LESSON SET

# 3 Expressions, Input, Output and Data Type Conversions

**PURPOSE**

1. To learn input and formatted output statements
2. To learn data type conversions (coercion and casting)
3. To work with constants and mathematical functions

**PROCEDURE**

1. Students should read the Pre-lab Reading Assignment before coming to lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab.

| Contents | Pre-requisites | Approximate completion time | Page number | Check when done |
|---|---|---|---|---|
| Pre-lab Reading Assignment | | 20 min. | 26 | |
| Pre-lab Writing Assignment | Pre-lab reading | 10 min. | 32 | |
| **LESSON 3A** | | | | |
| **Lab 3.1** Working with the `cin` Statement | Confidence in use of data types | 15 min. | 33 | |
| **Lab 3.2** Formatting Output | Basic understanding of `cout` and formatted output | 15 min. | 35 | |
| **Lab 3.3** Arithmetic Operations and Math Functions | Understanding of pre-defined functions `pow` and `sqrt` | 20 min. | 36 | |
| **LESSON 3B** | | | | |
| **Lab 3.4** Working with Type Casting | Understanding of type casting (implicit and explicit data conversion) | 20 min. | 37 | |

*continues*

| Lab 3.5 | | | |
|---|---|---|---|
| Student Generated Code Assignments | Understanding of all concepts covered in this section. | 30 min. | 39 |

## PRE-LAB READING ASSIGNMENT

### Review of the `cout` Statement

The **`cout`** statement invokes an output stream, which is a sequence of characters to be displayed to the screen.

*Example:* `cout << "Hi there";`

The **insertion operator** << inserts the string of characters `Hi there` into the output stream that goes to the screen. The `cout` statement can be thought of as an **`ostream`** (output stream) data type.

### Input Instructions

Just as the `cout` statement transfers data from the computer to the "outside" world, the **`cin`** statement transfers data into the computer from the keyboard.

*Example:* `cin >> grade;`

The **extraction operator** >> extracts an item from the input stream. In this case, since `grade` is an integer, this instruction will wait for an integer to be entered at the keyboard and then will place that number in the memory location called `grade`.

Just as `cout` is of type ostream, `cin` is considered to be an **istream** (input stream) data type. In order to use `cin` and `cout` in a C++ program, the `#include <iostream>` directive should be included in the header. The >> extraction operator also serves as a separator between input variables, allowing more than one memory location to be loaded with a single `cin` instruction. The values read must be the same data type as their corresponding variables, although a floating point variable could receive an integer since the conversion will be made automatically. Conversion is discussed later in this lesson.

*Example:*
```
float rate;
float hours;

cin >> rate >> hours;
```

The `cin` statement will wait for two floating point numbers (separated by at least one blank space) to be input from the keyboard. The first will be stored in `rate` and the second in `hours`.

There is one problem with the example above; it does not indicate to the user for what data the `cin` statement is waiting. Remember that the `cin` statement is expecting data from the user at the keyboard. For this reason, every `cin` statement should be preceded by a `cout` statement that indicates to the user the data to be input. Such a `cout` statement is called a **prompt**.

*Example:*

```
float rate, hours;
                          // More than one variable can be defined
                          // in a statement. Multiple variables are
                          // separated by a comma.
float grosspay;

cout << "Please input the pay rate per hour"
     << " and then the number of hours worked" << endl;
cin >> rate >> hours;

grosspay = rate * hours;          // finds the grosspay

cout << endl << "The rate is = " << rate << endl;
cout << "The number of hours = " << hours << endl;
cout << "The gross pay =  " << grosspay << endl;
```

When `cin` is reading numeric data, whitespace (blank spaces or unseen control characters) preceding the number are ignored and the read continues until a non-numeric character is encountered.

## Strings

It is often useful to store a string, such as a name, in a variable. Since the `char` data type holds only a single character, we must define a variable that is able to hold a whole sequence of characters. One way to do this is through an **array** of characters, often referred to as a **C-string** in C++. When using this method to define a string, the programmer must indicate how many characters it can hold. The last character must be reserved for the end-of-string character `'\0'` which marks the end of the string. In Example 2 below, the variable `name` can hold up to 11 characters even though the size of the array indicates 12. The extra character is reserved for the end-of-string marker. Arrays of characters are discussed in a later chapter. For now we can define a variable to be a **string object**: Example 1 below.

