Pointers

- All data is in memory. That means that each data has a memory address that maps to it. Pointers are variables that contain the address values.
- You can dereference a pointer (put a * in front of it) to obtain the data in the given address.
- When you initialize a pointer, you only make room in the memory for the space to hold the pointer, NOT the space to hold the data it's pointing to!
- You can obtain the address of any data by putting & in front of the variable name.

Write functions that achieve the given tasks. Not all of them necessarily have a solution.

1. Swaps the value of two ints declared in main.

2. Increments the value of an int declared in main by one.

3. Returns the number of bytes in the input string. Does not use strlen.

4. Returns the number of elements in the input array ARR of ints.

Memory Management in C

5. In which memory sections (code, static, heap, stack) do the following reside?

```
#define val 16
char arr[] = "foo";
void foo(int arg){
        char *str = (char *) malloc (val);
   char *ptr = arr;
}
```

arg _____ arr _____ str_____*str _____ val _____

6. What are two reasons we might need to use malloc in a C program?

7. What is wrong with the C code below?

```
int* ptr = malloc(4 * sizeof(int));
if(extra_large) {
        ptr = malloc(10 * sizeof(int));
}
return ptr;
```

MIPS

- 32 Registers, $16~$17 => $s0~$s7, $8~$15 => $t0~$t7. $0 is reserved for the value 0, and cannot be overwritten with other values.
  THERE ARE NO VARIABLES IN MIPS, JUST REGISTERS.
- MIPS Instruction Format: Operand Dest, Src1, Src2 (In most cases)
- Some example MIPS Instructions:

Instruction	Syntax	Example
add	`add dest, src0, src1`	`add $s0, $s1, $s2`
addi	`addi dest, src0, immediate`	`addi $s0, $s1, 12`
sll / srl	`sll dest, src, immediate`	`sll $t0, $s0, 4`
lw / lb	`lw dest, offset(base addr)`	`lw $t0, 4($s0)`
sw / sb	`sw src, offset(base addr)`	`sw $t0, 4($s0)`

C	MIPS

`// $s0 -> a (use $s0 for a),`
`// $s1 -> b`
`// $s2 -> c, $s3 -> z`
 	
`int a=4, b=5, c=6, z;`
`z = a+b+c+10;`	

`// $s0 -> int *p = (int *)malloc`
`//  (3*sizeof(int));`
`// $s1 -> a`
`p[0] = 0;`
`int a = 2;`
`p[1] = a;`
`p[a] = a;`	