Memory Management in C

1) 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 _____

2) What are two reasons we might need to use malloc in a C program?

3) What is wrong with the C code below:

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

4) For the singly linked list implementation below, fill out `free_ll`, which frees all of the memory allocated for the linked list.

```
typedef struct {
      ll_node* next;
      int element;
} ll_node;

void free_ll(ll_node* list) {




}
```

5) Write `append_ll` which appends a new node with element x. Assume list is not null. Return success.

```
int append_ll(ll_node* list, int x) {
```

6) You have two `ll_node` pointers. Write a function that swaps the two pointers.

MIPS cheat sheet (including instructions you might not have seen yet)

Instruction	Syntax	Example
add	`add dest, src0, src1`	`add $s0, $s1, $s2`
sub	`sub dest, src0, src1`	`sub $s0, $s1, $s2`
addi	`addi dest, src0, immediate`	`addi $s0, $s1, 12`
sll / srl	`sll dest, src, immediate`	`sll $t0, 4($s0)`
lw / lb	`lw dest, offset(base addr)`	`lw $t0, 4($s0)`
sw / sb	`sw src, offset(base addr)`	`sw $t0, 4($s0)`
bne	`bne src0, src1, branchAddr`	`bne $t0, $t1, notEq`
beq	`beq src0, src1, branchAddr`	`bne $t0, $t1, Eq`
j	`j jumpAddr`	`j jumpWhenDone`
jr	`jr reg`	`jr $ra`

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;`	

`// $s0 -> a, $s1 -> b`
`int a = 5, b = 10;`
`if (a + a == b) {`
`  a = 0;`
`} else {`
`  b = a - 1;`
`}`	

`What does this do (in English)?`	`  addi $s0, $0, 0`
	`  addi $s1, $0, 1`
	`  addi $t0, $0, 30`
	`loop: beq $s0, $t0, done`
	`  add $s1, $s1, $s1`
	`  addi $s0, $s0, 1`
	`  j loop`
	`done: # done!`