Pointer Review Homework

Due: April 2, 2020

Listed below are several programs, each of which have one or more questions that you need to answer. If the question is asking you to write code you are to write the code within the program. Any questions not asking for code should be answered within this document.

1.

advPtrHWPart1.c

Complete the code in the c file.

Discuss some interesting things you have learned about pointers in this exercise.

**The most interesting point of this exercise was proving that the pointer “ptr” was in fact one nibble in memory behind the array. I did not realize that &arr contained an address 5 times larger than an integer.**

2.

registerEx1.c

When ran, this program will cause an error. Explain why.

RegisterEx1.c throws an error because the pointer “a” is set to the memory address of the register int variable “i”. A register is not stored in memory, which is where the pointer “a” is trying to access.

3.

registerEx3.c

When ran, this program will cause an error. Explain why.

This program throws an error because register variables cannot be used with static.

4.

registerEx4.c

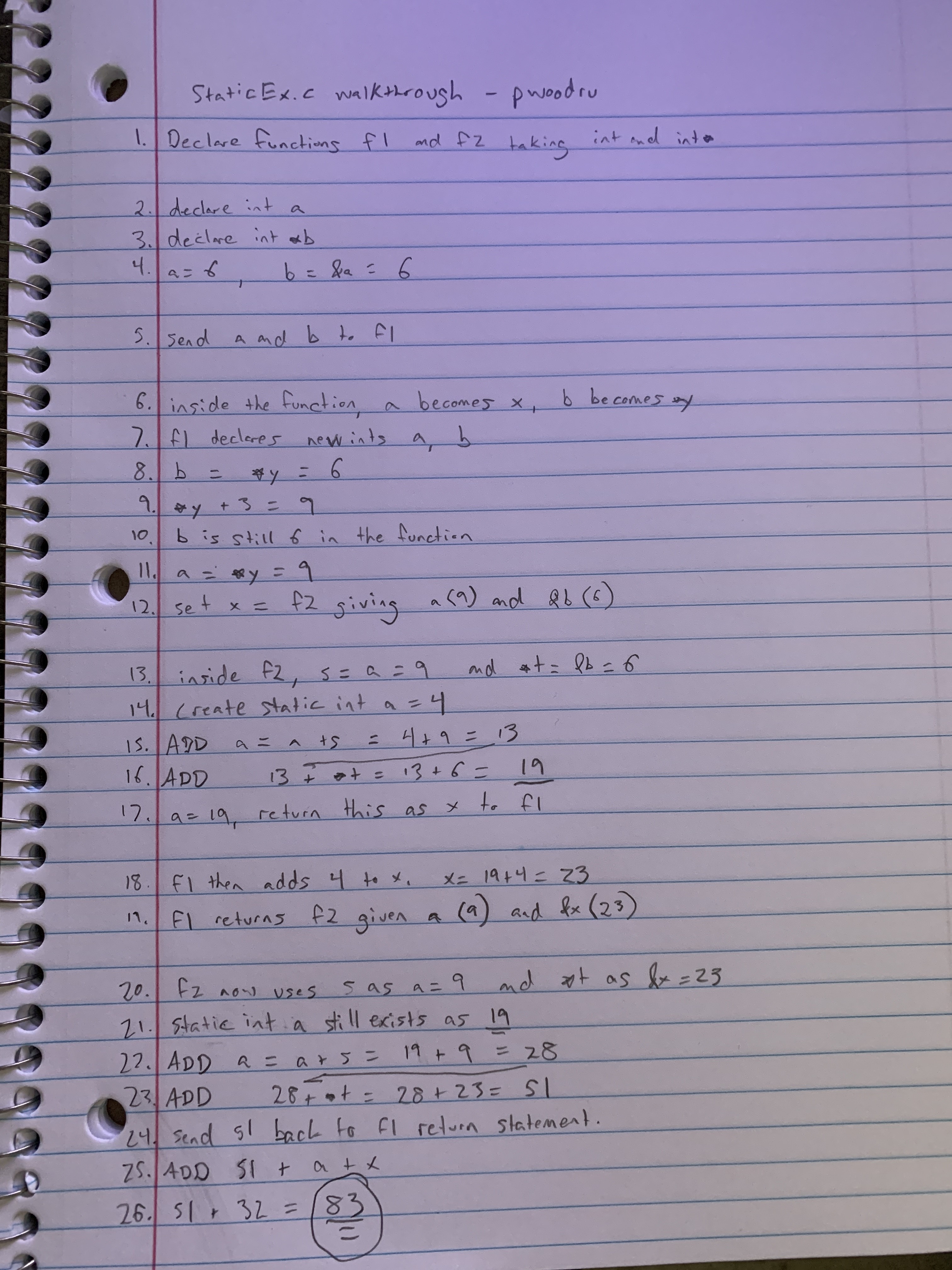
When ran, this program will cause an error. Explain why.

This program will not work because registers can only be used in a local space. The main function attempts to print a register variable that was declared outside main in the global space.

5.

staticEx.c

Trace through this program on paper. You must show the state of the variables at each step and a short explanation. You will need to convince me you actually walked through this and did not simply run it to get the answer.



6.

AdvPtrHW2.c

Run this program, then trace through the program on paper. Explain what each print statement is doing. Illustrate this by drawing out the memory. A big part of this is understanding, double pointers, dereferencing, and order of precedence rules. This one is somewhat difficult. You will need to convince me you know what is happening.

First print statement: \*\*++cpp

\*\*\*cpp points to the beginning of cp. The ++ tells it to access the second element of cp instead of the first. This then accesses the second element which is c+2. The statement then prints the element at c+2 in the original array of strings, printing “Two”.

Second print statement: \*--\*++cpp+3

This one was more confusing but I believe I understand it. The first portion “\*--” is telling the pointer to access the cp array – 1, which would wrap around to the end and cause it to access the element c+0, or the first element of c. The second portion is instructing the pointer to only print the characters after 3 letters into the element. Therefore the result is “loWorld”. This cut off the first three characters “Hel” which would print if the instruction removed the “+3”.

Third print statement: \*cpp[-2]+3

The first portion of the instruction, “\*cpp[-2] uses the \* to access the last element of cp, which results in accessing the c+0 element of the original array. The [-2] then instructs it to move back two elements to the “c+3”. This element reads “Three”. The final portion of instruction “+3” says to skip the first three characters and read the rest, resulting in printing “ee”, skipping “Thr”.

Fourth print statement: cpp[-1][-1]+1

Because cpp requires two dereferences to access elements, the subscripts [-1][-1] work together as the dereferencing. The first [-1] instructs it to access “c+2” element inside cp, and then the second [-1] decreases that one to the “c+1” element, which in turn accesses the second element of c, or “One”. Again, the +1 on the end of the instruction tells the print to ignore the first character and print the rest. The result is printing “ne” and skipping “O”.