CSCI 2270 Practice Midterm 1

This exam was taken from last semester to give you a feel for the format and difficulty of the actual exam

---

Read these directions carefully.
On Moodle there is a quiz with **6** multiple-choice problems. Submit answers for each of those.

In addition to the multiple choice, you will be solving two of the three programming problems detailed below. *Problem 1* is **mandatory**, but you only have to do **either** *Problem 2* **or** *Problem 3*. For each problem you must submit three documents:

1. A C++ program that solves the given problem.
2. A text file that contains the output of your program when it is run.
3. A short document that contains any issues or concerns you have about the given problem, as well as any information that we need to understand, compile, or run your solution.

Your submission should be valid C++ 11 code. The solutions should use similar types and functions to those in the given starter code, but you are welcome to make modifications as you see fit. We will be running this code on other computers, so make sure to avoid **any** undefined behavior such as uninitialized variables.

Problem 1 (Mandatory)

Task:

Write a program that creates a linked list of integers, and a function that returns `true` if the number of nodes in the list is even, and `false` if it is odd.

Requirements:

1. Implement a `lengthIsEven` function:

```
bool lengthIsEven(Node *head); // example declaration
```

This function should return `true` if the number of nodes in the linked list is even, and `false` if it is odd.

Examples:

* If the list is `3 -> 9 -> 2 -> 4 -> NULL`, calling `lengthIsEven(head)` should return `true`.
* If the list is `3 -> 9 -> 2 -> NULL`, calling `lengthIsEven(head)` should return `false`.
* If the list is `NULL`, calling `lengthIsEven(head)` should return `true`.

2. Write a main function that creates a linked list of integers, then calls the function defined in part (1) to determine if the length of the list is even or odd. Call it on multiple lists to show that it works in all cases, and print out the results.

3. **Test your function** to make sure that it works in every case, no matter how many nodes are in the linked list.

Problem 2

Task:

Write a program that creates a linked list of integers, and a function that deletes every node with a negative value in that list.

Requirements:

1. Implement a `deleteNegatives` function:

```
Node *deleteNegatives(Node *head); // example declaration
```

This function should delete every negative node in the linked list and return a pointer to the new head of the list.

Examples:

* If the list is `-3 -> 9 -> 2 -> -4 -> NULL`, calling `deleteNegatives(head)` should change the list to be `9 -> 2 -> NULL` and return a pointer to the new head.

* If the list is `7 -> 6 -> 5 -> NULL`, calling `deleteNegatives(head)` should leave the list the same and return a pointer to the old head.

2. Write a main function that creates a linked list of integers, then calls the function defined in part (1) to delete all the negative

values. Call it on multiple lists to show that it works in all cases, and print out the results.

3. **Test your function** to make sure that it works in every case, no matter how many nodes or negative values are in the linked list.

Problem 3

Task:

Write a stack of integers that is implemented with a dynamically doubling array, that also halves its size when the extra space isn't needed.

Requirements:

1. Implement a stack of integers with a dynamic array with a starting size of 5. Every time the stack fills all the available space in the array, the array should double its capacity. Every time the stack drops below half full, the array should halve its capacity, with a minimum length of 5. Your stack should implement the following functions:

```
void push(int n); // example declaration
```

The `push` function should insert a value onto the stack. If the stack is full, it should double its capacity.

```
void pop(); // example declaration
```

The `pop` function should remove the value at the top of the stack. If the stack is only half full, it should halve its capacity (down to a minimum capacity of 5).

```
void print(); // example declaration
```

The `print` function should print the contents of the stack.

2. Write a main function that creates an instance of your stack, then calls the functions defined in part (1) to add and remove elements from it. Clearly show that it can double and halve its capacity. Print the results.

3. **Test your code** to make sure that it is successfully changing the size of the array, and that it can handle any edge cases you can think of.