CSCI 2270 Final

Directions

Read these directions carefully.
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. It is required that your code files are well commented. `Note: you will not receive more than 25% of the possible points on any problem if your submitted code does not compile.`
2. If your code does not fully meet the specifications, then include a text file with your explanations for potential partial credit.

The third document you submit will help us understand your thought process. Mention anything you have done to write, test, and debug your code. Incomplete code can still receive points if you show that you have identified the errors and tried to debug them.

- Your submission should be valid C++ 11 code.
- Your solutions should use similar types and functions to the example code provided, but you are welcome to make modifications as you see fit.
- Show how you have tested your code. You will be graded higher for code with complete tests.
- 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:

Find the middle of a given linked list in C++. Given a singly linked list, find the middle of the linked list. For example, if given linked list is 1->2->3->4->5 then output should be 3. If there are an even number of nodes, then print the second of the middle elements. For example, if given linked list is 1->2->3->4->5->6, then the output should be 4.

Use the following algorithm: Traverse the linked list using two pointers. Advance the first pointer by one node at a time. Advance the second pointer by two nodes at a time. When the fast pointer reaches the end, the slow pointer will reach the middle of the linked list.

Requirements:

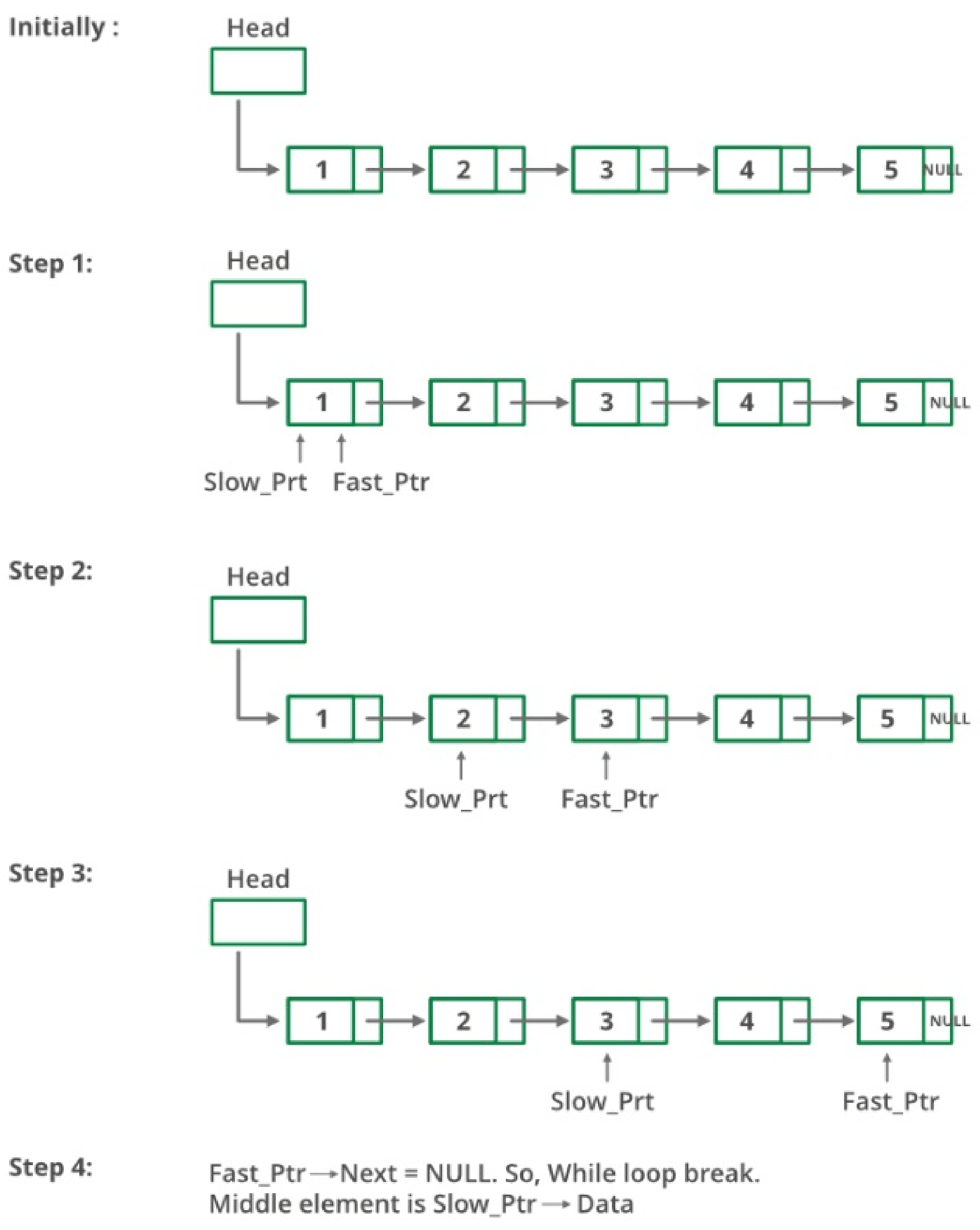
1. Implement a `getMiddle` function:

```
Node*  getMiddle();
```

