1) #include <stdio.h>

#include <string.h> #define MAX\_CHAR 256

void characterFrequency(const char \*s) { int frequency[MAX\_CHAR] = {0};

for (int i = 0; s[i] != '\0'; i++) { frequency[(unsigned char)s[i]]++;

}

for (int i = 0; i < MAX\_CHAR; i++) {

if (frequency[i] > 0) { printf("%c -> %d\n", i, frequency[i]);

}

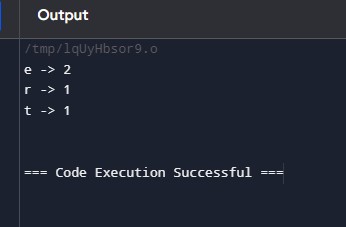
}

}

int main() { const char \*s = "tree"; characterFrequency(s); return 0;

}

Output:



2) #include <stdio.h>

#include <stdlib.h> int segregate(int arr[], int size) { int j = 0; for (int i = 0; i < size; i++) { if (arr[i] <= 0) { int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

j++;

}

}

return j;

}

int findMissingPositive(int arr[], int size) { for (int i = 0; i < size; i++) { int absVal = abs(arr[i]);

if (absVal - 1 < size && arr[absVal - 1] > 0) {

arr[absVal - 1] = -arr[absVal - 1];

}

}

for (int i = 0; i < size; i++) {

if (arr[i] > 0) { return i + 1;

}

}

return size + 1;

}

int findSmallestMissingPositive(int arr[], int size) { int shift = segregate(arr, size); return findMissingPositive(arr + shift, size - shift);

}

int main() { int arr1[] = {2, 3, 7, 6, 8, -1, -10, 15}; int size1 = sizeof(arr1) / sizeof(arr1[0]);

printf("The smallest missing positive number is %d\n", findSmallestMissingPositive(arr1,

size1));

int arr2[] = {2, 3, -7, 6, 8, 1, -10, 15}; int size2 = sizeof(arr2) / sizeof(arr2[0]);

printf("The smallest missing positive number is %d\n", findSmallestMissingPositive(arr2,

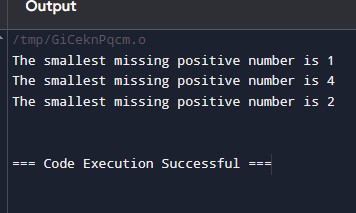
size2)); int arr3[] = {1, 1, 0, -1, -2}; int size3 = sizeof(arr3) / sizeof(arr3[0]);

printf("The smallest missing positive number is %d\n", findSmallestMissingPositive(arr3,

size3)); return 0;

}

Output:



3) #include <stdio.h>

#include <stdlib.h> struct TreeNode {

int val;

struct TreeNode \*left; struct TreeNode \*right; };

int search(int \*inorder, int start, int end, int value) { for (int i = start; i <= end; i++) { if (inorder[i] == value) { return i;

}

}

return -1;

}

struct TreeNode\* buildTreeHelper(int\* preorder, int\* inorder, int inorderStart, int inorderEnd, int\* preorderIndex) { if (inorderStart > inorderEnd) { return NULL;

}

int rootValue = preorder[\*preorderIndex];

(\*preorderIndex)++; struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode)); root->val = rootValue; root->left = NULL; root->right = NULL; if (inorderStart == inorderEnd) { return root;

}

int inorderIndex = search(inorder, inorderStart, inorderEnd, rootValue); root->left = buildTreeHelper(preorder, inorder, inorderStart, inorderIndex - 1, preorderIndex);

root->right = buildTreeHelper(preorder, inorder, inorderIndex + 1, inorderEnd,

preorderIndex); return root;

}

struct TreeNode\* buildTree(int\* preorder, int preorderSize, int\* inorder, int inorderSize) { int preorderIndex = 0; return buildTreeHelper(preorder, inorder, 0, inorderSize - 1, &preorderIndex);

}

void printLevelOrder(struct TreeNode\* root) {

if (!root) return;

struct TreeNode\* queue[100]; int front = 0, rear = 0; queue[rear++] = root; while (front < rear) { struct TreeNode\* currentNode = queue[front++]; if (currentNode) { printf("%d ", currentNode->val); queue[rear++] = currentNode->left; queue[rear++] = currentNode->right;

