1. Linkedlist

#include <stdio.h>

#include<bits/stdc++.h>

using namespace std;

class Node{

public:

int val;

Node \*next;

Node(){

this->val = INT\_MIN;

this->next = NULL;

}

Node(int x){

val = x;

this->next = NULL;

}

};

class LL{

Node \*head;

public:

LL(){

this->head = NULL;

}

void insertAtBegin(int x){

Node \*n1 = new Node(x);

n1->next = head;

head = n1;

}

void insertAtPos(int x, int pos){

int n = getSize();

if(pos>n || pos<0){return ;}

if(pos==0){insertAtBegin(x); return ;}

Node \*p1 = head;

while(--pos){

p1 = p1->next;

}

Node \*n1 = new Node(x);

n1->next = p1->next;

p1->next = n1;

}

void insertAtEnd(int x){

Node \*n1 = new Node(x);

if(head==NULL){

head=n1;

return ;

}

Node \*temp = head;

while(temp->next){

temp = temp->next;

}

temp->next = n1;

}

void deleteBegin(){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

head = head->next;

}

void deletValue(int x){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

Node \*prev=NULL , \*curr=head;

bool flag=true;

while(curr){

if(curr->val == x){

flag=false;

Node \*temp = curr;

prev->next = curr->next;

curr->next = prev;

// delete(curr);

// delete(temp);

}

prev = curr;

curr = curr->next;

}

if(flag){

cout<<"There exist no element which is asked to delete.\n";

}else{

cout<<"Deleted\n";

}

}

void deleteAtPos(int x, int pos){

int n = getSize();

if(pos>n || pos<=0){return ;}

if(pos==1){deleteBegin(); return ;}

Node \*p1 = head, \*prev=nullptr;

while(--pos){

prev = p1;

p1 = p1->next;

}

if(p1->val==x){

prev->next = p1->next;

free(p1);

}

}

void deleteEnd(){

if(!head){

cout<<"Underflow: No data to delete.";

return ;

}

Node \*temp = head;

while(temp->next->next){

temp = temp->next;

}

temp->next = NULL;

}

void sorting(){

Node \*temp1 = head, \*temp2 = head;

while(temp1->next){

temp2 = temp1->next;

while(temp2){

if(temp2->val < temp1->val){

int temp = temp2->val;

temp2->val = temp1->val;

temp1->val = temp;

}

temp2 = temp2->next;

}

temp1 = temp1->next;

}

}

void printList(){

if(!head){

cout<<"Underflow: No data to print.";

return ;

}

Node \*temp = head;

while(temp){

cout<<temp->val<<"->";

temp = temp->next;

}

}

Node\* getHead(){return this->head;}

int getSize(){

if(!head)return 0;

if(!head->next)return 1;

Node \*curr = head;

int ans=0;

while(curr){

curr = curr->next;

ans++;

}

return ans;

}

};

int main(){

LL l1;

l1.insertAtEnd(11); //

l1.insertAtEnd(13); //

l1.insertAtEnd(12); //

l1.insertAtEnd(19); //

l1.insertAtEnd(12); //

l1.insertAtEnd(13);

l1.insertAtPos(999,1);

l1.deleteAtPos(12,2);

l1.printList();

return 0;

}

1. Add 2 ll

#include <iostream>

using namespace std;

struct ListNode {

int val;

ListNode\* next;

ListNode(int x) : val(x), next(NULL) {}

};

ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {

ListNode\* dummy = new ListNode(0);

ListNode\* current = dummy;

int carry = 0;

while (l1 || l2 || carry) {

int sum = (l1 ? l1->val : 0) + (l2 ? l2->val : 0) + carry;

carry = sum / 10;

current->next = new ListNode(sum % 10);

current = current->next;

if (l1) {

l1 = l1->next;

}

if (l2) {

l2 = l2->next;

}

}

return dummy->next;

}

// Sample usage

int main() {

ListNode\* l1 = new ListNode(7);

l1->next = new ListNode(1);

l1->next->next = new ListNode(5);

ListNode\* l2 = new ListNode(5);

l2->next = new ListNode(9);

l2->next->next = new ListNode(2);

ListNode\* result = addTwoNumbers(l1, l2);

while (result) {

cout << result->val << "->";

result = result->next;

}

cout << "NULL\n";

return 0;

