**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 12: WAP to evaluate a postfix expression using stack.**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_SIZE 100

struct Stack

{

int top;

unsigned capacity;

int \*array;

};

int isEmpty(struct Stack \*stack)

{

return stack->top == -1;

}

void push(struct Stack \*stack, int value)

{

stack->array[++stack->top] = value;

}

int pop(struct Stack \*stack)

{

if (!isEmpty(stack))

return stack->array[stack->top--];

return -1; // Return -1 for an empty stack (error condition)

}

int evaluatePostfix(char \*postfix)

{

struct Stack \*stack = (struct Stack \*)malloc(sizeof(struct Stack));

stack->capacity = MAX\_SIZE;

stack->top = -1;

stack->array = (int \*)malloc(stack->capacity \* sizeof(int));

int i, operand1, operand2;

char ch;

for (i = 0; postfix[i] != '\0'; i++)

{

ch = postfix[i];

if (isdigit(ch))

{

push(stack, ch - '0');

}

else

{

operand2 = pop(stack);

operand1 = pop(stack);

switch (ch)

{

case '+':

push(stack, operand1 + operand2);

break;

case '-':

push(stack, operand1 - operand2);

break;

case '\*':

push(stack, operand1 \* operand2);

break;

case '/':

push(stack, operand1 / operand2);

break;

}

}

}

int result = pop(stack);

free(stack->array);

free(stack);

return result;

}

int main()

{

char postfix[MAX\_SIZE];

printf("Enter a postfix expression: ");

fgets(postfix, MAX\_SIZE, stdin);

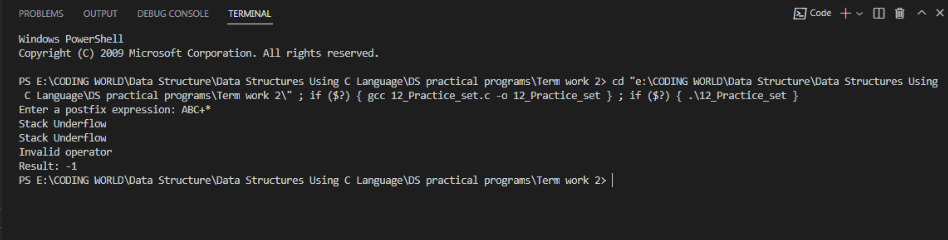
int result = evaluatePostfix(postfix);

printf("Result: %d\n", result);

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 13: Write a menu driven C program to implement of singly linked list with following operations:**

**• Create the list**

**• Insert a node in the beginning, in the end, at given position**

**• Delete a node in the beginning, in the end, at given position**

**• Search a node**

**• Display the list**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*insert\_beg(struct node \*head)

{

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for new node: ");

scanf("%d", &(new\_node->data));

new\_node->next = head;

head = new\_node;

return head;

}

struct node \*insert\_pos(struct node \*head)

{

int position;

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for new node: ");

scanf("%d", &(new\_node->data));

printf("Enter position for new node: ");

scanf("%d", &position);

if (position == 1)

{ // Insert at the beginning

new\_node->next = head;

return new\_node;

}

struct node \*temp = head;

for (int i = 1; i < position - 1; i++)

{

temp = temp->next;

}

if (temp == NULL)

{

printf("Invalid position\n");

return head;

}

new\_node->next = temp->next;

temp->next = new\_node;

return head;

}

struct node \*insert\_end(struct node \*head)

{

struct node \*temp = head;

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for new node: ");

scanf("%d", &(new\_node->data));

if (head == NULL)

{

head = new\_node;

}

while (temp->next != NULL)

{

temp = temp->next;

}

new\_node->next = NULL;

temp->next = new\_node;

return head;

}

struct node \*delete\_beg(struct node \*head)

{

if (head == NULL)

{

printf("Linked list is empty deletion is not possible\n");

return head;

}

struct node \*temp = head;

head = head->next;

free(temp);

return head;

}

struct node \*delete\_pos(struct node \*head)

{

int position;

printf("Enter position for deleting node: ");

scanf("%d", &position);

if (position == 1)

{

struct node \*temp = head;

head = head->next;

free(temp);

return head;

}

struct node \*temp = head;

for (int i = 1; i < position - 1; i++)

{

temp = temp->next;

}

struct node \*ptr = temp->next;

temp->next = ptr->next;

free(ptr);

return head;

}

struct node \*delete\_end(struct node \*head)

{

if (head == NULL)

{

printf("Liked list is empty. Deletion not possible\n");

return head;

}

struct node \*temp = head;

while (temp->next->next != NULL)

{

temp = temp->next;

}

struct node \*ptr = temp->next;

temp->next = NULL;

free(ptr);

return head;