} else {

printf("null ");

}

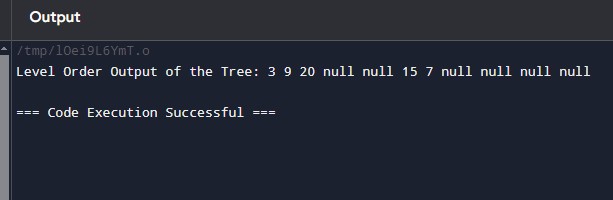
}

}

int main() { int preorder[] = {3, 9, 20, 15, 7}; int inorder[] = {9, 3, 15, 20, 7}; int preorderSize = sizeof(preorder) / sizeof(preorder[0]); int inorderSize = sizeof(inorder) / sizeof(inorder[0]); struct TreeNode\* root = buildTree(preorder, preorderSize, inorder, inorderSize); printf("Level Order Output of the Tree: "); printLevelOrder(root); return 0;

}

Output:



4) #include <stdio.h>

#include <stdlib.h> struct Node {

int data; struct Node\* next;

};

struct Node\* createNode(int data) { struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = data; newNode->next = NULL; return newNode;

}

void appendNode(struct Node\*\* head, int data) { struct Node\* newNode = createNode(data); if (\*head == NULL) {

\*head = newNode;

} else { struct Node\* temp = \*head; while (temp->next != NULL) { temp = temp->next;

}

temp->next = newNode;

}

}

void displayList(struct Node\* head) { struct Node\* temp = head; while (temp != NULL) {

printf("%d", temp->data);

temp = temp->next; if (temp != NULL) {

printf("->");

} } printf("\n");

}

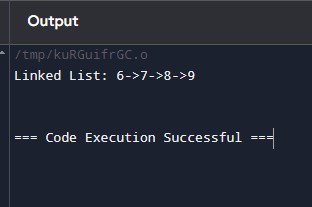
int main() {

struct Node\* head = NULL; appendNode(&head, 6); appendNode(&head, 7); appendNode(&head, 8); appendNode(&head, 9);

printf("Linked List: "); displayList(head); return 0;

}

Output:



5) #include <stdio.h> void bubbleSortDescending(int arr[], int n) { int i, j, temp;

for (i = 0; i < n - 1; i++) { for (j = 0; j < n - i - 1; j++) {

if (arr[j] < arr[j + 1]) { temp = arr[j]; arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

void printArray(int arr[], int n) {

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

printf("%d", arr[i]);

if (i < n - 1) {

printf(",");

} } printf("\n");

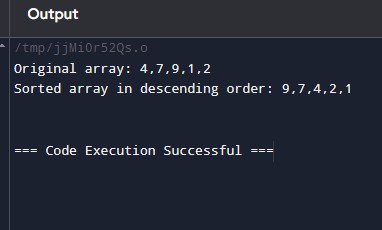
}

int main() { int arr[] = {4, 7, 9, 1, 2}; int n = sizeof(arr) / sizeof(arr[0]); printf("Original array: "); printArray(arr, n); bubbleSortDescending(arr, n); printf("Sorted array in descending order: ");

printArray(arr, n); return 0;

}

Output:



6) #include <stdio.h> int findMissingNumber(int arr[], int n) {

int expectedSum = n \* (n + 1) / 2; int actualSum = 0; for (int i = 0; i < n - 1; i++) { actualSum += arr[i];

}

return expectedSum - actualSum;

}

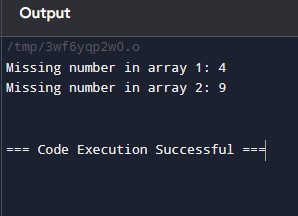
int main() { // Example 1

int arr1[] = {1, 2, 3, 5};

int n1 = 5; printf("Missing number in array 1: %d\n", findMissingNumber(arr1, n1)); int arr2[] = {6, 1, 2, 8, 3, 4, 7, 10, 5}; int n2 = 10; printf("Missing number in array 2: %d\n", findMissingNumber(arr2, n2)); return 0;

