



Arrays, Strings, Scope, and Other Magic Stuff

A Freshman Engineering Computer Programming Lab
Developed by Dr. P.K. Imbrie, Purdue University

Overview:

In this lab, students will be learning about computer programming using the C programming language. The goal of this lab is to provide students with exercises that accelerate learning of fundamental computer programming materials through repetitive practice and through looking at programming from more than one angle.

Learning Objectives or Student Outcomes:

By the end of this lab, students will be able to

1. demonstrate an understanding of increment and decrement operators as well as the prefix and postfix uses of these;
2. use a given code in a program to increase their understanding of operators;
3. add blocks of code to a program using the increment operator;
4. write a main program that will prompt a user for a magnitude and an angle (in degrees) for a vector, pass the angle (in degrees) and the magnitude to a function through the argument/parameter list;
5. define a 10-element single subscripted integer array that is initialized (via the declaration statement) to contain the numbers 1 through 10 in elements 0 through 9, respectively;
6. run a given code within a main program, and, then, after seeing what happens, alter the code to make it more clear;
7. write a flowchart for an algorithm that will ask a user to enter all the elements for two real matrices;
8. develop a simple test case by hand to ensure that your algorithm works before you attempt to actually write a program; and
9. verify their algorithm and convert it to an appropriate C program.

Length of Lesson:

Four to six hours of in-class time, including any pre- and post-activity discussions.

Prior Knowledge to Ensure Student Preparation:

Prior to participating in this lab, students must have some kind of knowledge of C programming—particularly the increment and decrement operators as well as the prefix and postfix uses of these; adding blocks of code to programs; writing programs; arrays; running and altering given code; writing and verifying algorithms, and converting those algorithms to a C program.

Mention of this prior knowledge or classroom instruction is indicated by notes within the lesson.

Team Size/Composition:

Teams of 4 work best; if necessary, a few teams of 3 or 5 students may be formed. Each team member should be assigned a given role prior to beginning the lab. These roles should be as follows: meeting coordinator, time keeper, task manager, and record/note keeper. In cases of teams of three, the roles of meeting coordinator and time-keeper should be combined. In cases of teams of five, two students may split the role of record/note keeper.

How is *positive interdependence* ensured?

Each team will turn in one file for each part of the lab—5 files total. Also, the assignment of roles is set up to encourage positive interdependence.

How is *individual accountability* ensured?

The assignment of roles will also serve to help ensure individual accountability, as each person has specific responsibilities. Also, at the conclusion of the lab, students will complete the [Individual Check for Understanding](#).

Components of Assessment:

At the end of each part of the lab, a product is due. An idea of how that product should look will have been given to the students during the instructions for that activity.

Team Skills Needed for Success:

Team members must have the ability to communicate, cooperate and collaborate; they must also feel free to contribute their ideas and to give and receive constructive feedback.

How Are These Skills Emphasized?

The organization of the lab—with a product at the end of each meeting—will really emphasize the needs for these skills.

Materials Needed:

- computers
- C programming language tools

Instructions to Students:**1****Part I**

While not required, it is suggested that this part should be done with all members of the team working at one computer; the team meeting coordinator should serve as the keyboard operator.

An important part of C programming to really understand is the increment and decrement operators as well as the prefix and postfix uses of these. **(Note: these should have been discussed in very recent lecture with the students.)**

The increment operator (++) and the decrement operator (- -) can be applied to variables. However they cannot be applied to constants or ordinary expressions.

	<p>When using the increment (or decrement) operator, the value of the variable in memory is altered. This is different from normal usage of the + or – operators. If you execute a+b, neither the value of a nor b is changed, however a++ changes the value of a.</p> <p>Use the following code in a program that will get you on track to fully understand and appreciate the usages of these operators.</p> <pre>int a, b, c = 0; a = ++c; b = c++; printf("%d %d %d\n", a, b, ++c);</pre> <p>Now, add to your program a block of code that will show that i += 1 is the same as i++. Make another block that will go through a for loop that prints out the numbers 1-10. (Note: Students should have done this a few weeks prior in class; however, tell them to use the increment operator this time.) Change this to a while loop in the next block.</p> <p>Take a few minutes to add two more blocks of code that will demonstrate a learning experience. Comment within your code why each block is significant. In total, your code should have 6 separate blocks of code. Each should have a print statement to demonstrate the working of each code.</p> <p>Name your file <i>lab10_p1.c</i></p> <p>TIMEKEEPER: Your team should spend 20-30 minutes working with this section (this is only a guideline). Remember that it is for your own understanding and that your team is mutually responsible for the information.</p>
<p>2</p>	<p>Part II</p> <p>The idea of <i>call by value</i> is basically that you can pass a variable to a function and not lose the value of the variable through the function.</p> <p>Your task is to write a main program that will prompt a user for a magnitude and an angle (in degrees) for a vector, pass the angle (in degrees) and the magnitude to a function through the argument/parameter list. Upon returning from the function, the program should output the magnitude and angle of the vector (in degrees), as well as the value of the x component of the vector as shown below. The function should convert the angle to radians (using the same variable name) and compute the x component of the vector, returning the value to the main program.</p> <p>Call your program <i>lab10_p2.c</i>.</p>

	<p>The main function should have an output strongly resembling the below:</p> <pre>% a.out Enter an Angle: 45 Enter a Magnitude: 42 The x value of a vector with a magnitude of 42 and angle of 45 degrees is 29.69.</pre> <p>TIMEKEEPER: This exercise should take 15-25 minutes. Remember this is only a guideline.</p>
<p>3</p>	<p>Part III</p> <p>(Note: Remind the students that they learned about arrays in their studies of Fortran.) C (like most other high-level programming languages) also supports arrays. However, the implementation is slightly different than what you saw in Fortran. The most noticeable difference is the notation. In C the first element of an array is denoted as element 0. This will be the source of many, many errors—just a hint.</p> <p>Your task is to define a 10-element single subscripted integer array that is initialized (via the declaration statement) to contain the numbers 1 through 10 in elements 0 through 9, respectively. The program should be written in such a way that it will repeatedly prompt the user to enter a specific element number of the array to be displayed on the computer monitor and then output the value stored in the element in a manner similar to the example shown below. You should make use of the value returned by scanf() to control when input is to be terminated (recall: a carriage return followed by control-d should return a value of false from scanf).</p> <p>Name your file <i>lab10_p3.c</i>.</p> <p>The output should look remarkably like this:</p> <pre>% a.out Which element of the array would you like to have printed out? 2 The value of element 2 is 3 Which element of the array would you like to have printed out? 1 The value of element 1 is 2. . . . ctrl-d</pre>

	<p>TIMEKEEPER: This exercise should take between 15 and 20 minutes. This is just a guideline.</p>
<p>4</p>	<p>Part IV</p> <p><i>This part should be done with all members of the team working at one computer; the team recorder should serve as the key-board operator.</i></p> <p>The scope of a variable refers to the part of the program for which the variable is known or accessible. A variable is known throughout a block of code. Variables are only available within the block of code where the variable is declared. A variable is not carried through a function because it has separate declarations.</p> <p>The following code is legal. Run it within a main program and see what happens. Alter the code to more clearly express the intents of the programmer. PLEASE NOTE: The following code does not express the coding practices of the administration of the course. As a matter of fact, the code is considered very bad programming practice. Consider this a learning experience.</p> <p>Name your code <i>lab10_p4.c</i>.</p> <p>We suggest that you take your time to really see what happens when you alter this code. Add a function and play with variable names.</p> <pre> { int a = 2; printf("%d\n", a); { int a = 7; printf("%d\n", a); } printf("%d\n", ++a); } </pre> <p>TIMEKEEPER: This exercise should take between 10 and 15 minutes. This is only a guideline.</p>
<p>5</p>	<p>Part V</p> <p>The product of two matrices is represented as $[c] = [a][b]$, where the elements of $[c]$ are defined as:</p> $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$ <p>where n equals the column dimension of $[a]$ and the row dimension $[b]$. That is, the c_{ij} element is obtained by adding the product of individual elements from the i^{th} row of the first matrix, in this case $[a]$, by the j^{th} column of the second matrix $[b]$.</p>

	<p>According to this definition, multiplication of two matrices can only be performed if the first matrix has as many columns as the number of rows in the second matrix. Thus, if [a] is an m-by-n matrix, [b] could be and n-by-l matrix. For this case the resulting [c] matrix would have dimension of m-by-l. However, if [b] were a l-by-n matrix, the multiplication could not be performed.</p> <p>As a team, write a flowchart for an algorithm that will ask a user to enter all the elements for two real matrices (since this is an algorithm we use the term matrices, not arrays): a that is of order ($m \times n$) and b that is of order ($n \times l$). The algorithm, should then take the product of the two matrices, storing the result in a new matrix called c (which is of order ($m \times l$)), using the definition of matrix multiplication shown above. Develop a simple test case by hand to ensure that your algorithm works before you attempt to actually write a program.</p> <p>After verifying your algorithm, convert it to an appropriate C program. It is safe to assume that the input matrices will not be larger than 10 by 10, however a #define should be used to ensure that this value can be easily changed later. Name your file <i>lab10_p5.c</i></p> <p>TIMEKEEPER: This should take between 20 and 25 minutes. Turn in your team's flowchart and hand-test case to your instructor or TA.</p>
6	<p>Turn in the following files by (insert date) to: (insert name)</p> <ul style="list-style-type: none"> • <i>lab10_p1.c</i> • <i>lab10_p2.c</i> • <i>lab10_p3.c</i> • <i>lab10_p4.c</i> • <i>lab10_p5.c</i>
7	<p>Individual Check for Understanding</p> <p>Complete the handout.</p>

Handouts:

- [Individual Check for Understanding](#)

Individual Check for Understanding

This is a written check for understanding. Put your name as well as the names of the team members present, your team number, your team role and lab time on your paper.

What is the output of the following program?

```
#include <stdio.h>

#define N 5

int main()
{
    int    a[N];
    int l, sum = 0;

    for (l = 0; l < N; ++l)
        a[l] = 7 + l * l;
    for (l = 0; l < N; ++l)
        printf("a[%d] = %d \n ", l, a[l]);
    for (l = 0; l < N; ++l)
        sum += a[l];
    printf("\nsum = %d\n", sum);
    return 0;
}
```