

Week 1 Lab A

1)

```
#include<iostream>
using namespace std;
struct node{
int data ;
struct node * next;
};
struct node* insert(struct node* head,int data){
    if(head==NULL){
        struct node* ptr=new struct node;
        ptr->data=data;
        ptr->next=NULL;
        head=ptr;
        return head;
    }
    else{
        struct node* ptr=new struct node;
        ptr->data=data;
        ptr->next=head;
        head=ptr;
        return head;
    }
}
struct node* insertatend(struct node* head, int data){
    struct node*p=new struct node;
    if(head==NULL)
    {
        p->data=data;
        p->next=NULL;
        return p;
    }
    struct node*ptr=head;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=p;
    p->data=data;
    p->next=NULL;
```

```

    return head;
}
void print(struct node* head){
    while(head!=NULL){
        cout<<head->data<<" ";
        head=head->next;
    }
}
int countnodes(struct node*head){
    int count=0,sum=0;
    struct node*ptr= head;
    while(ptr!=NULL){
        sum+=ptr->data;
        count++;
        ptr=ptr->next;
    }
    return count;
}
float average(struct node*head){
    int count=0,sum=0;
    struct node*ptr= head;
    while(ptr!=NULL){
        sum+=ptr->data;
        count++;
        ptr=ptr->next;
    }
    return ((float)sum/count);
}
void printfirst_m(struct node*head,int m){
    struct node*p=head;
    while(m!=0){
        cout<<p->data<<" ";
        p=p->next;
        m--;
    }
    cout<<endl;
}
int printelement(struct node*head, int a){
    struct node*ptr=head;
    while((a-1)!=0){
        ptr=ptr->next;
        a--;
    }
}

```

```

    }
    return ptr->data;
}
void middle(struct node*head){
    struct node*ptr=head;
    int a=countnodes(head);
    if (a%2!=0){
        int b=(a+1)/2;
        int x=pruntelement(head,b);
        if(x%2==0){
            cout<<"Middle element : " <<x<<" is even. "<<endl;
        }
        else{
            cout<<"Middle element : " <<x<<" is odd. "<<endl;
        }
    }
    else{
        int c=a/2;
        int d= c+1;
        int p=pruntelement(head,c);
        int q=pruntelement(head,d);
        if(p%2==0){
            cout<<"Middle elements are:\n " <<p<<" (even) & ";
        }
        else{
            cout<<"Middle elements are: " <<p<<" (odd) & ";
        }
        if(q%2==0){
            cout<<q<<" (even).\n ";
        }
        else{
            cout<<q<<" (odd).\n ";
        }
    }
}
void elementfromend(struct node* head, int a){
    struct node*ptr=head;
    int n= countnodes(head);
    int y=n-a;
    while(y!=0){
        ptr=ptr->next;
        y--;
    }
}

```

```

do{
    cout<<ptr->data<<" ";
    ptr=ptr->next;
}while(ptr!=NULL);
cout<<endl;
}
struct node* deleteelement(struct node * head, int value){
    struct node *p = head;
    struct node *q = head->next;
    if(p->data==value){
        head=p->next;
        free(p);
        return head;
    }
    while(q->data!=value && q->next!= NULL)
    {
        p = p->next;
        q = q->next;
    }

    if(q->data == value){
        p->next = q->next;
        free(q);
    }
    return head;
}
struct node* searchdelete(struct node*head, int f){
    struct node*ptr=head;
    while(ptr!=NULL){
        if(ptr->data==f){
            cout<<f<<" exists in the given linked list."<<endl;
            head = deleteelement(head,f);
            cout<<"Updated LL:\n";
            print(head);
            return head;
        }
        else{
            ptr=ptr->next;
        }
    }
    cout<<f<<" doesn't exist in the given LL.\n";
    return head;
}
int checkpair(struct node* head, int a, int b){

```

```

struct node*p=head;
struct node*q=head->next;
while(q->data!=b&&q->next!=NULL){
    p=p->next;
    q=q->next;
}
if(q->data==b&&p->data==a){
    return 1;}
return 0;
}
struct node* interchangepair(struct node*head,int a, int b, int c, int d)
{
    struct node*p=head;
    struct node*q=head->next;
    struct node*r=head;
    struct node*s=head->next;
    while(p->data!=a){
        p=p->next;
        q=q->next;
    }
    while(r->data!=c){
        r=r->next;
        s=s->next;
    }
    int temp1,temp2;
    temp1=p->data;
    p->data=r->data;
    r->data=temp1;
    temp2=q->data;
    q->data=s->data;
    s->data=temp2;
    return head;
}
int checksublist( struct node*h1, struct node*h2){
    int a=printelement(h2,1);
    int i=countnodes(h1);
    int j=countnodes(h2);
    struct node*p=h1;
    struct node*q=h2;
    for(int b=0;b<i;b++){

        if(p->data!=a){
            p=p->next;
            continue;