}

Output:



7) #include <stdio.h>

#include <stdlib.h> typedef struct Node {

int data; struct Node\* next;

} Node;

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node)); if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data; newNode->next = NULL; return newNode;

}

void printOddNumbers(Node\* head) { Node\* current = head; while (current != NULL) { if (current->data % 2 != 0) {

printf("%d ", current->data);

}

current = current->next;

} printf("\n");

}

void appendNode(Node\*\* head, int data) { Node\* newNode = createNode(data); if (\*head == NULL) { \*head = newNode;

return;

}

Node\* temp = \*head; while (temp->next != NULL) { temp = temp->next;

}

temp->next = newNode;

}

int main() {

Node\* head = NULL;

appendNode(&head, 1); appendNode(&head, 2); appendNode(&head, 3); appendNode(&head, 7); printf("Odd numbers in the linked list: "); printOddNumbers(head); Node\* temp; while (head != NULL) { temp = head; head = head->next;

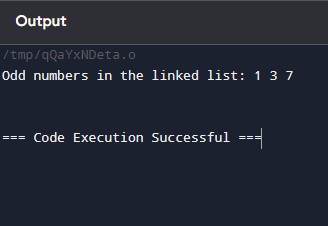
free(temp);

}

return 0;

}

Output:



8)#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct { int items[MAX];

int front; int rear; } Queue; void initializeQueue(Queue \*q) { q->front = -1; q->rear = -1;

}

int isEmpty(Queue \*q) { return q->front == -1;

}

int isFull(Queue \*q) { return (q->rear + 1) % MAX == q->front;

}

void enqueue(Queue \*q, int value) {

if (isFull(q)) {

printf("Queue is full! Cannot insert %d\n", value);

return;

}

if (isEmpty(q)) { q->front = 0;

}

q->rear = (q->rear + 1) % MAX;

q->items[q->rear] = value; printf("Inserted %d into the queue.\n", value);

}

int dequeue(Queue \*q) {

if (isEmpty(q)) { printf("Queue is empty! Cannot delete.\n"); return -1;

}

int data = q->items[q->front]; if (q->front == q->rear) { q->front = -1; q->rear = -1;

} else { q->front = (q->front + 1) % MAX;

}

return data;

}

void displayQueue(Queue \*q) { if (isEmpty(q)) { printf("Queue is empty!\n");

return;

}

printf("Queue contents: "); int i = q->front;

while (i != q->rear) { printf("%d ", q->items[i]);

i = (i + 1) % MAX;

}

printf("%d\n", q->items[i]);

}

int main() { Queue q;

initializeQueue(&q);

enqueue(&q, 12); enqueue(&q, 34); enqueue(&q, 56); enqueue(&q, 78);

printf("Initial queue: ");

displayQueue(&q); enqueue(&q, 60);

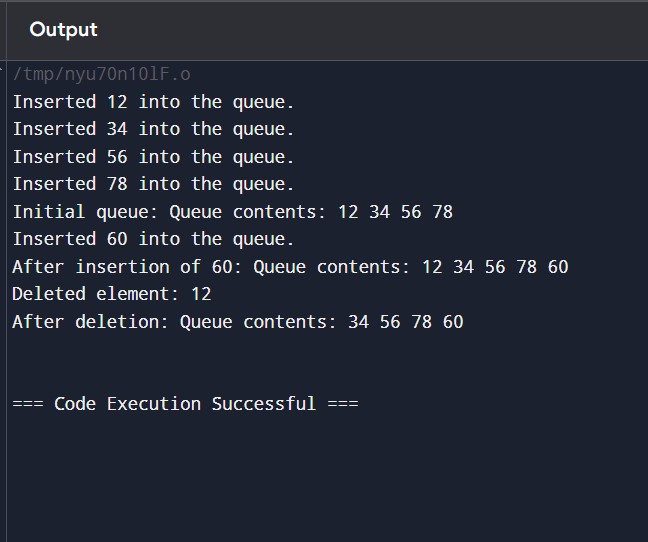
printf("After insertion of 60: "); displayQueue(&q); int deletedElement = dequeue(&q); printf("Deleted element: %d\n", deletedElement);

printf("After deletion: ");

displayQueue(&q); return 0;

