Week5 Batch3

Thursday, December 30, 2021 2:40 PM

1. Write a menu driven program to perform the following on a doubly linked list
a) Insert an element at the rear end of the list
b) Delete an element from the rear end of the list
c) Insert an element at a given position of the list
d) Delete an element from a given position of the list
e) Insert an element after another element
f) Insert an element before another element
g) Print the list

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

class Node
{
public:
   int data;
   Node *next;
   Node *prev;
};

class DoublyLinkedList
{
private:
   Node *head, *tail;

public:
   // constructor
   DoublyLinkedList()
   {
      head = NULL;
      tail = NULL;
   }

   // To Insert Element at rear end of the list
   void Insert_End(int Ele)
   {
      Node *temp = new Node;
      temp->data = Ele;
      temp->next = NULL;

      // if dll empty
      if (head == NULL)
      {
         temp->prev = NULL;
         head = temp;
      }

      else
      {
         // if list not empty toh new Element ko tail bana,
```

```
A:\C++\DLL.exe
1. Insert an Element at the rear end of the list
2. Delete an Element from the rear end of the list
3. Insert an Element at a given index of the list
4. Delete a given from given index from the list
5. Insert an Element after another Element
6. Insert an Element before another Element
7. Print the list
8. Exit

Enter a choice: 1
Enter the Element to insert at end :1

Enter a choice: 1
Enter the Element to insert at end :2

Enter a choice: 1
Enter the Element to insert at end :3

Enter a choice: 7
1 2 3

Enter a choice: 2

Enter a choice: 7
1 2

Enter a choice: 5
Enter the Element after which insertion is to be do
Enter the Element to insert:3

Enter a choice: 7
1 3 2

Enter a choice: 2

Enter a choice: 7
1 3

Enter a choice: 8

--------------------------------
Process exited after 40.41 seconds with return valu
Press any key to continue . . .
```

```
        // then that tail ke next to temp, then new tail = temp
        temp->prev = tail;
        tail->next = temp;
    }
    tail = temp;
}

void Delete_End()
{
    Node *temp = new Node;
    temp = tail;
    // Set temp to tail
    // then check wheter ek hi Element to nhi

    if (temp->prev == NULL)
    {
        head = tail = NULL;
        delete temp;
        return;
    }
    // make the second last Element point to null before deleting
    temp->prev->next = NULL;
    tail = temp->prev; // Make tail as the 2nd last Element
    delete temp;      // delete last Element
    return;
}

void Insert_Pos(int Ele, int Loc)
{
    // We'll need two nodes, one for traversal to reach the position, and one for that actual new
Element

    Node *temp = new Node;
    Node *temp2=new Node;
    Node *trav;
    trav = head;

    temp->data = Ele;

    if (Loc == 0)
{
    temp->next = head;
    head = temp;

    return;
}
    int count = 0; // counter to check when we reach Location
    // If list is empty
    if (trav == NULL)
    {
        Insert_End(Ele);
    }

    // If list not empty, we'll run a while loop till trav!=null i.e till end of list and then increment count
every node, if count=Loc then break;then set the data, addresses  properly
    while (trav != NULL)
    {
        if (count == Loc)
        {
```

```
            break;
        }
        count++;
        trav = trav->next; // move on to next node
    }

    for (int i = 0; i<Loc-2; i++)
{
if(temp2==NULL)
{
Insert_End(Ele);
break;
}
    temp2 = temp2->next;
}
    // now trav will be at Location where we have to insert, so well have to insert temp before trav
    trav->prev->next = temp; // trav ke preious ka next points to temp
    temp->prev = trav->prev; // temp ka previous becomes trav ka previous
    temp->next = trav;      // temp ke next ko trav
    trav->prev = temp;      // trav ke prev ko temp
}

void Delete_Pos(int Loc)
{ // To delete from an index we'll only need a traversing node

    Node *trav;
    trav = head; // set it to head

    // if Element to be deleted is the head Element;
    if (Loc == 0)
    {
        head = trav->next; // new head
        delete trav;
        return;
    }

    // traversing to find Location, seen aboove

    if (trav == NULL)
    {
        Delete_End();
    }

    int count = 0;
    while (trav != NULL)
    {
        if (count == Loc)
        {
            break;
        }
        count++;
        trav = trav->next;
    }

    trav->prev->next = trav->next; // trav is now at Location, so to delete trav ke previous ka next will
be trav ka next
    trav->next->prev = trav->prev; // trav ke next ka prev will be trav
}
```

```
void Insert_After(int Ele, int val)
{
    Node *temp = new Node;
    Node *trav = new Node; // node to traverse

    temp->data = Ele;

    trav = head;
    bool found = false; // boolean for searching the Element

    // if list not empty
    while (trav != NULL)
    {
        if (val == trav->data) // if we find the Element
        {
            found = true;

            // checking if the Element is not last Element
            if (trav->next != NULL)
                trav->next->prev = temp; // trav ke k=next ka prev ko temp, matlab trav aur next ke bichme
temp ajayega

            else                    // if last Element
                tail = temp;        // make tail as temp
            temp->next = trav->next; // temp ka next ko trav ka next
            trav->next = temp;      // trav ka next ko temp, so temp is inserted after trav
            temp->prev = trav;      // temp pe prevv ko trav
            break;
            // Insert_End(Ele);
        }
        trav = trav->next;
    }

    if (!found)
    {
        cout << "Element not found" << endl;
    }
}

void Insert_Before(int Ele, int val)
{ // this is nearly similar to insert after, we just traverse from end and add after use similar logic as
insert after, we do this because previous pointer is available in DLL

