CS61c Spring 2015 Discussion 2 – C Memory Management & MIPS

1 C Memory Management

1. In which memory sections (**C**ODE, S**T**ATIC, **H**EAP, **S**TACK) do the following reside?

```
#define C 2
const int val = 16;
char arr[] = "foo";
void foo(int arg){
    char *str = (char *) malloc (C*val);
    char *ptr = arr;
}
```

```
arg [         ]     str  [         ]

arr [         ]     *str [         ]

val [         ]     C    [         ]
```

2. What is wrong with the C code below?

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

3. Write code to prepend (add to the start) to a linked list, and to free/empty the entire list.
`struct ll_node { struct ll_node* next; int value; }`

`free_ll(struct ll_node** list)`	`prepend(struct ll_node** list, int value)`

Note: `list` *points to the first element of the list, or to* `NULL` *if the list is empty.*

2 MIPS Intro

1. Assume we have an array in memory that contains `int* arr = {1,2,3,4,5,6,0}`. Let the value of `arr` be a multiple of 4 and stored in register `$s0`. What do the following programs do?

```
a) lw $t0, 12($s0)
   add $t1, $t0, $s0
   sw $t0, 4($t1)

b) addiu $s1, $s0, 27
   lh $t0, -3($s1)

c) addiu $s1, $s0, 24
   lh $t0$, -3($s1)
```

```
d) addiu $t0, $0, 12
   sw $t0, 6($s0)

e) addiu $t0, $0, 8
   sw $t0, -4($s0)

f) addiu $s1, $s0, 10
   addiu $t0, $0, 6
   sw $t0, 2($s1)
```

2. In 1), what other instructions could be used in place of each load/store without alignment errors?

3. What are the instructions to branch to `label:` on each of the following conditions?

`$s0 < $s1`	`$s0 <= $s1`	`$s0 > 1`	`$s0 >= 1`

3 Translating between C and MIPS

Translate between the C and MIPS code. You may want to use the MIPS Green Sheet as a reference. In all of the C examples, we show you how the different variables map to registers – you don't have to worry about the stack or any memory-related issues.

C	MIPS

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

// $s0 -> int * p = intArr;
// $s1 -> a;
*p = 0;
int a = 2;
p[1] = p[a] = a;	

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

	addiu $s0, $0, 0
	    addiu $s1, $0, 1
	    addiu $t0, $0, 30
	loop:
	    beq $s0, $t0, exit
	    addu $s1, $s1, $s1
	    addiu $s0, $s0, 1
	    j loop
	exit:

// $a0 -> n, $v0 -> sum
int sum;
for(sum=0;n>0;sum+=n--);	