

Chapter 6

Basic Input & Output

Basic Output in C++

Output Stream - A sequence of characters from the computer to an output device. To output information in C++ we will use a variable called *cout* (pronounced "see-out"). The identifier *cout* stands for common output.

cout is a predefined variable in C++ that indicates you are going to output a stream of characters to an output device. *cout* uses an operator called the *insertion operator* (<<). You will often see << referred to as "put to". The insertion operator requires two operands. For our purposes the operand on the left will be the *cout* variable and the operand on the right will be an expression. Examples:

The following expression is a literal constant of type c-string.

```
cout << "Hello World!";
```

The following expression is a simple arithmetic expression.

```
cout << (num1 + num2) /2;
```

The following expression contains a literal constant of type c-string followed by a variable. Note the insertion operator must be used to separate the various items in the output stream..

```
cout << "The average is " << averageAge;
```

A **general form** specifies the proper syntax for a given statement. The general form for the output statement is shown below.

GENERAL FORM for cout:

```
cout << ExprOrString << ExprOrString ...;
```

Given:

```
const char SCHOOL[20] = "Saddleback";  
int num1, num2;
```

Examine the following output statements and what they produce. Assume num1 contains 3 and num2 contains 7.

<u>Statement</u>	<u>Output</u>
<code>cout << num1;</code>	3
<code>cout << num2;</code>	7
<code>cout << num1 + num2;</code>	10
<code>cout << "Code =" << num1;</code>	Code =3
<code>cout << num1 << "+" << num2;</code>	3+7
<code>cout << num1 << " + " << num2;</code>	3 + 7
<code>cout << "Number is" << num2;</code>	Number is7
<code>cout << "Number is" << " " << num2;</code>	Number is 7
<code>cout << "My school is " << SCHOOL;</code>	My school is Saddleback
<code>cout << "My school is" << endl << SCHOOL;</code>	My school is Saddleback

NOTE: `endl` (end line) causes the cursor to move to the beginning of the next line. This is one way to get blank lines in your output.

```
cout << num1 << endl << endl << num2;
```

3

7

Why did we get only one blank line?

Escape Sequences

Another way to cause *cout* to produce a new line is by inserting an *escape sequence* into a string. When you are dealing with strings this is an easier method. An escape sequence begins with a backslash (\) and is followed by a *control character*. The "\ " and the control character are not printed. Control characters allow you to specify the appearance of your output.

There are many escape sequences – following are a list of a few that we will use in this class.

\n	Newline	Moves the cursor to the next line
\t	Horizontal tab	Moves the cursor to the next tab stop (be careful)
\a	Alarm	Causes the computer to beep
\\	Backslash	Causes a backslash to be printed
\'	Single quote	Causes a single quotation mark to print
\"	Double quote	Causes a double quotation mark to print

Examples:

```
cout << "Stop lights have three colors\n";
cout << "Red\n";
cout << "Yellow\n";
cout << "Green\n";
```

```
cout << "Stop lights have three colors\n";
cout << "Red\nYellow\nGreen\n";
```

```
cout << "Stop lights have three colors";
cout << "\nRed\nYellow\nGreen\n";
```

Compare `\n` with `endl`

```
cout << "Stop lights have three colors" << endl;  
cout << "Red" << endl;  
cout << "Yellow" << endl;  
cout << "Green" << endl;
```

Rewrite the above output using a combination of `\n` and `endl`

Write the `cout` statements necessary to produce the following output.

1. Hello World
2. Hello
World
3. Hello
World
4. It's a beautiful day.
5. It's a "wacky" world!

Output Exercises

1.) Write a complete C++ program to output the three samples EXACTLY as they are shown below:

The Marine Corps is looking for a few good men.

The Marine Corps
is looking for
a few good men.

The Marine Corps

is looking
for a few

good men.

