C++ I/O Manipulators*

(CS 1002)

Dr. Mudassar Aslam

Cybersecurity Department

National University of Computer & Emerging Sciences,
Islamabad Campus

Acknowledgements: Prof. Tim Colburn, University of Minnesota-Duluth

Manipulators

A **manipulator** is a **simple function** that can be included in an **insertion** or **extraction chain**

- **C++ manipulators**
  - **must include iomanip** to use

Output Manipulators (no args)

Manipulators included like arguments:

- **endl** - **outputs** a **new line character**, **flushes output**
- **dec** - **sets int** output to **decimal**
- **hex** - **sets int** output to **hexadecimal**
- **oct** - **sets int** output to **octal**

Example:

```
#include <iostream>
#include <iomanip>
int x = 42;
cout << oct << x << endl; // Outputs 52\n
cout << hex << x << endl; // Outputs 2a\n
cout << dec << x << endl; // Outputs 42\n
```

Output Manipulators (1 arg)

Manipulators with 1 argument

setw(*int*) - **sets** the **width** to ***int*** value

setfill(*char*) - **sets fill char** to ***char*** **value**

left – Left Justified text

setprecision(*int*) - **sets precision** to ***int*** **value**

setbase(*int*) - **sets int** output to **hex** if *int* is 16, oct if *int* is 8, dec if *int* is 0 or 10

```
cout << setw(7) << setprecision(2) << setfill('_') << 34.267 <<
   endl;
  // outputs __34.27
```

Floating Point Format

- Can use flags **scientific** and **fixed** to **force floating point output** in **scientific** or **fixed format**

- Effect of precision depends on format
  - *scientific (total significant digits)*
  - *fixed (how many digits after decimal point)*

```
float a = 4.0;
int n=7;
cout<<"\nDefault:"<<setprecision(n)<<a<<endl;
cout<<"\nFixed:"<<setprecision(n)<<fixed<<a<<endl;
cout<<"\nScientific:"<<setprecision(n)<<scientific<<a<<endl;
```

Integer Input

If **none of the flags** (***hex***, ***dec***, ***oct*** set) then we can indicate how an **int** is **formatted** with **value typed**:

42 - decimal 42

052 - octal 52

0x2a - hexadecimal 2a

- **If** any of these flags set, all **input** will be **treated as being of only that type**
  - note, in explicit decimal format, **052** read as **52**, **0x2a** read as **0**

Character Input

- The **extraction operator** when applied to a **character** **ignores** **whitespace** (***space***, ***tab***, ***enter*** **key**)

- To **read any character** (i.e., ***space***, ***tab***, ***enter*** **key**) use the **get(*char*) function** with ***cin*** object, can also provide **no argument** (***works like getchar***)

```
char ch;
cin >> ch;     // Reads next non-whitespace char
cin.get(ch);   // Reads next character (any:
               //  space, newlines, tab, etc.)
```

Formatting cout output using flags

Setting Format Flags

- The **object** ***cout*** has **flags** that ***determine how objects are printed***

- **To set a flag(s)** we use the ***setf*** **function** (*associated with objects* **cout** and **cin**)

```
cout.setf(flags)
```

Setting Format Flags (cont)

- **To set flags**, we **may need** to **unset other flags** first:

```
cout.unsetf(flags)
```

A short-hand method:

```
cout.setf(OnFlags,OffFlags)
```

- ***turns off***, *the flags* ***OffFlags***
- ***turns on,*** *the flags* ***OnFlags***

Integer Base and Format Flags

Printing integer using different base formats:

ios::dec - show *ints* as *decimal* (the default)

ios::oct - show ints as octal

ios::hex - show ints as hexadecimal

- *Should only have one on at a time:*

```
cout.unsetf(ios::dec);
cout.unsetf(ios::oct);
cout.unsetf(ios::hex);
cout.setf(ios::oct);
```

Integer Base and Format Flags (cont)

