Assignment 5-Linked List by Gavisht Singh  
1. Develop a menu driven program for the following operations on a Singly Linked

List.

(a) Insertion at the beginning.

(b) Insertion at the end.

(c) Insertion in between (before or after a node having a specific value, say 'Insert a

new Node 35 before/after the Node 30').

(d) Deletion from the beginning.

(e) Deletion from the end.

(f) Deletion of a specific node, say 'Delete Node 60').

(g) Search for a node and display its position from head.

(h) Display all the node values.

Ans:

#include <iostream>

using namespace std;

class Node{

    public:

    int data;

    Node\* next;

    Node(int data1){

        data=data1;

        next=nullptr;

    }

};

class SLL{

   private:

   Node\* head;

   public:

   SLL(){

    head=nullptr;

   }

  void insertAtBeginning(int data) {

        Node\* newNode = new Node(data);

        newNode->next = head;

        head = newNode;

        cout<< data << " inserted at the beginning."<<endl;

  }

   void insertattail(int data){

    Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

        } else {

            Node\* temp = head;

            while (temp->next != NULL) {

                temp = temp->next;

            }

            temp->next = newNode;

        }

        cout<<data << " inserted at the end."<<endl;

   }

   void insertaftervalue(int targetval,int val){

    Node\* temp=head;

    while (temp != NULL && temp->data != targetval) {

            temp = temp->next;

        }

        if (temp == nullptr) {

            cout<< targetval << " not found"<<endl;

*return*;

        }

        Node\* newNode = new Node(val);

        newNode->next = temp->next;

        temp->next = newNode;

        cout<< val << " inserted after " << targetval << endl;

   }

   void insertbeforevalue(int targetval,int val){

    Node\* temp=head;

    Node\* x=temp;

    if (head != NULL && head->data == targetval) {

        insertAtBeginning(val);

*return*;

    }

    while (temp != NULL && temp->data != targetval) {

         x=temp;

            temp = temp->next;

        }

        if (temp == nullptr) {

            cout<< targetval << " not found"<<endl;

*return*;

        }

        Node\* newNode = new Node(val);

        newNode->next = temp;

        x->next = newNode;

        cout<< val << " inserted before " << targetval << endl;

   }

   void deletehead(){

     if(head==NULL){

        cout<<"Empty"<<endl;

    }

    else{

        Node\* temp=head;

        head=head->next;

        delete temp;

        cout<<"Deleted head"<<endl;

   }

   }

   void deletetail(){

    if(head==NULL){

        cout<<"Empty"<<endl;

*return*;

    }

    if (head->next == nullptr) {

        delete head;

        head = nullptr;

        cout << " The only node has been deleted."<<endl;

*return*;

    }

        Node\* temp=head;

        Node\* x=temp;

        while(temp->next!=NULL){

            x=temp;

            temp=temp->next;

        }

        x->next=nullptr;

         delete temp;

        cout<<"Deleted head"<<endl;

   }

   void deletebyval(int val){

    if(head==NULL){

        cout<<"Empty list"<<endl;

*return*;

    }

        if(head->data==val){

        Node\* temp=head;

        head=head->next;

        delete temp;

        cout<<"Deleted"<<val<<endl;

*return*;

    }

    Node\* temp=head;

    Node\* x=NULL;

        while(temp!=NULL && x!=0){

           if(temp->data==val){

            x->next=x->next->next;

            delete temp;

            cout<<"Deleted"<<val<<endl;

*return*;

        }

        else{

        x=temp;

        temp=temp->next;

        }

        }

   }

   void search(int val){

    Node\* temp=head;

    int count=0;

    if(head==NULL){

        cout<<"Empty list"<<endl;

*return*;

    }

    while(temp!=NULL){

        count++;

        if(temp->data==val){

            cout<<val<<" Found at position"<<count<<endl;

*return*;

        }

        else{

            temp=temp->next;

        }

    }

   }

   void display(){

    Node\* temp=head;

    if(head==NULL){

        cout<<"Empty list"<<endl;

    }

        while(temp!=NULL){

            cout<<temp->data<<" ";

            temp=temp->next;

        }

        cout<<endl;

   }

};

int main(){

   SLL list;

    int choice, value, target;

    char subChoice;

    do{

        cout << "1. Insert at Beginning"<<endl;

        cout << "2. Insert at End"<<endl;

        cout << "3. Insert In-Between"<<endl;

        cout << "4. Delete from Beginning"<<endl;

        cout << "5. Delete from End"<<endl;

        cout << "6. Delete a Specific Node"<<endl;

        cout << "7. Search for a Node"<<endl;

        cout << "8. Display List"<<endl;

        cout << "9. Exit"<<endl;

        cout << "Enter your choice: "<<endl;