*Example 1 (using a string object)*

```
string name;
cout << "What is your name";
cin  >> name;
cout << "Hi " << name << endl;
```

*Example 2 (using a C-string)*

```
char name[12]
cout << "What is your name";
cin >> name;
cout << "Hi " << name << endl;
```

Although Example 1 will work, we often do not use `cin >>` to read in strings. This is because of the way it handles **whitespace** (blank spaces, tabs, line breaks, etc.). When `cin >>` is reading numeric data, leading whitespace is ignored and the read continues until a non-numeric character is encountered. As one might expect, `cin >>` is a logical choice for reading numeric data. However, when `cin >>` is reading into a variable defined as a string, it skips leading whitespaces but stops if a blank space is encountered within the string. Names that have a space in it such

as Mary Lou, would not be read into one variable using `cin >>`. We can get around this restriction by using special functions for reading whole lines of input. The `getline` function allows us to input characters into a string object. In Example 1 above we could read a name like Mary Lou into the `name` variable with the statement

```
getline(cin, name);
```

The first word in the parentheses is an indication of "where" the data is coming from. In this case it is coming from the keyboard so we use `cin`. Data could come from other sources such as files (discussed later in this chapter) in which case the name of the file would be used instead of `cin`. The second word in parentheses is the name of the variable that will "receive" the string (`name` in this case).

When using C-strings, we can read whole lines of input using `cin.getline (string_name, length)`, where length specifies the number of characters the C-string can hold. In Example 2 above, we could read a name like Mary Lou into the `name` variable with the statement

```
cin.getline(name,12);
```

This allows a maximum of 11 characters to be read in and stored in `name`, reserving a space for the '\0' end-of-string character.

## Summary of storing and inputting strings

| | |
|---|---|
| `cin >> name;` | Skips leading whitespaces<br>Stops at the first trailing whitespace which is not consumed (ie. the whitespace is not placed in name) |
| `cin.getline(name,12);` | Does not skip leading whitespaces<br>Stops when either 11 characters are read or when an end-of-line '\n' character is encountered (which is not consumed) |

## Formatted Output

C++ has special instructions that allow the user to control output attributes such as spacing, decimal point precision, data formatting and other features.

*Example:*

```
cout << fixed                 // This displays the output in decimal
                              // format rather than scientific notation.

cout << showpoint;            // This forces all floating-point output to
                              // show a decimal point, even if the values
                              // are whole numbers

cout << setprecision(2);      // This rounds all floating-point numbers
                              // to 2 decimal places
```

The order in which these **stream manipulators** appear does not matter. In fact, the above statements could have been written as one instruction:

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

Spacing is handled by an indication of the width of the field that the number, character, or string is to be placed. It can be done with the `cout.width(n);` where `n` is the width size. However it is more commonly done by the `setw(n)` within a `cout` statement. The `#include <iomanip>` directive must be included in the header (global section) for features such as `setprecision()` and `setw()`.

*Example:*
```
float price = 9.5;
float rate = 8.76;
cout << setw(10) << price << setw(7) << rate;
```

The above statements will print the following:

```
    9.5   8.76
```

There are seven blank spaces before 9.5 and three blank spaces between the numbers. The numbers are right justified. The computer memory stores this as follows:

| | | | | | | | 9 | . | 5 | | | | 8 | . | 7 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Note: So far we have used `endl` for a new line of output. `'\n'` is an escape sequence which can be used as a character in a string to accomplish the same thing.

*Example:* Both of the following will do the same thing.

```
cout << "Hi there\n";            cout << "Hi there" << endl;
```

## Expressions

Recall from Lesson Set 2 that the assignment statement consists of two parts: a variable on the left and an expression on the right. The expression is converted to one value that is placed in the memory location corresponding to the variable on the left. These expressions can consist of variables, constants and literals combined with various operators. It is important to remember the mathematical precedence rules which are applied when solving these expressions.