}

void searchNode(struct node \*head)

{

int key;

printf("Enter the value of key: ");

scanf("%d", &key);

if (head == NULL)

{

return;

}

struct node \*temp = head;

while (temp != NULL)

{

if (temp->data == key)

{

printf("Node found\n"); // Node found

return;

}

temp = temp->next;

}

printf("Node not found\n"); // Node not found

}

void display(struct node \*head)

{

struct node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("\n");

}

int main()

{

int choice;

struct node \*head = NULL;

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = NULL;

struct node \*temp = new\_node;

head = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

display(head);

while (choice != 9)

{

printf("Enter your choice: \n1.Insert at beginning\n2.Insert at position\n3. Insert at the end\n4. Delete at beginning \n5. Delete at position\n6.Delete at the end\n7.Search a node\n8. display list\n9. Exit\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

head = insert\_beg(head);

break;

case 2:

head = insert\_pos(head);

break;

case 3:

head = insert\_end(head);

break;

case 4:

head = delete\_beg(head);

break;

case 5:

head = delete\_pos(head);

break;

case 6:

head = delete\_end(head);

break;

case 7:

searchNode(head);

break;

case 8:

display(head);

break;

default:

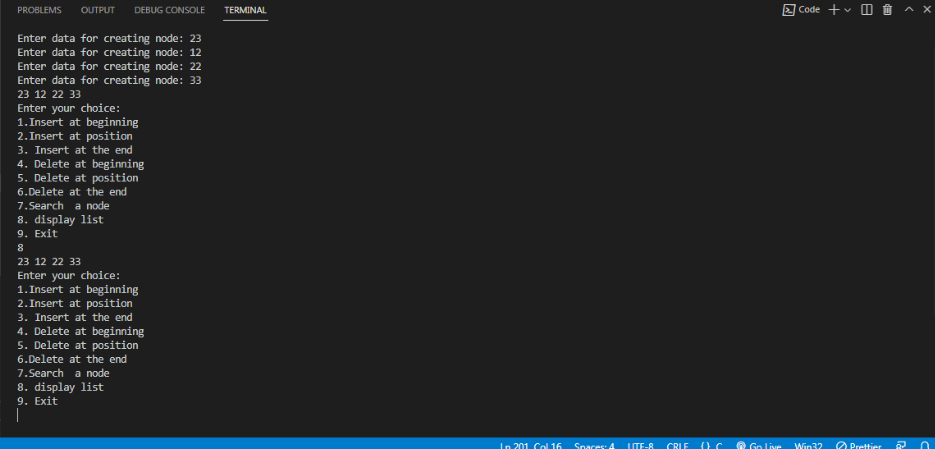
break;

}

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement: 14. Write a menu driven C program to implement of doubly linked list with following operations:**

**• Create the list**

**• Insert a node in the beginning, in the end, at given position.**

**• Delete a node in the beginning, in the end, at given position.**

**• Search a node**

**• Display the list**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*prev;

struct node \*next;

};

struct node \*deleteAtBeginning(struct node \*head)

{

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

struct node \*temp = head;

head = (head)->next;

if (head != NULL)

{

(head)->prev = NULL;

}

free(temp);

printf("Node at the beginning deleted successfully.\n");

return head;

}

// Function to delete a node at a given position in the doubly linked list

struct node \*deleteAtPosition(struct node \*head)

{

int position;

printf("Enter position\n");

scanf("%d", &position);

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

if (position <= 0)

{

printf("Invalid position.\n");

return head;

}

if (position == 1)

{

head = deleteAtBeginning(head);

return head;

}

struct node \*current = head;

int currentPosition = 1;

while (current != NULL && currentPosition < position)

{

current = current->next;

currentPosition++;

}

if (current == NULL)

{

printf("Invalid position.\n");

return head;

}

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

if (current->next != NULL)

{

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

}

free(current);

printf("Node at position %d deleted successfully.\n", position);

return head;

}

// Function to delete the node at the end of the doubly linked list

struct node \*deleteAtEnd(struct node \*head)

{

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

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

{

free(head);

head = NULL;

printf("Node at the end deleted successfully.\n");

return head;

}

struct node \*current = head;

while (current->next != NULL)

{

current = current->next;

}

current->prev->next = NULL;

free(current);

printf("Node at the end deleted successfully.\n");

return head;

}

void searchNode(struct node \*head)

{

int key;

printf("Enter the value of key: ");

scanf("%d", &key);

if (head == NULL)

{

return;

}

struct node \*temp = head;

while (temp != NULL)

{

if (temp->data == key)

{

printf("Node found\n"); // Node found

return;

}

temp = temp->next;

}

printf("Node not found\n"); // Node not found

}

void display(struct node \*head)

{

struct node \*current = head;

printf("Doubly linked list: ");

while (current != NULL)

{

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

current = current->next;

}

printf("\n");

}

int main()

{

int choice;

struct node \*head = NULL;

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->prev = head;

new\_node->next = NULL;

struct node \*temp = new\_node;

head = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->prev = temp;

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->prev = temp;

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->prev = temp;

new\_node->next = NULL;

temp->next = new\_node;

temp = new\_node;

while (choice != 6)

{

printf("\nEnter you choice:\n1.Deletion at starting\n2.Deletion at any valid position\n3. Deletion at the end\n4.Search a node\n5.Display liked-list\n6. Exit\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

head = deleteAtBeginning(head);

break;

case 2:

head = deleteAtPosition(head);

break;

case 3:

head = deleteAtEnd(head);

break;

case 4:

searchNode(head);

break;

case 5:

display(head);

case 6:

break;

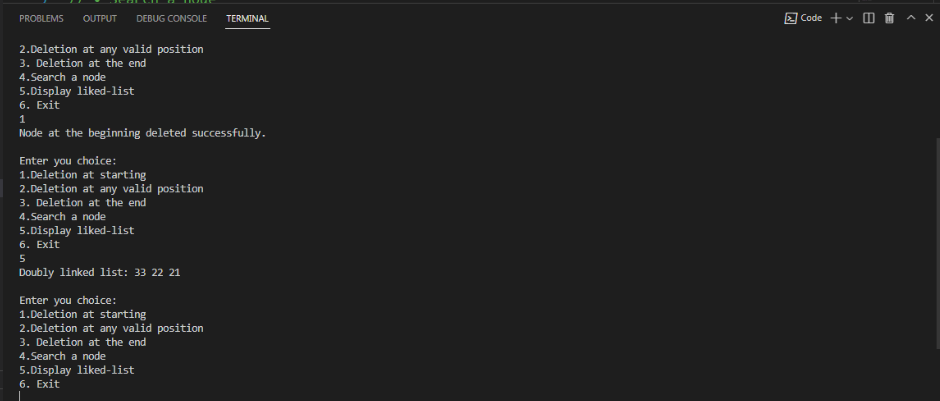
}

}

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Q15. Write a menu driven C program to implement of circular linked list with following operations:**

**• Create the list**

**• Insert a node in the beginning, in the end, at given position**

**• Delete a node in the beginning, in the end, at given position**

**• Search a node**

**• Display the list**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*insertAtBeginning(struct Node \*head)

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

int data;

if (head == NULL)

{

printf("Enter data: ");

scanf("%d", &data);

newNode->data = data;

head = newNode;

newNode->next = head;

return head;

}

else

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

printf("Enter data: ");

scanf("%d", &data);

newNode->data = data;

newNode->next = head;

struct Node \*temp = head;

while (temp->next != head)

{

temp = temp->next;

}

temp->next = newNode;

head = newNode;

return head;

}

printf("Node with data %d inserted at the beginning.\n", data);

}

struct Node \*insertAtEnd(struct Node \*head)

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

int data;

printf("Enter data: ");

scanf("%d", &data);

if (head == NULL)

{

newNode->data = data;

head = newNode;

newNode->next = head;

return head;

}

else

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = data;

struct Node \*temp = head;

while (temp->next != head)

{

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

return head;

}

printf("Node with data %d inserted at the end.\n", data);

}

struct Node \*insertAtPosition(struct Node \*head)

{

int position, data;

if (head == NULL)

{

printf("The list is empty. Inserting at the beginning instead.\n");

head = insertAtBeginning(head);

return head;

}

printf("Enter data: ");

scanf("%d", &data);

printf("Enter position: ");

scanf("%d", &position);

struct Node \*current = head;

int currentPosition = 1;

while (current->next != head && currentPosition < position - 1)

{

current = current->next;

currentPosition++;

}

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = current->next;

current->next = newNode;

printf("Node with data %d inserted at position %d.\n", data, position);

return head;

}

struct Node \*deleteAtBeginning(struct Node \*head)

{

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

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

{

free(head);

head = NULL;

return head;

}

else

{

struct Node \*temp = head;

while (temp->next != head)

{

temp = temp->next;

}

temp->next = head->next;

struct Node \*deleteNode = head;

head = head->next;

free(deleteNode);

return head;

}

printf("Node at the beginning deleted successfully.\n");

}

struct Node \*deleteAtEnd(struct Node \*head)

{

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

if (head->next == head)

{

free(head);

head = NULL;

return head;

}

else

{

struct Node \*current = head;

struct Node \*previous;

while (current->next != head)

{

previous = current;

current = current->next;

}

previous->next = head;

free(current);

}

printf("Node at the end deleted successfully.\n");

return head;

}

struct Node \*deleteAtPosition(struct Node \*head)

{

int position;

printf("Enter position: ");

scanf("%d", &position);

if (head == NULL)

{

printf("The list is already empty.\n");

return head;

}

if (position <= 0)

{

printf("Invalid position.\n");

return head;

}

if (position == 1)

{

head = deleteAtBeginning(head);

return head;

}

struct Node \*current = head;

struct Node \*previous;

int currentPosition = 1;

while (current->next != head && currentPosition < position)

{

previous = current;

current = current->next;

currentPosition++;

}

if (currentPosition != position)

{

printf("Invalid position.\n");

return head;

}

previous->next = current->next;

free(current);

printf("Node at position %d deleted successfully.\n", position);

return head;

}

void searchNode(struct Node \*head)

{

int key;

printf("Enter key value: ");

scanf("%d", &key);

if (head == NULL)

{

printf("The list is empty.\n");

return;

}

struct Node \*current = head;

int position = 1;

do

{

if (current->data == key)

{

printf("Node with data %d found at position %d.\n", key, position);

return;

}

current = current->next;

position++;

} while (current != head);

printf("Node with data %d not found in the list.\n", key);

}

void displayList(struct Node \*head)

{

if (head == NULL)

{

printf("The list is empty.\n");

return;

}

struct Node \*current = head;

printf("Circular linked list: ");

do

{

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

current = current->next;

} while (current != head);

printf("\n");

}

int main()

{

struct Node \*head = NULL;

struct Node \*new\_node = (struct Node \*)malloc(sizeof(struct Node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = head;

struct Node \*temp = new\_node;

head = new\_node;

new\_node = (struct Node \*)malloc(sizeof(struct Node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = head;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct Node \*)malloc(sizeof(struct Node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = head;

temp->next = new\_node;

temp = new\_node;

new\_node = (struct Node \*)malloc(sizeof(struct Node));

printf("Enter data for creating node: ");

scanf("%d", &new\_node->data);

new\_node->next = head;

temp->next = new\_node;

temp = new\_node;

int choice;

while (choice != 9)

{

printf("\nCircular Linked List Operations:\n");

printf("1. Insert a node at the beginning\n");

printf("2. Insert a node at the end\n");

printf("3. Insert a node at a given position\n");

printf("4. Delete a node from the beginning\n");

printf("5. Delete a node from the end\n");

printf("6. Delete a node from a given position\n");

printf("7. Search a node\n");

printf("8. Display the list\n");

printf("9. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

head = insertAtBeginning(head);

break;

case 2:

head = insertAtEnd(head);

break;

case 3:

head = insertAtPosition(head);

break;

case 4:

head = deleteAtBeginning(head);

break;

case 5:

head = deleteAtEnd(head);

break;

case 6:

head = deleteAtPosition(head);

break;

case 7:

searchNode(head);

case 8:

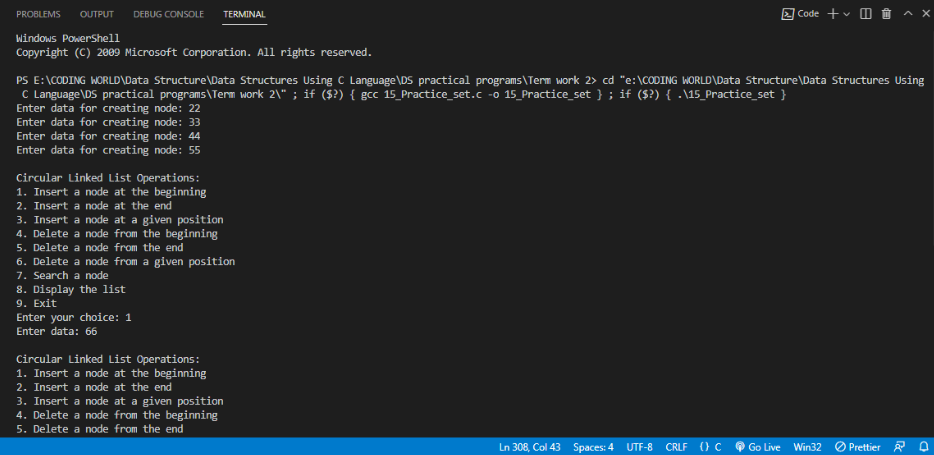
displayList(head);

}

}

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 16. Write a menu driven C program to implement of stack using linked list with following operations :**

**• Check if the stack is empty**

**• Check if the stack is full**

**• Display the contents of stack**

**• Push data**

**• Pop data**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*top = NULL;

int isEmpty()

{

return (top == NULL);

}

int isFull()

{

struct Node \*temp = (struct Node \*)malloc(sizeof(struct Node));

int isFullFlag = (temp == NULL);

free(temp);

return isFullFlag;

}

void push(int data)

{

if (isFull())

{

printf("Stack is full. Cannot push element.\n");

}

else

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = top;

top = newNode;

printf("%d pushed to stack.\n", data);

}

}

void pop()

{

if (isEmpty())

{

printf("Stack is empty. Cannot pop element.\n");

}

else

{

struct Node \*temp = top;

int poppedValue = top->data;

top = top->next;

free(temp);

printf("%d popped from stack.\n", poppedValue);

}

}

void display()

{

if (isEmpty())

{

printf("Stack is empty.\n");

}

else

{

struct Node \*temp = top;

printf("Stack elements: ");

while (temp != NULL)

{

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

temp = temp->next;

}

printf("\n");

}

}

int main()

{

int choice, data;

do

{

printf("Stack Operations\n");

printf("1. Check if the stack is empty\n");

printf("2. Check if the stack is full\n");

printf("3. Display the contents of stack\n");

printf("4. Push data\n");

printf("5. Pop data\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 0:

printf("Exiting...\n");

break;

case 1:

if (isEmpty())

{

printf("Stack is empty.\n");

}

else

{

printf("Stack is not empty.\n");

}

break;

case 2:

if (isFull())

{

printf("Stack is full.\n");

}

else

{

printf("Stack is not full.\n");

}

break;

case 3:

display();

break;

case 4:

printf("Enter data to push: ");

scanf("%d", &data);

push(data);

break;

case 5:

pop();

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

}

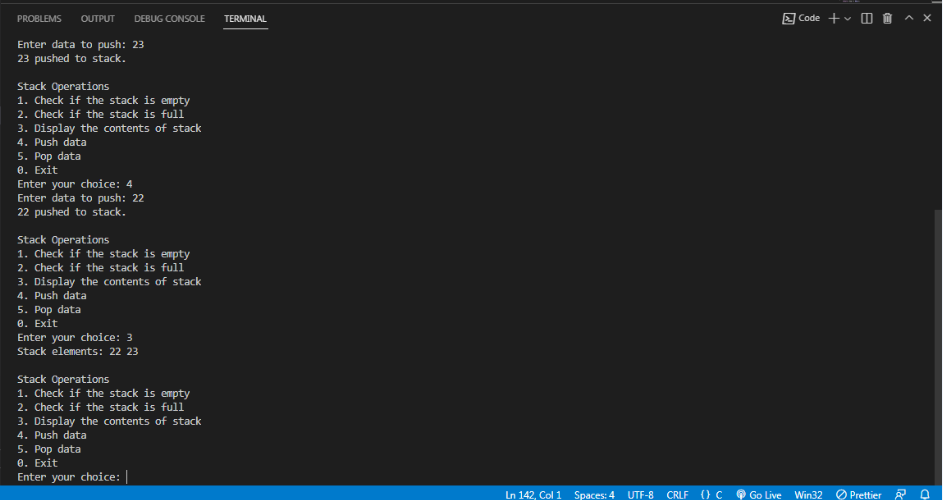
printf("\n");

} while (choice != 0);

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 17: Write a menu driven C program to implement of queue using linked list with following operations :**

**• Check if the stack is empty**

**• Check if the stack is full**

**• Display the contents of queue**

**Enqueue data**

**• Dequeue data**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front = NULL;

struct node \*rear = NULL;

void enqueue()

{

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

printf("Enter data: ");

scanf("%d", &new\_node->data);

new\_node->next = NULL;

if (front == NULL && rear == NULL)

{

front = rear = new\_node;

}

else

{

rear->next = new\_node;

rear = new\_node;

}

}

void dequeue()

{

if (front == NULL)

{

printf("Queue is empty.\n");

return;

}

struct node \*temp = front;

front = front->next;

if (front == NULL)

{

rear = NULL;

}

free(temp);

}

void display\_queue()

{

if (front == NULL)

{

printf("Queue is empty.\n");

return;

}

struct node \*temp = front;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("\n");

}

int is\_empty()

{

return (front == NULL);

}

int is\_full()

{

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

if (new\_node == NULL)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

int choice;

while (1)

{

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Check if empty\n5. Check if full\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display\_queue();

break;

case 4:

if (is\_empty())

{

printf("Queue is empty.\n");

}

else

{

printf("Queue is not empty.\n");

}

break;

case 5:

if (is\_full())

{

printf("Queue is full.\n");

}

else

{

printf("Queue is not full.\n");

}

break;

case 6:

exit(0);

default:

printf("Invalid choice.\n");

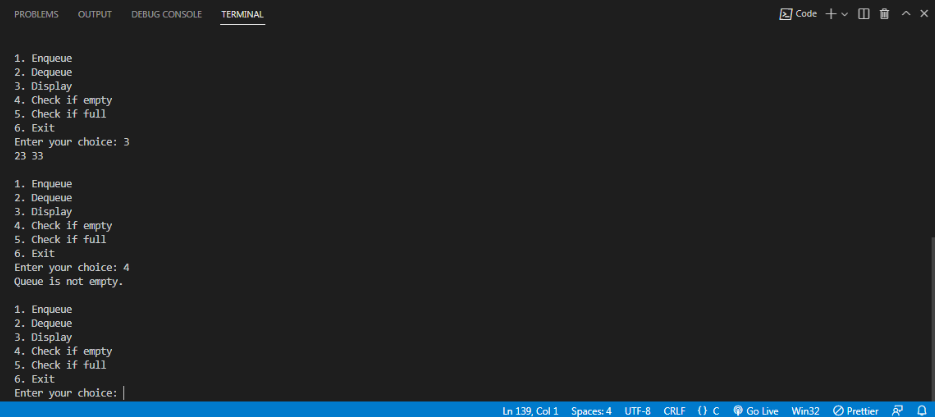
}

}

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 18: Write a C program that implements Linear Search on a set of n Numbers.**

#include <stdio.h>

int linear\_search(int \*array, int n, int x)

{

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

{

if (array[i] == x)

{

return i;

}

}

return -1;

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

int x = 20;

int index = linear\_search(array, n, x);

if (index == -1)

{

printf("Element not found.\n");

}

else

{

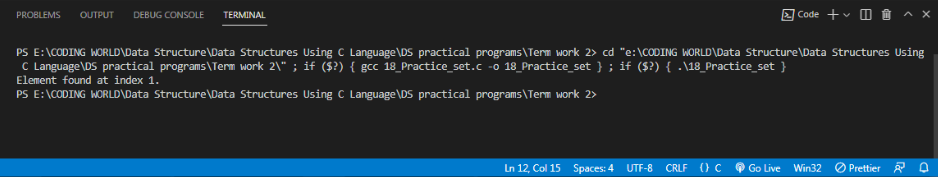
printf("Element found at index %d.\n", index);

}

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**// Problem Statement 19: Write a C program that implements Binary Search on a set of n Numbers.**

#include <stdio.h>

int binary\_search(int \*array, int n, int x)

{

int low = 0;

int high = n - 1;

while (low <= high)

{

int mid = (low + high) / 2;

if (array[mid] == x)

{

return mid;

}

else if (array[mid] < x)

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

return -1;

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

int x = 20;

int index = binary\_search(array, n, x);

if (index == -1)

{

printf("Element not found.\n");

}

else

{

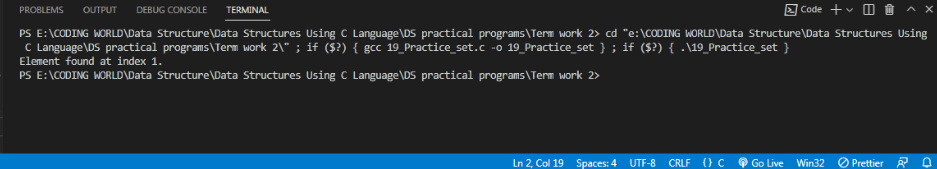
printf("Element found at index %d.\n", index);

}

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 20: Write a C program that implements Bubble Sort on a set of n Numbers.**

#include <stdio.h>

void bubble\_sort(int \*array, int n)

{

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

{

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

{

if (array[j] > array[j + 1])

{

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

bubble\_sort(array, n);

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

{

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

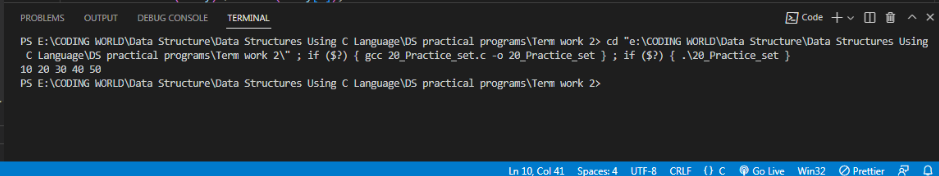
}

printf("\n");

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 21: Write a C program that implements Insertion Sort on a set of n Numbers.**

#include <stdio.h>

void insertion\_sort(int \*array, int n)

{

for (int i = 1; i < n; i++)

{

int current = array[i];

int j = i - 1;

while (j >= 0 && array[j] > current)

{

array[j + 1] = array[j];

j--;

}

array[j + 1] = current;

}

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

insertion\_sort(array, n);

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

{

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

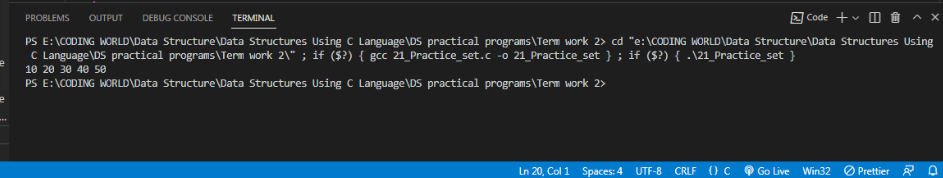
}

printf("\n");

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 22: Write a C program that implements Selection Sort on a set of n Numbers.**

#include <stdio.h>

void selection\_sort(int \*array, int n)

{

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

{

int min\_index = i;

for (int j = i + 1; j < n; j++)

{

if (array[j] < array[min\_index])

{

min\_index = j;

}

}

int temp = array[i];

array[i] = array[min\_index];

array[min\_index] = temp;

}

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

selection\_sort(array, n);

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

{

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

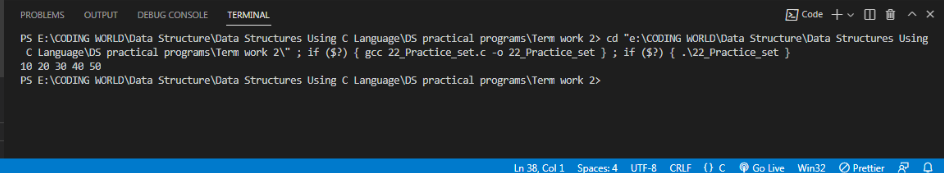
}

printf("\n");

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 23: Write a C program that implements Merge Sort on a set of n Numbers.**

#include <stdio.h>

void merge(int \*array, int low, int mid, int high)

{

int i = low;

int j = mid + 1;

int k = low;

int temp[high - low + 1];

while (i <= mid && j <= high)

{

if (array[i] <= array[j])

{

temp[k++] = array[i++];

}

else

{

temp[k++] = array[j++];

}

}

while (i <= mid)

{

temp[k++] = array[i++];

}

while (j <= high)

{

temp[k++] = array[j++];

}

for (i = low; i <= high; i++)

{

array[i] = temp[i];

}

}

void merge\_sort(int \*array, int low, int high)

{

if (low < high)

{

int mid = (low + high) / 2;

merge\_sort(array, low, mid);

merge\_sort(array, mid + 1, high);

merge(array, low, mid, high);

}

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

merge\_sort(array, 0, n - 1);

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

{

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

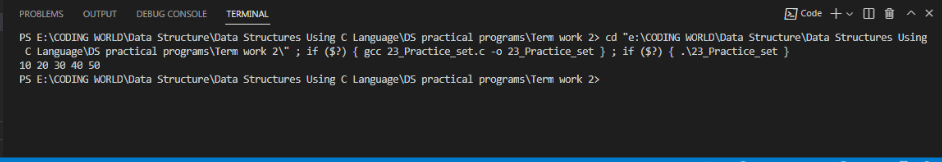
}

printf("\n");

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 24: Write a C program that implements Quick Sort on a set of n Numbers.**

#include <stdio.h>

int partition(int \*array, int low, int high)

{

int pivot = array[high];

int i = low - 1;

for (int j = low; j < high; j++)

{

if (array[j] <= pivot)

{

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

void quick\_sort(int \*array, int low, int high)

{

if (low < high)

{

int pivot = partition(array, low, high);

quick\_sort(array, low, pivot - 1);

quick\_sort(array, pivot + 1, high);

}

}

int main()

{

int array[] = {10, 20, 30, 40, 50};

int n = sizeof(array) / sizeof(array[0]);

quick\_sort(array, 0, n - 1);

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

{

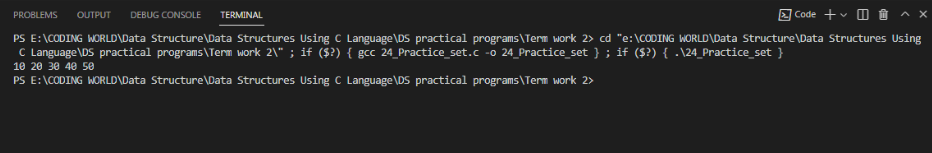
printf("%d ", array[i]);

}

printf("\n");

return 0;

}

// Output:

**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement 25: Write a C program for binary search tree traversal: in-order , post-order, pre-order.**

#include <stdio.h>

#include <stdlib.h>

// Binary Search Tree (BST) Node

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

};

// Function to create a new BST node

struct Node \*createNode(int data)

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function to insert a node into BST

struct Node \*insertNode(struct Node \*root, int data)

{

if (root == NULL)

{

return createNode(data);

}

if (data < root->data)

{

root->left = insertNode(root->left, data);

}

else if (data > root->data)

{

root->right = insertNode(root->right, data);

}

return root;

}

// Function for in-order BST traversal (left-root-right)

void inorderTraversal(struct Node \*root)

{

if (root != NULL)

{

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

// Function for pre-order BST traversal (root-left-right)

void preorderTraversal(struct Node \*root)

{

if (root != NULL)

{

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

}

// Function for post-order BST traversal (left-right-root)

void postorderTraversal(struct Node \*root)

{

if (root != NULL)

{

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

}

int main()

{

struct Node \*root = NULL;

root = insertNode(root, 50);

insertNode(root, 30);

insertNode(root, 20);

insertNode(root, 40);

insertNode(root, 70);

insertNode(root, 60);

insertNode(root, 80);

printf("In-order traversal: ");

inorderTraversal(root);

printf("\n");

printf("Pre-order traversal: ");

preorderTraversal(root);

printf("\n");

printf("Post-order traversal: ");

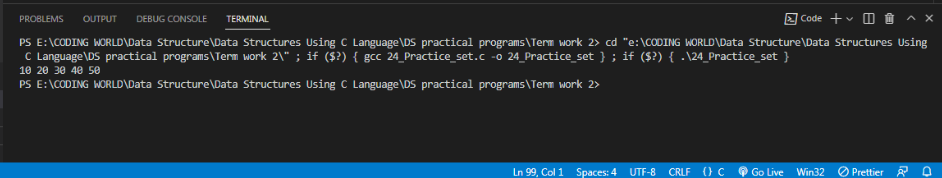
postorderTraversal(root);

printf("\n");

return 0;

}

// Output:



**Name – Ashish Kothari Subject code – PBC 201**

**Section – D1 Course – BCA(II)**

**Roll No. – 17 Subject – Data Structure.**

**Problem Statement : 26 Write a C program to implement BST with the following operations:**

**a) Insert an element into a binary search tree.**

**b) Delete an element from a binary search tree.**

**c) Search for a key element in a binary search tree.**

**d) Count the number of nodes the binary search tree.**

**e) Display the elements in in-order manner.**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

};

struct Node \*createNode(int data)

{

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct Node \*insertNode(struct Node \*root, int data)

{

if (root == NULL)

{

return createNode(data);

}

if (data < root->data)

{

root->left = insertNode(root->left, data);

}

else if (data > root->data)

{

root->right = insertNode(root->right, data);

}

return root;

}

struct Node \*findMinValueNode(struct Node \*node)

{

struct Node \*current = node;

while (current && current->left != NULL)

{

current = current->left;

}

return current;

}

struct Node \*deleteNode(struct Node \*root, int key)

{

if (root == NULL)

{

return root;

}

if (key < root->data)

{

root->left = deleteNode(root->left, key);

}

else if (key > root->data)

{

root->right = deleteNode(root->right, key);

}

else

{

if (root->left == NULL)

{

struct Node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

struct Node \*temp = root->left;

free(root);

return temp;

}

struct Node \*temp = findMinValueNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

struct Node \*searchNode(struct Node \*root, int key)

{

if (root == NULL || root->data == key)

{

return root;

}

if (key < root->data)

{

return searchNode(root->left, key);

}

return searchNode(root->right, key);

}

int countNodes(struct Node \*root)

{

if (root == NULL)

{

return 0;

}

return countNodes(root->left) + countNodes(root->right) + 1;

}

void inorderTraversal(struct Node \*root)

{

if (root != NULL)

{

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

int main()

{

struct Node \*root = NULL;

int choice, data, key;

do

{

printf("\nBinary Search Tree Operations:\n");

printf("1. Insert an element\n");

printf("2. Delete an element\n");

printf("3. Search for an element\n");

printf("4. Count the number of nodes\n");

printf("5. Display elements in in-order\n");

printf("0. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter element to insert: ");

scanf("%d", &data);

root = insertNode(root, data);

break;

case 2:

printf("Enter element to delete: ");

scanf("%d", &key);

root = deleteNode(root, key);

break;

case 3:

printf("Enter element to search: ");

scanf("%d", &key);

if (searchNode(root, key) != NULL)

{

printf("Element %d is found in the tree.\n", key);

}

else

{

printf("Element %d is not found in the tree.\n", key);

}

break;

case 4:

printf("Number of nodes in the tree: %d\n", countNodes(root));

break;

case 5:

printf("In-order traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 0:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please try again.\n");

break;

}

} while (choice != 0);

return 0;

}

// Output:

