Formatted Input-Output Operations in C++

Format flag	Group	Default	i/o	Effect
left	adjustfield	right	o	Left-align, padding with fill character on right
right				Right-align, padding with fill character on left
internal				padding between sign or base indicator and number [-12.34]
dec	basefield	dec	i,o	integer to decimal form
oct				integer to octal form
hex				integer to hexadecimal form
fixed	floatfield	fixed	o	Floating point in fixed point notation [123.45]
scientific				Floating point in scientific notation [1.2345E2]
skipws		1	i	When unset to 0, white spaces are not ignored
boolalpha		0	i,o	When set to 1, bool values converted to string “true” or “false”
showpos		0	o	When set, show + sign before non-negative integers
showpoint		0	o	Always show decimal point in floating point o/p [123.00]
showbase		0	o	Show base indicator (0 for octal, 0x for hex)
uppercase		0	o	Use uppercase X, E, and hex output letters (ABCDEF)
unitbuf		0	o	Flush o/p stream after each formatting operation

Formatting using ios functions

Function	Default	Effect
width()	0	Minimum field with to be used
precision()	6	Precision for number of digits after decimal point
fill()	Space character	Fill character to be used for padding
setf()	0	Set bit of format flag
unsetf()	0	Reset bit of format flag

width()

- Function width() is used to get/set width variable. Width specifies number of characters or positions to be used during input/output. Even though it can be used during input, it is more meaningful to use it for output purpose.
- Syntax: int ios::width (int num=0)

width()

- For example:
  - int i=1234; str[10]=”Hello”;
  - cout.width(7);
  - cout<<i << str;

- In the above example, value of the first item i will be displayed using a width of 7 positions and the width parameter will be reset to its default value 0. Now, value of str will be displayed using minimum width 0; i.e 5 positions as necessary.

width()

- cin.width(7); cin >> i;
- cin.width(7); cin>> str;

- Let the input be: 1234GujaratHello
- Here, extraction for integer i is independent of width parameter. Characters 1234 are extracted and transformed to binary number to be stored in variable i. Next, 7 characters will be extracted and stored in variable str.
- Thus specified width will be applied first time to item str and reset to 0.

fill()

- Function fill() is used to get/set fill character used as padding character to fill extra positions.
- Syntax: char ios::fill (char padding = ‘ ’)
  - cout.width(7);
  - cout.fill (‘*’);
  - cout << 1234;
  - cout.width(7);
  - cout << “Hello”;
  - Output will be: ***1234**Hello

precision()

- This function is used to specify number of significant digits in case of floating point numbers.

- Syntax: int ios::precision (int num = 6)

- For example:
  - cout.fill (‘*’);
  - cout.precision(7);
  - cout.width(10);
  - cout << 1234.567<< endl;
  - cout.precision(8);
  - cout.width(10);
  - cout << 1234.567<< endl;
  - cout.precision(5);
  - cout.width(10);
  - cout << 1234.567<< endl;
  - cout.precision(3);
  - cout.width(10);
  - cout << 1234.567<< endl;
  - **Output is:**
  - ****1234.567**
  - ****1234.567**
  - ******1234.6**
  - ***1.23e+003**

- Now try the following code.
  - cout.set(ios::fixed);
  - cout.precision(2);
  - cout.width(10);
  - cout << 1234.567 << endl;

- The result is ***1234.57. Here, the precision is 2; shorter than required. But, format flag is set to fixed point notation. Due to this, a number is formatted using fixed point notation instead of scientific notation. Also note that it has considered precision 2 as two significant digits after decimal point in fixed point notation.

setf()

- Format flags (bits of format flag variable) can be set using setf() function in two forms as follows.
- Syntax:
  - fmtflags ios::setf (fmtflags flg)
  - **Flag to set :**
    - left, right, internal
    - dec, oct, hex
    - scientific, fixed

unsetf()

- Format flags can be reset using unsetf() function.
- Syntax:
- fmtflags ios::unsetf (fmtflags flg)

Formatting using manipulators

- Like member functions of ios class, manipulators are also used to perform the task of formatting input/output. Manipulators are special type of non-member functions that can be included in an insertion or extraction chain.
- Manipulators are non-member functions returning a reference to a stream. So they can be used with shift operator

Manipulator

- setw()
- setprecision()
- setfill()
- setiosflags()
- resetiosflags()

- Note : To use manipulators, it requires to include <iomanip> in C++ program.

- cout<<“Numbers left justified in width 10 : ” << endl;
- cout.setf(ios_base::left,ios_base::adjustfield);
- cout.width(10);
- cout<<12;
- cout.width(10);
- cout<<123<< '\n';

- The above C++ statements can be written in a single statement using manipulators as follows:
- cout<< “Numbers left justified in width 10 : ” << endl << left << setw(10) << 12 << setw(10) << 123
  << '\n' ;

User-defined manipulators

- The code structure for user-defined manipulator is as follows:
  - ostream &<manipulator_name> (ostream &<parameter_name>)
  - {
    - // code for formatting …
  - return <parameter_name>;
  - }

- For example, define manipulator km to print KiloMeter as follows:
  – ostream& km (ostream & pout) { pout << " KiloMeter" ; return pout;}
  – Using cout << 200 << km; will output 200 KiloMeter

Using ios functions	Using manipulators
Functions are members of ios class	Manipulators are non-member functions
Require only <iostream> file to be included	Require <iomanip> file to be included additionally
Return the previous status of format parameters like width, precision, fill character and format Flags	Does not return previous status of format parameters, Return a reference to stream
Used as a standalone statement, ex. cout.width(10); cout.fill('*');	Can be cascaded with shift operator, ex. cout << setw(10) << setfill('*');
For some manipulators, there is no equivalent function available, for example, endl	Manipulators available corresponding to all ios functions
No shorthand names to use	Can use shorthand names
Not possible to define user-defined member functions in ios class, one may create another class inheriting from ios and add user-defined functions	User-defines manipulators can be easily created
Function calls always requires use of (), example: int w = width();	Manipulators without argument can be called without (), example: left, hex, endl