06: INPUT AND OUTPUT

Programming Technique I

(SCSJ1013)

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.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Stream Manipulators

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

Formatting Output: setw()

- Used to ouput 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

```cpp
// This program displays three rows of numbers.
#include <iostream>
#include <iomanip>      // Required for setw
using namespace std;

int main()
{
   int num1 = 2897, num2 = 5,    num3 = 837,
       num4 = 34,   num5 = 7,    num6 = 1623,
       num7 = 390,  num8 = 3456, num9 = 12;

   // Display the first row of numbers
   cout << setw(6) << num1 << setw(6)
        << num2 << setw(6) << num3 << endl;

   // Display the second row of numbers
   cout << setw(6) << num4 << setw(6)
        << num5 << setw(6) << num6 << endl;

```

Program 3-16 *(continued)*

```cpp
   // Display the third row of numbers
   cout << setw(6) << num7 << setw(6)
        << num8 << setw(6) << num9 << endl;
   return 0;
}
```

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;
}
```

Output:

```
*-17*
*   -17*

*Hi there!*
*           Hi there!*
*Hi there!*
```

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;
}
```

Output:

```
15   7634   Warm
   15   7634    Warm
```

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;
}
```

Output:

```
*-17*
*   -17*
*-17   *

*Hi there!*
*Hi there!           *
*           Hi there!*
```

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;
}
```

Output:

```
15.674000
235.730000
9525.987400
```

Example 2: fixed

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

int main()
{
  float small = 3.141592653589
  float large = 6.0234567e17;
  float whole = 2.000000000;
  cout << "Some values in gener
  cout << "small:  " << small <
  cout << "large:  " << large <
  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;
}
```

Output:

```
Some values in general format
small:  3.14159
large:  6.02346e+017
whole:  2

The same values in fixed format
small:  3.141593
large:  602345661202956290.000000
whole:  2.000000
```

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;
}
```

Output:

```
15.6740
235.730
9525.99
```

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Example 2: showpoint

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

int main()
{
  float lots = 3.1415926535, little1 = 2.
  float little2 = 1.5, whole = 4.00000;

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

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

Output:

```
Some values with
noshowpoint (the default)
lots:    3.14159
little1: 2.25
little2: 1.5
whole:   4

The same values with
showpoint
lots:    3.14159
little1: 2.25000
little2: 1.50000
whole:   4.00000
```

Example: fixed and showpoint

```cpp
#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;
}
```

Output:

```
15.674000
235.730000
9525.987400
```

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;
}
```

Output:

```
16
2.4e+002
9.5e+003
```

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;
}
```

Output:

```
156.74
157
1.6e+002
2e+002
156.74
235.76
9525.99
```

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;
   cout << setprecision(4) << val << endl;
   cout << setprecision(3) << val << endl;
   cout << setprecision(2) << val << endl;
   cout << setprecision(1) << val << endl;
   cout << "Apa Khabar \n Semua /n" << endl;
   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;
}
```

Output:

```
10.345
10.35
10.3
10
1e+001
Apa Khabar
 Semua /n
5
5e+001
    10.3
```

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;
}
```

Output:

```
Enter your name: LeeChongWei
LeeChongW
```

Example: Problem using cin

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

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

Output 1:

```
Enter your name: LeeChongWei
LeeChongWei
```

Output 2:

```
Enter your name: Lee Chong Wei
Lee
```

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;
}
```

Output:

