There is no need however to use symbolic names for constants (like zero, two, fifty.) Instead we may use literal values.

E.g. 0, 2, 50 (int)
    0.0, 2.0, 50.0, 3.14 (double)
    'a', 'b', 'e' (char)

Note: char literals must be enclosed in single quotes.

C/C++ allows the use of symbolic constants through the use of **const**.

E.g. const double pi = 3.14;

Symbolic constants, like constant macros, add to program readability and maintainability.

Executable statements are of three types: input, output, and calculation.

Simple keyboard input and screen output are accomplished in C++ by use of the **iostream cin** and the **iostream cout**, both defined in iostream
Ex:  
cout << variable;
cout << expression;
cout << literal value;

Ex:  
cin >> variable;

In this context << AND >> are called the extraction AND insertion operators, respectively.

In the first example above, the value of variable is printed to the standard output device (screen). Variable can be of any built-in C++ data type.

Ex:  
int count; double sum; char letter;

cout << count << sum << letter;

We can also print string literals.

Ex:  
cout << "ENTER TEMPERATURE IN FAHRENHEIT:";

Note: string literals must be enclosed in double quotes. A string is NOT a separate data type in C/C++. Instead it is an array of characters. More on arrays later.
Two special characters are the newline \n, and the tab \t.

Ex. int hours, minutes;

    cout << "The time is now \n" << hours << ":" << minutes << endl;

When placed in the output stream, \n\nlends a newline to be printed.

Ex. cin >> variable;

This causes execution to pause until a value is entered at the standard input device (keyboard) which is then assigned to variable.

Calculations can be performed by using the basic arithmetic operators +, -, *, / (and % for int only).

A statement of the form

variable = expression;

directs the computer to evaluate expression.
THEN ASSIGN THE RESULTING VALUE TO VARIABLE. WE CALL IT THE ASSIGNMENT OPERATOR.

Ex: int a, b, c, d;
    d = a + b - c;

DATA AREAS:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEFORE</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>AFTER</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>-3</td>
</tr>
</tbody>
</table>

NOTE: ASSIGNMENT = MUST BE DISTINGUISHED FROM COMPARE FOR EQUALITY ==.

Ex: WRITE A C++ PROGRAM TO CALCULATE THE CIRCUMFERENCE AND AREA OF A CIRCLE, GIVEN ITS RADIUS.

Algorithm
1.) SET VALUE FOR RADIUS FROM USER.
2.) SET AREA TO \[ \pi \times \text{radius} \times \text{radius} \]
3.) SET CIRCUMFERENCE TO \[ 2 \times \pi \times \text{radius} \]
4.) PRINT VALUES OF AREA AND CIRCUMFERENCE.
// File: circle.cpp
// Description: Get input value for the radius of a circle, 
// print out the circumference and area. 
// Compile: g++ -o circle circle.cpp

#include<iostream>
using namespace std;

#define PI (3.14159)

int main(void)
{
    double radius, // Input: radius of circle.
    area, // Output: area of circle
    circumference; // Output: circumference of circle

    // Get radius from user.
    cout << "Enter the circle radius: ";
    cin >> radius;

    // Compute area.
    area = PI*radius*radius;

    // Compute circumference.
    circumference = 2*PI*radius;

    // Print out area and circumference.
    cout << "The area is " << area << ", and the circumference" << " is " << circumference << "." << endl;

    return (0);
}

Compile:
% g++ -o circle circle.cpp

Output:
% a.out
Enter the circle radius: 37
The area is 4300.84, and the circumference is 232.478.
WE CAN USE A CONSTANT MACRO FOR THE PROGRAM CONSTANT $\pi$:

```c
#define PI 3.14
```

Or we can declare a symbolic constant within function main:

```c
const double PI = 3.14;
```

--- SEE EXAMPLE `circle.c` ---

**ARRAYS**

An array is a contiguous set of memory locations each of which stores data of the same type.

Arrays are declared as follows:

```c
int list[10];
double array[20];
char word[30];
```

These declarations set aside space in memory for 10 ints, 20 doubles, and 30 chars respectively.
Note: Array names must follow the rules for C/C++ identifiers. In particular, the word "array" is not a reserved word in C/C++, though it may be a poor choice for an array name.

The elements of array list are referred to as:

\[ \text{list}[0], \text{list}[1], \ldots, \text{list}[9]. \]

The valid array indices start at 0 and end at one less than the number of elements in the array.

The data area for array list may be pictured as:

<table>
<thead>
<tr>
<th>Address</th>
<th>list</th>
<th>Symbolic Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td></td>
<td>list[0]</td>
</tr>
<tr>
<td>1012</td>
<td></td>
<td>list[1]</td>
</tr>
<tr>
<td>1014</td>
<td></td>
<td>list[2]</td>
</tr>
<tr>
<td>1018</td>
<td></td>
<td>list[3]</td>
</tr>
<tr>
<td>1020</td>
<td></td>
<td>list[4]</td>
</tr>
<tr>
<td>1022</td>
<td></td>
<td>list[5]</td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td>list[6]</td>
</tr>
<tr>
<td>1026</td>
<td></td>
<td>list[7]</td>
</tr>
<tr>
<td>1028</td>
<td></td>
<td>list[8]</td>
</tr>
<tr>
<td>1030</td>
<td></td>
<td>list[9]</td>
</tr>
</tbody>
</table>
Observe that list[103] does not refer to a memory location in array list.

Warning: it is possible however to read or write past the end of an array by referencing say list[10], list[111], etc. This is not a syntax error in C/C++ but can cause run-time errors which are very difficult to track down.

An array of chars (like word above) is called a string in C/C++. There are certain special operations for processing strings in the library string.h, which we will not cover.

C/C++ allows multidimensional arrays

Ex. double table[5][6]

Declares an array of 30 doubles which we may think of as being arranged in 5 rows and 6 columns:

\[
\begin{align*}
\text{table[0][0]}, & \quad \ldots \quad \text{table[0][5]} \\
\text{\ldots}, & \\
\text{table[4][0]}, & \quad \ldots \quad \text{table[4][5]} 
\end{align*}
\]