

Chapter 4

Intro to Programming in C++

Computer Programming

Earlier we defined an algorithm to be a step by step process for solving a problem. **Computer programming** is the process of implementing the algorithm using a language the computer can understand. So a **computer program** is a step by step set of instructions performed by the computer.

A **programming language** is a set of special words, symbols and rules for constructing a program. There are many programming languages each with their own set of rules.

- **Syntax** rules govern how valid instructions are to be written in a given language.
- **Semantics** rules determine the meaning attached to instructions written in a given language.

There are many elements to a program. There are variables, named constants, data types and functions to name a few. A name or an *identifier* as it is called in a programming language will reference each of these elements.

Identifiers are used in C++ to name things. They may contain

- letters (a-z, A-Z)
- digits (0-9)
- the underscore (_) character

An identifier **must** begin with a letter or an underscore.

Identifiers are used to name variables, named constants and functions in a program. **It is very important that identifiers be meaningful.** In other words, grossPay would be a meaningful identifier in a program used to calculate a weekly or monthly payroll where the identifier gP would not.

NOTE: C++ is case-sensitive – that is to say the identifiers `grossPay` `GrossPay` `grosspay` and `GROSSPAY` would all be viewed as different identifiers by the compiler.

Variable – the symbolic name for a location in memory referenced by an identifier that contains a data value that may change during program execution. Variables must be declared.

Named Constant – the symbolic name for a location in memory referenced by an identifier that contains a data value that does NOT change during program execution. Named constants must be declared.

When we run programs we need to store values that will be used by the program. When variables and named constants are declared we are really requesting memory locations within the computer. We need to specify two things:

- how much memory,
- and how the contents of these memory locations are to be viewed. We do this with a data type.

Data Type – a set of valid data values along with the operations that may be performed on those values. The C++ data types we will use in this class are as follows:

char - one alphanumeric character enclosed in single quotes.

'a' '!' 'C' '\$' 'x' '*'

int - positive or negative integers with or without a sign (commas are not allowed)

23 -5 6 -100 0 etc.

float - positive or negative numbers containing an integer part and a fractional part with a decimal point in between.

9.5 4. 37895.75 .8 0.5 etc.

c-string - an array of characters used to hold data containing more than one character. The last character in a c-string is called the null terminator (\0). **NOTE: The word c-string is not a data type and does not appear in the declaration. See declarations below.**

char lastName[15] - the variable lastName could hold a maximum of 15 characters (14 plus the null terminator)

char response[3] - the variable response could hold a maximum of 3 characters (2 plus the null terminator)

C++ INTRO LABS AND EXERCISES

Learning a programming language such as C++ is like learning any other foreign language. The vocabulary and structure will take some time to get used to. The exercises on the following pages are designed to introduce you to the compiler and to provide worksheets that will assist you in learning the necessary vocabulary. The same topics are covered multiple times in the worksheets in the hope that this repetition will prove helpful. Review these examples on a regular basis until you become comfortable with the syntax, vocabulary, and program format.

Name _____

CS 1A Intro to C++ Lab

Type in the following program EXACTLY as it appears below.

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

void main( )
{
const char STUDENT[20] = "1A Student Again";
  char firstName[15];
  int age;

  cout << "This program was written by 1A Student.";
  cout << "This program was written by " << STUDENT;

  cout << "Enter your first name: ";
  cin >> firstName;
  cout << "Enter your age: ";
  cin >> age;

  cout << "Hello " << firstName << "How does it feel to be"
        << age << "years old?";
}
```

1. Run the program and comment on the output.

2. Change `cout << "This program was written by 1A Student.";` to
`cout << "This program was written by 1A Student.\n";`
Run the program and comment on the output.

3. Change `cout << "This program was written by " << STUDENT;` to
`cout << "This program was written by " << STUDENT`
`<< endl;`

Run the program and comment on the output. What do **endl** and **\n** do?

4. Change

```
cout << "Hello " << firstName << "How does it feel to be"  
      << age << "years old?;          to  
cout << "Hello " << firstName << "How does it feel to be"  
      << setw(4) << age << "years old?";  
Run the program and comment on the output.
```

5. Using your recently obtained knowledge of C++ make the necessary changes to the program so your output looks EXACTLY as shown below using your name and age when prompted.

This program was written by 1A Student.

This program was written by 1A Student Again!

Enter your first name: *Yourname*
Enter your age: *Yourage*

Hello, *Yourname*.
How does it feel to be *Yourage* years old?

Yourname signing off for now.

Turn In (IN THIS ORDER)

- This assignment sheet
- A listing of the program
- A listing of the output

STAPLE IN THE UPPER **LEFT** CORNER

PROGRAM COMPONENT EXAMPLE

Given the following:

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

void main( )
{
    float floatVal;
    int intVal;

    cout << "Enter a floating point number: ";
    cin >> floatVal;
    cout << "Enter an integer: ";
    cin >> intVal;

    cout << fixed << setprecision(2);
    cout << "\n\nThe floating point value is " << setw(8) << floatVal << endl;
    cout << "The integer value is " << setw(6) << intVal << endl;
}
```

The two lines below are called preprocessor directives. These are commands sent to the preprocessor to include code in your C++ program before it is sent to the compiler. Code for basic input and output routines is contained in a header file called **iostream** and formatting code is found in **iomanip**. Note that you do not put semicolons at the end of these directives because they are not C++ statements.

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

In CS 1A we will simply say that the statement below tells the compiler where to find the files `iostream` and `iomanip`.

```
using namespace std;
```

Every C++ program must contain a function called `main`. Program execution begins with this function. French braces are used to mark the beginning and end of this block.

```
void main( )
{
    body of the function
}
```

The statements below are called declarations. In this case we are declaring two variables (names for memory locations whose values may change). This is really a request for memory. The compiler keeps track of the actual physical memory location where they are stored and we "symbolically" reference them using the *identifiers* `floatVal` and `intVal`.

```
float floatVal;
int intVal;
```

When we request memory we need to tell the compiler how much memory we need and how it is supposed to interpret the bit patterns contained in the locations. This is done through the use of a *data type*. *int* is a data type used to store integer values such as 5 -3 26 0 etc. *float* is a data type used to store decimal numbers.

Floating point numbers may be expressed using scientific notation (E notation). The number

652.34 is written as 6.5234E2 where E means "times 10 to the power"
0.005238 is written as 5.238E-3

Express the following numbers in scientific notation:

1254.75

0.0000835

The next section is used to obtain input from the user of the program. The C++ identifier *cout* is used to send output to the standard output device, the monitor. One or more items can be inserted into the output stream. These items are inserted using the *insertion operator* (<<). *cin* is used to accept input from the standard input device, the keyboard. One or more items may be extracted using the *extraction operator* (>>).

```
cout << "Enter a floating point number: ";  
cin >> floatVal;  
cout << "Enter an integer: ";  
cin >> intVal;
```

In the statement below, the *string literal* "Enter a floating point number: " is inserted into the stream and appears on the screen as a user prompt.

```
cout << "Enter a floating point number: ";
```

In the statement below, the information extracted from the input buffer is placed into the memory location (variable) called floatVal.

```
cin >> floatVal;
```

Note that all C++ statements end with a semicolon. The semicolon (;) is called the statement separator.

The following statement is used to specify the format of floating point numbers when they are displayed. *fixed* causes all subsequent numbers to print in fixed point notation. When used with the *fixed* manipulator, *setprecision* is used to set the number of places displayed to the right of the decimal point.