This function should return a pointer to the middle node. If the list is empty, your function should return NULL.

Examples:

- Calling `getMiddle` on linkedList 1->2->3->4->5->6 should return 4
- Calling `getMiddle` on linkedList 1->2->3->4->5 should return 3



2. The function getMiddle skeleton code has been provided in the LinkedList.cpp file. Complete your implementation inside of this skeleton (you are not allowed to alter any other provided functions).

3. **Test your function** Test your code on the test cases provided in the driver file, LinkedListDriver.cpp (you are welcome to try your own test cases as well).

Problem 2

Task:

You are required to process a given string S that includes '#' symbol.
The '#' symbol represents backspace of a character. For instance given a string 'abc#def' you are required to transform it to 'abdef'. The transformation requires that the character before the # symbol and the symbol itself (i.e. c and #) be removed from the string.

Inside of your function, you will need to repeat this process on two strings. Once this is done, return true if both string are the same (post transformation). Otherwise return false.
You must use the provided stack class to execute this task.

Requirements:

1. The function stringManipulation skeleton code has been provided in the driver.cpp file. Fill in the function definition with your algorithm implementation. You `must` use the stack class provided in the starter code. `Do not change any of the class header or definition code.`

```
bool stringManipulation(string S, string T); // example declaration
```

This function should return true if the strings matched else false.

```
**Example 1**

    * Input: S = "ab#c", T = "ad#c"
    * Output: true
    * Explanation: Both S and T become "ac".

**Example 2**

    * Input: S = "ab##", T = "c#d#"
    * Output: true
    * Explanation: Both S and T become ""

**Example 3**

    * Input: S = "a##c", T = "#a#c"
    * Output: true
    * Explanation: Both S and T become "c".
```

```
**Example 4**
    * Input: S = "a#c", T = "b"
    * Output: false
    * Explanation: S becomes "c" while T becomes "b".
```

2. **Test your function** to make sure that it passes all the test cases.

Hint: the string class member function length() might be useful in your implementation.

Problem 3

Task:

Create a function that will append one array to another one.

Requirements:

The function prototype is given:

```
void append(int* &arr1, int* &arr2, int size1, int size2)
```

where

- `arr1` : pointer to the first array
- `arr2` : pointer to the second array
- `size1` : size of the array1
- `size2` : size of the array2

1. Function will append array2 to array1. After calling the function, elements of array2 will be appended to array1. Size of array1 should become `size1 + size2`
2. The main function will take size1 and size2 as command line arguments. Sample code for creating array1 is given in the starter code. You need to create the array2 in similar way. Then, call the append function.
3. After appending the arrays, **free up the memory space used by the original array(s)**.
4. Print the contents of the two arrays before the function call and the contents of the resulting array after the function call. Each element should be separated by a `,`. There should not be any `,` after the last element.

Example program output:

```
./a.out 5 3
arr1: 3,1,8,9,6
arr2: 5,7,12
arr1: 3,1,8,9,6,5,7,12
```