C Coding Guerilla Section Worksheet 1b (C coding)

1. BuggC (Spring 2014 MT1 #4)

A colleague of yours has implemented some homebrew C99 string manipulation functions, while steadfastly refusing to use any standard libraries, but they're buggy! We've marked each potentially problematic line with // <*number*>. Your job is to fill in a correct replacement line in the corresponding row of the following table, or write 'OK' if there is nothing wrong. DO NOT LEAVE ANY FIELDS BLANK.

Line number	Replacement Code
1	
2	
3	
4	
5	

```
/** Converts the string S to lowercase */
void string_to_lowercase(char *s) {
    for(char c = *s; c != '\0'; s++) {   // 1
        if(c >= 'A' && c <= 'Z') {
            s += 'a' - 'A';                    // 2
        }
    }
}

/** Returns the number of bytes in S before, but not counting, the null terminator. */
size_t string_length(char *s) {
    char *s2 = s;
    while(*s2++);                // 3
    return s2 - s - 1;           // 4
}


/** Return the number of odd numbers in a number array */
uint32_t number_odds(uint32_t *numbers, uint32_t size) {
    Uint32_t odds = 0;
    for (uint32_t i = 0; i < size; i++)
        odds += *numbers+i && 1;     // 5
```

```
    return odds;
}
```

2. **Insert TreeNode**

Implement the following function. TreeNode struct is reprinted below for your reference.

```
struct TreeNode {
    int32_t value;
    struct TreeNode *left;
    struct TreeNode *right;
    struct TreeNode *parent;
    uint64_t magic; // you are not required to modify this
}
```

Insert a TreeNode with value, val, under parent. If the parent does not have a left child, make the TreeNode the left child. If the parent has a left child, then make the TreeNode the right child. It is guaranteed that there will always be a spot under parent for you to insert the new node. Note: If parent is NULL, you are creating the root. This function should return a pointer to the newly inserted TreeNode. Not all lines are needed.

```
struct TreeNode * insertNode(struct TreeNode *parent, int val) {
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
}
```

3. **Number Pushing and Popping** (Fall 2016 M1 #3)

Your task is to implement a simple stack adding machine that uses a stack data structure. For example Push 2 and Push 3, PopAdd yields 5 in the top of the stack. Following this with Push 1, PopAdd would yield 6. Fill in the code for functions push and popadd so they meet the specifications stated in the comments. Do not make any other code modifications. Calls to malloc always returns a valid address.

```
/* Each item on the stack is represented
  by a pointer to the previous element
  (NULL if none) and its value. */
typedef struct stack_el {
    struct stack_el *prev;
    double val;
} stack_el;

/* PUSH: Push new value to top of the stack. Return
  pointer to new top of stack. */
stack_el push(stack_el *top_of_stack, double v) {
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
}

/* POPADD: Pop top stack element and add its value
  To the new top's value. Return new top of stack.
  Free no longer used memory. Do not change
  the stack if it has fewer than 2 elements. */
stack_el* popadd(stack_el *top_of_stack) {
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
}
```

4. A lot of Pointing

Assume you are given an int array arr, with a pointer p to its beginning:

```
int arr[] = {0x61c, 0x5008, 0xd, 0x4, 0x3, 0x4ffc};
int *p = arr;
```

Suppose arr is at location 0x5000 in memory, i.e., the value of p if interpreted as an integer is 0x5000. To visualize this scenario:

0x61c	0x5008	0xd	0x4	0x3	0x4ffc

arr[0] **…** **arr[5]**

Assume that integers and pointers are both 32 bits. What are the values of the following expressions? If an expression may cause an error, write "Error" instead.

a)*(p+3) = _____

b) p[4] = _____

c) *(p+5) + p[3] = _____

d) *(int*)(p[1]) = _____

e) *(int*)(*(p+5)) = _____

5. Reverse! (Fall 2017 MT1 #3.1)

Fill in the blanks to complete the reverse function which takes in a head_ptr to the head of a linked list and returns a new copy of the linked list in reverse order. You must allocate space for the new linked list that you return. An example program using reverse is also shown below.

```
struct list_node {
    int val;
    struct list_node* next;
};

struct list_node* reverse( _____   head_ptr ) {
    struct list_node* next = NULL;
    struct list_node* ret;
    while (*head_ptr != NULL) {
        ret = _____ ;
        ret->val = _____ ;
        ret->next = _____ ;
        next = _____ ;
        *head_ptr = (*head_ptr)->next;
    }
    return _____;
}
```