Assignment no 1

Data structures

20p-0480

Q1: Write a program that creates an even number of links in a linked list and then finds the middle of the link list and insert a data item in the list.

Code:

#include<iostream>

using namespace std;

class node

{

    public:

        node \*next;

        int data;

    node(int val)

    {

        data=val;

        next=NULL;

    }

};

class linkedlist

{

    private:

        node \*head;

        int length;

    public:

    linkedlist()

    {

        head=NULL;

        length=0;

    }

    void insert(int val, int pos)

{

    if(pos<1 || pos>length+1)

    {

        cout<<"Invalid position."<<endl;

        return;

    }

    node \*t=new node(val);

    if(pos==1)

    {

        t->next=head;

        head=t;

        length ++;

    }

    else

    {

        node \*t=new node(val);

        node \*curr=head;

        for(int i=1; i<pos-1; i++)

        {

            curr=curr->next;

        }

        t->next=curr->next;

        curr->next=t;

        length ++;

    }

}

void print()

{

    node \*temp=head;

    while(temp!=NULL)

    {

        cout<<temp->data<<" ";

        temp=temp->next;

    }

}

//for insertion in middle in link list

void midInsert(int val)

{

    // here we divide the length by 2 to find the mid pos

    int pos=length/2;

    // insert the value after the pos ,So we will increment the length

    pos++;

    //here using  previous function of insert instead of making new function and using updated pos it insert it in middle

    insert(val,pos);

}

};

int main()

{

    linkedlist obj;

    obj.insert(1,1);

    obj.insert(2,2);

    obj.insert(3,3);

    obj.insert(4,4);

    obj.print();

    cout<<endl;

    obj.midInsert(7);

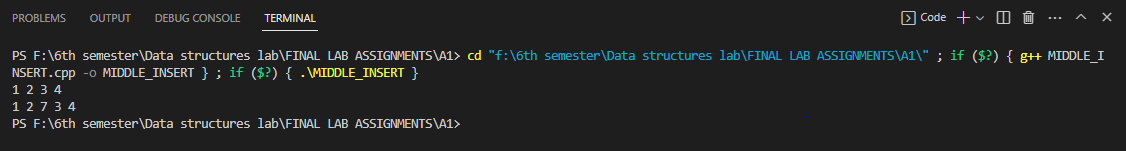
    obj.print();

    cout<<endl;

    return 0;

}

Output:



Q2: Write a function to check whether the given Singly Linked List is Palindrome or not.

Code:

#include<iostream>

using namespace std;

class node

{

    public:

        node \*next;

        int data;

    node(int val)

    {

        data=val;

        next=NULL;

    }

};

class linkedlist

{

    private:

        node \*head;

        int length;

    public:

    linkedlist()

    {

        head=NULL;

        length=0;

    }

    void Palindrome(linkedlist a, linkedlist b)

{

    node \*curr=a.head;

    node \* curr1=NULL;

    //first list in reverse order

    while(curr!=NULL)

    {

        node \*n=new node(curr->data);

        n->next=b.head;

        b.head=n;

        curr=curr->next;

    }

    node \*cp1=b.head;

    node \*cp=a.head;

    while(cp!=NULL)

    {

        if(cp->data!=cp1->data)

        {

            cout<<"Linklist is  not a palindrome."<<endl;

            // cout<<cp1->info<<" "<<cp->info<<endl;

            return;

        }

        cp=cp->next;

        cp1=cp1->next;

    }

    cout<<"Link list is a palindrome."<<endl;

}

void insert(int val, int pos)

{

    if(pos<1 || pos>length+1)

    {

        cout<<"Invalid position."<<endl;

        return;

    }

    node \*n=new node(val);

    if(pos==1)

    {

        n->next=head;

        head=n;

        length ++;

    }

    else

    {

        node \*n=new node(val);

        node \*curr=head;

        for(int i=1; i<(pos-1); i++)

        {

            curr=curr->next;

        }

        n->next=curr->next;

        curr->next=n;

        length ++;

    }

}

void print()

{

    node \*n=head;

    while(n!=NULL)

    {

        cout<<n->data<<" ";

        n=n->next;

    }

}

};

int main()

{

    linkedlist l;

    linkedlist l1;

    l.insert(1,1);

    l.insert(7,2);

    l.insert(1,3);

    l.print();

    cout<<endl;

    l.Palindrome(l, l1);

    linkedlist l2;

    linkedlist l3;

    cout<<endl;

    l2.insert(5,1);

    l2.insert(4,2);

    l2.insert(2,3);

    l2.print();

    cout<<endl;

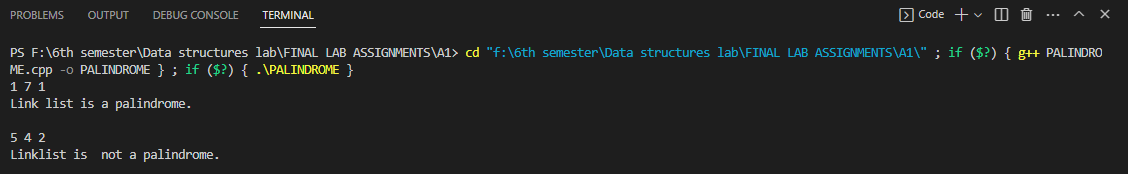
    l2.Palindrome(l2, l3);

    cout<<endl;

    return 0;

}

Output:



Q3: Write a removeDuplicates() function which takes a list sorted in increasing order and deletes any

duplicate nodes from the list. Ideally, the list should only be traversed once.

Code:

#include<iostream>

using namespace std;

class node

{

    public:

        node \*next;

        int data;

    node(int val)

    {

        data=val;

        next=NULL;

    }

};

class linkedlist

{

    private:

        node \*head;

        int length;

    public:

    linkedlist()

    {

        head=NULL;

        length=0;

    }

    void insert(int val, int pos)

{

    if(pos<1 || pos>length+1)

    {

        cout<<"Invalid position."<<endl;

        return;

    }

    node \*n=new node(val);

    if(pos==1)

    {

        n->next=head;

        head=n;

        length ++;

    }

    else

    {

        node \*n=new node(val);

        node \*curr=head;

        for(int i=1; i<(pos-1); i++)

        {

            curr=curr->next;

        }

        n->next=curr->next;

        curr->next=n;

        length ++;

    }

}

void print()

{

    node \*n=head;

    while(n!=NULL)

    {

        cout<<n->data<<" ";

        n=n->next;

    }

}

void RemoveDuplicates()

{

    //sorting

    for(int i=0; i<length; i++)

    {

        node \*temp=head;

        node \*temp1=temp->next;

        while(temp->next!=NULL)

        {

            if(temp->data>temp1->data)

            {

                // swapping

                int v=temp->data;

                temp->data=temp1->data;

                temp1->data=v;

            }

            temp=temp1;

            temp1=temp->next;

        }

    }

    node \*temp=head;

    //traversing

    while(temp->next!=NULL)

    {

        if(temp->data==temp->next->data)

        {

            if(temp==head)

            {

                head=head->next;

                node \*n=temp;

                temp=head;

                delete n;

            }

            else

            {

                node \*n=temp->next;

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

                delete n;

            }

        }

        else

        {

            temp=temp->next;

        }

    }

}

void remove(int p)

{

   if(head==NULL)

   {

       cout<<"List is empty."<<endl;

       return;

   }

   if(p>length)

   {

       cout<<"Invalid position."<<endl;

       return;

   }

   node \*prev=NULL;

   node \*curr=head;

   if(p==1)

   {

       head=head->next;

   }

   else

   {

       for(int i=1; i<p; i++)

       {

           prev=curr;

           curr=curr->next;

       }

       prev->next=curr->next;

   }

   delete curr;

   length--;