}

1. Palindrome

#include <bits/stdc++.h>

using namespace std;

class Node {

int data;

Node next;

Node(int value) {

data = value;

next = null;

}

}

public class PalindromeUsingReverse

{

Node head;

Node secondHalf = head;

// Insertion at Last

public void insertAtLast(int data)

{

// Make a new node

Node newNode = new Node(data);

// if this is the first node

if(head == null)

{

head = newNode;

return;

}

newNode.next = null;

Node temp = head;

while(temp.next != null)

{

temp = temp.next;

}

temp.next = newNode;

//return;

}

// A utility function to print the Linked List

public void printList(Node head)

{

System.out.println("Printing the Linked List");

Node temp = head;

while(temp != null)

{

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

// To check if Linked List is palindrome

boolean isPalindrome(Node head)

{

// This will move by one step

Node slow = head;

// This will move by two steps

Node fast = head;

// This will keep track of the node previous to

// the node pointed by slow

Node prev\_of\_slow = head;

/\*

In case of odd sized lists, the middle element

need not to be a part of the second half. So making

a separate variable to store it in case of odd-sized

lists. In even sized lists,this will be null

\*/

Node midNode = null;

boolean result = true;

// Proceeding further iff the List has atleast two elements

// This is checked by the following condition specified in t

// the if clause

if(head != null && head.next != null)

{

// STEP 1: FINDING THE MIDDLE ELEMENT

while(fast != null && fast.next != null)

{

fast = fast.next.next;

prev\_of\_slow = slow;

slow = slow.next;

}

/\* fast would become NULL when there are even elements

in the list and not NULL for odd elements.

the middle node is to be skipped for odd case

and store it somewhere so that the original list

can be restored

\*/

// Storing the middle element for odd size lists

if(fast != null)

{

midNode = slow;

slow = slow.next;

}

// Now regardless of odd or even elements

// the slow pointer would point to the starting

// of the second half of list

secondHalf = slow;

prev\_of\_slow.next = null;

// STEP 2: Reverse the second half

reverseList();

// STEP 3: Comparing the reverse of second half

// with the first half

result = compareList(head, secondHalf);

/\*

STEP 4: Constructing the original linked list back

1) Reverse the second half again.

2) If the list was odd sized, then the midNode will not be Null

The prev\_of\_slow.next will point to the midNode. The secondHalf will contain

the elements next to middle node

3) If the list was even sized, then the midNode will be null. The prev\_of\_slow

will point to the secondHalf.

\*/

reverseList();

if(midNode != null)

{

prev\_of\_slow = midNode;

midNode.next = secondHalf;

}

else{

prev\_of\_slow.next = secondHalf;

}

}

return result;

}

/\* Function to reverse the linked list \*/

void reverseList()

{

Node prev = null;

Node current = secondHalf;

Node next;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

secondHalf = prev;

}

/\* Function to check if two input lists have same data\*/

boolean compareList(Node head1, Node head2)

{

Node temp1 = head1;

Node temp2 = head2;

while (temp1 != null && temp2 != null) {

if (temp1.data == temp2.data) {

temp1 = temp1.next;

temp2 = temp2.next;

}

else

return false;

}

if (temp1 == null && temp2 == null)

return true;

/\* Will reach here when one is NUll and other is not \*/

return false;

}

public static void main(String[]args)

{

PalindromeUsingReverse ll = new PalindromeUsingReverse();

// 1->Null

ll.head = new Node(1);

// 1->2->Null

ll.insertAtLast(2);

// 1->2->1->Null

ll.insertAtLast(1);

// 1->2->1->2->Null

ll.insertAtLast(2);

// 1->2->1->2->3->Null

ll.insertAtLast(3);

ll.printList(ll.head);

if(ll.isPalindrome(ll.head))

System.out.println("Palindrome Linked List");

else

System.out.println("Not a Palindrome Linked List");

}

}

1. Text editor

#include <iostream>

#include <string>

using namespace std;

int main() {

string text;

cout << "Enter the text: ";

getline(cin, text);

string search, replace;

cout << "Enter the substring to search: ";

getline(cin, search);

cout << "Enter the substring to replace with: ";

getline(cin, replace);

size\_t pos = 0;

while ((pos = text.find(search, pos)) != string::npos) {

text.replace(pos, search.length(), replace);

pos += replace.length();

}

cout << "Modified text: " << text << endl;

return 0;

}