15B17CI371 – Data Structures

Lab ODD 2024

Week 1-LAB B

Practice Lab

[CO: C270.1]

**1. Given a linked list, write functions to a. Insert an element at the beginning of the linked list. b. Insert an element at a specific location in a linked list. c. Take input an integer number, split the number in its digits and stores the digits in a linked list structure.**

#include <iostream>

#include <vector>

using namespace std;

struct Node

{

int data;

Node\* next;

Node(int val):data(val),next(nullptr){}

};

class LinkedList

{

Node\* head;

public:

LinkedList():head (nullptr){}

void insertAtBeginning(int data)

{

Node\* newNode=new Node(data);

newNode->next=head;

head=newNode;

}

void insertAtPosition(int data,int pos)

{

Node\* newNode=new Node(data);

if(pos==0)

{

newNode->next=head;

head=newNode;

}

else

{

Node\* temp=head;

for(int i=0;i<pos-1&&temp->next;i++)

temp=temp->next;

newNode->next=temp->next;

temp->next=newNode;

}

}

void printList()

{

cout<<"List :\n";

Node\* temp=head;

while(temp)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

void split\_store(int data)

{

while(data>0)

{

insertAtBeginning(data%10);

data/=10;

}

}

};

int main()

{

LinkedList list;

int val,pos;

list.insertAtBeginning(7);

list.insertAtBeginning(6);

list.insertAtBeginning(4);

list.insertAtBeginning(3);

list.insertAtBeginning(2);

list.printList();

cout<<"Input the value to be inserted in the beginning : ";

cin>>val;

list.insertAtBeginning(val);

list.printList();

cout<<"Input the position : ";

cin>>pos;

cout<<"Input the value to be inserted at the given position : ";

cin>>val;

list.insertAtPosition(val,pos);

list.printList();

cout<<"Input an integer : ";

cin>>val;

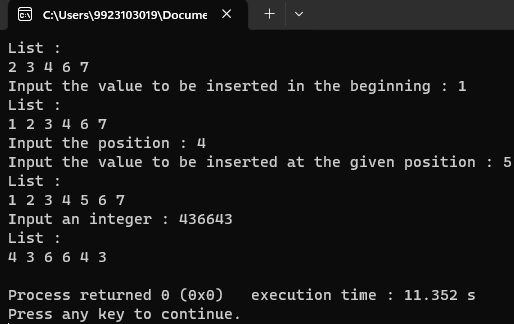
LinkedList list2;

list2.split\_store(val);

list2.printList();

}

**Output :**



**2. Write a program which reads a name and generates the link list of the characters**

**in that name. Later it removes the vowels from the link list and displays the**

**modified link list.**

#include <iostream>

#include <vector>

using namespace std;

struct Node

{

char data;

Node\* next;

Node(char val):data(val),next(nullptr){}

};

class LinkedList

{

Node\* head;

public:

LinkedList():head(nullptr){}

void insertAtBeginning(char data)

{

Node\* newNode=new Node(data);

newNode->next=head;

head=newNode;

}

void printList()

{

cout<<"List :\n";

Node\* temp=head;

while(temp)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

bool is\_vowel(char x)

{

if(x=='a'||x=='e'||x=='i'||x=='o'||x=='u'||x=='A'||x=='E'||x=='I'||x=='O'||x=='U')

return true;

return false;

}

void deleteNode()

{

Node\* temp=head;

Node\* prev=nullptr;

if(is\_vowel(temp->data))

{

head=temp->next;

delete temp;

return;

}

while(temp)

{

while(!is\_vowel(temp->data))

{

prev=temp;

temp=temp->next;

}

prev->next=temp->next;

Node\* temp2=temp;

temp=temp->next;

delete temp2;

}

}

};

int main()

{

LinkedList list;

string name;

cout<<"Input your name : ";

getline(cin,name);

int len=name.length();

for(int i=len-1;i>=0;i--)

list.insertAtBeginning(name[i]);

list.printList();