```

```

}
else{
    struct node*r=p;

    for(int c=0;c<j;c++){
        if(r->data==q->data)
        {
            r=r->next;
            q=q->next;
            continue;
        }
        else{
            q=h2;
            goto pin;
        }
    }
    cout<<"The given sub lists exists in the linked list at position "<<b+1<<endl;
    return b+1;
}
pin:
p=p->next;
}
cout<<"The given sub lists doesnt exists in the linked list . "<<endl;
return 0;
}
struct node* reverse(struct node*h){
int arr[countnodes(h)];
int i=0;
struct node*p= h;
while(p!=NULL){
arr[i]=p->data;
p=p->next;
i++;
}
p=h;
int j= countnodes(h)-1;
while(p!=NULL){
p->data=arr[j];
p=p->next;
j--;
}
return h;
}

```

```

struct node* updatereversedsublist(struct node*h1, struct node*h2, int g){
    int l=countnodes(h2);
    struct node*p=h1;
    struct node*q=h2;
    while((g-1)!=0)
    {
        p=p->next;
        g--;
    }
    while(l!=0){
        p->data=q->data;
        p=p->next;
        q=q->next;
        l--;
    }
    return h1;
}

```

```

int main()
{
    struct node*head;
    head=NULL;
    cout<<"Enter no. of elements to be inserted:\n";
    int a;
    cin>>a;
    int s=a;
    cout<<"Enter elements:\n";
    while(a!=0){
        int k=0;
        cin>>k;
        head=insert(head,k);
        a--;
    }
    print(head);
    cout<<endl;
    cout<<"The number of nodes:"<<countnodes(head)<<endl;
    cout<<"Average of nodes:"<<average(head)<<endl;
    cout<<endl;
    cout<<"Enter m for first m elements to be printed:\n";
    int m;
    cin>>m;
    if(m>s){
        cout<<"incorrect value of m\n";
    }
}

```

```

else{
    printfirst_m(head,m);
}
middle(head);
cout<<"Enter n for last n elements to be printed:\n";
int c;
cin>>c;
elementfromend(head,c);
cout<<endl<<"Enter a number to search and delete:\n";
int f;
cin>>f;
head=searchdelete(head,f);

a=s=m=c=0;
cout<<endl<<"Enter 1st pair:\n";
cin>>a>>s;
if(checkpair(head,a,s)){
    cout<<"Enter 2nd pair:\n";
    cin>>m>>c;
    if(checkpair(head,m,c)){
        head=interchange(head,a,s,m,c);
        print(head);
    }
    else{
        cout<<"Pair doesnt exist.\n";
    }
}
else{
    cout<<"Pair doesnt exist.\n";
}
cout<<"\nEnter No. of elements in sublist:\n";
cin>>a;
if(a>countnodes(head)){
    cout<<endl<<"The size of given sublist is more than parent list.\n";
}
else{
    struct node*h2= NULL;
    cout<<"Enter elements:\n";
    for(int i=0;i<a;i++)
    {
        cin>>s;
        h2=insertatend(h2,s);
    }
}

```



```
int g=checksublist(head,h2);  
h2=reverse(h2);  
head=updatereversedsublist(head,h2,g);  
print(head);  
}  
return 0;  
}
```

a)

```
Enter no. of elements to be inserted:  
7  
Enter elements:  
1  
4  
6  
7  
9  
2  
0  
0 2 9 7 6 4 1
```

b)

```
The number of nodes:7  
Average of nodes:4.14286
```

c)

```
Enter m for first m elements to be printed:  
4  
0 2 9 7
```

d)

```
Middle element : 7 is odd.
```

e)

```
Enter n for last n elements to be printed:  
3  
6 4 1
```

f)

```
Enter a number to search and delete:
0
0 exists in the given linked list.
Updated LL:
2 9 7 6 4 1
```

g)

```
Enter 1st pair:
2 9
Enter 2nd pair:
6 4
6 4 7 2 9 1
```

h)

```
Enter No. of elements in sublist:
3
Enter elements:
4 7 2
The given sub lists exists in the liinked list at position 2
```

i)

```
The given sub lists exists in the linked list at position 2
Updated LL with reversed sublist:
5 3 9 2 4 7 1 %
```

2)

```
#include<iostream>
#include<string.h>
using namespace std;
struct node
{
    string data;
    struct node *next;
```

```

};
void traversal(struct node *ptr)
{
    while(ptr!=NULL)
    {
        cout<<ptr->data<<" ";
        ptr = ptr->next;
        cout<<endl;
    }
}
struct node* insert(struct node* head,string data){
    if(head==NULL){
        struct node* ptr=new struct node;
        ptr->data=data;
        ptr->next=NULL;
        head=ptr;
        return head;
    }

    else{
        struct node* ptr=new struct node;
        ptr->data=data;
        ptr->next=head;
        head=ptr;
        return head;
    }
}

void alphabet(struct node *ptr, char data)
{
    while(ptr!=NULL)
    {
        if(ptr->data[0]==data)
        {
            cout<<ptr->data<<" ";
        }
        ptr = ptr->next;
    }
}
int exist(struct node *ptr, string data)
{
    while(ptr!=NULL)

```

```

{
    if(ptr->data==data)
    {
        return 1;
    }
    ptr = ptr->next;
}
return 0;
}

```

```

void max(struct node *ptr)

{ int max;

string data=ptr->data;
max=ptr->data.length();
while(ptr!=NULL)

{
    if(ptr->data.length()>max)
    {
        max=ptr->data.length();
        data=ptr->data;
    }
    ptr=ptr->next;
}
cout<<"The max length element is: "<<data<<endl;
}

```

```

void checksubstring(string a, struct node*head){
    int y;
    struct node*ptr=head;
    while(ptr!=NULL)
    {
        int k=ptr->data.length();
        int j=0;

        int count=0;
        for(int i=0;i<k;i++)
        {
            if(ptr->data[i]==a[j])

```

```

        {
            j++;
            count++;
            continue;
        }
        else{
            j=0;
            count=0;
        }

    }
    if(count==a.length()){
        cout<<"The given substring exists in "<<ptr->data<<endl;
        y++;
    }
    ptr=ptr->next;
    j=0;
}
if(y==0){cout<<"The given substring doesnt exists " <<endl;
}
return ;
}

```

```

int main()
{
    struct node* head;
    head=NULL;
    cout<<"Enter no. of elements to be inserted:\n";
    int a;
    cin>>a;
    int k=a;
    cout<<"Enter elements:\n";
    while(a!=0){
        string s;
        cin>>s;
        head=insert(head,s);
        a--;
    }
}

```

```

traversal(head);
char s;
cout<<"\nenter alphabet\n";
cin>>s;
alphabet(head,s);

```

```

string m;
cout<<"\nEnter string you want to find:";
cin>>m;
if(exist(head,m))
{
cout<<"It exists\n";
}
else{
    cout<<"It doesnot exist\n";
}
max(head);
string x;
cout<<"Enter substring:\n";
cin>>x;
checksubstring(x,head);
}

```

```

Enter no. of elements to be inserted:

```

```

5

```

```

Enter elements:

```

```

oggy

```

```

jack

```

```

bob

```

```

olly

```

```

crack

```

```

crack

```

```

olly

```

```

bob

```

```

jack

```

```

oggy

```

```

enter alphabet

```

```

o

```

```

olly oggy

```

```

Enter string you want to find:bob

```

```

It exists

```

```

The max length element is: crack

```

```

Enter substring:\nack

```

```

The given substring exists in crack

```

```

The given substring exists in jack

```

3)

```

#include <iostream>

```

```

#include <cmath>

```

```

using namespace std;
struct Node {
    int value;
    Node* next;

    Node(int val) : value(val), next(nullptr) {}
};

void insert(Node*& head, int value);
void printElements(Node* head);
int countElements(Node* head);
bool hasNegative(Node* head);
int countGreaterThan15(Node* head);
void deleteValue(Node*& head, int value);
void updateValue(Node* head, int oldValue, int newValue);
void insertAtPosition(Node*& head, int value, int position);
void deletePrimes(Node*& head);
void deleteFibonacci(Node*& head);
bool isPrime(int n);
bool isFibonacci(int num);
bool isPerfectSquare(int x);
void deleteNode(Node*& head, Node* target, Node* prev);

int main() {
    Node* head = nullptr;
    int n;
    cout << "Enter the number of elements to insert: ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        int value;
        cout << "Enter element " << i + 1 << ": ";
        cin >> value;
        insert(head, value);
    }
    cout << "Elements in list: ";
    printElements(head);
    cout << "Number of elements: " << countElements(head) << endl;

    cout << "List has negative value: " << (hasNegative(head) ? "Yes" : "No") << endl;

    cout << "Number of elements greater than 15: " << countGreaterThan15(head) << endl;

    int oldValue, newValue;
    cout << "Enter value to update: ";
    cin >> oldValue;
    cout << "Enter new value: ";
    cin >> newValue;
    updateValue(head, oldValue, newValue);
    cout << "List after updating " << oldValue << " to " << newValue << ": ";
    printElements(head);
}

```

```

    int position, insertValue;
    cout << "Enter position to insert new value: ";
    cin >> position;
    cout << "Enter value to insert: ";
    cin >> insertValue;
    insertAtPosition(head, insertValue, position);
    cout << "List after inserting " << insertValue << " at position " << position << ": ";
    printElements(head);

    int deleteValueInput;
    cout << "Enter value to delete: ";
    cin >> deleteValueInput;
    deleteValue(head, deleteValueInput);
    cout << "List after deleting " << deleteValueInput << ": ";
    printElements(head);

    deletePrimes(head);
    cout << "List after deleting prime numbers: ";
    printElements(head);

    deleteFibonacci(head);
    cout << "List after deleting Fibonacci numbers: ";
    printElements(head);

    return 0;
}

void insert(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        head->next = head;
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
}

void printElements(Node* head) {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->value << " ";
        temp = temp->next;
    }

```



```

    } while (temp != head);
    cout << endl;
}
int countElements(Node* head) {
    if (!head) return 0;
    int count = 0;
    Node* temp = head;
    do {
        count++;
        temp = temp->next;
    } while (temp != head);
    return count;
}
bool hasNegative(Node* head) {
    if (!head) return false;
    Node* temp = head;
    do {
        if (temp->value < 0) return true;
        temp = temp->next;
    } while (temp != head);
    return false;
}
int countGreaterThan15(Node* head) {
    if (!head) return 0;
    int count = 0;
    Node* temp = head;
    do {
        if (temp->value > 15) count++;
        temp = temp->next;
    } while (temp != head);
    return count;
}
void deleteValue(Node*& head, int value) {
    if (!head) return;

    Node* temp = head;
    Node* prev = nullptr;

    do {
        if (temp->value == value) {
            deleteNode(head, temp, prev);
            return;
        }
        prev = temp;
        temp = temp->next;
    } while (temp != head);
}
void updateValue(Node* head, int oldValue, int newValue) {
    if (!head) return;

```

```

Node* temp = head;
do {
    if (temp->value == oldValue) {
        temp->value = newValue;
        return;
    }
    temp = temp->next;
} while (temp != head);
}

void insertAtPosition(Node*& head, int value, int position) {
    if (position < 0) return;

    Node* newNode = new Node(value);

    if (!head) {
        if (position == 0) {
            head = newNode;
            head->next = head;
        }
        return;
    }

    if (position == 0) {
        newNode->next = head;
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        head = newNode;
        return;
    }

    Node* temp = head;
    int index = 0;

    while (temp->next != head && index < position - 1) {
        temp = temp->next;
        index++;
    }

    newNode->next = temp->next;
    temp->next = newNode;
}

void deletePrimes(Node*& head) {
    if (!head) return;

    Node* temp = head;

```

```

Node* prev = nullptr;

do {
    if (isPrime(temp->value)) {
        deleteNode(head, temp, prev);
        temp = prev ? prev->next : head;
    } else {
        prev = temp;
        temp = temp->next;
    }
} while (temp != head);
}

void deleteFibonacci(Node*& head) {
    if (!head) return;

    Node* temp = head;
    Node* prev = nullptr;

    do {
        if (isFibonacci(temp->value)) {
            deleteNode(head, temp, prev);
            temp = prev ? prev->next : head;
        } else {
            prev = temp;
            temp = temp->next;
        }
    } while (temp != head);
}

bool isPrime(int n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) return false;
    }
    return true;
}

bool isFibonacci(int num) {
    if (num < 0) return false;
    int x = 5 * num * num;
    return isPerfectSquare(x + 4) || isPerfectSquare(x - 4);
}

bool isPerfectSquare(int x) {
    int s = static_cast<int>(sqrt(x));
    return s * s == x;
}

void deleteNode(Node*& head, Node* target, Node* prev) {
    if (prev) {
        prev->next = target->next;
    }
}

