Albert Owusu-Asare & Helen Dougherty

A Review of C

Part A

1. The code will likely crash because "char *temp" isn't directed at anything or place in memory.

2. A buffer overflow occurs because it goes over the buffer boundary for temp. Because temp has nine spaces, the longest a name can be is eight characters to allow for the boundary.

3. This program crashes because *buffer cannot be rewritten from "Hello" to "World." The char * is immutable.

4. Technically yes, but it would be simpler to just point buffer to b. The name of the array implicitly points to the first element in the array.

5. The output will be "program1 5 abc". In the print statement, *argv points to argv[0], the string "program1"; *(argv + 1) points to "-n5", and the + 2 on the end advances two spaces inside that array, giving "5"; and *(argv+ 2) points to "abc".

6. myfunc returns a pointer to buffer, which is only defined within myfunc; once the function completes, buffer essentially disappears and the rest of the program has nothing definite to which to refer.

7. We believe it will crash instead of print infinitely because 'iter' is incremented once for each run through the while loop in main, and eventually it will reach the 32-bit limit of 2^31 -1 and not be able to iterate any longer.

8. There is no 'break' after the 'y' case hence control flows to the 'default' case as well hence printing both messages for the 'y' case and the 'default' case.

Part B:Fixing the bugs

1. The function "strcmp" returns " < 0" or "> 0" when one of the strings is less than or greater than the other. It also "0" when the strings are equal. Hence, we want the 'if statement'to be true only when they are equal and false for everything else. Anything that is not "false or 0 " is truish so to ensure false status for the ">0" and "<0" returns we must invoke the '!' operator. Notice that the use of '!' makes the statement true when the strings are equal. Thus '!0' = 1.

2. The length macro works by finding the total byte size of a certain structure and divides it by the number of bytes a pointer to that strucuture.(Normally the number of bytes of a pointer to a type is equal to the number of bytes to the type itself.eg int * = 4bytes , int = 4bytes.) Hence, by doing the division we have an idea of exactly how many of that type exists in the structure and therefore the length.

For a dynamically allocated array this macro would not work because we do not know exactly how the structure(array) is relayed in memory, we just happen to know of a way to get to that structure(the pointer to the array).Because we dont know much information about the array itself, we dont know its total amount of bytes occupied hence we cant tell the size.

3. There is no 'break' after the statement in case 'h'. As a result, each time the '-h' option is used the statement under case 'h' is executed, control is transfered to the default case, and then the statement under defaut is exectuted.

Fix: add 'break' after the case 'h' and 'default' executions.

4. 'entry_count' is never 0 because there will always be at least one word in the command line, the program name used to call the program. To fix this, we incremented each of the cases by 1 in the if statements testing for the presence of other arguments.

5. We altered the loop so it printed from element 0 to element entry_count, instead of the original way which was flipped.

6. See main.c code.