```
Enter your name: Lee Chong Wei
Lee Chong Wei
```

Example 2: getline()

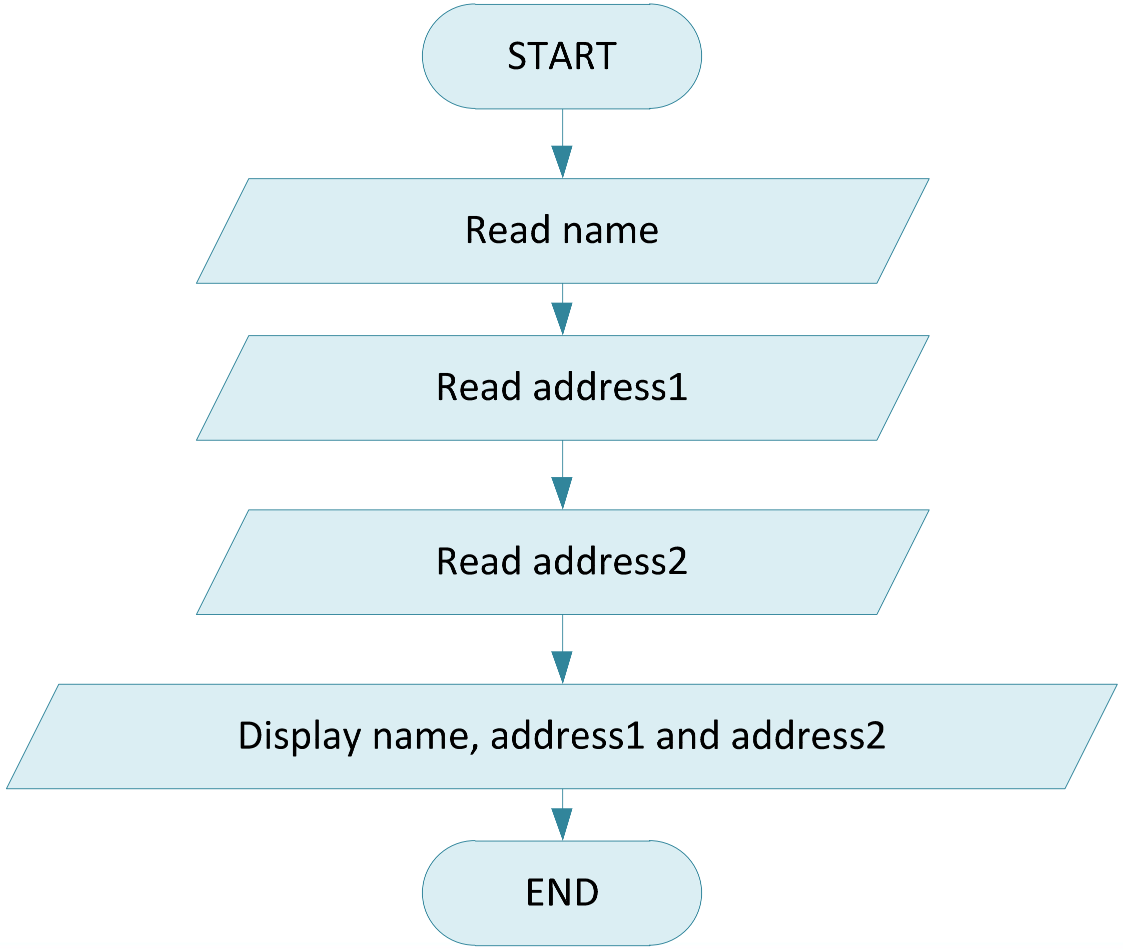
```
#include <iostream>
using namespace std;

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

Output:

```
Enter your name: Lee Chong Wei
Lee Chong Wei
```

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()**.

