NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

Assignment – 02

Name : Joy Kumar Ghosh
Student ID : 2211424 6 42
Course No. : CSE 225
Course Title : Data Structures and Algorithm
Section : 06
Date : 28 March 2023

Code:

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


//Class Declaration
template <class T>
class SingleLinkedList{
   struct Node{
      T data;
      Node* next;
   };

private:
   Node* head;
   int length;

public:
   SingleLinkedList();
   int getLength();
   bool isMemoryFull();
   bool insertAtBeginning(T);
   bool insertAtEnd(T);
   void display();
   bool search(T);
   bool deleteFromBeginning();
   bool deleteFromEnd();
};
```

```
//Implementation
template <class T>
SingleLinkedList<T>::SingleLinkedList(){
   head = NULL;
   length = 0;
}

template <class T>
int SingleLinkedList<T>::getLength(){
   return length;
}

template <class T>
bool SingleLinkedList<T>::isMemoryFull(){
   Node* temp;
   try{
      temp = new Node;
      delete temp;
      return false;
   }
   catch(bad_alloc& exception){
      return true;
   }
}

template <class T>
bool SingleLinkedList<T>::insertAtBeginning(T value){
   if(!isMemoryFull()){
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = head; //Point the next of the new node to the current head
      head = newNode; //Set the head to the new node
      length++;
      return true;
   }
```

```
    else{
        return false;
    }
}

template <class T>
bool SingleLinkedList<T>::insertAtEnd(T value){
     if(!isMemoryFull()){
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL; // Set the next of the new node to NULL as it is the
last node
        if(head == NULL){ // If the linked list is empty, set the head to the new node
            head = newNode;
            length++;
            return true;
        }

        Node* last = head; // Traverse the linked list to find the last node
        while(last->next != NULL){
            last = last->next;
        }
        last->next = newNode; // Set the next of the last node to the new node
        length++;
        return true;
     }

     else{
        return false;
     }
}

template <class T>
void SingleLinkedList<T>::display(){
    if(head == NULL){ // If the linked list is empty, print a message
```

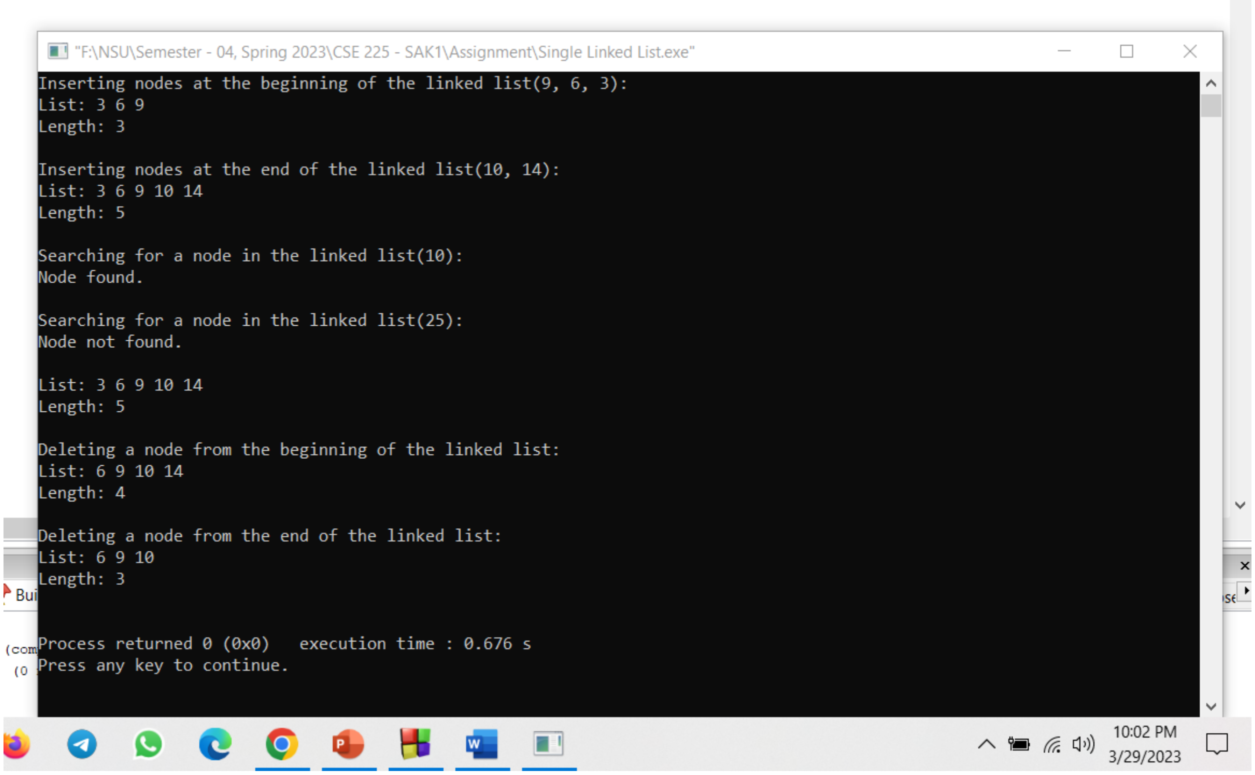
```
        cout << "Linked list is empty." << endl;
    }

