

# 06: INPUT AND OUTPUT

Programming Technique I  
(SECJ1013)

# Formatting Output

# Introduction to Output Formatting

✿ Can control how output displays for numeric and string data:

- ◆ size
- ◆ position
- ◆ number of digits

✿ Done through the use of manipulators, special variables or objects that are placed on the output stream.

✿ Most of the standard manipulators are found in **<iostream>**, some requires **<iomanip>** header file.

# Stream Manipulators

Stream Manipulator	Description
<b>setw(<i>n</i>)</b>	Establishes a print field on <i>n</i> spaces.
<b>fixed</b>	Displays floating-point numbers in fixed point notation.
<b>showpoint</b>	Causes a decimal point and trailing zeros to be displayed, even there is no fractional part.
<b>setprecision(<i>n</i>)</b>	Sets the precision of floating-point numbers.
<b>left</b>	Causes subsequent output to be left justified.
<b>right</b>	Causes subsequent output to be right justified.

# Formatting Output: setw()

✿ Used to output the value of an expression in a specific number of columns

✿ **setw(x)** - outputs the value of the next expression in x columns

✿ The output is right-justified

- ◆ Example: if you specify the number of columns to be 8 and the output requires only 4 columns, then the first four columns are left blank

✿ If the number of columns specified is less than the number of columns required by the output, the output automatically expands to the required number of columns

# Example 1: setw()

## Program 3-16

```
1 // This program displays three rows of numbers.
2 #include <iostream>
3 #include <iomanip>      // Required for setw
4 using namespace std;
5
6 int main()
7 {
8     int num1 = 2897, num2 = 5,    num3 = 837,
9         num4 = 34,   num5 = 7,    num6 = 1623,
10        num7 = 390,  num8 = 3456, num9 = 12;
11
12    // Display the first row of numbers
13    cout << setw(6) << num1 << setw(6)
14         << num2 << setw(6) << num3 << endl;
15
16    // Display the second row of numbers
17    cout << setw(6) << num4 << setw(6)
18         << num5 << setw(6) << num6 << endl;
19
```

## Program 3-16 (continued)

```
20    // Display the third row of numbers
21    cout << setw(6) << num7 << setw(6)
22         << num8 << setw(6) << num9 << endl;
23    return 0;
24 }
```

## Program Output

```
2897      5    837
   34      7  1623
  390  3456    12
```

## Example 2: setw()

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    cout << "*" << -17 << "*" << endl;
    cout << "*" << setw(6) << -17 << "*" << endl << endl;

    cout << "*" << "Hi there!" << "*" << endl;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;
    cout << "*" << setw(3) << "Hi there!" << "*" << endl;

    return 0;
}
```

# Example 1: left and right

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

int main()
{
    int x = 15;
    int y = 7634;
    cout << left;
    cout << setw(5) << x << setw(7) << y << setw(8) << "Warm"
         << endl;

    cout << right;
    cout << setw(5) << x << setw(7) << y << setw(8) << "Warm"
         << endl;
    return 0;
}
```



## Example 2: left and right

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    cout << "*" << -17 << "*" << endl;
    cout << "*" << setw(6) << -17 << "*" << endl;
    cout << left;
    cout << "*" << setw(6) << -17 << "*" << endl << endl;

    cout << "*" << "Hi there!" << "*" << endl;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;
    cout << right;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;

    return 0;
}
```

# Example 1: fixed

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

int main()
{
    double x = 15.674;
    double y = 235.73;
    double z = 9525.9874;

    cout << fixed;
    cout << x << endl << y << endl << z << endl;

    return 0;
}
```

## Example 2: fixed

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

int main()
{
    float small = 3.1415926535897932384626;
    float large = 6.0234567e17;
    float whole = 2.0000000000;
    cout << "Some values in general format" << endl;
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    cout << fixed;
    cout << "The same values in fixed format" << endl;
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;
    return 0;
}
```

# Example 1: showpoint

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

int main()
{
    double x = 15.674;
    double y = 235.73;
    double z = 9525.9874;

    cout << showpoint;
    cout << x << endl << y << endl << z << endl;
    return 0;
}
```

## Example 2: showpoint

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

int main()
{
    float lots = 3.1415926535, little1 = 2.25;
    float little2 = 1.5, whole = 4.00000;

    cout << "Some values with noshowpoint (the default)" << endl;
    cout << "lots:      " << lots << endl;
    cout << "little1: " << little1 << endl;
    cout << "little2: " << little2 << endl;
    cout << "whole:    " << whole << endl << endl;

    cout << "The same values with showpoint" << endl;
    cout << showpoint;
    cout << "lots:      " << lots << endl;
    cout << "little1: " << little1 << endl;
    cout << "little2: " << little2 << endl;
    cout << "whole:    " << whole << endl;
    return 0;
}
```

# Example: fixed and showpoint

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

int main()
{
    double x = 15.674;
    double y = 235.73;
    double z = 9525.9874;

    cout << fixed << showpoint;
    cout << x << endl << y << endl << z << endl;

    return 0;
}
```

# setprecision() Manipulator

✿ To control the number of significant digits (or precision) of the output, i.e., the total number of digits before and after the decimal point.

