DATA STRUCTURES AND ALGORITHMS ASSIGNMENT ON LINKED LIST

NAME : RAHUL GOEL

REG NO: RA1911030010094

BATCH : O2-TP304

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

class linkedList{
   public:

   struct Node
   {
      int data;
      Node *next;
   };

   struct Node* head = NULL;

   //function to check if list is empty
   void Empty_List(){
      if(head==NULL){
         cout<<"True";
      }
      else{
         cout<<"False";
      }
   }

   //function to insert node at the end
   void Insert(int nData){
      Node* new_node = new Node();
      Node *last = head;
      new_node->data = nData;
      new_node->next = NULL;
      if (head == NULL)
      {
         head = new_node;
         return;
```

```
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
    return;

}

//function to insert node at specified postion
void Insert(int nData,int pos){

    Node* new_node = new Node();
    Node* pNode = head;
    if(pos==1||pNode==NULL){
        new_node->data=nData;
        new_node->next=pNode;
        head = new_node;
    }
    else{
        pos=pos-2;
        while(pos!=0){
            pNode=pNode->next;
            pos--;
            if(pNode==NULL)
                break;
        }
        if(pNode==NULL){
            cout<<"invalid position"<<endl;
            return;
        }

        new_node->data=nData;
        new_node->next=pNode->next;
        pNode->next=new_node;
    }
}

//fuction to delete node at specified position
void Delete(int pos){

    Node* pNode = head;
    if(pos==1){
        head=pNode->next;
        free(pNode);
    }
    else{
        pos=pos-2;
        while(pos--){
```

```
            pNode=pNode->next;
            if(pNode==NULL)
                break;
        }
        if(pNode==NULL){
            cout<<"invalid position"<<endl;
            return;
        }
        Node* temp = pNode->next;
        pNode->next=pNode->next->next;
        free(temp);
    }
}

//function to delete node with specified data
void Delete(int dData,int start){
    Node* pNode = head;
    Node* prevNode = pNode;
    if(pNode==NULL)
        cout<<"List is empty"<<endl;
    else{
        while(pNode->data != dData){
            prevNode=pNode;
            pNode=pNode->next;
            if(pNode->next==NULL){
                break;
            }
        }
        if(pNode->next==NULL && pNode->data==dData){
            prevNode->next=NULL;
            free(pNode);
        }
        else if(pNode->next==NULL)
            cout<<"Element not present in list"<<endl;
        else if(prevNode==pNode){
            head = prevNode->next;
            free(prevNode);
        }
        else{
            Node* temp = prevNode->next;
            prevNode->next=prevNode->next->next;
            free(temp);
        }
    }
}

//function to display linked list
void Display() {
```

```
        struct Node* ptr;
        ptr = head;
        while (ptr != NULL) {
            cout<<"->"<< ptr->data;
            ptr = ptr->next;
        }
    }

    //funtion to merge and sort the linked lists
    void Merge(linkedList &ob1, linkedList &ob2){
        //merge part
        Node *last = ob1.head;
        if (last==NULL)
            last=ob2.head;
        else{
            while (last->next != NULL)
            {
                last=last->next;
            }
            last->next = ob2.head;
        }

        //sort part
        int temp;
        for(Node *i = ob1.head;i->next!=NULL;i=i->next){
            for(Node *j = ob1.head; j->next!=NULL; j=j->next){
                if(j->data>j->next->data){
                    temp=j->data;
                    j->data=j->next->data;
                    j->next->data=temp;
                }
            }
        }
    }
};

int main(){
    //initializing variables and linked lists
    int n1,n2,element,pos;
    linkedList l1,l2;

    //taking number of elements in each list
    cout<<"Enter size of 2 linked lists"<<endl;
    cin>>n1>>n2;

    //inputing elements for 1st list
    cout<<"Enter elements of 1st linked list"<<endl;
    while(n1--){
```

```
    cin>>element;
    l1.Insert(element);
}

//inputing elements for 2nd list
cout<<"Enter elements of 2nd linked list"<<endl;
while(n2--){
    cin>>element;
    l2.Insert(element);
}

//output list 1
cout<<"Linked list 1: ";
l1.Display();
cout<<endl<<endl;

//output list 2
cout<<"Linked list 2: ";
l2.Display();
cout<<endl<<endl;

//inserting element at specified position
cout<<"Enter element to insert and the position "<<endl;
cin>>element>>pos;
l1.Insert(element,pos);
cout<<"Updated linked list 1: ";
l1.Display();
cout<<endl<<endl;

//deleting element at specified position
cout<<"Insert position of element to delete "<<endl;
cin>>pos;
l1.Delete(pos);
cout<<"Updated linked list 1: ";
l1.Display();
cout<<endl<<endl;

//deleting specified element
cout<<"Insert element to delete "<<endl;
cin>>element;
l1.Delete(element,1);
cout<<"Updated linked list 1: ";
l1.Display();
cout<<endl<<endl;

//checking if list is empty
cout<<"Checking if list is empty - ";
l1.Empty_List();
```

```
    cout<<endl<<endl;

    //merging 2 lists
    cout<<"Merging lists"<<endl;
    l1.Merge(l1,l2);
    cout<<"New merged list :";
    l1.Display();

    return 0;
}
```

Explanation for code

First, I made a linked list class, this is so that I can make multiple linked lists by using objects.

Using a structure, I created a node with a data field and address field. Then initialized the head to NULL.

Empty_List()

Empty_List() function checks if the value of head is NULL, if it is NULL then the list is empty and therefore it prints true, if not it prints false. The time complexity is O(1).

Insert()

I made two Insert() functions using function overloading, one to insert a new element at the end and one to insert an element at a specified position.

The first Insert() function takes the data as input, it creates a new node and assigns the input data to the data field of new node, it traverses to the end of the list and using pointer ‘last’, then changes the address field of last element from NULL to new node. The time complexity is O(n).

The second Insert() function takes data and position as input. It creates a new node and assigns the input data to the data field of new node, it traverses to the position using pNode and changes new node address field to pNode->next and pNode address field to new node. The time complexity is O(n).

Delete()

Similar to Insert(), I made two Delete() functions using function overloading, one to delete according to position and one to delete according to the given element.

The first Delete() function takes position as input and traverses to the inputted position. A new pointer called temp is created which points to the node at inputted position. The previous nodes address field is changed to pNode->next->next, then temp is freed. The time complexity is O(n).

The second Delete() function takes data as input, it also takes an integer as input but this was just to enable overloading of the function and the integer serves no purpose. pNode is assigned value of head and it continues to traverse the list until a nodes data matches with inputted data. If the data matches then the node is deleted using a temporary pointer. The time complexity is O(n).

Display()

Display() function simply displays the elements of the linked list, by traversing it and printing the data field of each node. The time complexity is O(n).

Merge()

Merge() function traverses to the end of the first linked list and then points it to the head of the second linked list, this concludes the merging. As for sorting, I used bubble sort to sort the merged list. The time complexity is $O(n^2)$.