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

Once the fixed and setprecision manipulators have been specified they remain in effect until the program ends or they are changed by the programmer.

```
cout << "\n\nThe floating point value is " << setw(8) << floatVal << endl;  
cout << "The integer value is " << setw(6) << intVal << endl;
```

The setw() manipulator must be specified for each item in the output stream that needs formatting. The statement

```
cout << "\n\nThe floating point value is " << setw(8) << floatVal << endl;
```

uses the *escape sequence* `\n` to specify two newlines, followed by the *string literal* "The floating point value is ", followed by a manipulator that states that the next item output is to be right justified in a field width of 8, followed by the actual value stored in variable floatVal, followed by the endl manipulator. The value floatVal will be right justified in a field width of 8 with two places displayed to the right of the decimal point. How will the second statement be output?

C++ Program Guide

- Open CodeWarrior and go to the File Menu. From there select Preferences, then Fonts & Tabs. Select Courier New as your font and set the tab stop to 2. You may need to do this each time you enter CodeWarrior.

All C++ programs written in this class must begin with the following _____
_____.

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

Next begin the function named **main**. Precede this declaration with a description of what the program does.

```
// This program calculates the sum & average of 2 ages
void main(void)
{
```

Notice that the French brace sits directly under the v in void. After you type in the brace hit enter to move to the next line then press the tab key once. You are ready to start the declaration section. This is where declare any named constants and variables you will use in your program.

```
void main(void)
{
    int firstAge;      // first age – INPUT
    int secAge;        // second age – INPUT
    int ageSum;        // sum of the ages – CALC & OUTPUT
    float avgAge;      // average of ages – CALC & OUTPUT
```

When the declaration section is complete (data table and all), write the code to output the class headings. Use the tab key to line up the items in the data table.

```
    cout << "\n\nYour Name\nCS 1A TTh\nStyle Assignment\nDue Date\n\n";
```

Next, implement the steps outlined in the flowchart. The first block of statements will be your input block. For each parallelogram on the flowchart you will need a pair of cout/cin statements. cout to prompt the user, and cin to read the input. Begin this section with a comment describing what the code does.

```
// Obtain 2 ages from the user
cout << "Enter first age: ";
cin >> firstAge;
cout << "Enter second age: ";
cin >> secAge;
```

Next, move to the processing section and document your code.

```
// Calculate the sum and average of the ages
ageSum = firstAge + secAge;
avgAge = ageSum / 2.0;
```

Finally, output the results and end the main. Make sure the closing brace lines up with the beginning brace.

```
// output the calculated results
cout << "\nThe sum of the ages is " << ageSum << " and the average is "
    << avgAge;
}
```

Complete Program

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

// This program calculates the sum & average of 2 ages
void main(void)
{
    int firstAge;        // first age – INPUT
    int secAge;         // second age – INPUT
    int ageSum;         // sum of the ages – CALC & OUTPUT
    float avgAge;       // average of ages – CALC & OUTPUT

    cout << "\n\nYour Name\nCS 1A TTh\nStyle Assignment\nDue Date\n\n";

    // Obtain 2 ages from the user
    cout << "Enter first age: ";
    cin >> firstAge;
    cout << "Enter second age: ";
    cin >> secAge;

    // Calculate the sum and average of the ages
    ageSum = firstAge + secAge;
    avgAge = ageSum / 2.0;

    // output the calculated results
    cout << "\nThe sum of the ages is " << ageSum << " and the average is "
        << avgAge;
}
```

Declarations of Variables and Named Constants

Memory locations are associated with identifiers in a program via declarations. Recall, a compiler is the program that translates code written in a high level language into machine language. It is necessary to declare all variables and named constants before they are used in a program. We will place all declarations at the beginning of the program.

Named Constant Declarations:

```
const int DAY = 5;
const float PI = 3.14159;
const char CODE = 'H';
const char GREETING[6] = "Hello";
```

Memory is allocated for named constants and values are placed into the locations at *compilation time*. The value of a named constant cannot be changed during program execution. **This means that the identifier for a named constant may NOT appear on the left side of an assignment operator or in a *cin* statement.**

Variable Declarations:

```
int ageOne;
int ageTwo;
float averageAge;
char answer;
char userName[20];
```

NOTE: The variable `answer` does not use square brackets ([]) in its declaration. This variable will hold one single character such as 'Y' or 'N' where variable `userName` may contain up to 20 characters. Values such as "Sam Spade" or "Sally Smith" may be stored in variable `userName`.

Begin variable names with a lower case letter and capitalize the first letter of each new word contained in the identifier. Remember, no blank spaces.

Variable declarations should contain a **data table**. This table contains comments that describe what the variable represents in the program and how its value was obtained.

```
int ageOne;           // first age from user - INPUT
int ageTwo;          // second age from user - INPUT
float averageAge;    // average of two input ages - CALC & OUTPUT
char answer;        // holds 'Y' or 'N' response from user - INPUT
char userName[20];  // name of program user - INPUT
```

Memory is allocated for variables at *compilation time* and values are placed into the locations at *run time*. This is accomplished using an assignment statement or reading from the keyboard (*cin*).

Named Constant Worksheet

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

int main(void)
{
    const char MYNAME[12] = "Sally Smart";
    const int MYAGE = 19;
    const float MYGPA = 4.0;

    cout << "Hello, my name is Sally Smart.\n";
    cout << "Hello, my name is " << MYNAME << endl;
    cout << "My age is " << MYAGE << endl;
    cout << "My age next year will be " << MYAGE + 1 << endl;

    // write a statement to output the grade point average

}

const char MYNAME[12] = "Sally Smart";
const int MYAGE = 19;
const float MYGPA = 4.0;
```

MYNAME, MYAGE, and MYGPA are called **named constants**. They represent the names of memory locations that contain the values indicated in the declaration. When these identifiers appear in `cout` statements, the actual values stored in memory are output.

When `endl` is placed after an insertion operator (`<<`), the cursor moves to the next line.

When `\n` is placed in the middle of a string (characters enclosed in double quotes) the cursor moves to the next line. `endl` is a _____ and `\n` is an _____

Write a C++ program that has named constants for your name, the current semester (Spring), and the starting date for spring break. Produce well formatted and spaced output using these constants and literal strings to indicate whether or not you are looking forward to the break and what you plan to do during the break.

Turn In (IN THIS ORDER – **STAPLED IN THE FAR LEFT CORNER**)

- this sheet
- a listing of the C++ program
- a sample run

Assignment Statement

General Form:

```
variable = expression;
```

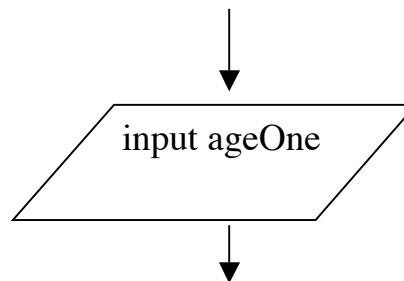
Assignment Statement Examples:

```
ageOne = 15;  
ageTwo = 23;  
averageAge = (ageOne + ageTwo) / 2.0;  
answer = 'y';
```

cin Examples:

```
cout << "Enter first age: ";  
cin >> ageOne;  
cout << "Enter second age: ";  
cin >> ageTwo;
```

Note: While only the *cin* statement actually places a value into a memory location, the *cout* placed on the line before the *cin* statement provides a user prompt so the user knows what data to enter. Each input shown on a flowchart will have a corresponding *cout/cin* pair in the program.



```
cout << "Enter first age: ";  
cin >> ageOne;
```

```

// This program outlines the basic parts of a C++ program.  "/" indicates a comment
// and causes the compiler to ignore text appearing on that line.
// Following are examples of preprocessor directives
#include <iostream>      // needed in all programs for basic input/output
using namespace std;

void main(void)
{
    // Declaration of named constants
    const char MY_NAME[25] = "Your name here";    // programmer name
    const char SECTION[25] = "Your 1A section here";    // class section
    const char DATE[10] = "Month dd, yyyy";        // current date

    // Declaration of variables
    // Note the data table to the right of variable names
    int num1;          // first value to average - INPUT
    int num2;          // second value to average - INPUT
    float average;     // average of the two values - CALC & OUTPUT

    // Output class headings
    cout<< "*****\n";
    cout<< MY_NAME << endl;
    cout<< SECTION << endl;
    cout<< DATE << endl;
    cout<< "*****\n\n";

    // Get numbers to average from user - prompt using cout and input using cin
    cout<< "Enter first integer to average: ";
    cin>> num1;
    cout<< "Enter second integer to average: ";
    cin>> num2;

    // Calculate the average - this is an assignment statement
    average = (num1 + num2) / 2.0;

    // Output the average
    cout<< "\n\nThe average of the numbers is: " << average << endl;
}

