# CS31 Written Homework 4: C Pointers and Functions

Due Thurs, March 5th at the beginning of class

**Required**: Your Name(s)/Lab Section(s):

## Question 1

Consider the following declarations and assignments:

```
int *a, b[5], c, *d;

for (c=0; c < 5 ; c++)  {
    b[c]= 1+c;
}
d=b;
a = &c;
c = d[3];
```

What are the TYPE and VALUE of each of the following expressions (if the expression is invalid, write "Illegal Expression", and if it is an address describe what it is the address of):

| | TYPE | VALUE |
|---|---|---|
| 1. a | | |
| 2. b | | |
| 3. c | | |
| 4. &b[1] | | |
| 5. d | | |
| 6. *d | | |

## Question 2

Trace through the following C code, and draw memory contents (heap and stack) at the execution point indicated in `foo`, and show the output produced by a complete run of the program. (Assume `stdio.h` and `stdlib.h` have been included, and that malloc succeeds.)

```
                                                    MEMORY
                                                    ------
int *foo(int *a, int *b, int s);

int main () {
    int *arr = NULL, x = 6, y = 7, i;

    arr = foo(&x, &y, 5);
    printf("x = %d  y = %d\n", x, y);
    if(arr != NULL) {
        for(i=0; i < 5; i++) {
            printf("arr[%d] = %d\n",
                    i, arr[i]);
        }
    }
    free(arr);
    return 0;
}
/**********************************/
int *foo(int *a, int *b, int s) {
    int *tmp, i;

    tmp = malloc(sizeof(int)*s);
    if(tmp != NULL) {
        for(i=0; i < s; i++) {
            tmp[i] = i + *b;
        }
        *a = tmp[2];
        *b = 8;
    }
    // DRAW MEMORY WHEN YOU GET HERE
    return tmp;
}

OUTPUT
------
```

# Question 3

Trace through the following IA32 code. Show the contents of the given memory and registers right before the instruction at point A is executed. Assume the `addl` instruction in `main` that is immediately after the `call` instruction is at memory address `0x1234`. Hints:

- remember to start execution in `main`.
- %esp points to the item on the top of the stack, so a push will grow the top of the stack and then move in the pushed value. A pop will move the value on top of the stack and then shrink the stack.
- The sequence of instructions `leave`; `ret` is equivalent to the sequence `movl %ebp, %esp`; `popl %ebp`; `popl %eip`.

```
foo:
    pushl  %ebp
    movl   %esp, %ebp
    subl   $16, %esp
    movl   8(%ebp), %eax
    addl   %eax, %eax
    movl   %eax, -4(%ebp)
    movl   -4(%ebp), %eax
    leave                       # A
    ret
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    $6, -4(%ebp)
    pushl   -4(%ebp)
    call    foo
    addl    $4, %esp       # at addr 0x1234
    movl    %eax, -4(%ebp)
    movl    $0, %eax
    leave
    ret
```

| Memory Address | at A value |
|---|---|
| 0x8880 | |
| 0x8884 | |
| 0x8888 | |
| 0x888c | |
| 0x8890 | |
| 0x8894 | |
| 0x8898 | |
| 0x889c | |
| 0x88a0 | |
| 0x88a4 | |
| 0x88a8 | |
| 0x88ac | |
| 0x88b0 | |
| 0x88b4 | |
| 0x88b8 | |
| 0x88bc | |
| 0x88c0 | |

| Register | Initial | at A |
|---|---|---|
| %eax | 2 | |
| %edx | 3 | |
| %esp | 0x88b0 | |
| %ebp | 0x88c0 | |