✿ However, when used with fixed, it specifies the number of floating-points (i.e., the number of digits after the decimal point).

✿ Without fixed, the setprecision() is set to a lower value, it will print floating-point value using scientific notation.

✿ `setprecision(n)` – *n* is the number of significant digits or the number of floating-point (if used with fixed).

# Example 1: setprecision()

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

int main()
{
    double x = 15.674;
    double y = 235.73;
    double z = 9525.9874;

    cout << setprecision(2);
    cout << x << endl << y << endl << z << endl;
    return 0;
}
```



## Example 2: setprecision()

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

int main()
{
    double x = 156.74, y = 235.765, z = 9525.9874;

    cout << setprecision(5) << x << endl;
    cout << setprecision(3) << x << endl;
    cout << setprecision(2) << x << endl;
    cout << setprecision(1) << x << endl;

    cout << fixed << setprecision(2);
    cout << x << endl << y << endl << z << endl;

    return 0;
}
```

# In-Class Exercise



What is the output of the following program:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main( )
{   double val = 10.345;
    cout << setprecision(5) << val << endl;           //(a)
    cout << setprecision(4) << val << endl;           //(b)
    cout << setprecision(3) << val << endl;           //(c)
    cout << setprecision(2) << val << endl;           //(d)
    cout << setprecision(1) << val << endl;           //(e)
    cout << "Apa Khabar \n Semua /n" << endl;        //(f)
    cout << static_cast<int>(val)/2 << endl;           //(g)
    cout << setw(6) << val*5 << endl;                 //(h)
    cout << showpoint << fixed << setw(8) << val << endl; //(i)
    return 0;
}
```

# Formatted Input

# Introduction to Input Formatting

❁ Can format field width for use with cin.

❁ Useful when reading string data to be stored in a character array:

```
const int SIZE = 10;  
char firstName[SIZE];  
cout << "Enter your name: ";  
cin >> setw(SIZE) >> firstName;
```

❁ cin reads one less character than specified with the setw() manipulator.

# Example: Input Formatting

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

int main()
{
    const int SIZE = 10;
    char firstName[SIZE];

    cout << "Enter your name: ";
    cin >> setw(SIZE) >> firstName;
    cout << firstName << endl;

    return 0;
}
```

# Example: Problem using cin

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

int main()
{
    string name;
    cout << "Enter your name: ";
    cin >> name;
    cout << name << endl;
    return 0;
}
```

# Input Formatting: `getline()`

🌸 To read an entire line of input, use `getline()`.

🌸 When reading string data to be stored in a character array, use `getline()` with two arguments:

- ◆ Name of array to store string
- ◆ Size of the array

🌸 When reading string data to be stored as an object of string, use `getline()` with two arguments:

- ◆ istream object, i.e cin
- ◆ string object

# Example 1: getline()

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

int main()
{
    const int SIZE = 20;
    char firstName[SIZE];

    cout << "Enter your name: ";
    cin.getline (firstName, SIZE);
    cout << firstName << endl;

    return 0;
}
```