## Precedence Rules of Arithmetic Operators

1. Anything grouped in parentheses is top priority
2. Unary negation (example: –8)
3. Multiplication, Division and Modulus * / %
4. Addition and Subtraction + –

*Example:*
```
( 8 * 4/2 + 9 - 4/2 + 6 * (4+3) )
( 8 * 4/2 + 9 - 4/2 + 6 * 7 )
( 32 /2   + 9 - 4/2 + 6 * 7 )
(   16    + 9 - 4/2 + 6 * 7 )
(   16    + 9 -  2  + 6 * 7 )
(   16    + 9 -  2  + 42 )
(      25     -  2  + 42 )
(           23      + 42 )  =    65
```

## Converting Algebraic Expressions to C++

One of the challenges of learning a new computer language is the task of changing algebraic expressions to their equivalent computer instructions.

*Example:* `4y(3-2)y+7`

How would this algebraic expression be implemented in C++?

```
4 * y * (3-2) * y + 7
```

Other expressions are a bit more challenging. Consider the quadratic formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

We need to know how C++ handles functions such as the square root and squaring functions.

There are several predefined math library routines that are contained in the `cmath` library. In order to use these we must have the `#include <cmath>` directive in the header.

Exponents in C++ are handled by the `pow(number,exp)` function, where `number` indicates the base and `exp` is the exponent. For example,

$2^3$ would be written as `pow(2,3)`

$5^9$ would be written as `pow(5,9)`

Square roots are handled by `sqrt(n)`. For example,

$\sqrt{9}$ would be written as `sqrt(9)`

Look at the following C++ statements and try to determine what they are doing.

```
formula1 = ( -b + sqrt(pow(b,2) -(4 * a * c))) / (2 * a);
formula2 = ( -b - sqrt(pow(b,2) -(4 * a * c))) / (2 * a);
```

(These are the roots from the quadratic formula in C++ format.)

## Data Type Conversions

Recall the discussion of data types from Lesson Set 2. Whenever an integer and a floating point variable or constant are mixed in an operation, the integer is changed temporarily to its equivalent floating point. This automatic conversion is called **implicit type coercion**.

Consider the following:

```
int count;
count = 7.8;
```

We are trying to put a floating point number into an integer memory location. This is like trying to stuff a package into a mailbox that is only large enough to contain letters. Something has to give. In C++ the floating point is **truncated** (the entire fractional component is cut off) and, thus, we have loss of information.

Type conversions can be made explicit (by the programmer) by using the following general format: static_cast<DataType>(Value). This is called **type casting** or **type conversion**.

*Example:*

```
int count;
float sum;

count = 10.89;                    // Float to integer  This is Type coercion
                                  // 10 is stored in count

count = static_cast<int>(10.89);  // Also float to integer; however this is
                                  // type casting
```

If two integers are divided, the result is an integer that is truncated. This can create unexpected results.

*Example:*

```
int num_As = 10;
int totalgrade = 50;
float percent_As;

percent_As = num_As / totalgrade;
```

In this problem we would expect `percent_As` to be .20 since 10/50 is .20. However since both `num_As` and `totalgrade` are integers, the result is integer division which gives a truncated number. In this case it is 0. Whenever a smaller integer value is divided by a larger integer value the result will always be 0. We can correct this problem by type casting.

```
percent_As = static_cast<float>(num_As)/totalgrade;
```

Although the variable `num_As` itself remains an integer, the type cast causes the divide operation to use a copy of the `num_As` value which has been converted to a float. A float is thus being divided by the integer `totalGrade` and the result (through type coercion) will be a floating-point number.

## PRE-LAB WRITING ASSIGNMENT

### Fill-in-the-Blank Questions

1. What is the final value (in C++) of the following expression?

   `(5 - 16 / 2 * 3 + (3 + 2 / 2) - 5)` _____

2. How would the following expression be written in C++?

   $2x + 3^4$

   _____

3. Implicit conversion is also known as data type _____.
4. Explicit type conversion is also known as type _____.
5. List the preprocessor directive that must be included for `cin` and `cout` to be used in a C++ program. _____
6. Blank spaces or unseen control characters in a data file are referred to as _____.
7. The << in a `cout` statement is called the _____operator.
8. The `#include<`_____> is needed for formatted output.
9. The `'\n'` is a special character that _____.

