**Department of Computing Sciences**

**SUNY Brockport**

**CSC 219 (Programming in C)**

**Lab Exercises 3**

This lab provides an introduction to pointers in C.

In this lab you will be using a C program to answer questions about pointers in C.

You will need to first write code to declare two integer variables (i1 and i2) and two double variables (d1 and d2). Assign value 50 to variable ‘i2’. Assign value ’80.0’ to variable ‘d2’. Then:

1. Write code to *print out* the *addresses* (not the content – i.e., not the value) of the following variables: i1, i2, d1 and d2. Use printf() to do the printing out, with the appropriate embedded conversion specification – i.e., %d, %f, etc. – for printing addresses.
2. Declare 4 pointer variables (intptr1, intptr2, dubptr1, dubptr2), with the first two in this list being *pointers to integers*, and the next two being *pointers to double* values. Write code, as in (1) above, to print out the *addresses* (again, note – *addresses*, not *contents*) of these pointer variables.
3. Write code to record the addresses of the (simple) variables mentioned below in appropriate pointer variables. Then write code to print out the *contents* of these *pointer variables*. What you should actually do is:

i1 => Write code to hold its address in a pointer to integer variable, which is: ‘intptr1’

i2 => Write code to hold its address in a pointer to integer variable: ‘intptr2’

d1 => Write code to hold its address in a pointer to a double variable: ‘dubptr1’

d2 => Write code to hold its address in a pointer to a double variable: ‘dubprt2’

Then, write code to print out the *contents* of the pointer variables ‘inptr1’, ‘intptr2’, ‘dubptr1’ and ‘dubptr2’.

1. Write code to assign ‘intptr2’ to ‘intptr1’. Then write code to print out the *contents* (aka *value*) of both ‘intptr1’ and ‘intptr2’.
2. Write code to use typecasting to cast the type of pointer variable ‘intptr1’ to type (double \*) and assign this to ‘dubptr1’. Now, write code to record the *content* of pointer variable ‘dubptr1’ and also of pointer variable ‘intptr1’.
3. Write code to assign the value NULL to pointer variable ‘intptr1’ and then print out the *content* of ‘intptr1’.
4. **Dereferencing pointers:** Write code to dereference the pointer ‘intptr2’ and print the result of dereferencing. Use ‘printf()’ to print the result. When writing this printf(), think what the appropriate embedded conversion specification you should use to do this – i.e., should it be ‘%d’, ‘%f’, ‘%p’, etc.
5. Write code to assign the value 100 to the dereferencing of pointer ‘intptr2’ – i.e, to ‘\*intptr2’, with ‘\*intptr2’ appearing on the left hand side of the assignment operator, as discussed in class. After doing this assignment, dereference ‘intptr2’ again (as done in (7) above) and print out the result of this dereferencing.
6. Write code to add 1 (one) to the *content* of pointer variable ‘intptr2’ (i.e., increment the address held in ‘intptr2’ by 1). Then write code to print (using ‘printf()’, of course) the *content* of ‘intptr2’. Then, dereference ‘intptr2’ and write code to print out the result of this dereferencing – i.e., repeat what you did in (7) above.
7. Write code to subtract 1 (one) from the *content* of pointer variable ‘intptr2’ (i.e., decrement the address held in ‘intptr2’ by 1). This code should execute *after* the code in (9) does. Then write code to print (using ‘printf()’, of course) the *content* of ‘intptr2’. Then, dereference ‘intptr2’ and write code to print out the result of this dereferencing.
8. Write code to dereference the pointer ‘dubptr2’ and print the result of dereferencing. Use ‘printf()’ to print the result (this is *similar* to (7) above).
9. Write code to add 1 (one) to the *content* of pointer variable ‘dubptr2’ (i.e., increment the address held in ‘dubptr2’ by 1). Then write code to print (using ‘printf()’, of course) the *content* of ‘dubptr2’. Then, dereference ‘dubptr2’ and write code to print out the result of this dereferencing.
10. Write code to subtract 1 (one) from the *content* of pointer variable ‘dubptr2’ (i.e., decrement the address held in ‘dubptr2’ by 1). This code should execute *after* the code in (12) does. Then write code to print (using ‘printf()’, of course) the *content* of ‘dubptr2’. Then, dereference ‘dubptr2’ and write code to print out the result of this dereferencing.
11. Write code to set the content of pointer ‘intptr1’ to the address of simple variable ‘i1’. Write code that will enable you to see the answers to the following questions in the output of the program:

Does intptr1 == intptr2?

Does \*intptr1 == \* intptr2?

After writing this code, write code to set the content of ‘intptr1’ to the address of simple variable ‘i2’. Then, write the code again that will enable you to see the answers to the following questions in the output of the program:

Does intptr1 == intptr2?

Does \*intptr1 == \* intptr2?

1. The ‘malloc()’ function assigns new memory to a pointer variable. It returns type void \* so you must typecast it to the correct type. Declare a new pointer to type double, call it ‘pi\_ptr’. Then, write code to assign a block of memory to this pointer using malloc (sizeof (double)). Using the pointer, write appropriate code (remember, pointers can be dereferenced and used on both sides of an assignment statement) to assign the value of 3.1416 to this block of memory. Write code to see the following information in your output:

Value located (i.e., *the content of*) in variable ‘pi\_ptr’

Value obtained by dereferencing ‘pi\_ptr’ – i.e., the value of \*pi\_ptr

1. To deallocate the dynamic memory of a pointer, we use the free () function. This function requires us to typecast our pointer to type void \*. The syntax is: free ((void \*) pi\_ptr); Now, first write code to store the content of variable ‘pi\_ptr’ in another pointer variable of the same type called ‘pi\_ptr\_2’. Deallocate the memory for ‘pi\_ptr’ and reallocate it again using ‘malloc()’. Then, write code to print the content of the pointer variable ‘pi\_ptr’ again. Next, write code to see if the current content of ‘pi\_ptr’ the same as it was in the previous question – i.e., is the current content of ‘pi\_ptr’ the same as the content of (i.e., the value held) in ‘pi\_ptr\_2’?
2. After writing all the above code, compiling and running your program, and getting its output, and saving the output somewhere suitable where you can copy paste it into (say) a Word file you can print and submit, edit your code by writing some new code at the end of the main program in your ‘.c’ file. This code should simply do one small thing – assign the content of ‘intptr1’ to ‘dubptr1’ without doing the type casting mentioned in question (5) above. Compile and run your program and see what happens. Copy paste the result of your compilation (if it shows anything significant) and/or the output of your code when executing this statement (if the output shows anything significant) in the Word file (say) you will use to print your code and output from.

Submit all the code you wrote, *followed by* the output of your code.