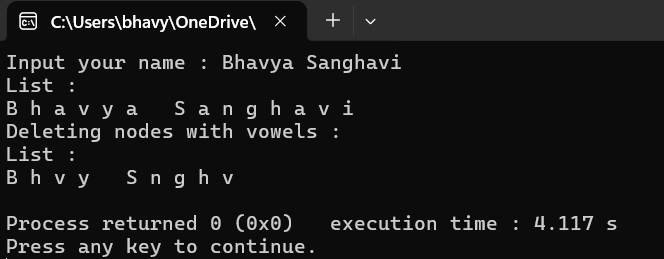
cout<<"Deleting nodes with vowels :\n";

list.deleteNode();

list.printList();

}

**Output :**

****

**3. Create a link list of users supplied ten characters to store a name. Create a**

**second link list of same type of user supplied five characters. Now using a**

**function remove(), traverse first link list and if any three consecutive characters**

**of second link list appears as consecutive characters of first link list, remove**

**those from first link list.**

#include <iostream>

#include <vector>

using namespace std;

struct Node

{

char data;

Node\* next;

Node(char val):data(val),next(nullptr){}

};

class LinkedList

{

Node\* head;

public:

LinkedList():head(nullptr){}

void insertAtBeginning(char data)

{

Node\* newNode=new Node(data);

newNode->next=head;

head=newNode;

}

void printList()

{

cout<<"List :\n";

Node\* temp=head;

while(temp)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

void remove(LinkedList& second)

{

Node\* temp1=head;

Node\* prev=nullptr;

while(temp1!=nullptr&&temp1->next!=nullptr&&temp1->next->next!=nullptr)

{

Node\* temp2=second.head;

while(temp2!=nullptr&&temp2->next!=nullptr&&temp2->next->next!=nullptr)

{

if(temp1->data==temp2->data&&temp1->next->data==temp2->next->data&&temp1->next->next->data==temp2->next->next->data)

{

Node\* t1=temp1;

Node\* t2=temp1->next;

Node\* t3=temp1->next->next;

if(prev==nullptr)

head=temp1->next->next->next;

else

prev->next=temp1->next->next->next;

temp1=temp1->next->next->next;

delete t1;

delete t2;

delete t3;

break;

}

temp2=temp2->next;

}

if(temp2==nullptr||temp2->next==nullptr||temp2->next->next==nullptr)

{

prev=temp1;

temp1=temp1->next;

}

}

}

};

int main()

{

LinkedList list1;

char ch;

cout<<"Input character for List 1 :\n";

for(int i=0;i<10;i++)

{

cin>>ch;

list1.insertAtBeginning(ch);

}

list1.printList();

LinkedList list2;

cout<<"Input character for List 2 :\n";

for(int i=0;i<5;i++)

{

cin>>ch;

list2.insertAtBeginning(ch);

}

list2.printList();

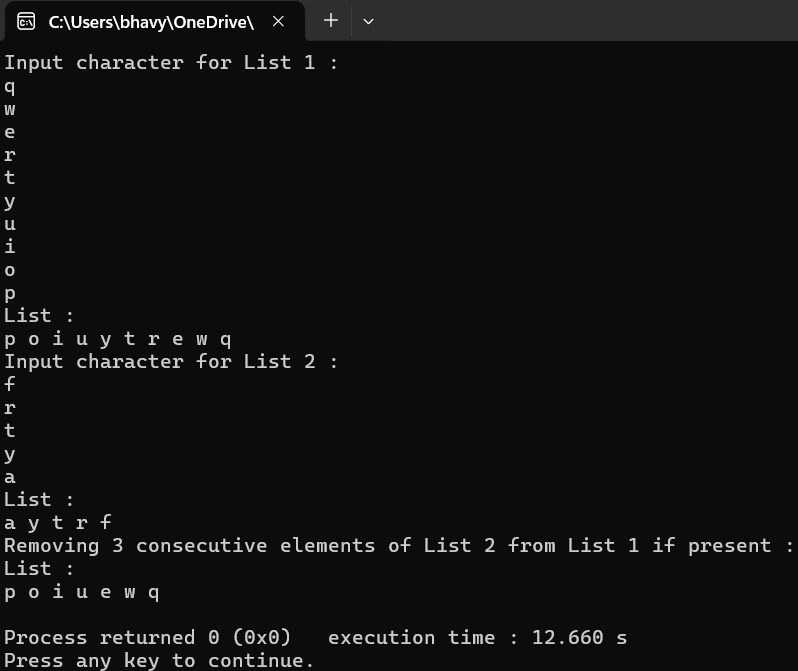
cout<<"Removing 3 consecutive elements of List 2 from List 1 if present :\n";

list1.remove(list2);

list1.printList();

}

**Output :**

****

**4. Write a program to insert an element at specific location in doubly linked list.**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* prev;

Node\* next;

Node(int val):data(val),prev(NULL),next(NULL){}

};

class DoublyLinkedList{

Node\* head;

Node\* tail;

public:

DoublyLinkedList():head(NULL),tail(NULL){}

void addNode(int val)

{

Node\* newNode=new Node(val);

if(tail==NULL)

head=tail=newNode;

else

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

}

void print()

{

cout<<"Doubly Linked List :\n";

Node\* temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

void insertAtPos(int val,int pos)

{

Node \*newNode=new Node(val);

if(pos==0)

{

newNode->next=head;

head->prev=newNode;

head=newNode;

}

else

{

Node\* temp=head;

for(int i=0;i<pos-1;i++)

temp=temp->next;

newNode->next=temp->next;

newNode->prev=temp;

temp->next->prev=newNode;

temp->next=newNode;

}

}

};

int main()

{

DoublyLinkedList dll;

int val,pos;

dll.addNode(1);

dll.addNode(2);

dll.addNode(4);

dll.addNode(5);

dll.addNode(6);

dll.print();

cout<<"Input the position to insert the value : ";

cin>>pos;

cout<<"Input the value : ";

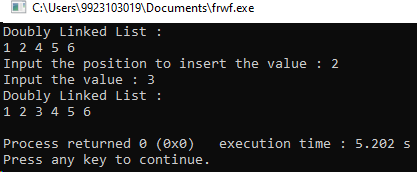
cin>>val;

dll.insertAtPos(val,pos);

dll.print();

}

**Output :**



**5. Write a program to delete last element from the doubly linked list.**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* prev;

Node\* next;

Node(int val):data(val),prev(NULL),next(NULL){}

};

class DoublyLinkedList{

Node\* head;

Node\* tail;

public:

DoublyLinkedList():head(NULL),tail(NULL){}

void addNode(int val)

{

Node\* newNode=new Node(val);

if(tail==NULL)

head=tail=newNode;

else

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

}

void print()

{

cout<<"Doubly Linked List :\n";

Node\* temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

void deleteFromEnd()

{

Node\* temp=tail;

tail=temp->prev;

tail->next=NULL;

delete temp;

}

};

int main()

{

DoublyLinkedList dll;

int val,pos;

dll.addNode(1);

dll.addNode(2);

dll.addNode(3);

dll.addNode(4);

dll.addNode(5);

dll.print();

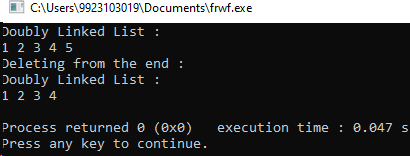
cout<<"Deleting from the end :\n";

dll.deleteFromEnd();

dll.print();

}

**Output :**

****

**6. Given a doubly linked list of any number of nodes, write a function ExtremeSwap(), which will swap values of the node at extreme pairs.**

#include <iostream>

using namespace std;

struct Node

{

int data;

Node\* prev;

Node\* next;

Node(int val):data(val),prev(NULL),next(NULL){}

};

class DoublyLinkedList{

Node\* head;

Node\* tail;

public:

DoublyLinkedList():head(NULL),tail(NULL){}

void addNode(int val)