```

```

    } else {
        if (target->next == head) {
            head = nullptr;
        } else {
            Node* last = head;
            while (last->next != head) {
                last = last->next;
            }
            last->next = target->next;
            head = target->next;
        }
    }
    delete target;
}

```

```

Enter the number of elements to insert: 7
Enter element 1: 11
Enter element 2: 22
Enter element 3: 33
Enter element 4: 44
Enter element 5: 55
Enter element 6: 66
Enter element 7: 77
Elements in list: 11 22 33 44 55 66 77
Number of elements: 7
List has negative value: No
Number of elements greater than 15: 6
Enter value to update: 66
Enter new value: 69
List after updating 66 to 69: 11 22 33 44 55 69 77
Enter position to insert new value: 5
Enter value to insert: 59
List after inserting 59 at position 5: 11 22 33 44 55 59 69 77
Enter value to delete: 59
List after deleting 59: 11 22 33 44 55 69 77
List after deleting prime numbers: 22 33 44 55 69 77
List after deleting Fibonacci numbers: 22 33 44 69 77

```

4)

```

#include <iostream>

using namespace std;

struct Node {
    int value;
    Node* next;
}

```

```

Node* prev;

Node(int val) : value(val), next(nullptr), prev(nullptr) {}

};

void insert(Node*& head, int value);

void printList(Node* head);

void traverseAndCheckDivisibility(Node* head, int m);

void deleteNodesGreaterThan(Node*& head, int x);

int countElementsBetweenDuplicates(Node* head, int value);

int main() {
    Node* head = nullptr;

    int n;

    cout << "Enter the number of elements to insert: ";

    cin >> n;

    for (int i = 0; i < n; i++) {
        int value;

        cout << "Enter element " << i + 1 << ": ";

        cin >> value;

        insert(head, value);
    }

    cout << "Doubly Linked List: ";

    printList(head);

    int m;

    cout << "Enter the number to check divisibility: ";

    cin >> m;

    cout << "Nodes divisible by " << m << ": ";

    traverseAndCheckDivisibility(head, m);

    int x;

    cout << "Enter the value to delete nodes greater than: ";

    cin >> x;

    deleteNodesGreaterThan(head, x);
}

```

```

    cout << "List after deleting nodes greater than " << x << ": ";
    printList(head);
    int duplicateValue;
    cout << "Enter the duplicate value to find elements between: ";
    cin >> duplicateValue;
    int count = countElementsBetweenDuplicates(head, duplicateValue);
    cout << "Number of elements between first pair of " << duplicateValue << " = " << count << endl;
    return 0;
}

void insert(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

void printList(Node* head) {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    while (temp) {
        cout << temp->value << " ";
    }
}

```

```

        temp = temp->next;
    }
    cout << endl;
}

void traverseAndCheckDivisibility(Node* head, int m) {
    if (!head) return;
    Node* temp = head;
    while (temp) {
        if (temp->value % m == 0) {
            cout << temp->value << " ";
        }
        temp = temp->next;
    }
    cout << endl;
}

void deleteNodesGreaterThan(Node*& head, int x) {
    Node* temp = head;
    while (temp) {
        Node* nextNode = temp->next;
        if (temp->value > x) {
            if (temp->prev) {
                temp->prev->next = temp->next;
            } else {
                head = temp->next;
            }
            if (temp->next) {
                temp->next->prev = temp->prev;
            }
            delete temp;
        }
    }
}

```

```

        temp = nextNode;
    }
}

int countElementsBetweenDuplicates(Node* head, int value) {
    if (!head) return 0;
    Node* first = nullptr;
    Node* second = nullptr;
    Node* temp = head;
    while (temp) {
        if (temp->value == value) {
            if (!first) {
                first = temp;
            } else if (!second) {
                second = temp;
                break;
            }
        }
        temp = temp->next;
    }
    if (!first || !second) return 0;
    int count = 0;
    temp = first->next;
    while (temp && temp != second) {
        count++;
        temp = temp->next;
    }
    return count;
}

```


Enter the number of elements to insert: 7

Enter element 1: 1

Enter element 2: 2

Enter element 3: 3

Enter element 4: 4

Enter element 5: 1

Enter element 6: 4

Enter element 7: 7

Doubly Linked List: 1 2 3 4 1 4 7

Enter the number to check divisibility: 2

Nodes divisible by 2: 2 4 4

Enter the value to delete nodes greater than: 6

List after deleting nodes greater than 6: 1 2 3 4 1 4

Enter the duplicate value to find elements between: 4

Number of elements between first pair of '4' = 1