Practice Midterm

Directions

Read these directions carefully.
Below are three programming problems. *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 w hen it is run.
3. A short document that contains any issues or concerns you have about the given problem, as w ell as any information that w e need to understand, compile, or run your solution.

Your submission should be valid C++ 11 code, and you should include comments that explain your reasoning. We w ill 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 finds the maximum value in that list.

Requirements:

1. Implement the `maximum` function:

```
int maximum(Node *head);
```

This function should find and return the maximum value in w hichever linked list is passed in. If it is called on an empty list, this function should return `0`.
Examples:

- If the list is `3 -> 9 -> 2 -> 4 -> NULL`, calling `maximum(head)` should return `9`.
- If the list is `NULL`, calling `maximum(head)` should return `0`.

1. Write a main function that creates a linked list of integers, then calls the function defined in part (1) to find the maximum value of that list. Call it on multiple lists to show that it w orks on any possible list.
2. **Test your function** to make sure that it w orks in every case, no matter how many nodes are in the linked list or w here the maximum value is w ithin the list.

Problem 2

Task:

Write a program that creates a linked list of integers, and a function that takes an index and deletes the node at that index, if it exists.

Requirements:

1. Implement the `deleteIndex` function:

```
bool deleteIndex(Node *head, int index);
```

This function takes a linked list and an index in that list (indices start at 0). The function should delete the node at that index. If the index is outside the bounds of the list, it should not delete anything. The function should return `true` if it successfully deleted a node, and `false` if the index w as out of bounds. It **must** delete the nodes from memory, rather than just removing them from the list.
Examples:

- If the list is `6 -> 3 -> 9 -> 6 -> NULL`, calling `deleteIndex(head, 2)` should return `true` and change the list to be `6 -> 3 -> 6 -> NULL`.
- If the list is `6 -> 3 -> NULL`, calling `deleteIndex(head, 2)` should return `false` and make keep the list the same.
- If the list is `6 -> NULL`, calling `deleteIndex(head, 0)` should return `true` and change the list to be `NULL`.

1. Write a main function that creates a linked list of integers, then calls the function defined in part (1) to delete various indices from that list. It should call the function multiple times, to show that your function can handle any possible index.
2. **Test your function** to make sure that it w orks in every case, no matter w hich index the user passes in!

Problem 3

Task:

Write a program that creates a linked list of integers, and a function that reverses that list.

Requirements:

1. Implement the `reverse` function:

```
void reverse(Node *head);
```

This function should reverse the order of the values in the linked list w hich is passed as an argument.
Example:

- If the list is `2 -> 7 -> 3 -> 3 -> NULL`, calling `reverse(head)` should change the list to be `3 -> 3 -> 7 -> 2 -> NULL`.

1. Write a main function that creates a linked list of integers, then calls the function defined in part (1) to reverse that list. It should call the function multiple times on different lists, to show that it can reverse a list of any length.
2. **Test your function** to make sure that it w orks in every case, no matter w hat list is passed in.