{

Node\* newNode=new Node(val);

if(tail==NULL)

head=tail=newNode;

else

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

}

void print()

{

cout<<"Doubly Linked List :\n";

Node\* temp=head;

while(temp!=NULL)

{

cout<<temp->data<<" ";

temp=temp->next;

}

cout<<endl;

}

void ExtremeSwap()

{

int i=0;

Node\* x=head;

Node\* y=tail;

while(x!=y&&x->next!=y)

{

int temp=x->data;

x->data=y->data;

y->data=temp;

cout<<"After swapping "<<++i<<" times :\n";

print();

x=x->next;

y=y->prev;

}

if(x->next==y)

{

int temp=x->data;

x->data=y->data;

y->data=temp;

cout<<"After swapping "<<++i<<" times :\n";

print();

}

}

};

int main()

{

DoublyLinkedList dll;

int val,pos;

dll.addNode(1);

dll.addNode(2);

dll.addNode(3);

dll.addNode(4);

dll.addNode(5);

dll.addNode(6);

dll.addNode(7);

dll.addNode(8);

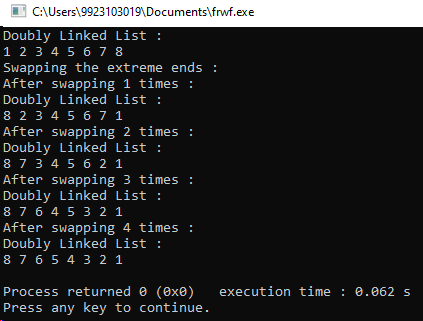
dll.print();

cout<<"Swapping the extreme ends :\n";

dll.ExtremeSwap();

}

**Output :**

****

**7. Write a program to implement addition of two polynomials. Each node must contain the value of the coefficient as well as its power as data components.**

#include <iostream>

using namespace std;

struct Node

{

int coeff,power;

Node\* prev;

Node\* next;

Node(int c,int p):coeff(c),power(p),prev(NULL),next(NULL){}

};

class polynomial{

Node\* head;

Node\* tail;

public:

polynomial():head(NULL),tail(NULL){}

void insertNode(int c,int p)

{

Node\* newNode=new Node(c,p);

if(tail==NULL)

head=tail=newNode;

else

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

}

void print()

{

Node\* temp=head;

while(temp!=NULL)

{

cout<<(temp->coeff>=0?"+":"")<<temp->coeff<<"x^"<<temp->power<<" ";

temp=temp->next;

}

cout<<endl<<endl;

}

polynomial operator+(polynomial& obj)

{

polynomial result;

Node\* p1=head;

Node\* p2=obj.head;

while(p1&&p2)

{

if(p1->power>p2->power)

{

result.insertNode(p1->coeff,p1->power);

p1=p1->next;

}

else if(p1->power<p2->power)

{

result.insertNode(p2->coeff,p2->power);

p2=p2->next;

}

else

{

result.insertNode(p1->coeff+p2->coeff,p1->power);

p1=p1->next;

p2=p2->next;

}

}

while(p1)

{

result.insertNode(p1->coeff,p1->power);

p1=p1->next;

}

while(p2)

{

result.insertNode(p2->coeff,p2->power);

p2=p2->next;

}

return result;

}

};

int main()

{

polynomial p1,p2,sum;

p1.insertNode(1,5);

p1.insertNode(2,3);

p1.insertNode(5,0);

cout<<"Polynomial 1:"<<endl;

p1.print();

p2.insertNode(2,4);

p2.insertNode(1,3);

p2.insertNode(4,1);

cout<<"Polynomial 2:"<<endl;

p2.print();

cout<<"Adding the two polnomials :\n";

