the name of the manipulator under creation.

EXAMPLE:

- The following function defines a manipulator called unit that dispalys"inches":

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

- The statement

```
cout<<36<<unit;
```

- will produce the following output

```
36 inches
```

Program Example

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

ostream & currency(ostream & output)
{
        output << "$";
        return output;
}
```

Program Example Cont'd

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

Program Example Cont'd

```
int main()
{
        cout << currency << form << 7864.5;
        cout << endl << endl;
        return 0;
}
```

The Output will be:

$+7864.50**

THANK YOU!

Any Questions Please?