CS61c Fall 2016 Discussion 2 – C Memory Management & MIPS

1 C Memory Management

1. Match the items on the left with the memory segment in which they are stored. Answers may be used more than once, and more than one answer may be required.

1. Static variables B	A. Code
2. Local variables D	B. Static
3. Global variables B	C. Heap
4. Constants A, B	D. Stack
5. Machine Instructions A	
6. Data B	
7. malloc() C	
8. String Literals B	
9. Characters A, B, C, D	

2. What is wrong with the C code below?

```
int* ptr = malloc(4 * sizeof(int));
if(extra_large) ptr = malloc(10 * sizeof(int)); // Memory leak if extra_large is true
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)`

```
if(*list) {
    free_ll(&((*list)->next));
    free(*list);
}

*list = NULL;
```

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

```
struct ll_node* item = (struct ll_node*)
    malloc(sizeof(struct ll_node));
item->value = value;
item->next = *list;
*list = item;
```

Note: `*list` *points to the first element of the list, or is* `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)  // lb,lh
   add $t1, $t0, $s0
   sw $t0, 4($t1) // arr[2] <- 4; sb,sh

b) addiu $s1, $s0, 27
   lh $t0, -3($s1) // $t0 <- 0; lw,lb

c) addiu $s1, $s0, 24
   lh $t0$, -3($s1) // alignment error; lb
```

```
d) addiu $t0, $0, 12
   sw $t0, 6($s0) // alignment error; sh,sb

e) addiu $t0, $0, 8
   sw $t0, -4($s0) // out of bounds; sh,sb

f) addiu $s1, $s0, 10
   addiu $t0, $0, 6
   sw $t0, 2($s1) // arr[3] <- 6; sh,sb
```

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`
`slt $t0, $s0, $s1` `bne $t0, $0, label`	`slt $t0, $s1, $s0` `beq $t0, $0, label`	`sltiu $t0, $s0, 2` `beq $t0, $0, label`	`bgtz $s0, label`

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;`
	`addiu $s0, $0, 4`
	`addiu $s1, $0, 5`
	`addiu $s2, $0, 6`
	`addu $s3, $s0, $s1`
	`addu $s3, $s3, $s2`
	`addiu $s3, $s3, 10`

`// $s0 -> int * p = intArr;`
`// $s1 -> a;`
`*p = 0;`
`int a = 2;`
`p[1] = p[a] = a;`
	`sw $0, 0($s0)`
	`addiu $s1, $0, 2`
	`sw $s1, 4($s0)`
	`sll $t0, $s1, 2`
	`add $t0, $t0, $s0`
	`sw $s1, 0($t0)`

`// $s0 -> a, $s1 -> b`
 	
`int a = 5, b = 10;`
`if(a + a == b) {`
`  a = 0;`
`} else {`
`  b = a - 1;`
`}`
	`  addiu $s0, $0, 5`
	`  addiu $s1, $0, 10`
	`  addu $t0, $s0, $s0`
	`  bne $t0, $s1, else`
	`  xor $s0, $0, $0`
	`  j exit`
	`else:`
	`  addiu $s1, $s0, -1`
	`exit:`

`// computes s1 = 2^30`
`s1 = 1;`
`for(s0=0;s0<30;s++) {`
`  s1 *= 2;`
`}`
	`  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--);`
	`  xor $v0, $0, $0`
	`loop:`
	`  blez $a0, exit`
	`  addu $v0, $v0, $a0`
	`  addiu $a0, $a0, -1`
	`  j loop`
	`exit:`