1. Read stack.c, which should be in this directory. What is it intended to do? What would the output be if the program worked as expected?

It intends to initialize the data pointed to by a pointer (array in main()), by using a function, named foo. Each value in the array is set to 42 by the function.

Another function bar, which does not serve any purpose, creates an array within the function.

Output:

Address of array 1

Address of array 2

42

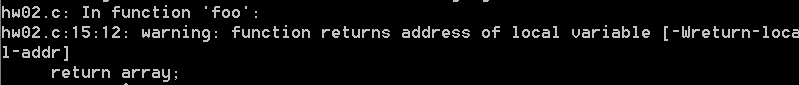
42

42

42

42

1. Compile it. Do you get a warning? What does it mean?



It means that the function attempts to return an address (which is declared within the function).

Memory of the array might be deallocated since the function ends (destructor?)

1. Run it. What happens? Can you explain it? You might find it helpful to draw a stack diagram.

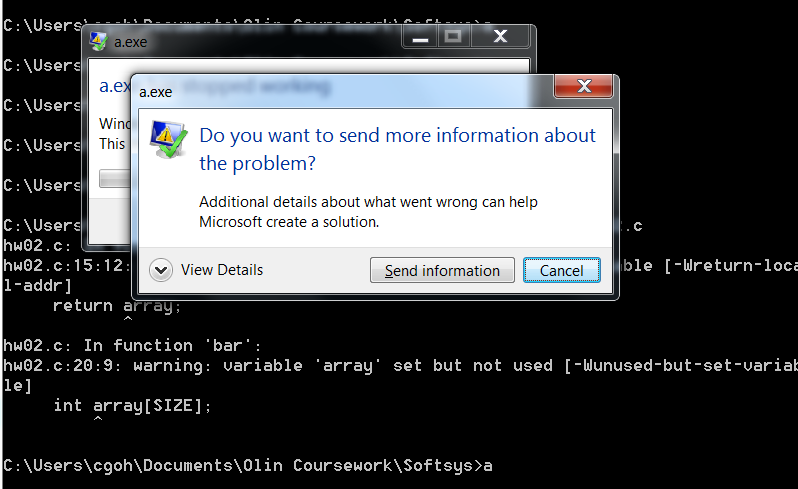
2 hexadecimal values are printed. (Likely to be from within foo, and bar). The address of the array allocated within the functions.

The first function, foo, first declare an array, which allocates SIZE(5) size of memory for the array. The memory is deallocated once it exits the function since it not of static data type.

When the 2nd function, bar, attempts to allocate memory for a new array within the function, it is allocated the same address as in foo, since the array in foo does not exist anymore (deallocated).

1. Comment out the print statements in foo() and bar() and run it again. What happens now?

It does not print out any value, it stops running. Most likely there might be segmentation fault. If the function foo, passing an address which is deallocated, it might be reading into some random, or system memory which is not intended for the pointer to read.



1. Add comments to the code to explain these experiments and the results, then check the code in to the repo.

Uploaded as stack.c

Moral: Don't return pointers to stack allocated data!