}

Output:



9)#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#define MAX 100

typedef struct { char items[MAX];

int top; } Stack; void initializeStack(Stack \*s) { s->top = -1;

}

bool isEmpty(Stack \*s) { return s->top == -1;

}

bool isFull(Stack \*s) {

return s->top == MAX - 1;

}

void push(Stack \*s, char value) {

if (isFull(s)) { printf("Stack overflow!\n"); return;

}

s->items[++s->top] = value;

}

char pop(Stack \*s) {

if (isEmpty(s)) { printf("Stack underflow!\n"); return '\0'; }

return s->items[s->top--];

}

char peek(Stack \*s) {

if (isEmpty(s)) {

return '\0';

}

return s->items[s->top];

}

bool isValid(char \*s) { Stack stack;

initializeStack(&stack); int length = strlen(s); for (int i = 0; i < length; i++) { char current = s[i]; if (current == '(' || current == '{' || current == '[') { push(&stack, current);

}

else if (current == ')' || current == '}' || current == ']') { if (isEmpty(&stack)) { return false;

}

char top = pop(&stack);

if ((current == ')' && top != '(') ||

(current == '}' && top != '{') || (current == ']' && top != '[')) { return false;

}

}

}

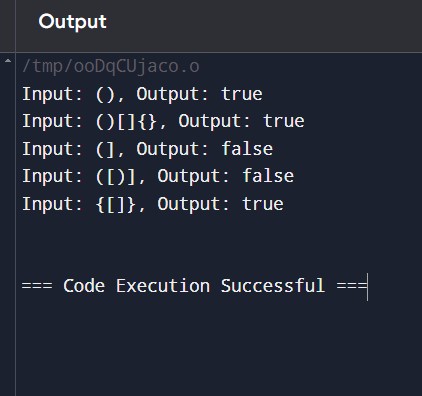
return isEmpty(&stack);

}

int main() { char s1[] = "()"; char s2[] = "()[]{}"; char s3[] = "(]"; char s4[] = "([)]"; char s5[] = "{[]}"; printf("Input: %s, Output: %s\n", s1, isValid(s1) ? "true" : "false"); printf("Input: %s, Output: %s\n", s2, isValid(s2) ? "true" : "false"); printf("Input: %s, Output: %s\n", s3, isValid(s3) ? "true" : "false"); printf("Input: %s, Output: %s\n", s4, isValid(s4) ? "true" : "false"); printf("Input: %s, Output: %s\n", s5, isValid(s5) ? "true" : "false"); return 0;

}

Output:



10)#include <stdio.h> int main() { int n = 10; int fib1 = 0, fib2 = 1; int nextTerm; int sum = fib1 + fib2;

printf("Fibonacci series: %d, %d", fib1, fib2); for (int i = 3; i <= n; i++) { nextTerm = fib1 + fib2;

printf(", %d", nextTerm);

sum += nextTerm;

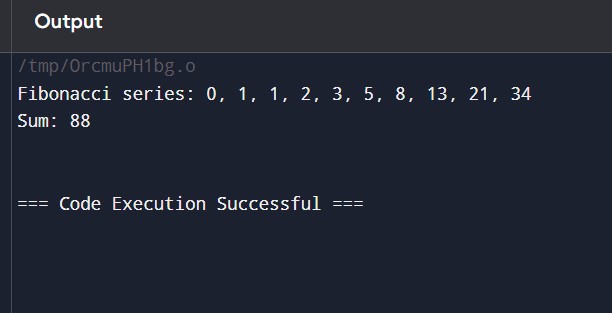
fib1 = fib2; fib2 = nextTerm;

}

printf("\nSum: %d\n", sum); return 0;

}

Output:



11)#include <stdio.h>

#include <stdlib.h> struct Node {

int data; struct Node\* next;

};

struct Node\* createNode(int data) { struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = data; newNode->next = NULL; return newNode;

}

int countNodes(struct Node\* head) { int count = 0; struct Node\* current = head; while (current != NULL) { count++; current = current->next;

