

12. Since strlen() returns the length excluding the NULL character, len = 6. That means that str[6], which is where the current NULL character is, is changed to '\n', and then str[len + 1] which is out of the array, is set to 0. Accessing out of the bounds of an array is not allowed because other variables could be stored there, and this could be potentially a serious problem which will cause strange errors elsewhere.

12. It is possible that the program will segfault if the byte after the initial terminating character is in the wrong segment (marked readonly or something), but the program will likely continue without raising the error and print "a dog\n". However, the change to the value one after the end of the array, could cause problems elsewhere in the program as explained in the answer to the first part of Question 12

---------------------------------------------------------------------------------------------------------

Given the following code:

{{{

int q;

int \*p;

q = 24;

\*p = 42;

q = \*p;

printf("The value of q is %d\n",q);

}}}

What is the main problem you would raise in reviewing the above code?

What result would you expect from compiling/running the code and why?

11. It trys to asign a value to what p points to before p has been assigned an address to point to. p, uninitialised will likely point to some random address in memory, and changing the value at this address could caused undefined/strange behaviour in the program or other programs.

11. It will likely segfault since it will be accessing and assigning a value to memory which it probably isn't allowed to (likely in a separate segment which is marked as read only)

// int sum = 0, store at 251

10 0 //put 0 in R0

12 251 //save R0 to add. 251

// int current = 200 store at 252

10 200 //put 200 in R0

12 252 //save to add. 252

// while (current != 251) {

14 252 //swap current into R0

12 252 //save it back to its address

11 251 //put 251 into R1

6 //R0 = R0 - R1, if 0 then equal

9 36 //if equal, then break from loop

5 // restore current in R0

// sum += value at current

12 23 //save current to instruction to put thing at address into R1

14 251 // swap sum into R0

15 0 //swap value at current into R1, 0 will be current by time executed

5 //add value at addr. curr. to sum

12 251 //save sum back to address

// current++;

14 252 //swap current into R0

1 //add 1 to current

12 252 //save current back to its address

//}

8 8 9 8 //jump back to start of loop

//print sum

14 251 //swap sum into R0

7 //print R0

0 //return 0