    Node *temp = new Node;
    Node *trav = new Node;
    temp->data = Ele;
    trav = tail;
    bool found = false;
    while (trav != NULL)
    {
        if (val == trav->data)
        {
            found = true;
            if (trav->prev != NULL)
                trav->prev->next = temp;
            else
                head = temp;
            temp->prev = trav->prev;
            trav->prev = temp;
```

```
            temp->next = trav;
            break;
          }
          trav = trav->prev;
        }
        if (!found)
        {
          cout << "Element not found" << endl;
        }
    }

    void Display()
    {
        Node *trav = new Node;

        trav = head;
        while (trav != NULL)
        {
          cout << trav->data << ' ';
          trav = trav->next;
        }
        cout << endl;
    }
};

int main()
{
    DoublyLinkedList D;
    int Choose;
    int data;
    int Loc, Ele;
    cout << "1. Insert an Element at the rear end of the list" << endl;
    cout << "2. Delete an Element from the rear end of the list" << endl;
    cout << "3. Insert an Element at a given index of the list" << endl;
    cout << "4. Delete a given from given index from the list" << endl;
    cout << "5. Insert an Element after another Element " << endl;
    cout << "6. Insert an Element before another Element " << endl;
    cout << "7. Print the list" << endl;
    cout << "8. Exit" << endl;

    while (1)
    {
        cout << "\nEnter a choice: ";
        cin >> Choose;
        switch (Choose)
        {
        case 1:
          cout << "Enter the Element to insert at end :";
          cin >> Ele;
          D.Insert_End(Ele);
          break;
        case 2:
          D.Delete_End();
          break;
        case 3:
          cout << "Enter the index of the Element to be inserted:";
          cin >> Loc;
          cout << "Enter the Element to insert:";
          cin >> Ele;
```

```
            D.Insert_Pos(Ele, Loc);
            break;
        case 4:
            cout << "Enter the index of the Element to be deleted: ";
            cin >> Loc;
            D.Delete_Pos(Loc);
            break;
        case 5:
            cout << "Enter the Element after which insertion is to be done :";
            cin >> Loc;
            cout << "Enter the Element to insert:";
            cin >> Ele;
            D.Insert_After(Ele, Loc);
            break;
        case 6:
            cout << "Enter the Element before which insertion is to be done :";
            cin >> Loc;
            cout << "Enter the Element to insert:";
            cin >> Ele;
            D.Insert_Before(Ele, Loc);
            break;
        case 7:
            D.Display();
            break;

        case 8:
            exit(0);
        default:
            cout << "Invalid Choice!" << endl;
            break;
        }
    }
}
```

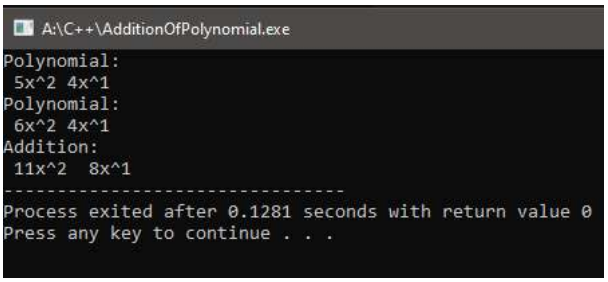
2. Write a program to add two polynomials using doubly linked list.

```
#include <bits/stdc++.h>
#include<iostream>
using namespace std;

class Node {
public:
    int coeff, power;
    Node* next;
    Node(int coeff, int power) {
        this->coeff = coeff;
        this->power = power;
        this->next = NULL;
    }
};

void addPolynomials(Node* head1, Node* head2) {

    if (head1 == NULL && head2 == NULL)
        return;
```



```
    else if (head1->power == head2->power) {
        cout << " " << head1->coeff + head2->coeff << "x^" << head1->power << " ";
        addPolynomials(head1->next, head2->next);
    }
    else if (head1->power > head2->power) {
        cout << " " << head1->coeff << "x^" << head1->power << " ";
        addPolynomials(head1->next, head2);
    }
    else {
        cout << " " << head2->coeff << "x^" << head2->power << " ";
        addPolynomials(head1, head2->next);
    }
}

void insert(Node* head, int coeff, int power) {
    Node* new_node = new Node(coeff, power);
    while (head->next != NULL) {
        head = head->next;
    }
    head->next = new_node;
}

void printList(Node* head) {
    cout << "Polynomial: " << endl;
    while (head != NULL) {
        cout << " " << head->coeff << "x" << "^" << head->power;
        head = head->next;
    }
}

int main() {

    Node* head = new Node(5, 2);
    insert(head, 4, 1);
    Node* head2 = new Node(6, 2);
    insert(head2, 4, 1);
    printList(head);
    cout << endl;
    printList(head2);
    cout << endl << "Addition:" << endl;
    addPolynomials(head, head2);


    return 0;
}
```