**CPSC 1071 Lab 4 - Pointers**

The purpose of this Lab is to give you additional practice in working with pointers.

In this lab, you will:

* Observe the basic pointer operators.
* Master the mechanism for simple pointer manipulation.
* Use pointers to pass parameters by reference.
* Use pointers in conjunction with dynamic data.

**Background**

A *pointer* type is a simple data type that contains the address of a variable. A pointer provides a convenient means of passing arguments to functions and for referring to more complex data types such as structures. Pointers are also essential if you want to use dynamic data.

The specific value in a pointer is seldom important, as long as it contains the address of the variable that you want. You need to declare and initialize pointers just as you would other variables, but there are special operators that you need to use.

**Pointer Operators**

The following table shows the special characters used in C to declare and use pointers.

|  |  |  |
| --- | --- | --- |
| \* | dereference operator or indirection operator | This operator is used to declare a variable as a pointer. It is also used when you want to access the value pointed to by the pointer variable. |
| & | reference operator or address-of operator | This operator is used before a variable to indicate the address of that variable. |
| -> | member selection operator | This is used to refer to members of structures |

**Simple Pointer Use**

* Once a pointer is declared, the programmer must initialize the pointer before using it. Either make the pointer point to something or assign it the value *NULL*. It is dangerous to use a pointer that has not been initialized.
* Like regular variables, uninitialized pointers will not cause a compiler error, but using an uninitialized pointer could result in unpredictable and potentially disastrous outcomes.
* Until a pointer holds a valid address, it isn't useful.vim
* Notice the difference between memory address and the actual data value, which is stored at the memory address. Do not worry about the addresses, they are determined and provided by the system.
* We can:
  1. Access the content of a variable by using the variable name, e.g. given

*int num;*

we can access the content of *num* simply by using *num*. e.g.

*int y = num;* or   
*printf("%d\n", num);*

This is called direct access.

* 1. Access the content of a variable by using a pointer to the variable, e.g., given

*int \*ptr1 = &num;*

we can access the content of num by using *\*ptr1*. e.g.,

*int y = \*ptr1;* or   
*printf("%d\n", \*ptr1);*

This is called indirect access or indirection; it is also called dereferencing the pointer.

* To summarize, if a pointer named *ptr* of type *int* has been initialized to point to the variable named *var*, i.e.,
* int var;
* int \*ptr;
* ptr = &var;
* the following are true:
  1. *\*ptr* and *var* both refer to the content of *var* (i.e., whatever data value is stored there).
  2. *ptr* and *&var* refer to the address of *var* (*ptr* only holds the address of variable *var* not the actual data value).

So, a pointer name without the dereferencing/indirection operator (\*) accesses the pointer value itself, which is of course, the address.

**Preparing for today's lab::**

Create a lab4 directory and *cd* to this directory. Use the following command to copy the files for this lab:

lab1071copy 4

List your files (*ls*). Your directory should now contain the files:

*lab4.c*, *ptrlab.c*, and *worksheet.txt*.

**Task 1: Tracing Exercise**

What does the following code print? Step through it as if you were the machine, and write your answers in worksheet.txt.

This is sometimes referred to as "tracing" the code. You may find it helpful to draw diagrams with boxes for memory locations and arrows for pointers. DO NOT compile and execute the code.

int a = 42;

int b = 7;

int c = 999;

int \*t = &a;

int \*u = NULL;

printf("%d %d\n", a, \*t);

c = b;

u = t;

printf("%d %d\n", c, \*u);

a = 8;

b = 8;

printf("%d %d %d %d\n", b, c, \*t, \*u);

\*t = 235;

printf("%d %d %d %d %d\n", a, b, c, \*t, \*u);

**Task 2**

**DO THIS:**

1. Open the file, *ptrlab.c*, compile, and execute it. Copy the output and paste it in worksheet.txt
2. After executing the code, add more lines of code to do the following:
   1. Update *t* to point to *c*. Dereference pointer *t* to change the value of *c* to 555. Verify that it worked by adding a print statement to print *c*. Does this change any of the other values? (You may copy and paste the printf statement that prints all values). Write your answer in worksheet.txt. If the answer is yes, indicate the values that were changed and explain why.
   2. Change the value of *c* again using a direct assignment. Verify that the pointer *t* still points to the value by printing the result of dereferencing the pointer *t*.
3. **Question:** What would happen if you tried to execute the following code? Type your answer in *worksheet.txt*
4. int \*v = &t;

printf("%d\n", \*v);

Fix the problem.

**Using Pointers to Simulate Pass by Reference.**

The previous example was used to help you understand pointers. A more realistic example of pointer use is to simulate pass by reference in C.

C does not have reference parameters; however, when a function needs to pass back more than one value, pointers can be used to simulate reference parameters. In other words, you want to pass the addresses of the data to the function rather than the values of the data.

Let's look at both pass by value and pass by reference to make sure we understand the difference. Below is an example that demonstrates how parameters are passed by value.

/\* function prototype \*/

void convertInches(int totalInches, int yards, int feet, int inches);

int main()

{

int totalInches = 213;

int yards;

int feet;

int inches;

convertInches(totalInches, yards, feet, inches); // function call

printf("\n%d total inches = %d yards, %d feet, %d inches\n\n",

totalInches, yards, feet, inches);

return 0;

}

void convertInches(int totalInches, int yards, int feet, int inches)

{

yards = totalInches / 36;

totalInches = totalInches % 36;

feet = totalInches / 12;

inches = totalInches % 12;

}

This seems simple and direct; however, this code does not work as it should. The function computes yards, feet, and inches, but the values in *main()* are not changed. In order for yards, feet, and inches in *main()* to change, we need to use pass by pointer. To do this, we need to change the function prototype and header to accept pointers for the values that should be changed in *main()*.

We will also need to change the call to *convertInches()* to pass addresses, and change the body of *convertInches()* to use pointers.

**Task 3**

1. Using the editor of your choice, open lab4.c and make the necessary changes to the call to *convertInches()* in *main()*, and make the necessary changes to the prototype, header, and body of *convertInches()*.

Compile using

gcc -Wall -o lab4 lab4.c

Test using

./lab4

For total inches of 213, the program should print

213 total inches = 5 yards, 2 feet, 9 inches

**Documentation**

Before submitting your code, make sure that you have provided necessary header documentation in each file, and function documentation for the *convertInches()* function in lab4.c

**Submission**

After you have completed and tested your code and completed worksheet.txt, use [Handin](https://handin.cs.clemson.edu/) to submit *worksheet.txt, ptrlab.c,* and *lab4.c*