}

};

int main()

{

    linkedlist l , l1;

    l.insert(7,1);

    l.insert(10,2);

    l.insert(4,3);

    l.insert(7,4);

    l.insert(1,5);

    l.insert(4,6);

    l.print();

    l.RemoveDuplicates();

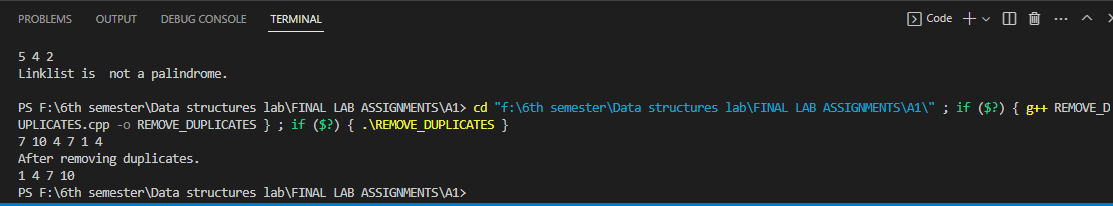
    cout<<"\nAfter removing duplicates."<<endl;

    l.print();

    return 0;

}

Output:

Q4: Write a function to swap the adjacent nodes of the linked list by swapping

the links not there values

Code:

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* next;

};

void swapNodes(int lPos, int rPos, Node\* head) {

    if (lPos == rPos) {

        return;

    }

    Node\* prevL = NULL;

    Node\* currL = head;

    for (int i = 1; i < lPos && currL != NULL; i++) {

        prevL = currL;

        currL = currL->next;

    }

    Node\* prevR = NULL;

    Node\* currR = head;

    for (int i = 1; i < rPos && currR != NULL; i++) {

        prevR = currR;

        currR = currR->next;

    }

    if (currL == NULL || currR == NULL) {

        return;

    }

    if (prevL != NULL) {

        prevL->next = currR;

    } else {

        head = currR;

    }

    if (prevR != NULL) {

        prevR->next = currL;

    } else {

        head = currL;

    }

    Node\* temp = currR->next;

    currR->next = currL->next;

    currL->next = temp;

}

int main() {

    // create a sample linked list

    Node\* head = new Node{1, new Node{2, new Node{3, new Node{4, new Node{5, NULL}}}}};

    // print the original linked list

    Node\* curr = head;

    cout<<"Before  Linked List"<<endl;

    while (curr != NULL) {

        cout <<curr->data << "  ";

        curr = curr->next;

    }

    cout << endl;

cout << endl;

    // swap nodes at positions 2 and 5

    swapNodes(2, 5, head);

    // print the modified linked list

    cout<<"After  the  swaping link list "<<endl;

    curr = head;

    while (curr != NULL) {

        cout << curr->data << " ";

        curr = curr->next;

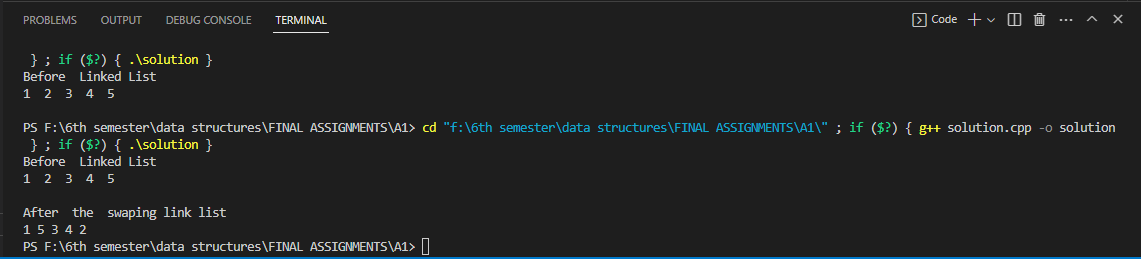
    }

    cout << endl;

    return 0;

}

Output:



THE END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_