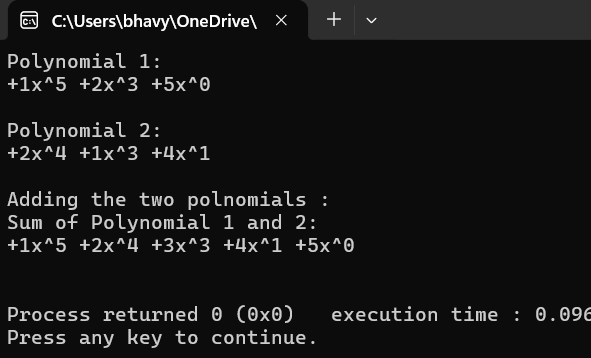
sum=p1+p2;

cout<<"Sum of Polynomial 1 and 2:"<<endl;

sum.print();

}

**Output :**

****

**8. Write a program to implement multiplication of two polynomials. Each node must**

**contain the value of the coefficient as well as its power as data components. Take**

**care of law of exponent multiplication.**

#include <iostream>

using namespace std;

struct Node

{

int coeff,power;

Node\* prev;

Node\* next;

Node(int c,int p):coeff(c),power(p),prev(NULL),next(NULL){}

};

class polynomial

{

Node\* head;

Node\* tail;

public:

polynomial():head(NULL),tail(NULL){}

void insertNode(int c,int p)

{

Node\* newNode=new Node(c,p);

if(tail==NULL)

head=tail=newNode;

else

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

}

void deleteAtPos(int c,int p)

{

Node\* temp=head;

while(temp)

{

if(temp->coeff==c&&temp->power==p)

{

temp->prev->next=temp->next;

delete temp;

break;

}

temp=temp->next;

}

}

void print()

{

Node\* temp=head;

while(temp!=NULL)

{

cout<<(temp->coeff>=0?"+":"")<<temp->coeff<<"x^"<<temp->power<<" ";

temp=temp->next;

}

cout<<endl<<endl;

}

void addTerm(int coeff,int power)

{

Node\* current=head;

while(current&&current->power>power)

current=current->next;

if(current&&current->power==power)

current->coeff+=coeff;

else

{

Node\* newNode=new Node(coeff, power);

if(!current)

{

if(tail)

{

tail->next=newNode;

newNode->prev=tail;

tail=newNode;

}

else

{

head=tail=newNode;

}

}

else if(current==head)

{

newNode->next=head;

head->prev=newNode;

head=newNode;

}

else

{

newNode->next=current;

newNode->prev=current->prev;

if(current->prev)

{

current->prev->next=newNode;

}

current->prev=newNode;

}

}

}

polynomial operator\*(polynomial& obj)

{

polynomial result;

Node\* p1=head;

Node\* p2=obj.head;

while(p1)

{

while(p2)

{

result.addTerm(p1->coeff\*p2->coeff,p1->power+p2->power);

p2=p2->next;

}

p2=obj.head;

p1=p1->next;

}

return result;

}

};

int main()

{

polynomial p1,p2,product;

p1.insertNode(1,5);

p1.insertNode(2,3);

p1.insertNode(5,0);

cout<<"Polynomial 1:"<<endl;

p1.print();

p2.insertNode(2,4);

p2.insertNode(1,3);

p2.insertNode(4,1);

cout<<"Polynomial 2:"<<endl;

p2.print();

cout<<"Multiplying the two polnomials :\n";

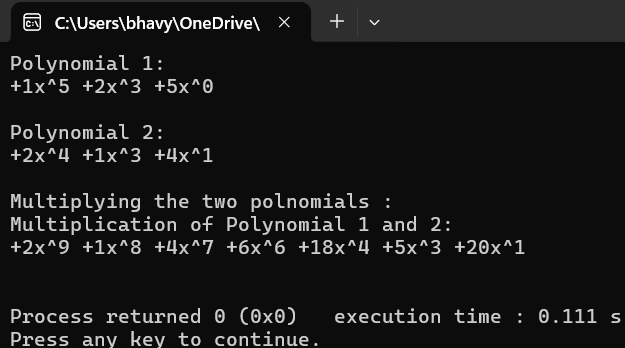
product=p1\*p2;

cout<<"Multiplication of Polynomial 1 and 2:"<<endl;

product.print();

}

**Output :**

****