# LESSON 3A

## LAB 3.1 Working with the `cin` Statement

Bring in the program `bill.cpp` from the Lab 3 folder. The code is listed below:

```
// This program will read in the quantity of a particular item and its price.
// It will then print out the total price.
// The input will come from the keyboard and the output will go to
// the screen.

//  PLACE YOUR NAME HERE

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


int main()

{
      int   quantity;         // contains the amount of items purchased
      float itemPrice;        // contains the price of each item
      float totalBill;        // contains the total bill.


      cout << setprecision(2) << fixed << showpoint;  // formatted output

      cout << "Please input the number of items bought" << endl;

      // Fill in the input statement to bring in the quantity.

      // Fill in the prompt to ask for the price.

      // Fill in the input statement to bring in the price of each item.

      // Fill in the assignment statement to determine the total bill.

      // Fill in the output statement to print total bill,
      // with a label to the screen.

      return 0;
}
```

*Exercise 1:* Complete the program so that a sample run inputting 22 for the number of items bought and 10.98 for the price of each item will produce the results below.

Sample run of the program.

```
Please input the number of items bought
22
```

```
Please input the price of each item
10.98

The total bill is $241.56
```

*Exercise 2:* Once you have the program working, change the instruction:

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

to

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

Rerun the program with the same data given in Exercise 1 above and record your results. What do you think the fixed attribute in the `cout` statement does?

*Exercise 3:* Now put the `fixed` attribute back in and change the instruction to make the precision 4. Rerun the program with the same data given in Exercise 1 and record your results. What do you think the `setprecision( )` attribute in the `cout` statement does?

The attribute `showpoint` forces all floating point output to show a decimal point even if the values are whole numbers. In some environments this is done automatically.

(optional)

Exercise 4: Add the following directive to the program: `#include <string>` in the header. Alter the program so that the program first asks for the name of the product (which can be read into a string object) so that the following sample run of the program will appear.

**Please input the name of the item**
Milk
**Please input the number of items bought**
4

**Please input the price of each item**
1.97

**The item that you bought is Milk**
**The total bill is $7.88**

Now altar the program, if you have not already done so, so that the name of an item could include a space within its string.

**Please input the name of the item**
Chocolate Ice Cream

**Please input the number of items bought**
4

**Please input the price of each item**
1.97

**The item that you bought is Chocolate Ice Cream**
**The total bill is $7.88**

## LAB 3.2 Formatting Output

Look at the following table:

```
PRICE    QUANTITY

 1.95     8
10.89     9
```

Assume that from the left margin, the price takes up fifteen spaces. We could say that the numbers are right justified in a 15-width space. Starting where the price ends, the next field (quantity) takes up twelve spaces. We can use the formatted output from Lab 3.1 and the statement `setw(n)` where `n` is some integer to indi- cate the width to produce such tables.

Bring in the program `tabledata.cpp` from the Lab 3 folder. The code is as follows:

```
//  This program will bring in two prices and two quantities of items
//  from the keyboard and print those numbers in a formatted chart.

//PLACE YOUR NAME HERE

#include <iostream>
#include ___________  // Fill in the code to bring in the library for
                      // formatted output.
using namespace std;

int main()

{
      float price1, price2;         // The price of 2 items
      int   quantity1, quantity2;   // The quantity of 2 items


      cout << setprecision(2) << fixed << showpoint;
      cout << "Please input the price and quantity of the first item" << endl;

      // Fill in the input statement that reads in price1 and
      // quantity1 from the keyboard.

      // Fill in the prompt for the second price and quantity.

      // Fill in the input statement that reads in price2 and
      // quantity2 from the keyboard.

      cout << setw(15) << "PRICE" << setw(12) << "QUANTITY\n\n";

      // Fill in the output statement that prints the first price
      // and quantity. Be sure to use setw() statements.

      // Fill in the output statement that prints the second price
      // and quantity.

      return 0;

}
```

*Exercise 1:* Finish the code above by filling in the blanks and the instructions necessary to execute the following sample run. Note that two or more data items can be input at one time by having at least one blank space between them before hitting the enter key.

```
Please input the price and quantity of the first item
1.95 8

Please input the price and quantity of the second item
10.89 9

          PRICE    QUANTITY

          1.95        8
         10.89        9
```