}

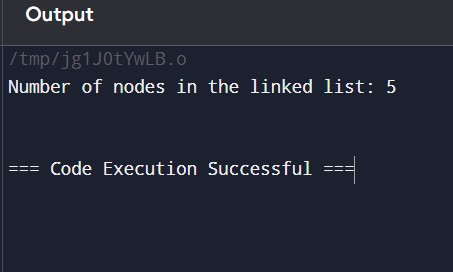
return count;

}

int main() { struct Node\* head = createNode(1); head->next = createNode(2); head->next->next = createNode(3); head->next->next->next = createNode(5); head->next->next->next->next = createNode(8); int nodeCount = countNodes(head); printf("Number of nodes in the linked list: %d\n", nodeCount); return 0;

}

Output:



12)#include <stdio.h> int fibonacci(int n) { if (n == 0) return 0; if (n == 1) return 1; return fibonacci(n - 1) + fibonacci(n - 2);

}

int fibonacciSum(int n) {

if (n == 0) return 0; return fibonacci(n) + fibonacciSum(n - 1);

}

int main() { int n = 10; int sum = 0;

printf("Fibonacci series: ");

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

int fib = fibonacci(i); printf("%d", fib);

if (i < n - 1) { printf(", "); } sum += fib;

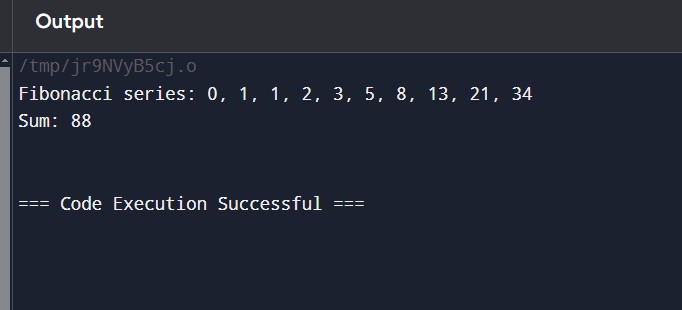
}

printf("\nSum: %d\n", sum); // Print the sum of the Fibonacci series

return 0;

}

Output:



13)#include <stdio.h> int binarySearch(int arr[], int size, int x) { int low = 0; int high = size - 1; while (low <= high) { int mid = low + (high - low) / 2;

if (arr[mid] == x) { return mid;

}

if (arr[mid] < x) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() { int arr[] = {1, 5, 6, 7, 9, 10}; int size = sizeof(arr) / sizeof(arr[0]);

int x = 6; // Element to search for

int result = binarySearch(arr, size, x); if (result != -1) { printf("Element %d found at location %d\n", x, result);

} else { printf("Element %d not found in the array\n", x);

}

x = 11;

result = binarySearch(arr, size, x); if (result != -1) { printf("Element %d found at location %d\n", x, result);

} else {

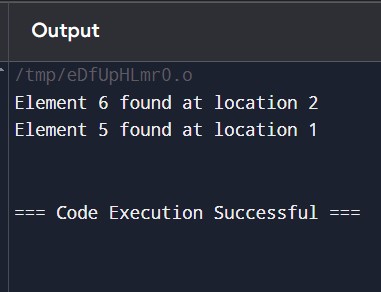
printf("Element %d not found in the array\n", x);

}

return 0;

}

Output:



14)#include <stdio.h>

#include <string.h> #include <stdbool.h> void sortString(char s[]) { int len = strlen(s); for (int i = 0; i < len - 1; i++) { for (int j = i + 1; j < len; j++) {

if (s[i] > s[j]) {

char temp = s[i];

s[i] = s[j];

s[j] = temp;

}

}

}

}

void findStartingIndices(char s[]) { int len = strlen(s); bool repeated = false; for (int i = 0; i < len; i++) {

if (i < len - 1 && s[i] == s[i + 1]) {

if (!repeated) {

printf("%d", i);

repeated = true;

}

} else { if (repeated) { printf(", %d", i + 1); repeated = false;

}

} } printf("\n");

