

**National University of Computer and Emerging**

**Sciences**

**Chiniot-Faisalabad Campus BS (Artificial Intelligence)**

|  |  |
| --- | --- |
| **Name** | **M Abdul Hanan** |
| **Reg.NO.** | **22F-3104** |
| **Section** | **BS(Ai)-3A** |
| **Course** | **Programming for Ai (2001)** |
| **Department** | **CS Department** |
| **Lab** | **#4** |

**Task no 1:**

**Code:**

#include <iostream>

using namespace std;

//In this task i make a singly link list which points to the next node and next node stores the value in the prevoius node and print the link list in reverse order

class node {

public:

int data;

node\* next;

// Constructor to initialize a node with a given value

node(int val) {

data = val;

next = nullptr;

}

};

class LinkedList {

public:

node\* head;

// Constructor to initialize the linked list with an empty head

LinkedList() {

head = nullptr;

}

// Function to insert a new node at the end of the linked list

void insert(int val) {

node\* newNode = new node(val);

// If the list is empty set the new node as the head

if (head == nullptr) {

head = newNode;

return;

}

// Traverse the list to find the last node and link the new node

node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

// Function to reverse the linked list

void reverse() {

node\* previous = nullptr;

node\* current = head;

node\* nn = nullptr;

// Reverse the links between nodes to reverse the list

while (current != NULL) {

nn = current->next;

current->next = previous;

previous = current;

current = nn;

}

// Update the head to point to the new first node

head = previous;

}

// Function to display the elements of the linked list

void display() {

node\* temp = head;

// Traverse the list and print each element

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << endl;

}

};

int main() {

LinkedList list1;

int num=0;

while (num != -1) {

cout << "Enter nodes :";

cin >> num;

// Insert characters into the list

if (num != -1) {

list1.insert(num);

int nodes = num;

}

else {

break;

}

}

// Display the original list

cout << "Original Singly List is: ";

list1.display();

list1.reverse();

// Display the reversed copy

cout << "Reversed Singly List is: ";

list1.display();

return 0;

}

**Screen Shot:**

****

**Task no 2:**

**Code:**

#include <iostream>

using namespace std;

//in this code i create a double link in which node points to the next node and pervious points to the node current and copy pointer head tail size in copy constructor and intialize it by deap copy

class Node {

public:

int data;

Node\* next;

Node\* prev;

// Constructor to initialize the data and pointers

Node(int value)

{

data = value;

next = nullptr;

prev = nullptr;

}

};

class double\_link {

public:

Node\* head;

Node\* tail;

int size;

// Constructor to initialize the linked list

double\_link()

{

head = nullptr;

tail = nullptr;

size = 0;

}

// Copy constructor to create a new linked list by copying elements from another list.

double\_link(double\_link& obj) : head(nullptr), tail(nullptr), size(0) {

Node\* current = obj.head;

while (current != nullptr) {

push\_back(current->data);

current = current->next;

}

}

// Function to add a new element to the end of the list.

void push\_back(int value) {

Node\* newNode = new Node(value);

if (tail == nullptr) {

head = tail = newNode;

}

else {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

}

size++;

}

// Function to print the elements in the list.

void display() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};

int main() {

double\_link originalList;

int num = 0;

// Input loop to read integers from the user and add them to the originalList.

while (num != -1) {

cout << "Enter nodes -1 to exit : ";

cin >> num;

if (num != -1) {

originalList.push\_back(num);

}

else {

break;

}

}

// Create a copiedList by using the copy constructor.

double\_link copiedList(originalList);

// Print the original and copied lists to verify the copying process.

cout << "Original List: ";

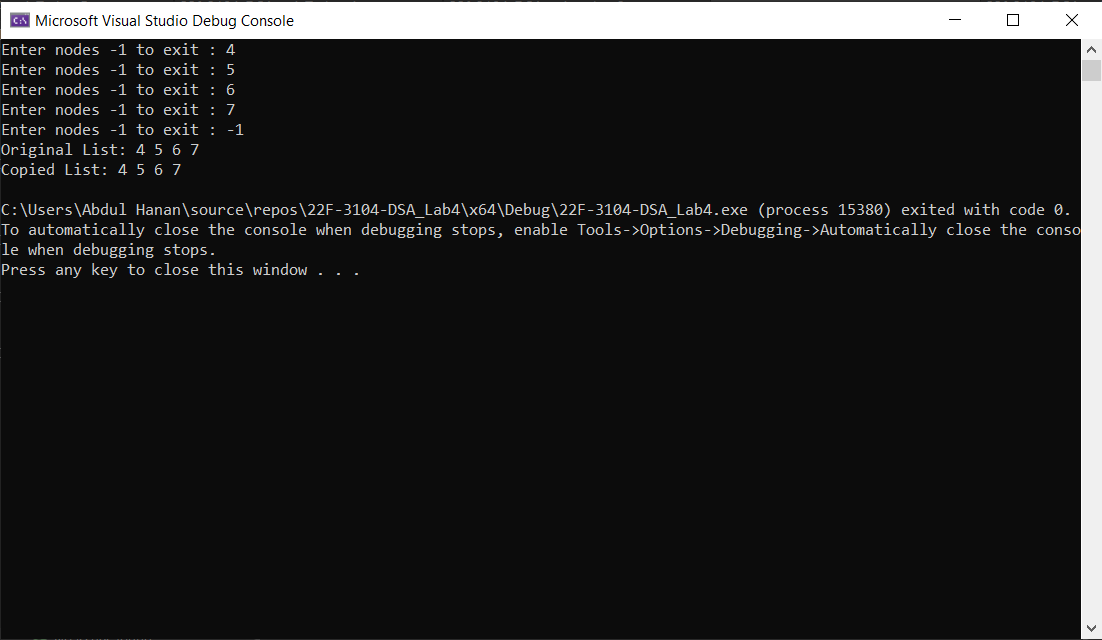
originalList.display();

cout << "Copied List: ";

copiedList.display();

return 0;

}

**Screen Shot: **

**Task no 3:**

**Code:**

#include <iostream>

using namespace std;

//In this code i write the circular list in which data of the nodes points to even nodes and intialize it to even head and odd data to oddhead and if it is even then it will sent at the tail and if odd then it will insert at head

class node {

public:

int data;

node\* next;

node(int val) {

data = val;

next = NULL;

}

};

// Function to insert a node at the end of a circular linked list

void insert\_end(node\*& head, int val) {

node\* n = new node(val);

if (head == NULL) {

n->next = n;

head = n;

}

else {

node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = n;

n->next = head;

}

}

// Function to rearrange the circular linked list

void rearrangeCircularList(node\*& head) {

if (head == NULL || head->next == head) {

return;

}

// Initialize pointers for odd and even nodes

node\* oddHead = head;

node\* evenHead = head->next;

node\* oddCurrent = oddHead;

node\* evenCurrent = evenHead;

// Rearrange the nodes separating odd index and even index nodes

while (evenCurrent->next != head && evenCurrent->next->next != head) {

oddCurrent->next = evenCurrent->next;

oddCurrent = oddCurrent->next;

evenCurrent->next = oddCurrent->next;

evenCurrent = evenCurrent->next;

}

// Handle the case when the number of nodes is odd

if (evenCurrent->next == head) {

oddCurrent->next = evenCurrent->next;

oddCurrent = oddCurrent->next;

}

// Make the odd indexed list circular

oddCurrent->next = evenHead;

// Make the even indexed list circular

evenCurrent->next = head;

// Update the head to the new head

head = oddHead;

}

// Function to display the circular linked list

void display(node\* head) {

if (head == NULL) {

return;

}

node\* temp = head;

do {

cout << temp->data << " ";

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

node\* head = NULL;

int num=0;

while (num != -1) {

cout << "Enter nodes :";

cin >> num;

insert\_end(head,num);

int nodes = num;

}

cout << "Original Circular Linked List: ";

display(head);

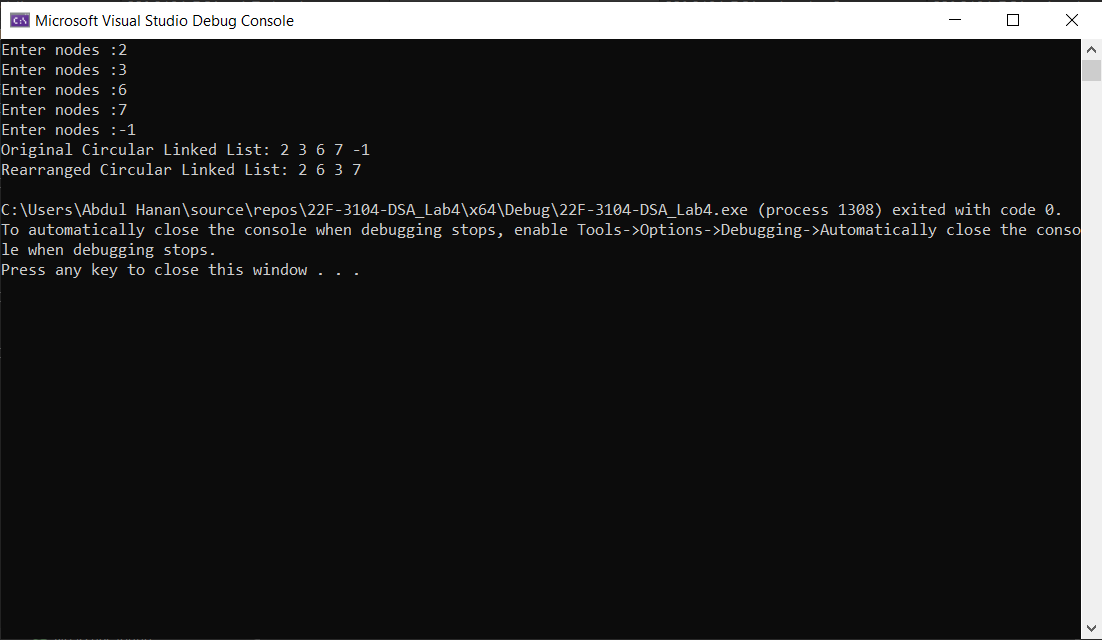
rearrangeCircularList(head);

cout << "Rearranged Circular Linked List: ";

display(head);

return 0;

}

**Screen Shot: **

**Task no 4:**

**Code:**

#include <iostream>

using namespace std;

//in this code i write the circular list i which i make a duplicate function which compare the node to the next node if it find any duplicate it will remove it and count the iteam reamove

class node {

public:

int data;

node\* next;

node(int val) {

data = val;

next = NULL;

}

};

void insert\_head(node\*& head, int val) {//insert nodes at the head

node\* n = new node(val);

if (head == NULL) {

n->next = n;

head = n;

return;

}

node\* temp = head;

// Traverse to the last node in the list

while (temp->next != head) {

temp = temp->next;

}

temp->next = n; // Connect the last node to the new node

n->next = head; // Make the new node point to the head

head = n; // Update the head to the new node

}

// Function to insert a new node at the end of the list

void insert\_end(node\*& head, int val) {

node\* n = new node(val); // If the list is empty insert head

if (head == NULL) {

insert\_head(head, val);

return;

}

node\* temp = head;

// Traverse to the last node in the list

while (temp->next != head) {

temp = temp->next;

}

temp->next = n; // Connect the last node to the new node

n->next = head; // Make the new node point to the head

}

// Function to remove duplicate elements list

node\* remove\_Duplicates(node\* head) {

if (head == NULL) {

return head; // If the list is empty return it as it is

}

node\* current = head;

int duplicate\_count = 0;

while (current->next != head) {

// If the current node data is the same as the next node data remove the next node

if (current->data == current->next->data) {

node\* duplicate = current->next;

current->next = current->next->next;

delete duplicate; // Delete the duplicate node

duplicate\_count++;

}

else {

current = current->next; // Move to the next node

}

}

// Display the number of duplicates removed

cout << "Number of duplicates removed: " << duplicate\_count << endl;

return head;

}

// Function to display the linked list

void display(node\* head) {

if (head == NULL) {

return; // If the list is empty, nothing to display

}

node\* temp = head;

do {

cout << temp->data << "->";

temp = temp->next;

} while (temp != head);

cout << endl;

}

int main() {

node\* head = NULL;

int num=0;

while (num != -1) {

cout << "Enter nodes :";

cin >> num;

if (num != -1) {

insert\_end(head, num);

int nodes = num;

}

else {

break;

}

}

cout << "Original list:" << endl;

display(head);

head = remove\_Duplicates(head);

cout << "List after removing duplicates:" << endl;

display(head);

return 0;

}

**Screen Shot:**

****

**Task no 5:**

**Code:**

#include <iostream>

using namespace std;

//in this code i convert double list to circular list by last node pointing to head node

class Node {

public:

int data;

Node\* next;

Node\* prev;

Node(int value) {

data = value;

next = this; // Initialize next and prev to itself for circularity.

prev = this;

}

};

class CircularDoublyLinkedList {

public:

Node\* head;

CircularDoublyLinkedList() {

head = nullptr;

}

// Function to insert a new element at the end of the circular doubly linked list.

void insert(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

}

else {

Node\* tail = head->prev; // Get the current tail

tail->next = newNode;

newNode->prev = tail;

newNode->next = head; // Circular connection

head->prev = newNode; // Update the head's previous pointer

}

}

// Function to delete a node with a given value from the circular doubly linked list.

void remove(int value) {

if (head == nullptr)

return;

Node\* current = head;

do {

if (current->data == value) {

Node\* prevNode = current->prev;

Node\* nextNode = current->next;

prevNode->next = nextNode;

nextNode->prev = prevNode;

if (current == head) {

head = nextNode; // Update the head if deleting the head node

}

delete current;

return; // Exit after deleting the first occurrence

}

current = current->next;

} while (current != head);

}

// Function to display the circular doubly linked list.

void display() {

if (head == nullptr)

return;

Node\* current = head;

do {

cout << current->data << " ";

current = current->next;

} while (current != head);

cout << endl;

}

};

int main() {

CircularDoublyLinkedList myList;

int num=0;

while (num != -1) {

cout << "Enter nodes :";

cin >> num;

if (num != -1) {

myList.insert(num);

int nodes = num;

}

else {

break;

}

}

cout << "Original List: ";

myList.display();

int remove;

cout << "Enter elment to remove :";

cin >> remove;

myList.remove(remove);

cout << "Enter elment to remove :";

cin >> remove;

myList.remove(remove);

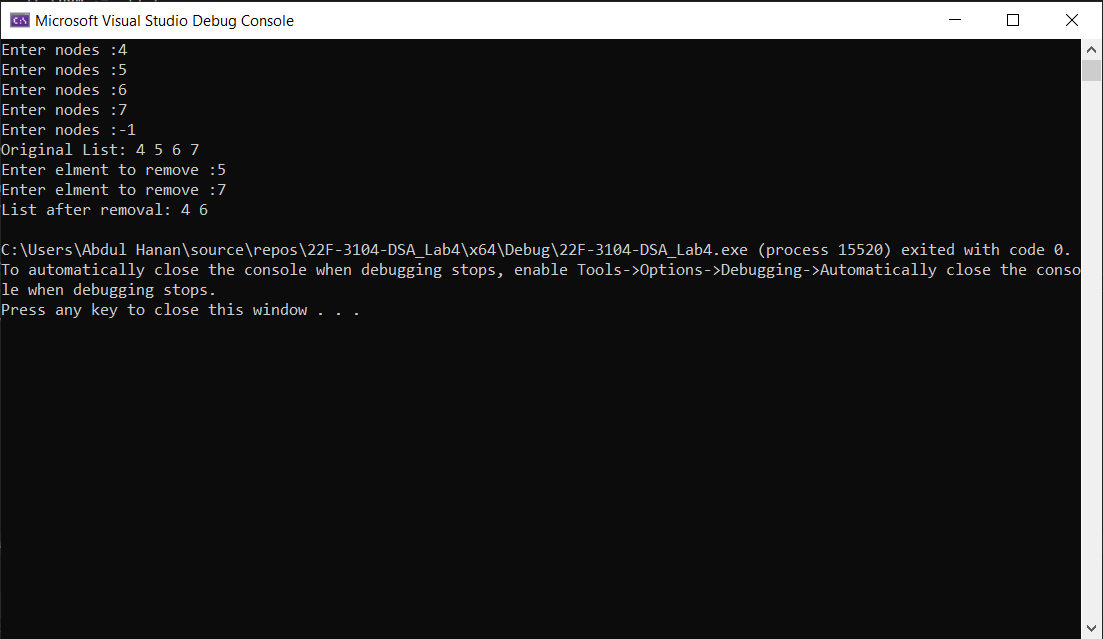
cout << "List after removal: ";

myList.display();

return 0;

}

**Screen shot:**

****