Question 1)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

void insert(Node\*\* head, int newData) {

Node\* newNode = new Node();

newNode->data = newData;

newNode->next = (\*head);

(\*head) = newNode;}

bool detectCycle(Node\* head) {

Node\* slow = head;

Node\* fast = head;

while (fast != nullptr && fast->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

return true;}}

return false;}

int main() {

Node\* head = nullptr;

insert(&head, 10);

insert(&head, 20);

insert(&head, 30);

insert(&head, 40);

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

if (detectCycle(head)) {

cout << "Cycle detected" << endl;

} else {

cout << "No cycle detected" << endl;}

return 0;}

Question 2)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

void insert(Node\*\* head, int newData) {

Node\* newNode = new Node();

newNode->data = newData;

newNode->next = (\*head);

(\*head) = newNode;}

void segregateEvenOdd(Node\*\* head) {

Node\* evenStart = nullptr;

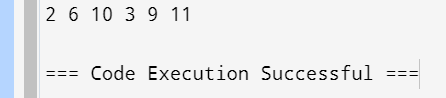
Node\* evenEnd = nullptr;

Node\* oddStart = nullptr;

Node\* oddEnd = nullptr;

Node\* current = \*head;

while (current != nullptr) {

 int val = current->data;

if (val % 2 == 0) {

if (evenStart == nullptr) {

evenStart = current;

evenEnd = evenStart;

} else {

evenEnd->next = current;

evenEnd = evenEnd->next;}

} else {

if (oddStart == nullptr) {

oddStart = current;

oddEnd = oddStart;

} else {

oddEnd->next = current;

oddEnd = oddEnd->next;}}

current = current->next;}

if (evenStart == nullptr || oddStart == nullptr)

return;

evenEnd->next = oddStart;

oddEnd->next = nullptr;

\*head = evenStart;}

void printList(Node\* head) {

while (head != nullptr) {

cout << head->data << " ";

head = head->next;}}

int main() {

Node\* head = nullptr;

insert(&head, 11);

insert(&head, 10);

insert(&head, 9);

insert(&head, 6);

insert(&head, 3);

insert(&head, 2);

segregateEvenOdd(&head);

printList(head);

return 0;}

Question 3)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

int getLength(Node\* head) {

int len = 0;

while (head != nullptr) {

len++;

head = head->next;}

return len;}

Node\* getIntersectionNode(Node\* head1, Node\* head2) {

int len1 = getLength(head1);

int len2 = getLength(head2);

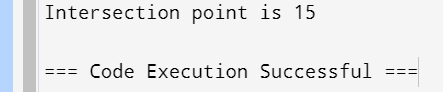
int diff = abs(len1 - len2);

if (len1 > len2) {

while (diff--) {

head1 = head1->next;}

} else {

 while (diff--) {

head2 = head2->next;}}

while (head1 != nullptr && head2 != nullptr) {

if (head1 == head2) {

return head1;}

head1 = head1->next;

head2 = head2->next;}

return nullptr;}

int main() {

Node\* newNode;

Node\* head1 = new Node();

head1->data = 10;

Node\* head2 = new Node();

head2->data = 3;

newNode = new Node();

newNode->data = 6;

head2->next = newNode;

newNode = new Node();

newNode->data = 9;

head2->next->next = newNode;

newNode = new Node();

newNode->data = 15;

head1->next = newNode;

head2->next->next->next = newNode;

newNode = new Node();

newNode->data = 30;

head1->next->next = newNode;

head1->next->next->next = nullptr;

Node\* intersection = getIntersectionNode(head1, head2);

if (intersection != nullptr)

cout << "Intersection point is " << intersection->data;

else

cout << "No intersection point";

return 0;}

Question 4)

#include <iostream>

#include <unordered\_set>

using namespace std;

struct Node {

int data;

Node\* next;};

void removeDuplicates(Node\* head) {

unordered\_set<int> seen;

Node\* current = head;

Node\* prev = nullptr;

while (current != nullptr) {

if (seen.find(current->data) != seen.end()) {

prev->next = current->next;

delete current;

} else {

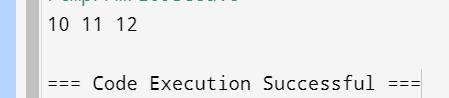
seen.insert(current->data);

prev = current;}

current = prev->next;}}

void insert(Node\*\* head, int new\_data) {

Node\* new\_node = new Node();

 new\_node->data = new\_data;

new\_node->next = (\*head);

(\*head) = new\_node;}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " ";

node = node->next;}} int main() {

Node\* head = nullptr;

insert(&head, 10);

insert(&head, 12);

insert(&head, 11);

insert(&head, 11);

insert(&head, 12);

insert(&head, 11);

insert(&head, 10);

removeDuplicates(head);

printList(head);

return 0;}

Question 5)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

void rotate(Node\*\* head, int k) {

if (!\*head || k == 0) return;