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert: ";

                cin >> value;

                list.insertAtBeginning(value);

                break;

            case 2:

                cout << "Enter value to insert: ";

                cin >> value;

                list.insertattail(value);

                break;

            case 3:

                cout << "Enter target node value: ";

                cin >> target;

                cout << "Enter new value to insert: ";

                cin >> value;

                cout << "Insert (B)efore or (A)fter " << target << "? ";

                cin >> subChoice;

                if (subChoice == 'B' || subChoice == 'b') {

                    list.insertbeforevalue(target, value);

                } else if (subChoice == 'A' || subChoice == 'a') {

                    list.insertaftervalue(target, value);

                } else {

                    cout << "Invalid choice.";

                }

                break;

            case 4:

                list.deletehead();

                break;

            case 5:

                list.deletetail();

                break;

            case 6:

                cout << "Enter value to delete: ";

                cin >> value;

                list.deletebyval(value);

                break;

            case 7:

                cout << "Enter value to search: ";

                cin >> value;

                list.search(value);

                break;

            case 8:

                list.display();

                break;

            case 9:

                cout << "Exiting program";

*return* 0;

            default:

                cout << "Invalid choice";

                break;

        }

    }

    while(choice!=9);

*return* 0;

}

2. Write a program to count the number of occurrences of a given key in a singly linked list and then delete all the occurrences.

Ans:

#include <iostream>

using namespace std;

class Node{

    public:

    int data;

    Node\* next;

    Node(int data1){

        data=data1;

        next=nullptr;

    }

};

class SLL{

   private:

   Node\* head;

   public:

   SLL(){

    head=nullptr;

   }

   void insertatend(int data){

       Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

        } else {

            Node\* temp = head;

            while (temp->next != NULL) {

                temp = temp->next;

            }

            temp->next = newNode;

        }

   }

   void occurence(int k){

    int count=0;

   while (head != nullptr && head->data == k) {

        Node\* toDelete = head;

        head = head->next;

        delete toDelete;

        count++;

    }

    if (head == nullptr) {

        cout << count << " is the number of times key occurred" << endl;

    }

    Node\* current = head;

    while (current->next != nullptr) {

        if (current->next->data == k) {

            Node\* toDelete = current->next;

            current->next = toDelete->next;

            delete toDelete;

            count++;

        } else {

            current = current->next;

        }

    }

    cout << count << " is the number of times key occurred" << endl;

   }

   void display(){

       Node\* temp=head;

       while(temp!=NULL){

        cout<<temp->data<<" ";

        temp=temp->next;

       }

       cout<<endl;

   }

};

int main(){

    SLL list;

    int key;

    int value;

    cin>>key;

    list.insertatend(1);

    list.insertatend(2);

    list.insertatend(1);

    list.insertatend(2);

    list.insertatend(1);

    list.insertatend(3);

    list.insertatend(1);

    list.occurence(key);

    list.display();

}

3. Write a program to find the middle of a linked list.

Input: 1->2->3->4->5

Output: 3

#include <iostream>

using namespace std;

class Node{

     public:

    int data;

    Node\* next;

    Node(int data1){

        data=data1;

        next=nullptr;

    }

};

class SLL{

  private:

  Node\* head;

  public:

  SLL(){

    head=nullptr;

  }

  void insertatend(int data){

       Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

        } else {

            Node\* temp = head;

            while (temp->next != NULL) {

                temp = temp->next;

            }

            temp->next = newNode;

        }

    }

    void findmiddle(){

        if(head==NULL){

            cout<<"List is empty"<<endl;

        }

        Node\* x=head;

        Node\* y=head;

        while(y!=NULL && y->next!=NULL){

            x=x->next;

            y=y->next->next;

        }

        cout<<x->data <<" is the middle value"<<endl;

    }

};

int main(){

SLL list;

list.insertatend(1);

list.insertatend(2);

list.insertatend(3);

list.insertatend(4);

list.insertatend(5);

list.findmiddle();

*return* 0;

}

4. Write a program to reverse a linked list.

Input: 1->2->3->4->NULL

Output: 4->3->2->1->NULL

Ans:

#include <iostream>

using namespace std;

class Node{

     public:

    int data;

    Node\* next;

    Node(int data1){

        data=data1;

        next=nullptr;

    }

};

class SLL{

  private:

  Node\* head;

  public:

  SLL(){

    head=nullptr;

  }

  void insertatend(int data){

       Node\* newNode = new Node(data);

        if (head == NULL) {

            head = newNode;

        } else {

            Node\* temp = head;

            while (temp->next != NULL) {

                temp = temp->next;

            }

            temp->next = newNode;

        }

    }

    void reverse(){

        Node\* prev=nullptr;

        Node\* current=head;

        Node\* next=nullptr;

        while(current!=NULL){

            next=current->next;

            current->next=prev;

            prev=current;

            current=next;

        }

        head=prev;

    }

    void display(){

        Node\* temp=head;

       while(temp!=NULL){

        cout<<temp->data<<" ";

        temp=temp->next;

       }

       cout<<”NULL”<<endl;

    }

};

int main(){

SLL list;

list.insertatend(1);

list.insertatend(2);

list.insertatend(3);

list.insertatend(4);

list.reverse();

list.display();

*return* 0;

}