

Ideal Answers to Homework #1:

Example Homework Question:	Ideal Answer:
<pre>void reset(int *x) { x = &0; } void main() { x = 1; reset(&x); printf("x is now 0.\n"); }</pre>	<p>(a) This code attempts to reset the value of a variable to 0 by passing the variable by reference to a function reset().</p> <p>(b) This code does not work correctly for two reasons:</p> <ol style="list-style-type: none"> 1) the reset() function sets the memory address of the storage of the pointer passed into reset() equal to NULL rather than setting the value of the pointer passed into reset() to 0 2) the main() function makes use of the variable `x` without declaring it first. <p>(c) The code may be corrected by the declaring `x` to be an int in the main() function and by modifying reset() to set the value of x equal to zero rather than the memory address of x equal to NULL.</p>
Example Homework Question:	Ideal Answer:
<pre>char c = "a";</pre>	<p>(a) This code attempts to assign the character 'a' to the char `c`.</p> <p>(b) The code does not work correctly because the character 'a' is assigned by "double quotes" resulting in C returning a (char *) rather than a char.</p> <p>(c) The code may be corrected by changing the double quotes around the character 'a' to single quotes, resulting in:</p> <pre>char c = 'a';</pre>
Example Homework Question:	Ideal Answer:
<pre>char *str = "abc"; strcat(str, "def");</pre>	<p>(a) This code attempts to create a C-string `str` with the value "abcdef".</p> <p>(b) This code does not work because `str` contains a pointer to memory that contains only the literal "abc" (represented in memory as "abc\0"). Concatenating to the C-string would exceed the allocated amount of memory and would (likely) result in a segmentation fault.</p> <p>(c) This code may be corrected by allocating seven bytes of heap space so that the entire string may be stored. The modified code may be:</p> <pre>char *str = (char *)malloc((char)*7); strcat(str, "abcdef");</pre>

Static, Heap, and Stack Memory Allocations

Remember, in CS 241, we focused on three different methods of memory allocation done in C:

- **Static:** Only allocated with the `static` keyword, initialized once, and never freed.
- **Heap:** Only allocated when `malloc()` or other memory-allocating function is called, freed only when `free()` is called, and is able to be referenced from anywhere within your code.
- **Stack:** Allocated by variables declared in your C code, initialized within the function that they're declared in, and freed when the function finishes executing.

How many bytes of each type of memory allocation are done in the following code segments? Assume no padding and that `sizeof(char) = 1`, `sizeof(int) = 4`, `sizeof(double) = 8`, and `sizeof(void *) = 8`.

Example #1	
<pre>char* a, b; int i, j = 2; for (i=0; i < j; i++) b = (char)('a' + i);</pre>	<p>Static Memory: _____ bytes</p> <p>Heap Memory: _____ bytes</p> <p>Stack Memory: _____ bytes</p>

Example #2	
<pre>double e; double *f = (double *)malloc(24); static double h, i; int j; int *k=(int *)malloc(4 * sizeof(int)); static int l, m;</pre>	<p>Static Memory: _____ bytes</p> <p>Heap Memory: _____ bytes</p> <p>Stack Memory: _____ bytes</p>

Example #3	
<pre>int i, j=10; for (i = 0; i < j; i++) { static int s = i; if (s == i) malloc(4 * sizeof(double)); else malloc(2 * sizeof(char *)); }</pre>	<p>Static Memory: _____ bytes</p> <p>Heap Memory: _____ bytes</p> <p>Stack Memory: _____ bytes</p>

Example #4	
<pre>char *a = "the world"; char *b = "go round"; char *c = (char *)malloc(100); sprintf(c, "%s %s", a, b);</pre>	<p>Static Memory: _____ bytes</p> <p>Heap Memory: _____ bytes</p> <p>Stack Memory: _____ bytes</p>