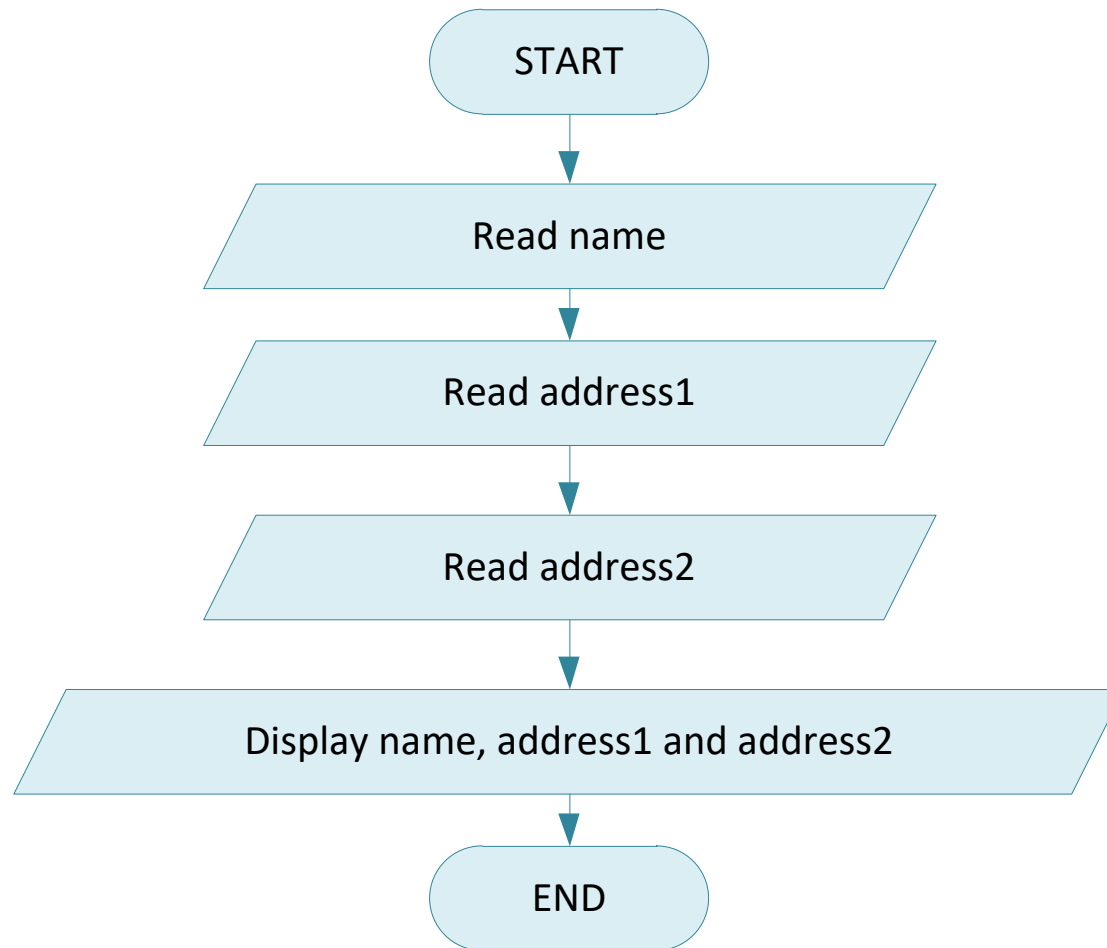
## Example 2: getline()

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

int main()
{
    string name;
    cout << "Enter your name: ";
    getline (cin, name);
    cout << name << endl;
    return 0;
}
```

# In-Class Exercise

✿ Write C++ program to solve the flow chart:



# Input Formatting: get()

❁ To read a single character, use cin.

