Chapter 2

Intro to Programming in C++
(CS 1A Quick Review)
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. The word c-string is used to indicate a "C like" string. 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)
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.

```plaintext
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**).
Assignment "Statement"
  \[ \text{variable} = \text{expression}; \]

Assignment Statement Examples:

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

\textit{cin} Examples:

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

Note: While only the \textit{cin} statement actually places a value into a memory location, the \textit{cout} placed on the line before the \textit{cin} statement provides a user prompt so the user knows what data to enter. Each input shown on a flowchart will have a corresponding \textit{cout}/\textit{cin} pair in the program.
Basic Structure of a C++ Program

We have previously defined a module or subprogram as one small part of the solution to a problem. In C++, subprograms or modules are called functions. Every C++ program must have a function called main. We will think of this as the body of the program – the section of your program containing the executable statements outlined in your flowchart. The beginning and end of the statements contained in this function are marked with a left French brace ( { ) to mark the beginning and a right French brace ( } ) to mark the end. The statements between the braces are referred to as the function body.

- **Directive(s)** - information the program needs (a list of all necessary header files used in the program)

- **Heading** - void main (void) - functions by definition return a value (the above heading indicates that this function will not return a value)

- **main function** - 
  { 
  named constant declarations 
  variable declarations 
  executable statements 
  }

A sample C++ program follows.
// 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<< "*************************
";
    cout<< MY_NAME << endl;
    cout<< SECTION << endl;
    cout<< DATE << endl;
    cout<< "*************************
";

    // 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<< "The average of the numbers is: " << average << endl;
}
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)