CS 305 Prelab 2: Pointers and arrays
Fall 2019

Name: _____(done individually) score: _____ / 30

This prelab is due on Friday, NLT beginning of the class.

Readings (do before the lab)

Read the following in GNU C tutorial: pages 89 – 107 and in the textbook sections 2.1 – 2.8 and 3.3

Exercises (do before the lab)

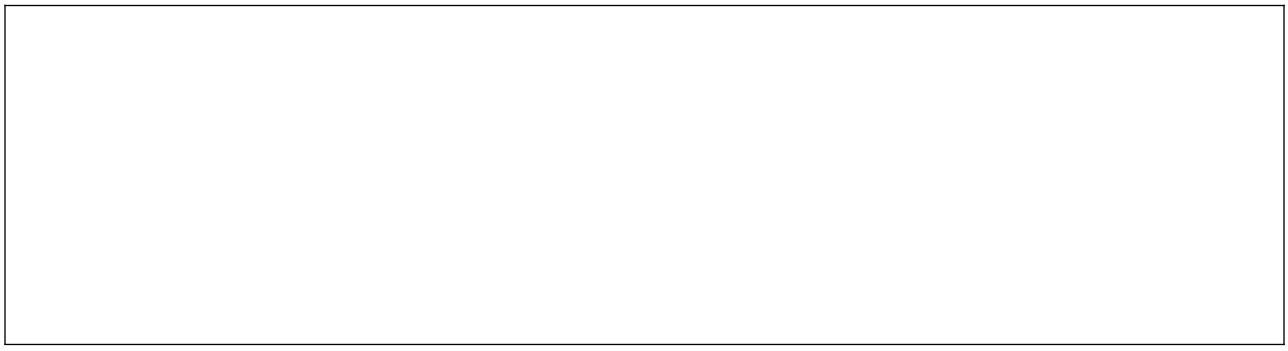
The comparison operators, ==, !=, <, >, <=, and >= can be used with pointers. The equality comparisons (`==` and `!=`) tell whether the pointers are the same pointer.

The non-equality operators (<, >, <=, and >=) tell whether the result of subtracting the two pointers is less than, equal to, or greater than zero. They should be used only when both pointers refer to elements in (or just beyond the end of) the same array. Thus, `p2 > p1` evaluates to true if `(p2 - p1)` is greater than 0.

1. (10 points) Examine the following code-fragment that compares pointers. What is printed?

```
int a[20];
int *p1 = a;
int *p2 = &a[0];
int *p3 = a + 5;
if (p1 == p2) printf("statement 1\n");
p1 += 3;
if (p1 > p3) printf("statement 2\n");
p1 += 3;
if (p1 > p3) printf("statement 3\n");
p1 += 3;
if (p1 > p3) printf("statement 4\n");
p2 += 3;
p3 -= 3;
if (p2 < p3) printf("statement 5\n");
```

Printed to screen:



2. (10 points) One of the dangers of using pointers (including pointer parameters in functions) is that it makes it somewhat more difficult to reason about programs. Consider the following code-snippet:

```
/* put your code here */




/* end your code here */

if (*a > 0) {
  *b = *b - 1;
}

else {
  *a = 10;
}

if (*a <= 0) {
  printf("this should never happen\n");
}
```

Determine a scenario involving pointers to variable(s) where "this should never happen" ends up getting printed out, even though it appears on the surface that `a` will always be a positive number after the first if-statement. Add code above the code that makes “this should never happen” print to the screen. Hint: more than one pointer can point to the same variable. Hint: you can write and test code when working on these prelab questions.

Having multiple pointers referencing the same data is called aliasing.

3. Background Information on pointer arithmetic

There are three kinds of pointer-arithmetic operations. These are generally valid only when the pointer operands are pointers to an element of an array.

1. `ptrVal + intVal` (or `intVal + ptrVal`)
   If ptrVal is pointing to the N^{th} element of an array, then the expression yields a pointer to the $(N+intVal)^{th}$ element of the array. This yields a valid pointer only if the pointer stays within the bounds of the array, or is a pointer just beyond the last element of the array. In the latter case, the pointer may be used in comparisons, but not for indirections. Operators that are abbreviations for '+' (e.g., +=, ++) may also be used here.
2. `ptrVal - intVal`
   If ptrVal is pointing to the N^{th} element of an array, then the expression yields a pointer to the $(N-intVal)^{th}$ element of the array. This is equivalent to ptrVal + (-intVal). Operators that are abbreviations for '-' (e.g., -=, --) may also be used here.
3. `ptrVal1 - ptrVal2`
   Here, the pointers should point to elements in the same array (or just beyond the end of it). The result is the integer value that denotes the number of slots in the array by which the pointers differ. This is generally not the same as the

number of bytes by which they differ, because most array-elements are more than one byte in size.

One mistake that a programmer might make is to ignore the restriction that the two pointers in pointer-pointer subtraction need to reference the **same** array. Most C compilers will allow this to occur, but will give "non-deterministic" results for the answer. The result will typically depend on how the compiler happens to allocate the arrays in memory.

1. Download the file `ptr_diff.c` from moodle. Save `ptr_diff.c` into your cs305 folder on the P: drive. Open Mobaxterm and navigate to your cs305 folder. Use emacs to open the file. Determine what will be printed when running the program.

We can compile programs using gcc with several different options.

```
gcc -o ptr_diff1 ptr_diff.c
gcc -O -o ptr_diff2 ptr_diff.c
gcc -O4 -o ptr_diff3 ptr_diff.c
gcc -g -o ptr_diff4 ptr_diff.c
```

- In the options above `-o` (the little o) means to direct the output the executable named `ptr_diffX`.
- In the options above `-O` (the big O, as the letter O) means to compile with optimization. The `-O4` means compile with high optimization.
- In the options above `-g` means to compile with debugging.

(6 points) Build and execute these 4 versions; report which of the 6 expressions give consistent results and which give inconsistent results.

Expression	**Check if consistent**	**Check if inconsistent**
`ptr2-ptr1`		
`ptr3-ptr1`		
`ptr4-ptr1`		
`ptr3-ptr2`		
`ptr4-ptr2`		
`ptr4-ptr3`		

(4 points) Why are the consistent expressions consistent across the four versions?

Bring to lab

- GNU C tutorial.
- A pencil/pen and scratch paper.
- Your completed prelab, done on this sheet.