## LAB 3.3 Arithmetic Operations and Math Functions

Bring in the program `righttrig.cpp` from the Lab 3 folder. The code is as follows:

```
// This program will input the value of two sides of a right triangle and then
// determine the size of the hypotenuse.

// PLACE YOUR NAME HERE

#include <iostream>
#include <cmath>          // needed for math functions like sqrt()
using namespace std;


int main()

{

      float a,b;          // the smaller two sides of the triangle
      float hyp;          // the hypotenuse calculated by the program

      cout << "Please input the value of the two sides" << endl;
      cin >> a >> b;


      // Fill in the assignment statement that determines the hypotenuse

      cout << "The sides of the right triangle are " << a << " and " << b << endl;

      cout << "The hypotenuse is " << hyp << endl;

      return 0;
}
```

The formula for finding the hypotenuse is $hyp = \sqrt{a^2 + b^2}$.

How can this be implemented in C++? Hint: You will use two pre-defined math functions (one of them twice) learned in this lesson. One of them will be "inside" the other.

*Exercise 1:* Fill in the missing statement so that the following sample run is implemented:

```
Please input the value of the two sides
9 3
The sides of the right triangle are 9 and 3
The hypotenuse is 9.48683
```

*Exercise 2:* Alter the program so that the sample run now looks like the following:

```
Please input the value of the two sides
9 3
The sides of the right triangle are 9 and 3
The hypotenuse is 9.49
```

Note: This is not a trivial change. You must include another directive as well as use the formatted features discussed in the earlier labs of this lesson. Notice that the change is made only to the value of the hypotenuse and not to the values of 9 and 3.

## LESSON 3B

### LAB 3.4 Working with Type Casting

Bring in the program `batavg.cpp` from the Lab 3 folder. The code follows.

```
// This program will determine the batting average of a player.
// The number of hits and at bats are set internally in the program.

// PLACE YOUR NAME HERE

#include <iostream>
using namespace std;


const int AT_BAT = 421;
const int HITS = 123;

int main()
{
      int batAvg;

      batAvg = HITS / AT_BAT                                  // an assignment statement
      cout << "The batting average is " << batAvg << endl;    // output the result

      return 0;
}
```

*Exercise 1:* Run this program and record the results. The batting average is _____.

*Exercise 2:* There is a logic error in this program centering around data types. Does changing the data type of `batavg` from `int` to `float` solve the problem? Make that change and run the program again and record the result.

The batting average is _____.

*Exercise 3:* Continue to work with this program until you get the correct result. The correct result should be 0.292162. Do not change the data type of the two named constants. Instead, use a typecast to solve the problem.

## LAB 3.5 Student Generated Code Assignments

*Option 1:* Write a program that will read in 3 grades from the keyboard and will print the average (to 2 decimal places) of those grades to the screen. It should include good prompts and labeled output. Use the examples from the earlier labs to help you. You will want to begin with a design. The Lesson Set 1 Pre-lab Reading Assignment gave an introduction for a design similar to this problem. Notice in the sample run that the answer is stored in fixed point notation with two decimal points of precision.

*Sample run:*

```
Please input the first grade
97

Please input the second grade
98.3

Please input the third grade
95

The average of the three grades is 96.77
```

*Option 2:* The Woody furniture company sells the following three styles of chairs:

| Style | Price Per Chair |
|---|---|
| American Colonial | $ 85.00 |
| Modern | $ 57.50 |
| French Classical | $127.75 |

Write a program that will input the amount of chairs sold for each style. It will print the total dollar sales of each style as well as the total sales of all chairs in fixed point notation with two decimal places.

*Sample run:*

```
Please input the number of American Colonial chairs sold
20
Please input the number of Modern chairs sold
15
Please input the number of French Classical chairs sold
5
```

```
The total sales of American Colonial chairs $1700.00
The total sales of Modern chairs $862.50
The total sales of French Classical chairs $638.75
The total sales of all chairs $3201.25
```

*Option 3:* Write a program that will input total sales (sales plus tax) that a business generates for a particular month. The program will also input the state and local sales tax percentage. It will output the total sales plus the state tax and local tax to be paid. The output should be in fixed notation with 2 decimal places.

*Sample run:*

```
Please input the total sales for the month
1080
Please input the state tax percentage in decimal form (.02 for 2%)
.06
Please input the local tax percentage in decimal form (.02 for 2%)
.02

The total sales for the month is $1080.00
The state tax for the month is $64.80
The local tax for the month is $21.60
```