One can combine flags using | operator

```
cout.unsetf(ios::dec | ios::oct | ios::hex);
cout.setf(ios::oct);
```

or

```
cout.setf(ios::oct,ios::dec | ios::oct | ios::hex);
```

C++ also includes a shorthand for the second (combination) flag:

ios::basefield:

```
cout.setf(ios::oct, ios::basefield);
```

Turns all of the base flags off and the octal flag on

Integer Base Example

```
int x = 42;

cout.setf(ios::oct,ios::basefield);
cout << x << '\n'; // Outputs 52\n

cout.setf(ios::hex,ios::basefield);
cout << x << '\n'; // Outputs 2a\n

cout.setf(ios::dec,ios::basefield);
cout << x << '\n'; // Outputs 42\n
```

Showing the Plus Sign

The **flag ios::showpos** can be **set** (*its default is off*) to **print** a **+ sign** with positive values (*int* or *float*) :

```
int x = 42;
int y = 3.1415;

cout.setf(ios::showpos);
cout << x << '\n'; // Outputs +42\n
cout << y << ‘\n’; // Outputs +3.1415\n
```

Showing Upper Case Hex Ints

The **flag** **ios::uppercase** (*default off*) - **letters** making up **hexadecimal numbers** should be **shown as upper case**:

```
int x = 42;

cout.setf(ios::uppercase);
cout.setf(ios::hex,ios::basefiSeld);
cout << x << '\n'; // Outputs 2A\n
```

Setting the Width

- You can use the **width(int)** function to **set the width** for **printing** a **value** (*but it only works for the next insertion command)*:

```
int x = 42;

cout.width(5);
cout << x << '\n';  // Outputs    42
cout << x << '\n';  // Outputs 42
```

Setting the Fill Character

Use the **fill(*char*) function** to **set** the **fill character** (*character remains as the fill character until set again*).

```
int x = 42;

cout.width(5);
cout.fill('*');
cout << x << '\n';  // Outputs ***42
```

Justification

Set **justification** using **flags ios::left**, **ios::right**, and **ios::internal** (*after sign or base*) - only one

Use **ios::adjustfield** to turn all three flags off

```
int x = 42;
cout.setf(ios::showpos);
cout.fill('*');
cout.setf(ios::right,ios::adjustfield);
cout.width(6);
cout << x << '\n';  // Outputs ***+42
cout.setf(ios::left,ios::adjustfield);
cout.width(6);
cout << x << '\n';  // Outputs +42***
cout.setf(ios::internal,ios::adjustfield);
cout.width(6);
cout << x << '\n';  // Outputs +***42
```

Decimal Points in Floats

Set **flag ios::showpoint** to make **decimal point appears** in **output:**

```
float y = 3.0;

cout << y << '\n';  // Outputs 3
cout.setf(ios::showpoint);
cout << y << '\n';  // Outputs 3.00000
```

Format of Float

Floating point values are printed out in fixed or scientific notation based on how they are stored/initialized:

```
cout << 2.3;  // Outputs 2.3
cout << 5.67e8; // Outputs 5.67e+08
cout << 0.0;  // Outputs 0
```

Significant Digits in Float

Use function **precision(*int*)** to **set** the **number of significant digits** printed (may convert from fixed to scientific to print):

```
float y = 23.1415;
cout.precision(1);
cout << y << '\n'; // Outputs 2e+01
cout.precision(2);
cout << y << '\n'; // Outputs 23
cout.precision(3);
cout << y << '\n'; // Outputs 23.1
```

Floating Point Format

- Can use flags **ios::scientific** and **ios::fixed** to **force floating point output** in **scientific** or **fixed format**

- Only one flag at a time, **ios::floatfield** to turn off

```
cout.setf(ios::scientific,ios::floatfield);
cout << 123.45 << '\n'; // Outputs 1.2345e+02

cout.setf(ios::fixed,ios::floatfield);
cout << 5.67E1 << '\n'; // Outputs 56.7
```

- Effect of precision depends on format
  - *scientific (total significant digits)*
  - *fixed (how many digits after decimal point)*

Displaying bools

- To print 0 (**false**) or 1 (**true**) for **bool type**

- To print out words (false, true) use flag **ios::boolalpha**

```
bool b = true;
cout.setf(ios::boolalpha);
cout << b << '\n';    // Outputs true
cout << (!b) << '\n'; // Outputs false
```