SC1008 Lab 2 of C++ – Linked List

1. **(getNthNodeValue) [10 marks]** Suppose a linked list is used to store students' marks of SC1008 lab 2, and each node's structure as follows:

```
struct MarkNode {
    int mark;
    ListNode* next;
};
```

You are asked to write a function called `locateNthNode()` to return the mark of the n-th node in the linked list. The function prototype is as follows:

```
int getNthNodeValue(const MarkNode* head, int n);
```

The parameter n is guaranteed to be >= 1. If n == 1, it means to return the mark value of the first node; If n is larger than the total length of the linked list, you should return -1, indicating exceeding the maximum length of the linked list.

Below is the main function and related functions with different test cases:

```
#include <iostream>
using namespace std;

struct MarkNode {
    int mark; // Changed to int
    MarkNode* next;
};

// Function to insert a node at the end of the linked list
void insertNode2ListEnd(MarkNode*& head, int newValue) {
    MarkNode* newNode = new MarkNode;
    newNode->mark = newValue;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
        return;
    }

    MarkNode* temp = head;
    while (temp->next != nullptr) { // Traverse to the last node
        temp = temp->next;
    }
    temp->next = newNode; // Link last node to new node
}
void destroyList(MarkNode*& head)
{
    MarkNode *nodePtr = head;  // Start at head of list
    MarkNode *garbage = nullptr;
```

```
    while (nodePtr != nullptr)
    {
        // garbage keeps track of node to be deleted
        garbage = nodePtr;
        // Move on to the next node, if any
        nodePtr = nodePtr->next;
        // Delete the "garbage" node
        delete garbage;
        garbage = nullptr;
    }
    head = nullptr;
}

// Function to get the value of the n-th node
int getNthNodeValue(const MarkNode* head, int n) {
    // TO-DO: WRITE Your code here
    //
    //
    //




}

int main() {
    MarkNode* head = nullptr; // Initialize an empty linked list

    // Insert nodes into the linked list
    insertNode2ListEnd(head, 10); // Insert 10
    insertNode2ListEnd(head, 20); // Insert 20
    insertNode2ListEnd(head, 30); // Insert 30
    insertNode2ListEnd(head, 40); // Insert 40

    // Test cases
    cout << getNthNodeValue(head, 1) << endl; // Output: 10
    cout << getNthNodeValue(head, 2) << endl; // Output: 20
    cout << getNthNodeValue(head, 4) << endl; // Output: 40
    cout << getNthNodeValue(head, 5) << endl; // Output: -1 (exceeds length)
    cout << getNthNodeValue(nullptr, 1) << endl; // Output: -1 (empty list)

    // Clean up memory
    destroyList(head);
    return 0;
}
```

Sample output should be:

```
10
20
40
-1
-1
```

2. **(reverseLinkedList) [10 marks]** Write a C++ function to reverse the order of a linked list. For example, a linked list like “A->B->C” will be changed to a linked list like “A<-B<-C” (i.e., “C->B ->A”). If the input linked list is empty, your function should leave it as it is. The structure of each node is as follows:

```
struct StringNode {
    string name;
    StringNode* next;
};
```

The function prototype is as follows:

```
void reverseLinkedList(StringNode*& head);
```

Below is the code for the main function with test cases and other relevant functions.

```
#include <iostream>
#include <string>
using namespace std;

struct StringNode {
    string name;
    StringNode* next;
};

void destroyList(StringNode*& head)
{
    StringNode *nodePtr = head;  // Start at head of list
    StringNode *garbage = nullptr;

    while (nodePtr != nullptr)
    {
        // garbage keeps track of node to be deleted
        garbage = nodePtr;
        // Move on to the next node, if any
        nodePtr = nodePtr->next;
        // Delete the "garbage" node
        delete garbage;
        garbage = nullptr;
    }
    head = nullptr;
}
```

```
void printLinkedList(const StringNode* head) {
    const StringNode* current = head;
    while (current != nullptr) {
        cout << current->name;
        if (current->next != nullptr) {
            cout << " -> ";
        }
        current = current->next;
    }
    cout << endl;
}

void insertNode2ListEnd(StringNode*& head, const string& newName) {
    StringNode* newNode = new StringNode;
    newNode->name = newName;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
        return;
    }

    StringNode* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to reverse the linked list
void reverseLinkedList(StringNode*& head) {
    // TO-DO: WRITE YOUR CODE HERE
    //
    //
    //


}

int main() {
    StringNode* head = nullptr; // Initialize an empty linked list

    /////// Case 1 ////////////
    // Print the original linked list
    cout << "Original Linked List: ";
    printLinkedList(head);
    // Reverse the linked list
```

```
    reverseLinkedList(head);
    // Print the reversed linked list
    cout << "Reversed Linked List: ";
    printLinkedList(head);
    cout << endl;

    /////// Case 2 ////////////
    insertNode2ListEnd(head, "Michael");
    cout << "Original Linked List: ";
    printLinkedList(head);
    // Reverse the linked list
    reverseLinkedList(head);
    // Print the reversed linked list
    cout << "Reversed Linked List: ";
    printLinkedList(head);
    cout << endl;

    /////// Case 3 ////////////
    insertNode2ListEnd(head, "Emily");
    insertNode2ListEnd(head, "James");
    insertNode2ListEnd(head, "William");
    // Print the original linked list
    cout << "Original Linked List: ";
    printLinkedList(head);
    // Reverse the linked list
    reverseLinkedList(head);
    // Print the reversed linked list
    cout << "Reversed Linked List: ";
    printLinkedList(head);
    cout << endl;

    destroyList(head);
    return 0;
}
```

Sample output should be:

```
Original Linked List:
Reversed Linked List:

Original Linked List: Michael
Reversed Linked List: Michael

Original Linked List: Michael -> Emily -> James -> William
Reversed Linked List: William -> James -> Emily -> Michael
```