```

Matters of Style

Programming style can greatly enhance the readability of a program or it can detract from it and make a program confusing, difficult to read, debug and later modify. Style elements include the position of internal documentation, capitalization of identifiers, indentation and spacing. Spacing and indentation add to the visual organization of a program where capitalization gives a visual clue to a programmer as to what an identifier represents. We will use the following style requirements in CS 1A and CS 1B. **YOU ARE RESPONSIBLE FOR LEARNING THIS STYLE AND USING IT IN ALL PROGRAMS TURNED IN. REFER TO CHAPTER 5 IN THIS SHRINK WRAP BEFORE TURNING IN YOUR PROGRAMS.**

- Identifiers representing *variables* will begin with a lower case letter and each successive English word will be capitalized.

areaOfCircle hrsWorked hrlyRate length width rectangleArea

- Identifiers representing *named constants* are represented in all upper case.

PROGRAMMER CLASS_SECTION PI TAX_RATE

- Identifiers belonging to the C++ language are represented in lower case.

cin cout int float char const #include

- Indentation will be 2 spaces (press the tab key once - the tab should already be set to 2 in Codewarrior)

- Every program will contain a data table. The data table contains an explanation of the use of variables and named constants and how the variable values are obtained.

- Documentation in the body of the program will be on the line ABOVE the code it references , not on the same line. A data table is not executable code and is the only exception to this rule. Additionally, the data table is not part of the body but rather it is in the declaration section.

REVIEW THE PROGRAM ON THE PREVIOUS PAGE AND NOTICE THE USE OF THESE STYLE ELEMENTS IN THE PROGRAM. USE THESE 2 PAGES AND CHAPTER 5 EVERY TIME YOU WRITE A PROGRAM.

Review

1. A compiler translates code written in a _____ language into _____ language.
2. Identifiers are associated with memory locations via _____.
3. A _____ is the name of a location in memory that has a data value that may be changed. Values for these identifiers are obtained at _____ time using an _____ statement or a _____ statement.
4. A _____ is the name of a location in memory that has a data value that may not be changed. These identifiers get their values at _____ time and may not appear on _____ or in a _____ statement.
5. The documentation next to the declarations for variables and named constants is called the _____. It tells the reader _____ and _____ their values are obtained.
6. Each input block shown on a flowchart requires a _____ statement to prompt the user and a _____ statement to place the input value into the specified memory location.
7. Explain the difference between the following declarations
char charVal;
char strVal[10];

Declaration Section Exercise

Write the necessary declaration section for a program requiring the following *variables* and *named constants*. Use the proper style and be sure to include the data table. Also, remember that identifiers must be descriptive.

- a location to hold the name of the programmer (an unchanging value)
- a location to hold the current date (an unchanging value)
- locations to hold the names of two users of the program (input from the keyboard)
- locations to hold the ages of each of the two users (input from the keyboard)
- a location to hold the older of the two (calculated & output)
- a location to hold the average of the ages (calculated & output)

Types & Declarations

Using the chart below, for each data type, give an example of a literal constant of that type, an example of the declaration of a variable of that type, and an example of the declaration of a named constant of that type.

<u>Data Type</u>	<u>Literal Constant</u>	<u>Variable Declaration</u>	<u>Named Constant Declaration</u>
------------------	-------------------------	-----------------------------	-----------------------------------

int

float

char

c-string