CSC 223: Stacks Quiz **Name ____**

1. Define stack underflow and stack overflow. How does stack overflow differ between array and linked list stack implementations?

2. A stack can be described as a linear, LIFO data structure. Explain what this means.

3. Write a function: void copy_stack(Stack** source, Stack** copy), that copies the contents of the stack source into the stack copy, preserving element order and leaving source unchanged.

CSC 223: Stacks Quiz **Name ____**

1. Define stack underflow and stack overflow. How does stack overflow differ between array and linked list stack implementations?

2. A stack can be described as a linear, LIFO data structure. Explain what this means.

3. Write a function: void copy_stack(Stack** source, Stack** copy), that copies the contents of the stack source into the stack copy, preserving element order and leaving source unchanged.

CSC 223: Stacks Quiz **Name _________Solutions___________________**

1. Define stack underflow and stack overflow. How does stack overflow differ between array and linked list stack implementations? (2 pts).

   Stack underflow occurs when an attempt is made to pop from an empty stack. Stack overflow occurs when an attempt is made to push onto a stack that is full. Overflow occurs with an array stack implementation when trying to push one more element onto the stack than the size of the array. With a linked list, overflow occurs when memory cannot be allocated from the heap (i.e. if the program statement: `StackNode* new = malloc(sizeof(StackNode));` in `llstack.c` fails).

2. A stack can be described as a linear, LIFO data structure. Explain what this means. (2 pts.)

   LIFO is an acronym for "last in, first out". It refers to the way in which data operations work in stack. The push operation adds a new element to the stack. The pop operation, which is the only way to remove an element, removes the element that was added by the most recent push operation. Stacks are "linear" data structures because they store data in a sequence (i.e. "linearly").

3. Write a function: void copy_stack(Stack** source, Stack** copy), that copies the contents of the stack source into the stack copy, preserving element order and leaving source unchanged. (3 pts).

```
void copy_stack(StackNode** source, StackNode** copy) {
    if (source == NULL || (*source) == NULL) return;
    StackNode* node_p = *source;
    Stack reversed = NULL;
    while (node_p != NULL) {
        llstack_push(&reversed, node_p->val);
        node_p = node_p->next;
    }
    while (!llstack_is_empty(&reversed)) {
        llstack_push(copy, llstack_pop(&reversed));
    }
}

or Nate's version:

void nate_copy(Stack* source, Stack* copy) {
    Stack tmp = NULL;
    while (!llstack_is_empty(source)) {
        int val = llstack_pop(source);
        llstack_push(&tmp, val);
    }
    while (!llstack_is_empty(&tmp)) {
        int val = llstack_pop(&tmp);
        llstack_push(source, val);
        llstack_push(copy, val);
    }
}
```

```
Which pass these tests:

    TEST("Test copy_stack works") {
        /* Setup original stack */
        Stack original = new_llstack_node(1);
        llstack_push(&original, 2);
        llstack_push(&original, 3);
        llstack_push(&original, 4);
        Stack copy;
        Stack ncopy;
        copy_stack(&original, &copy);
        nate_copy(&original, &ncopy);
        /* Confirm copy has what it should */
        ASSERT_EQ(llstack_top(&copy), 4);
        llstack_pop(&copy);
        ASSERT_EQ(llstack_top(&copy), 3);
        llstack_pop(&copy);
        ASSERT_EQ(llstack_top(&copy), 2);
        llstack_pop(&copy);
        ASSERT_EQ(llstack_top(&copy), 1);
        /* Confirm nate's copy has what it should */
        ASSERT_EQ(llstack_top(&ncopy), 4);
        llstack_pop(&ncopy);
        ASSERT_EQ(llstack_top(&ncopy), 3);
        llstack_pop(&ncopy);
        ASSERT_EQ(llstack_top(&ncopy), 2);
        llstack_pop(&ncopy);
        ASSERT_EQ(llstack_top(&ncopy), 1);
        /* Confirm original still has what it should */
        ASSERT_EQ(llstack_top(&original), 4);
        llstack_pop(&original);
        ASSERT_EQ(llstack_top(&original), 3);
        llstack_pop(&original);
        ASSERT_EQ(llstack_top(&original), 2);
        llstack_pop(&original);
        ASSERT_EQ(llstack_top(&original), 1);
    }
```