```
char ch;  
cout << "Strike any key to continue";  
cin >> ch;
```

Problem: will skip over blanks, tabs, <ENTER>

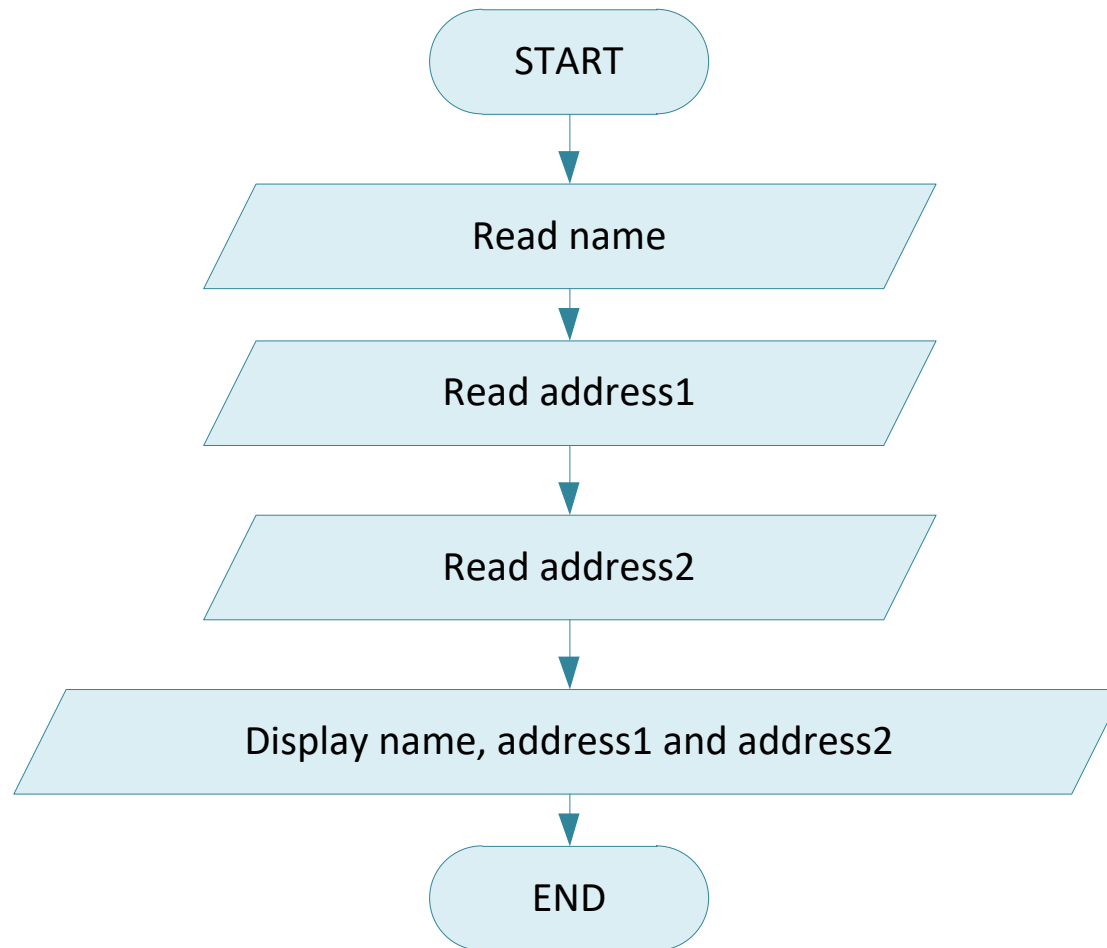
❁ Solution to read a single character, use get().

```
char ch;  
cout << "Strike any key to continue";  
cin.get(ch) ;
```

Advantage: Will read the next character entered, even whitespace.

# In-Class Exercise

✿ Write C++ program to solve the flow chart:



# Input Formatting: ignore()

✿ Mixing cin >> and cin.get() in the same program can cause input errors that are hard to detect.

✿ To skip over unneeded characters that are still in the keyboard buffer, use cin.ignore():

```
//skip next char
```

```
cin.ignore() ;
```

```
//skip the next 10 char. @ until a '\n'
```

```
cin.ignore(10, '\n') ;
```

# In-Class Exercise

🌸 What will be displayed if the user enters the following input:

202

L

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

int main()
{
    int id;
    char code;
    cout << "Enter an integer id: ";
    cin >> id;
    cout << "Enter a code: ";
    cin.get(code);
    cout << "Output\n" << id << "\t" << code;
    return 0;
}
```

# Introduction to Files

# File Input and Output

- ✿ Can use files instead of keyboard and monitor screen for program input and output.
- ✿ File: a set of data stored on a computer, often on a disk drive.
  - ◆ Allows data to be retained between program runs.
- ✿ Programs can read from and/ or write to files.
- ✿ Used in many applications: word processing, databases, spreadsheets, compilers.
- ✿ Steps: (1) Open the file (2) Use the file (read from, write to, or both) (3) Close the file.



# File Operations

✿ Requires **fstream** header file:

- ◆ use ifstream data type for input files.
- ◆ use ofstream data type for output files.
- ◆ use fstream data type for both input, output files.

✿ **ifstream:**

- ◆ Open for input only and file cannot be written to.
- ◆ Open fails if file does not exist.

✿ **ofstream:**

- ◆ Open for output only and file cannot be read from.
- ◆ File created if no file exists.
- ◆ File contents erased if file exists.

# File Operations (cont.)

✿ **fstream** object can be used for either input or output.

✿ **fstream**: must specify mode on the open statement. Sample modes:

- ◆ **ios::in** for input mode.
- ◆ **ios::out** for output mode.
- ◆ **ios::binary** for binary mode.
- ◆ **ios::app** for append mode. All output operations are performed at the end of the file, appending the content to the current content of the file.

# Opening Files

✿ Create a link between file name (outside the program) and file stream object (inside the program).

✿ Filename may include drive and/or path info.

✿ **ifstream** and **ofstream** - use the **open()** member function:

```
infile.open("inventory.dat");  
outfile.open("report.txt");
```

✿ **fstream** - use the **open()** member function and **mode(s)**:

```
infile.open("inventory.dat", ios::in);  
outfile.open("report.txt", ios::out);
```

# Opening Files (cont.)

✿ **fstream** - can be combined on open call:

```
dFile.open("class.txt", ios::in | ios::out);
```

✿ Can open file at declaration:

```
ifstream gradeList("grades.txt");  
fstream infile("inventory.dat", ios::in);  
fstream file("class.txt", ios::in | ios::out);
```

✿ Output file will be created if necessary; existing file will be erased first.

✿ Input file must exist for open to work.

# Opening Files (cont.)

🌸 File stream object set to **0(false)**, if **open failed**. Example:

```
if (!input)
{ cout << "ERROR: Cannot open file\n";
  exit(1); }
```

🌸 Can use **fail()** member function to detect file open error:

```
if (input.fail())
{ cout << "ERROR: Cannot open file\n";
  exit(1); }
```

🌸 Can use **is\_open()** member function to check if a file is open:

```
if (!input.is_open())
{ cout << "ERROR: Cannot open file\n";
  exit(1); }
```

# Using Files

✿ Can use **output file object** and **<<** to send data to a file:

```
outfile << "Inventory report";
```

✿ Can use **input file object** and **>>** to copy data from file to variables:

```
infile >> partNum;
```

```
infile >> qtyInStock >> qtyOnOrder;
```

✿ Can use **eof()** member function **to test for end of input file.**

# Closing Files

✿ Use the **close()** member function:

```
infile.close();  
outfile.close();
```

✿ Don't wait for operating system to close files at program end:

- ◆ may be limit on number of open files.
- ◆ may be buffered output data waiting to send to file.

# Example 1: File Operations

```
#include <iostream> //copy 10 numbers between files
#include <fstream>
using namespace std;

int main()
{
    fstream infile("input.txt", ios::in); // open the files
    fstream outfile("output.txt", ios::out);
    int num;

    for (int i = 1; i <= 10; i++) {
        infile >> num;      // use the files
        outfile << num; }

    infile.close();        // close the files
    outfile.close();

}
```



# Example 2: File Operations

## Program 3-28

```
1 // This program writes data to a file.
2 #include <iostream>
3 #include <fstream>
4 using namespace std;
5
6 int main()
7 {
8     ofstream outputFile;
9     outputFile.open("demofile.txt");
10
11     cout << "Now writing information to the file.\n";
12
13     // Write 4 great names to the file
14     outputFile << "Bach\n";
15     outputFile << "Beethoven\n";
16     outputFile << "Mozart\n";
17     outputFile << "Schubert\n";
18
```

## Program 3-28 (continued)

```
19     // Close the file
20     outputFile.close();
21     cout << "Done.\n";
22     return 0;
23 }
```

### Program Screen Output

Now writing data to the file.  
Done.

### Output to File demofile.txt

Bach  
Beethoven  
Mozart  
Schubert

# Example 3: File Operations

## Program 3-29

```

1  // This program reads information from a file.
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main()
7  {
8      ifstream inFile;
9      const int SIZE = 81;
10     char name[SIZE];
11
12     inFile.open("demofile.txt");
13     cout << "Reading information from the file.\n\n";
14
15     inFile >> name;           // Read name 1 from the file
16     cout << name << endl;     // Display name 1
17
18     inFile >> name;           // Read name 2 from the file
19     cout << name << endl;     // Display name 2
20
21     inFile >> name;           // Read name 3 from the file
22     cout << name << endl;     // Display name 3
23

```

## Program 3-29 (continued)

```

24     inFile >> name;           // Read name 4 from the file
25     cout << name << endl;     // Display name 4
26
27     inFile.close();           // Close the file
28     cout << "\nDone.\n";
29     return 0;
30 }

```

## Program Screen Output

Reading data from the file.

Bach  
 Beethoven  
 Mozart  
 Schubert

Done.

# Example 4: File Operations

```
#include <fstream>
using namespace std;
int main()
{
    ifstream input("inputfile.txt");
    char str[80];

    if (!input)
    {
        cout << "While opening a file an error is encountered" << endl;
        return 0;
    }
    else
        cout << "File is successfully opened" << endl;
    while(!input.eof())
    {
        input.getline(str, 80);
        cout << str << endl;
    }
    input.close();
    return 0;
}
```

# Example 5: File Operations

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
    int num;
    ifstream inp("input.txt"); // open the input file
    ofstream out("output.txt"); // open the output file
    if (!inp.is_open()) // check for successful opening
    {
        cout << "Input file could not be opened! Terminating!\n";
        return 0;
    }
    while (inp >> num)
        out << num * 2 << endl;

    inp.close();
    out.close();
    cout << "Done!" << endl;
    return 0;
}
```