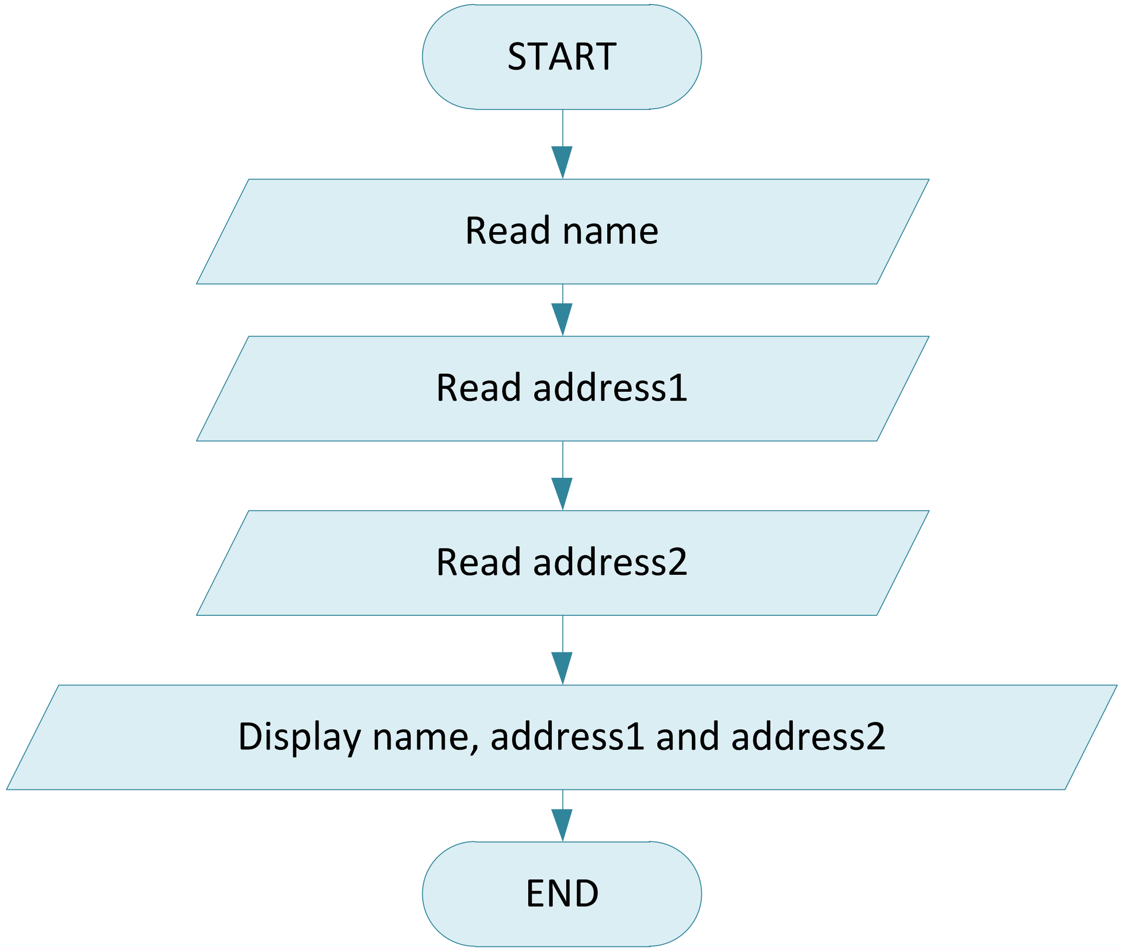
```
char ch;
cout << "Strike any key to continue";
cin.get(ch);
```

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

UTM UNIVERSITI TEKNOLOGI MALAYSIA

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;
}
```

Output:

```
Enter an integer id: 202
Enter a code: Output
202
```

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

```cpp
// This program writes data to a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ofstream outputFile;
   outputFile.open("demofile.txt");

   cout << "Now writing information to the file.\n";

   // Write 4 great names to the file
   outputFile << "Bach\n";
   outputFile << "Beethoven\n";
   outputFile << "Mozart\n";
   outputFile << "Schubert\n";

```

Program 3-28 *(continued)*

```cpp
   // Close the file
   outputFile.close();
   cout << "Done.\n";
   return 0;
}
```

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

```
// This program reads information from a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ifstream inFile;
   const int SIZE = 81;
   char name[SIZE];

   inFile.open("demofile.txt");
   cout << "Reading information from the file.\n\n";

   inFile >> name;          // Read name 1 from the file
   cout << name << endl;    // Display name 1

   inFile >> name;          // Read name 2 from the file
   cout << name << endl;    // Display name 2

   inFile >> name;          // Read name 3 from the file
   cout << name << endl;    // Display name 3

```

Program 3-29 *(continued)*

```
   inFile >> name;          // Read name 4 from the file
   cout << name << endl;    // Display name 4

   inFile.close();          // Close the file
   cout << "\nDone.\n";
   return 0;
}
```

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;
}
```