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

```
// This program demonstrates output formatting
```

```
void main(void)
```

```
{
```

```
    const char PROGRAMMER[20] = "Your Name";
```

```
    cout << "This program was written by " << PROGRAMMER << endl << endl;
```

```
    // First output sample
```

```
    cout << "The Marine Corps is looking for a few good men.\n\n";
```

```
    // Write the code below for the last two outputs shown above.
```

```
}
```

2) Show the output from the following cout statements using the named constants provided.

```
const char OLDGOLFER[14] = "Jack Nicklaus";  
const char YOUNG_GOLFER[14] = "Tiger Woods";
```

```
cout << "\nMany years ago " << OLDGOLFER << " took the golf world\nby storm.";  
cout << "\n\n\nNow, many years later, " << endl << YOUNG_GOLFER << " is doing"  
    << " the same thing.";
```

Note: A cout statement that is too long to fit on a single line in the editor may be broken into several lines. Just make sure that you do not break the c-string in the middle of open quotation marks.

line 1: _____
line 2: _____
line 3: _____
line 4: _____
line 5: _____
line 6: _____
line 7: _____
line 8: _____
line 9: _____
line 10: _____
line 11: _____
line 12: _____

Using Manipulators to Format Output

We have been using the `endl` manipulator to insert a new line in a `cout` statement. We are going to examine the manipulators `setw()`, `setprecision()`, `fixed`, and `showpoint`.

setw(n)

`setw(n)` is used to specify a field for output display where `n` is an integer expression representing the field width. The output is right justified in a field width of `n`. This manipulator may be used with `int`, `float`, and `cstring` data types. It is not valid with data type `char`.

```
int val;  
val = 25;  
cout << "The value is " << setw(5) << val;
```

The value is ---25

The dashes (-) are used to show a blank space in this example and will NOT appear on the actual output.

NOTE: `setw(n)` applies ONLY to the next item in the stream. It will revert to a default of 1 after each use.

Formatting floating point values:

Often times we want to output floating point values in a particular format that is consistent throughout the program. Following are three manipulators to use in this process. To use the following manipulators, you must add the following preprocessor directive

```
#include <iomanip>
```

precision – the total number of digits that appear before and after the decimal point. (default on most systems is 6 significant digits)

setprecision(n)

This manipulator allows us to control the number of digits displayed and when used with the manipulator `fixed`, the number of places displayed to the right of the decimal point. Argument `n` is used to indicate the number of places displayed. Unlike `setw()`, once the precision is set, it remains at that value until changed in the program.

showpoint

Showpoint forces the decimal point and any trailing zeros to be displayed. The number of places displayed depends on the current precision.

If a float variable called `fVal` contains the value 5, the statement

```
cout << showpoint << setprecision(2) << fVal;
```

will now output `5.00` rather than `5`

The `fixed` and `showpoint` manipulators may be specified in the same statement.

```
cout << fixed << showpoint;
```

fixed

Fixed causes all subsequent numbers to print in fixed point notation with the number of digits to the right of the decimal specified by the `setprecision` manipulator. The default is 6.

```
cout << fixed;
```

The fixed format will remain until it is disabled with the statement

```
cout.unsetf(ios::fixed);
```

Sample program:

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

void main(void)
{
    float n1 = 3;
    float n2 = 3.14159;
    float n3 = 12.75;

    cout << fixed << setprecision(2);
    cout << setw(10) << n1 << endl;
    cout << setw(8) << n2 << endl;
    cout << n3 << endl;
    cout << setw(10) << setprecision(5) << n3;
}
```

Output:

```
          3.00
         3.14
12.75
    12.75000
```

:

Basic Input in C++

Input Stream - A sequence of characters from an input device to the computer. To input information in C++ we will use a variable called *cin* (pronounced "see-in"). The identifier *cin* stands for common input.

cin is a predefined variable in C++ that indicates you are going to input a stream of characters from an input device (*cin* is associated with the standard input device which is the keyboard). *cin* uses an operator called the *extraction operator* (>>). You will often see >> referred to as "get from". The extraction operator requires two operands. For our purposes the operand on the left will be the *cin* variable and the operand on the right will be a variable (char, int, float etc.). The general form for the input statement is shown below:

GENERAL FORM for cin: cin >> variable >> variable ...;

NOTE: Unlike the output statement which may contain constants, variables or more complicated expressions, the **only items that may be specified in an input statement are the names of one or more variables.** The purpose of an input statement is to fill a memory location. The only memory locations that may be modified while the program is running are those named by variables.

Examples:

```
cin >> length;  
cin >> width;
```

or

```
cin >> length >> width;
```

Although you may use a sequence of extraction operators in the same input statement, we will typically prompt for one item at a time using *cout* and read into variables one at a time using *cin*.

```
cout << "Enter the length: ";  
cin >> length;  
cout << "Enter the width: ";  
cin >> width;
```

It is important that the input entered at the keyboard match the data type of the variable specified in the input statement. Variables of type `int` require an integer literal constant. Variables of type `float` may receive either an integer or a floating point literal constant. If an integer is entered at the keyboard, the compiler will convert it to a float and store it appropriately. Variables of type `char` require a single printable character other than a blank. When a sequence of extraction operators appear in the same input statement, the various data items may be delimited with *whitespace characters*. Whitespace characters are blanks and certain nonprintable characters such as the end-of-line character and the tab character.

It is very important that you carefully enter data from the keyboard. Entering data of a type not compatible with the input variable specified in the `cin` statement can cause very unexpected results.

Given the following declarations:

```
int n1;  
int n2;  
int n3;  
float aFloatNum;  
char theChar;
```

For each input statement and the indicated data from the keyboard, indicate the contents of the memory locations after the input statement has been executed.

<u>Input Statement</u>	<u>Data</u>
cin >> n1;	12
cin >> n1 >> n2;	8 16
cin >> theChar >> aFloatNum;	X 3.75
cin >> n1 >> n2 >> n3;	12 24 36
cin >> aFloatNum >> theChar >> n1;	5X3
cin >> n1 >> n2 >> n3	5 6

C++ Program Worksheet Review

We are going to take one last look at the flowchart for calculating and outputting the sum and average of two numbers and look at the program construction piece by piece.

1. All C++ programs will begin with

This is a preprocessor directive that tells the compiler that we need to use some code contained in a file called `iostream`. Specifically, we need this to get input from the user (*cin*) and provide output (*cout*).

2. All C++ programs must contain a function called `main`. This function begins
3. We need to use some memory locations to store the two numbers, their sum, and their average. We will use names to represent these memory locations. In C++ these names are called *identifiers*.

Identifiers are associated with memory locations via declarations.

Our programs will begin with a declaration section. In this section we will

- give names of variables and named constants
- tell the compiler how much memory to allocate (by specifying the data type)
- tell the compiler how the contents of the memory locations are to be interpreted
- tell the compiler whether the identifier represents a location that can be changed, or one that cannot

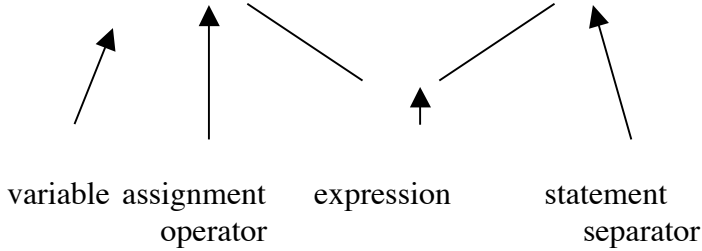
The necessary memory locations for this program will hold numeric values. We will choose between type *int*, and type *float*. If we wish to store values that contain a decimal point we will choose *float*. For integer values, type *int*. Which type will provide the maximum flexibility for the user?