Node\* current = \*head;

int count = 1;

while (count < k && current) {

current = current->next;

count++;}

if (!current) return;

Node\* kthNode = current;

while (current->next) {

current = current->next;}

current->next = \*head;

\*head = kthNode->next;

kthNode->next = nullptr;}

void insert(Node\*\* head, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = (\*head);

(\*head) = new\_node;}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " ";

node = node->next;}}

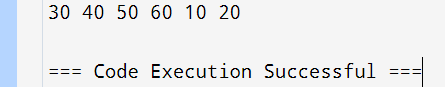
int main() {

Node\* head = nullptr;

insert(&head, 60);

insert(&head, 50);

insert(&head, 40);

 insert(&head, 30);

insert(&head, 20);

insert(&head, 10);

int k = 2;

rotate(&head, k);

printList(head);

return 0;}

Question 6)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

void insert(Node\*\* head, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = (\*head);

(\*head) = new\_node;}

void sortList(Node\*\* head) {

if (!\*head) return;

Node\* zeroD = new Node();

Node\* oneD = new Node();

Node\* twoD = new Node();

Node\* zero = zeroD;

Node\* one = oneD;

Node\* two = twoD;

Node\* curr = \*head;

while (curr) {

if (curr->data == 0) {

zero->next = curr;

zero = zero->next;

} else if (curr->data == 1) {

one->next = curr;

one = one->next;

} else {

two->next = curr;

two = two->next;}

curr = curr->next;}

zero->next = (oneD->next) ? (oneD->next) : (twoD->next);

one->next = twoD->next;

two->next = nullptr;

\*head = zeroD->next;

delete zeroD;

delete oneD;

delete twoD;}

void printList(Node\* node) {

while (node != nullptr) {

cout << node->data << " ";

node = node->next;}}

int main() {

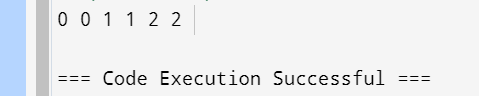
Node\* head = nullptr;

insert(&head, 2);

insert(&head, 1);

insert(&head, 0);

insert(&head, 1);

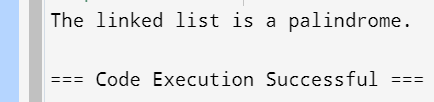
 insert(&head, 2);

insert(&head, 0);

sortList(&head);

printList(head);

return 0;}

Question 7)

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;};

void insert(Node\*\* head, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = (\*head);

(\*head) = new\_node;}

Node\* reverse(Node\* head) {

Node\* prev = nullptr;

Node\* curr = head;

Node\* next = nullptr;

while (curr != nullptr) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;}

return prev;}

bool isPalindrome(Node\* head) {

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

return true;

Node\* slow = head;

Node\* fast = head;

while (fast != nullptr && fast->next != nullptr) {

slow = slow->next;

fast = fast->next->next;}

slow = reverse(slow);

Node\* secondHalf = slow;

Node\* firstHalf = head;

while (secondHalf != nullptr) {

if (firstHalf->data != secondHalf->data)

return false;

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;}

return true;}

int main() {

Node\* head = nullptr;

insert(&head, 1);

insert(&head, 2);

insert(&head, 3);

insert(&head, 2);

insert(&head, 1);

if (isPalindrome(head))

cout << "The linked list is a palindrome.";

else

cout << "The linked list is not a palindrome.";

return 0;}

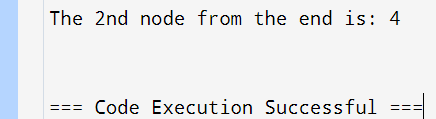
Question 8)

#include <iostream>

using namespace std;

struct Node {

int data;

 Node\* next;};

void insert(Node\*\* head, int new\_data) {

Node\* new\_node = new Node();

new\_node->data = new\_data;

new\_node->next = (\*head);

(\*head) = new\_node;}

void findNthFromEnd(Node\* head, int n) {

Node\* main\_ptr = head;

Node\* ref\_ptr = head;

int count = 0;

while (count < n) {

if (ref\_ptr == nullptr) {

cout << "List has fewer than " << n << " nodes." << endl;

return;}

ref\_ptr = ref\_ptr->next;

count++;}

while (ref\_ptr != nullptr) {

main\_ptr = main\_ptr->next;

ref\_ptr = ref\_ptr->next;}

cout << "The " << n << "nd node from the end is: " << main\_ptr->data << endl;}

int main() {

Node\* head = nullptr;

insert(&head, 20);

insert(&head, 4);

insert(&head, 15);

insert(&head, 35);

int n = 2;

findNthFromEnd(head, n);

return 0;}