    else{
        cout << "List: ";
        Node* temp = head; // Traverse the linked list and print the data of each
node
        while(temp != NULL){
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
}

template <class T>
bool SingleLinkedList<T>::search(T value){
    Node* temp = head; // Traverse the linked list and check if the value matches
with the data of any node
    while(temp != NULL){
        if(temp->data == value){
            return true; // Return true if value is found
        }
        temp = temp->next;
    }
    return false; // Return false if value is not found
}

template <class T>
bool SingleLinkedList<T>::deleteFromBeginning(){
    if(head == NULL){ // If the linked list is empty
        return false;
    }
    else{
        Node* temp = head; // Set a temporary node to the head
        head = head->next; // Set the head to the next node
        delete temp; // Delete the temporary node
        length--;
        return true;
    }
}

template <class T>
bool SingleLinkedList<T>::deleteFromEnd(){
    if(head == NULL){ // If the linked list is empty
        return false;
    }
    else{
        if(head->next == NULL){ // If the linked list has only one node, delete it and set the head to NULL
            delete head;
            head = NULL;
            length--;
            return true;
        }

        Node* temp = head; // Traverse the linked list to find the second last node
        while(temp->next->next != NULL){
            temp = temp->next;
        }

        delete temp->next; // Delete the last node
        temp->next = NULL; // Set the next of the second last node to NULL as it is the new last node
        length--;
        return true;
    }
}
```

```
//main driver file
int main()
{
   SingleLinkedList<int> list;

   // Test the insertAtBeginning() function
   cout << "Inserting nodes at the beginning of the linked list(9, 6, 3):" << endl;
   if(!list.insertAtBeginning(9)){
      cout << "Memory Full!!" << endl;
   }
   if(!list.insertAtBeginning(6)){
      cout << "Memory Full!!" << endl;
   }
   if(!list.insertAtBeginning(3)){
      cout << "Memory Full!!" << endl;
   }
   list.display();

   cout << "Length: " << list.getLength() << endl << endl;

   // Test the insertAtEnd() function
   cout << "Inserting nodes at the end of the linked list(10, 14):" << endl;
   if(!list.insertAtEnd(10)){
      cout << "Memory Full!!" << endl;
   }
   if(!list.insertAtEnd(14)){
      cout << "Memory Full!!" << endl;
   }
   list.display();

   cout << "Length: " << list.getLength() << endl << endl;

   // Test the search() function
   cout << "Searching for a node in the linked list(10):" << endl;
   if(list.search(10)){
```

```
   cout << "Node found." << endl;
}
else{
   cout << "Node not found." << endl;
}

cout << endl;

cout << "Searching for a node in the linked list(25):" << endl;
if(list.search(25)){
   cout << "Node found." << endl;
}
else{
   cout << "Node not found." << endl;
}

cout << endl;

// Test the deleteFromBeginning() function
list.display();

cout << "Length: " << list.getLength() << endl << endl;

cout << "Deleting a node from the beginning of the linked list:" << endl;
list.deleteFromBeginning();
list.display();

cout << "Length: " << list.getLength() << endl << endl;

// Test the deleteFromEnd() function
cout << "Deleting a node from the end of the linked list:" << endl;
list.deleteFromEnd();
list.display();

cout << "Length: " << list.getLength() << endl << endl;
```

```
    return 0;
}
```

Screenshot:


"F:\NSU\Semester - 04, Spring 2023\CSE 225 - SAK1\Assignment\Single Linked List.exe"
10:02 PM
3/29/2023

```
Inserting nodes at the beginning of the linked list(9, 6, 3):
List: 3 6 9
Length: 3

Inserting nodes at the end of the linked list(10, 14):
List: 3 6 9 10 14
Length: 5

Searching for a node in the linked list(10):
Node found.

Searching for a node in the linked list(25):
Node not found.

List: 3 6 9 10 14
Length: 5

Deleting a node from the beginning of the linked list:
List: 6 9 10 14
Length: 4

Deleting a node from the end of the linked list:
List: 6 9 10
Length: 3


Process returned 0 (0x0)   execution time : 0.676 s
Press any key to continue.
```