```
sum = num1 + num2
```

Assignment Statement

```
sum = num1 + num2;
```

```
sum = num1 + num2;
```



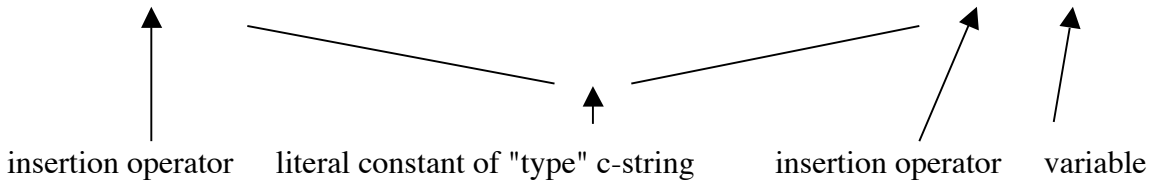
The statement above assigns to the memory location referenced by the identifier `sum`, the value of the expression on the right side of the assignment operator (`=`). Repeat for the remaining assignment statements (processes).

Finally, output the results.

```
output sum
```

```
cout << "The sum of the numbers is " << sum;
```

```
cout << "\n\nThe sum of the numbers is " << sum;
```



Even though this program was introduced earlier, the complete program is on the next page. Compare this program carefully with your flowchart and study the implementation of the algorithm in C++.

```

#include <iostream>
using namespace std;

// This program calculates the sum and average of two input values.

void main(void)
{
    float num1;    // First value to process - INPUT
    float num2;    // Second value to process - INPUT
    float sum;     // Sum of two values - CALC & OUTPUT
    float average; // Average of two values - CALC & OUTPUT

    cout << "\n\n\nYour name here";
    cout << "\nCS 1A - days & time";
    cout << "\nTitle of homework assignment";
    cout << "\nDue date: mm/dd/yy\n\n";

    // obtain two numbers from user
    cout << "Enter first value: ";
    cin >> num1;
    cout << "Enter second value: ";
    cin >> num2;

    // calculate the sum and average
    sum = num1 + num2;
    average = sum / 2.0;

    //output the sum and average
    cout << "\nThe sum of the numbers is " << sum;
    cout << "\nThe average of the numbers is " << average;
}

```

Pay close attention to the style (program heading, spacing, indentation, blank lines between sections of the program, internal comments) shown above. **NOTE: When you wish to indent, use the tab key, NOT the spacebar. Make sure the tab is set to 2. If you are not sure, check the preferences settings.**

Gross Pay Calculation (flowchart from Ch. 3)

Problem: Obtain from the user an hourly pay rate and the number of hours worked for the week. Calculate their pay for the week (no overtime, and no taxes). Output the result.

```
// place the preprocessor directives here
```

```
// This program calculates a weekly paycheck given the hours worked and the hourly rate.
```

```
void main(void)
```

```
{
```

```
    // declare your variables here (include a data table)
```

```
    // input the hours worked and hourly pay rate (use cout to prompt, cin to input)
```

```
    // calculate the gross pay using an assignment statement
```

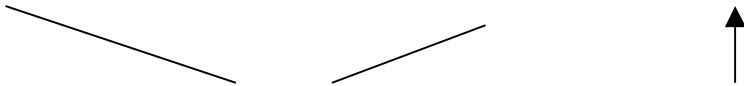
```
    // output the gross pay (descriptive and well spaced)
```

```
}
```

Review

1. << is called the _____ operator and is used with _____.
2. >> is called the _____ operator and is used with _____.
3. Given the following:

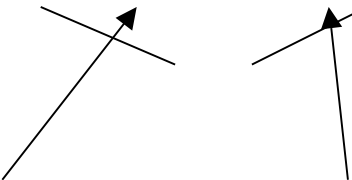
```
cout << "This program was written by " << PROGRAMMER;
```



Identify the indicated parts of the above statement.

4. Given the following:

```
cout << (3 + 4 + 5) / 3.0;
```



Identify the indicated parts of the above statement.

5. \n, \t, \" are examples of _____