CS61C Spring 2018 Discussion 2 – C Memory Management & RISC-V

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
   2. Local variables D
   3. Global variables B
   4. Constants A, B, D
   5. Machine Instructions A
   6. Result of malloc() C
   7. String Literals B

   A. Code

   B. Static

   C. Heap

   D. Stack

   4 Explained: With DEFINE, you can replace something in the code. With const declaration, it's stored in the static or stack depending on being declared in a function or not.

2. Write the code necessary to properly allocate memory (on the heap) in the following scenarios

   1. An array `arr` of k integers

   ```
   arr = (int *) malloc(sizeof(int) * k);
   ```

   2. A string `str` containing p characters

   ```
   str = (char *) malloc(sizeof(char) * (p + 1)); // Don't forget the null terminator!
   ```

   3. An $n \times m$ matrix `mat` of integers initialized to zeros

   ```
   mat = (int *) calloc(n*m, sizeof(int));
   ```

   Alternative solution. This might be needed if you wanted to efficiently permute the rows of the matrix.

   ```
   mat = (int **) calloc(n, sizeof(int *));
   for (int i = 0; i < n; i++) {
       mat[i] = (int *) calloc(m, sizeof(int));
   }
   ```

3. What is wrong with the C code below?

```
int* pi = malloc(314 * sizeof(int));
if(!raspberry) {
  pi = malloc(1 * sizeof(int)); // Memory leak if not raspberry
}
return pi;
```

4. 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; }`

`void prepend(struct ll_node** lst, int val)`	`void free_ll(struct ll_node** lst)`

`struct ll_node* item = (struct ll_node*)`
`    malloc(sizeof(struct ll_node));`
`item->value = val;`
`item->next = *lst;`
`*lst = item;`
	`if(*lst) {`
	`    free_ll(&((*lst)->next));`
	`    free(*lst);`
	`}`
	
	`*lst = NULL;`

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

2 Data Structures in C

In this question, we will implement a array-based stack of integers in C. The stack will be represented by the struct below.

```
struct stack {
    int size;        // Number of element in stackArray
    int topIndex;    // Index of the array that is the top of the stack
    int *stackArray; // Array holding the elements of the stack
}
```

Implement the functions below.

```
// Create a new stack with the given array size
struct stack *init_stack(int size) {
    struct stack *stk = (struct stack *) malloc(sizeof(struct stack));
    if (!stk) {
        return NULL;
    }
    stk->size = size;
    stk->topIndex = -1;
    stk->stackArray = (int *) malloc(sizeof(int) * size);
    if (!(stk->stackArray)) {
        free(stk);
        return NULL;
    }
    return stk;
}

// Add the given element to the stack. Resize by doubling the array size if full.
// Return 1 on success, 0 on failure. stk should be unchanged on failure.
int push(int x, struct stack* stk) {
    if (stk->topIndex == stk->size - 1) {
        int *stackArray = (int *) realloc(stk->stackArray, stk->size * 2)
        if (!stackArray) {
            return 0;
        }
        stk->stackArray = stackArray;
        stk->size = stk->size * 2;
    }
    stk->topIndex++;
    stk->stackArray[stk->topIndex] = x;
}

// Remove the top element from the stack. Return 0 if empty.
int pop(struct stack *stk) {
    if (stk->topIndex < 0) {
        return 0;
    }
    int x = stk->stackArray[stk->topIndex];
    // Setting to 0 is not strictly needed, but can be useful for
    // debugging/assertions
    stk->stackArray[stk->topIndex] = 0;
    stk->topIndex--;
    return x;
}
```

3 RISC-V 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 snippets of RISC-V code do? Note that these snippets all run immediately after each other on the same core (snippet a runs, then snippet b and so on).

```
a) lw t0, 12(s0)
Sets register t0 equal to arr[3]

b) slli t1, t0, 2
   add t2, s0, t1
   lw  t3, 0(t2)
   addi t3, t3, 1
   sw t3, 0(t2)
Increments the array element specified by t0 (i.e. arr[t0]) by 1

c) lw t0, 0(s0)
   xori t0, t0, 0xFFF ;Immediates are sign-extended (0xFFF -> all 1s);
   addi t0, t0, 1
Sets the register t0 to the two's complement negation of arr[0]
```

2. What are the instructions to branch to `label` on each of the following conditions? The only branch instructions you may use are `beq` and `bne`.

s0 < s1	s0 <= s1	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

4 Translating between C and RSIC-V

Translate between the C and RISC-V code. You may want to use the RISC-V Reference Card for more information on the instruction set and syntax. 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. You may assume all registers are initialized to zero.

C	RISC-V

// s0 -> a, s1 -> b
// s2 -> c, s3 -> z
int a = 4, b = 5, c = 6, z;
z = a + b + c + 10;
	addi s0, x0, 4
	addi s1, x0, 5
	addi s2, x0, 6
	add s3, s0, s1
	add s3, s3, s2
	addi s3, s3, 10

// s0 -> int * p = intArr;
// s1 -> a;
*p = 0;
int a = 2;
p[1] = p[a] = a;
	sw x0, 0(s0)
	addi s1, x0, 2
	sw s1, 4(s0)
	slli 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;
}
	addi s0, x0, 5
	addi s1, x0, 10
	add t0, s0, s0
	bne t0, s1, else
	xor s0, x0, x0
	jal x0, exit
	else:
	addi s1, s0, -1
	exit:

// computes s1 = 2^30
s1 = 1;
for(s0=0;s0<30;s++) {
s1 *= 2;
}
	addi s0, x0, 0
	addi s1, x0, 1
	addi t0, x0, 30
	loop:
	beq s0, t0, exit
	add s1, s1, s1
	addi s0, s0, 1
	jal x0, loop
	exit:

// s0 -> n, s1 -> sum
// assume n > 0 to start
for(int sum = 0; n > 0; n--) {
sum += n;
}
	addi s1, s1, 0
	loop:
	beq s0, x0, exit
	add s1, s1, s0
	add s0, s0, -1
	jal x0, loop
	exit: