Tutorial 2 – Linked List

1. **(getListLen)** A linked list consists of nodes with each node's structure as follows:

```
struct ListNode {
    int value;
    ListNode* next;
};
```

You are asked to write a C++ function called `getListLen()` to count the total number of nodes in the linked list. The function prototype is as follows:

```
int getListLen(const ListNode* head);
```

Below is the main function with different test cases:

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

struct ListNode {
    int value;
    ListNode* next;
};

// TO-DO: You need to implement this function
int getListLen(const ListNode* head){


}

int main() {
    // Creating a simple linked list: 10 -> 20 -> 30 -> 40
    ListNode* head = new ListNode;
    head->value = 10;
    head->next = nullptr;
    cout << "Length of the linked list: " << getListLen(head) << endl;

    head->next = new ListNode;
    head->next->value = 20;
    head->next->next=nullptr;
    cout << "Length of the linked list: " << getListLen(head) << endl;

    head->next->next=new ListNode;
    head->next->next->value = 30;
    head->next->next->next = nullptr;
    cout << "Length of the linked list: " << getListLen(head) << endl;

    head->next->next->next=new ListNode;
    head->next->next->next->value = 40;
    head->next->next->next->next = nullptr;
    cout << "Length of the linked list: " << getListLen(head) << endl;
    // Free allocated memory
    while (head) {
        ListNode* temp = head;
        head = head->next;
        delete temp;
    }
    head = nullptr;

    return 0;
}
```

Sample output should be:

```
Length of the linked list: 1
Length of the linked list: 2
Length of the linked list: 3
Length of the linked list: 4
```

2. **(stringArray2List)** Write C++ code to convert an array of strings (e.g., student names) to a linked list. The structure of each node in the linked list is defined as follows:

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

You are asked to insert elements of the input array one by one at the beginning of the linked list, instead of at the end of the linked list. The function prototype is as follows:

```
void arrayToLinkedList(const string* arr, int size, StringNode*& head);
```

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

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

// Define the structure of a linked list node
struct StringNode {
    string name;
    StringNode* next;
};

// Function to print the linked list
void printList(StringNode* head) {
    StringNode* temp = head;
    cout << "Linked list: ";
    while (temp) {
        cout << temp->name << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

// Function to free allocated memory
void deleteList(StringNode*& head) {
    while (head) {
        StringNode* temp = head;
        head = head->next;
        delete temp;
    }
    head = nullptr;
}

// To-do: Create a linked list from an array of strings
void arrayToLinkedList(const string* arr, int size, StringNode*& head) {
     // Please write down your code here
     //
     //


}

int main() {
    // Case 1
    string students[] = {"Alice", "Bob", "Charlie", "David"};
    int size = sizeof(students) / sizeof(students[0]);
    StringNode* head1 = nullptr;
    arrayToLinkedList(students, size, head1);
    printList(head1);

    // Case 2
    string companyNames[] = {"Microsoft", "Google", "Tecent", "Alibaba", 
"HP"};
    size = sizeof(companyNames) / sizeof(companyNames[0]);
    StringNode* head2 = nullptr;
    arrayToLinkedList(companyNames, size, head2);
    printList(head2);

    deleteList(head1);
    deleteList(head2);
    return 0;
}
```

Sample output should be:

```
Linked list: David -> Charlie -> Bob -> Alice -> NULL
Linked list: HP -> Alibaba -> Tecent -> Google -> Microsoft -> NULL
```

3. **(insert2SortedList)** For the linked list discussed in the lecture of "Linked List",

```
struct Node {
    double value;
    Node* next;
};
```

write a function to in C++ to insert a new number into a sorted linked list at the appropriate location. For simplicity, let's assume that the values of the sorted linked list are already in ascending order. The function prototype has been provided here:

```
void insertNode2SortedList(Node*& head, double number);
```

The main function with a few test cases as well as other related functions are given below.

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

struct Node {
    double value;
    Node* next;
};

void printList(Node* head) {
    Node* current = head;// Start at the head of the list
    while (current) {
        cout << current->value << " -> ";
        current = current->next;
    }
    cout << "NULL" << endl;
}

void destroyList(Node*& head)
{
    Node *nodePtr = head;
    Node *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 insertNode2ListEnd(Node*& head, double newValue) {
    Node* newNode = new Node;
    newNode->value = newValue;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) { // Traverse to the last node
        temp = temp->next;
    }
    temp->next = newNode; // Link last node to new node
}


// TO-DO: Finish the implementation of this function
//        Assumption: the linked list is in ascending order
void insertNode2SortedList(Node*& head, double number)
{
    // TO-DO: Write your code here
    //
    //
    //


}



int main() {
    // Create the linked list
    Node* head = nullptr; // Start with an empty list

    // Insert values
    insertNode2ListEnd(head, 2.5);
    insertNode2ListEnd(head, 7.9);
    insertNode2ListEnd(head, 12.6);

    cout << "Original list: ";
    printList(head);

    insertNode2SortedList(head, 10.5);
    cout << "New list: ";
    printList(head);

    insertNode2SortedList(head, 1.5);
    cout << "New list: ";
    printList(head);

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

Sample output should be:

```
Original list: 2.5 -> 7.9 -> 12.6 -> NULL
New list: 2.5 -> 7.9 -> 10.5 -> 12.6 -> NULL
New list: 1.5 -> 2.5 -> 7.9 -> 10.5 -> 12.6 -> NULL
```

4. **(concatenateTwoLists)** For the linked list discussed in the lecture of "Linked List",

```
struct Node {
    double value;
    Node* next;
};
```

Write a C++ function to concatenate two separate linked lists. The function prototype is as follows:

```
void concateTwoLists (Node*& firstList, Node*& secondList);
```

The function should append the **second linked list** to **the end of the first linked list**. After the concatenation, `secondList` should be set as `nullptr`. The main function with two test cases and other related functions are given below:

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

struct Node {
    double value;
    Node* next;
};

void printList(Node* head) {
    Node* current = head;

    while (current) {
        cout << current->value << " -> ";
        current = current->next;
    }
    cout << "NULL" << endl;
}


void destroyList(Node*& head) {
    Node* nodePtr = head;
    Node* 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 insertNode2ListEnd(Node*& head, double newValue) {
    Node* newNode = new Node;
    newNode->value = newValue;
    newNode->next = nullptr;

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

    Node* temp = head;
    while (temp->next != nullptr) { // Traverse to the last node
        temp = temp->next;
    }
    temp->next = newNode; // Link last node to new node
}

// To-DO: concatenate two lists
void concateTwoLists(Node*& firstList, Node*& secondList) {
     // To-DO: write your code here
     //
     //
     //



}

int main() {
    // Create first list: 1 -> 2 -> 3 -> NULL
    Node* firstList = nullptr;
    insertNode2ListEnd(firstList, 1);
    insertNode2ListEnd(firstList, 2);
    insertNode2ListEnd(firstList, 3);
    cout << "First List: ";
    printList(firstList);

    // Create second list: 4 -> 5 -> 6 -> NULL
    Node* secondList = nullptr;
    insertNode2ListEnd(secondList, 4);
    insertNode2ListEnd(secondList, 5);
    insertNode2ListEnd(secondList, 6);
    cout << "Second List: ";
    printList(secondList);

    // Create third list: 100 -> 200 -> 300 -> NULL
    Node* thirdList = nullptr;
    insertNode2ListEnd(thirdList, 100);
    insertNode2ListEnd(thirdList, 200);
    insertNode2ListEnd(thirdList, 300);
    cout << "Third List: ";
    printList(thirdList);

    // Create fourth list: 900 -> 800 -> NULL
    Node* fourthList = nullptr;
    insertNode2ListEnd(fourthList, 900);
    insertNode2ListEnd(fourthList, 800);
    cout << "Fourth List: ";
    printList(fourthList);


    // Concatenate secondList to firstList
    concateTwoLists(firstList, secondList);
    cout << "Concatenated First List: ";
    printList(firstList);

    // Concatenate thirdList to fourthList
    concateTwoLists(fourthList, thirdList);
    cout << "Concatenated Fourth List: ";
    printList(fourthList);


    // Destroy the concatenated list
    destroyList(firstList);
    destroyList(fourthList);

    return 0;
}
```

Sample output should be:

```
First List: 1 -> 2 -> 3 -> NULL
Second List: 4 -> 5 -> 6 -> NULL
Third List: 100 -> 200 -> 300 -> NULL
Fourth List: 900 -> 800 -> NULL
Concatenated First List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
Concatenated Fourth List: 900 -> 800 -> 100 -> 200 -> 300 -> NULL
```