CS 305 Lab 2: pointers and arrays
Spring 2019
VanDeGrift

This lab introduces you to pointers, arrays, malloc and free.
This lab has a total of 100 possible points (30 points from pre-lab and 70 points from lab).

Objectives

Upon completion of the laboratory, you will be able to do the following:

- Explain the results of pointer arithmetic and comparisons (prelab).
- Access array elements using pointers (lab).
- Use `malloc` or `calloc` to create dynamically sized arrays (lab).
- Free memory (lab).

Part 1: Log in (choose one person of the pair to log in)

1. Follow the procedure you used in lab 1 to log into the computer.

2. Go into your cs305 directory and make a new directory called `lab2` in your `cs305` directory.
```
cd cs305
mkdir lab2
```

3. Get the lab2 files. Download `starter.zip` from moodle to your lab2 folder. On Mobaxterm navigate to your lab2 folder. In the terminal window, type:
```
ls
```

Hopefully, you can see your zip file there. To unzip it, type:
```
unzip starter.zip
```

List your directory contents again. You should see a new directory called `starter` in addition to the zip file.
```
ls
```

Part 2: Pointers and arrays

In C, there is a close relationship between pointers and arrays.

First, with just a couple of exceptions, any time an array name is given in a program, it is converted (by the compiler) to be a pointer to the first element of the array. The exceptions are when the array is the immediate operand of the `sizeof` function or the "create-pointer-to" (unary `&`) operator. The semantics of the `[]` operator in `a[i]` mean `*((a)+(i))`:

- `a` is converted to be a pointer to its first element,
- `i` is added to this pointer, giving a pointer to the element in the ith position of `a`, and
- the `*` operator makes the reference.

One of the anomalies of C is that the semantics of `[]` allows `i[a]` to be used as an alternative to `a[i]`. (You could experiment with this: instead of `a[0]`, write `0[a]` when you want to access the first element in array a. Just because you can do this **does not mean you should** in your code…Tammy does not recommend using the `0[a]` form.)

Second, any time a function's formal parameter is given as an array type, it is converted (by the compiler) to be a pointer type: "array of int" is converted to "pointer to int"; "array of pointer to char" is converted to "pointer to pointer to char".

1. Given this knowledge, examine the program `array.c`. Compile and run the program. Explain why:
    - no type-check errors are reported for any of the three declarations of the function `abc`, even though the parameter declarations appear to be inconsistent. The “`extern`” keyword declares the type for the function `abc`, while the actual function definition is defined once below the three declarations.

Compile and run the program:

```
gcc –o array array.c
./array
```

2. Why does the array seem to change size when the function is called?

Part 3: Pointers accessing elements out-of-bounds (switch driver)

C does not do bounds-checking on arrays for you, so it is possible to use pointers to access memory that ***does not store*** contents of the array. This is a bit scary, so you should be **vigilant** in writing C code by doing bounds-checking.

Examine the program `bad_pointers.c`. This program creates two arrays (one of size 4 and one of size 8) and creates pointers to each array. Compile and run the code and explain how you think the memory for array `a` and array `b` are allocated given the printing you see. Draw a picture of the memory layout.

Add to the bottom of the main function:

- Print the values of `a[0]` through `a[25]`. (Note that the array `a` really contains 8 integers. The variable i is already declared in the file.)

Does printing the values confirm your memory layout above? YES NO

What are the second half of the 25 values coming from?
____

Checkpoint 1 [25 points]: Show your answers to the above questions and execute your code for your lab instructor/assistant.

Part 4: Allocate an array on the heap (dynamically-sized array) (switch driver)

Open and examine the file `array_alloc.c`.

The *main* function in *array_alloc.c*:

- prompts the user for a non-negative integer; re-prompts if necessary
- calls the *get_array* function with the given argument, which returns (a pointer to) a dynamically-allocated array. If the result of *get_array* is non-*null*, the *main* function will assume that it is to an array of *double* whose length is the integer argument.
- prints the values in the array by calling the *printArray* function
- frees the memory allocated for the_array; note that you should also free memory that is allocated on the heap when your program is done using that memory

As implemented, the *get_array* function simply returns *null*. Modify it so that it:

- checks the argument for being negative. If it is negative, it is treated as if the argument was 0
- allocates an array of *doubles* of the given length (hint: use the `calloc` function or `malloc` function to do this)
- initializes the array so that for all values *n* in its range, the value at index *n* of the array is `n * 10.0`.

Compile and run your program.

Checkpoint 2 [20 points]: Show your modified array_alloc.c program to your lab instructor/assistant. Demonstrate its execution.

Part 5: Allocate a 2-dimensional array on the heap (switch driver)

1. Open and inspect the file `2darray.c.`

The *main* function in 2darray.c:

- prompts the user for the height and width of the 2D array
- calls the *make2Darray* function to create a 2D array of `int`s with that size and returns a pointer to the dynamically-allocated array
- prints the values in the array to the screen using *print2Darray*

As implemented, the *make2Darray* function allocates memory to store the 2D array. Inspect this function to see how the allocation and freeing of memory works.

Why is the type of `a (int **)`?____

Why is the type of `a[i] (int *)`?____

As implemented, the *free2Darray* function frees the memory for the 2D array. Inspect this function to see how the array is "freed": first each row is "freed" and then the array is "freed".

2. Compile and run the program.

3. Add code to the function *make2Darray* to do the following:

- Initialize each element in the 2D array with its row number. For example, a height 4 and width 3 array would have the values:

```
0       0       0
1       1       1
2       2       2
3       3       3
```

Checkpoint 3 [25 points]: Show your answers above and your modified 2darray.c program to your instructor or lab assistant. Demonstrate its execution.

Part 6: Finish up

Close any emacs windows. Close Mobaxterm and log off the PC.

If any checkpoints are not finished, they are due in one week. You may submit screenshots and printed files of code to complete the checkpoints. Be sure your name is on the submitted work.