}

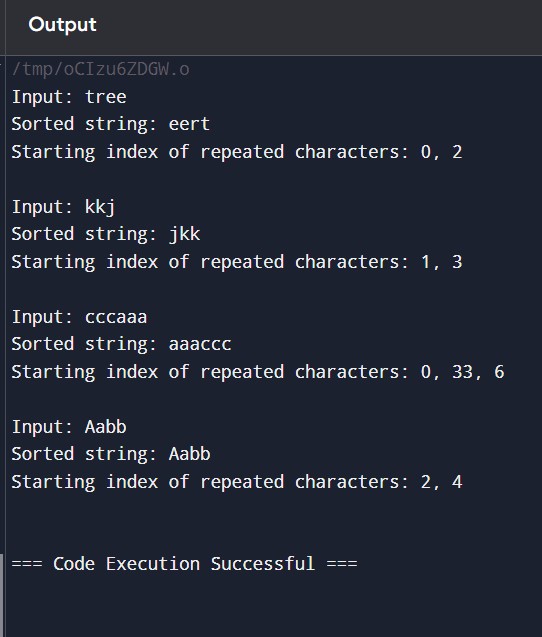
int main() { char s1[] = "tree"; char s2[] = "kkj"; char s3[] = "cccaaa"; char s4[] = "Aabb";

printf("Input: %s\n", s1); sortString(s1); printf("Sorted string: %s\n", s1); printf("Starting index of repeated characters: "); findStartingIndices(s1); printf("\nInput: %s\n", s2); sortString(s2);

printf("Sorted string: %s\n", s2); printf("Starting index of repeated characters: "); findStartingIndices(s2); printf("\nInput: %s\n", s3); sortString(s3); printf("Sorted string: %s\n", s3); printf("Starting index of repeated characters: "); findStartingIndices(s3); printf("\nInput: %s\n", s4); sortString(s4); printf("Sorted string: %s\n", s4); printf("Starting index of repeated characters: "); findStartingIndices(s4); return 0;

}

Output:



15)#include <stdio.h>

#include <stdlib.h> #include <stdbool.h> struct Node { char data; struct Node\* next;

};

struct Node\* createNode(char data) { struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); newNode->data = data; newNode->next = NULL; return newNode;

}

struct Node\* reverseList(struct Node\* head) { struct Node\* prev = NULL; struct Node\* current = head; struct Node\* next = NULL; while (current != NULL) { next = current->next; current->next = prev; prev = current; current = next;

}

return prev;

}

bool isPalindrome(struct Node\* head) { if (head == NULL || head->next == NULL) {

return true;

}

struct Node\* slow = head; struct Node\* fast = head; while (fast != NULL && fast->next != NULL) { slow = slow->next; fast = fast->next->next;

}

struct Node\* secondHalf = reverseList(slow); struct Node\* firstHalf = head; struct Node\* copySecondHalf = secondHalf; while (secondHalf != NULL) {

if (firstHalf->data != secondHalf->data) { // Restore the list before returning false reverseList(copySecondHalf); return false;

}

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;

}

reverseList(copySecondHalf); return true;

}

void printList(struct Node\* head) { struct Node\* current = head; while (current != NULL) {

printf("%c->", current->data); current = current->next;

}

printf("NULL\n");

}

int main() { struct Node\* head1 = createNode('1'); head1->next = createNode('2'); head1->next->next = createNode('2'); head1->next->next->next = createNode('1'); printf("Input: 1->2->2->1->NULL\n"); printf("Output: %s\n", isPalindrome(head1) ? "true" : "false"); struct Node\* head2 = createNode('1'); head2->next = createNode('2'); printf("Input: 1->2->NULL\n"); printf("Output: %s\n", isPalindrome(head2) ? "true" : "false"); struct Node\* head3 = createNode('R'); head3->next = createNode('A'); head3->next->next = createNode('D'); head3->next->next->next = createNode('A'); head3->next->next->next->next = createNode('R');

printf("Input: R->A->D->A->R->NULL\n"); printf("Output: %s\n", isPalindrome(head3) ? "Yes" : "No"); struct Node\* head4 = createNode('C'); head4->next = createNode('O'); head4->next->next = createNode('D'); head4->next->next->next = createNode('E'); printf("Input: C->O->D->E->NULL\n"); printf("Output: %s\n", isPalindrome(head4) ? "Yes" : "No"); return 0;

}

Output:

