Q1.Basic Programs - Control Flow Structure (Expression, Conditional, looping, data Manipulation.

A. Expression and Basic Arithmetic

#Algorithm:

#Code:

#include <stdio.h>

int main() {

int a = 10, b = 5;

int sum = a + b;

int product = a \* b;

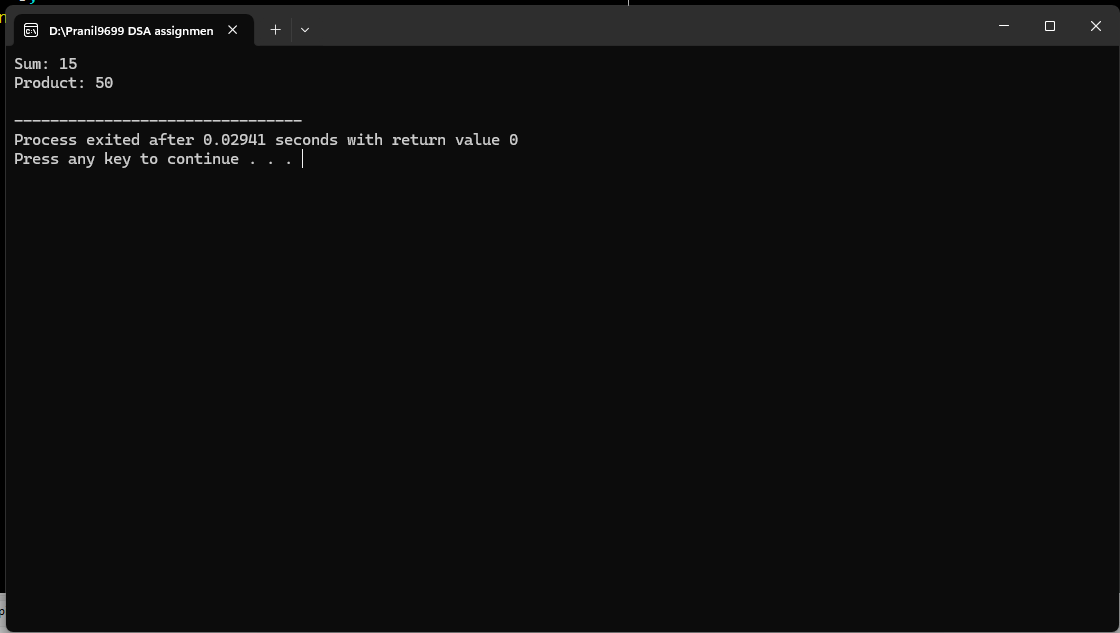
printf("Sum: %d\n", sum);

printf("Product: %d\n", product);

return 0;

}

#Output:



B. Conditional Statement (if-else)

#Algorithm:

#Code:

#include <stdio.h>

int main() {

int number;

printf("Enter a number: ");

scanf("%d", &number);

if (number % 2 == 0) {

printf("The number %d is even.\n", number);

} else {

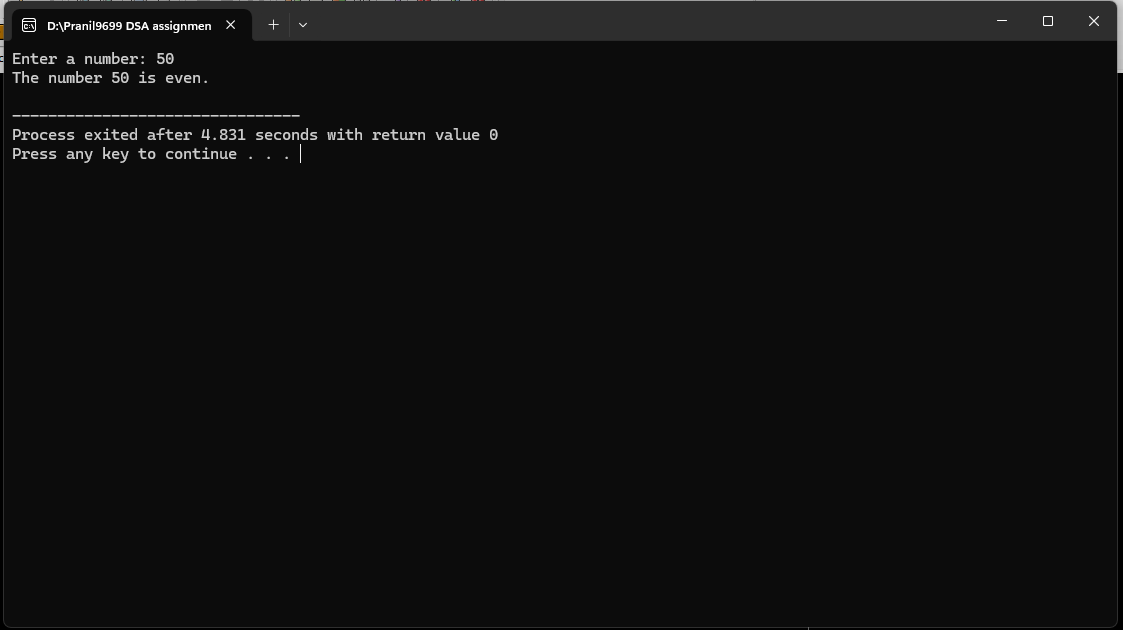
printf("The number %d is odd.\n", number);

}

return 0;

}

#Output:



C. Looping Structure (for loop)

#Algorithm:

#Code:

#include <stdio.h>

int main() {

printf("Numbers from 1 to 5:\n");

int i;

for (i = 1; i <= 5; i++) {

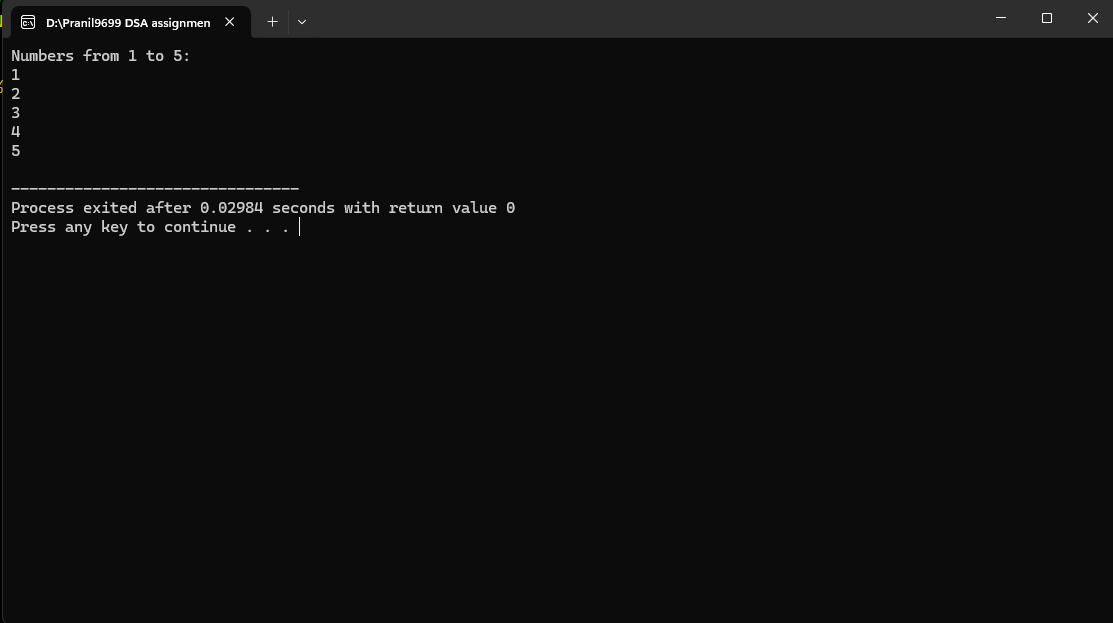
printf("%d\n", i);

}

return 0;

}

#Output:



D. Looping Structure (while loop)

#Algorithm:

#Code :

#include <stdio.h>

int main() {

int count = 1;

printf("Counting from 1 to 5 using while loop:\n");

while (count <= 5) {

printf("%d\n", count);

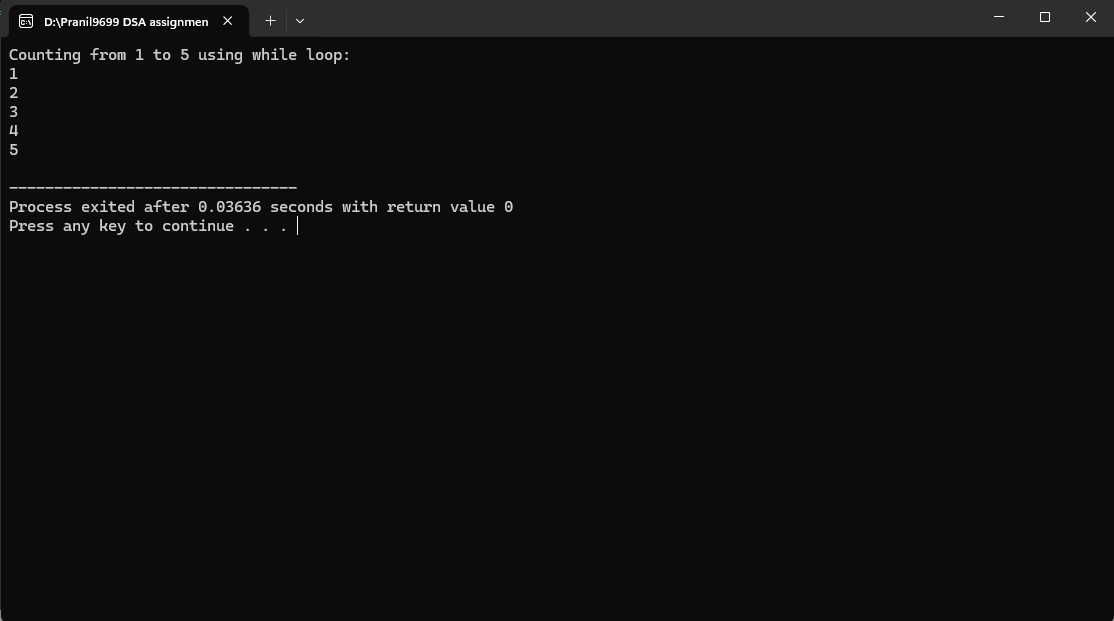
count++;

}

return 0;

}

#Output



E. Looping and Conditional (Multiplication Table using Nested Loop)

#Algorithm:

#Code:

#include <stdio.h>

int main() {

printf("Multiplication Table (1 to 5):\n");

int i,j;

for ( i = 1; i <= 5; i++) {

for (j = 1; j <= 5; j++) {

printf("%d x %d = %d\t", i, j, i \* j);

}

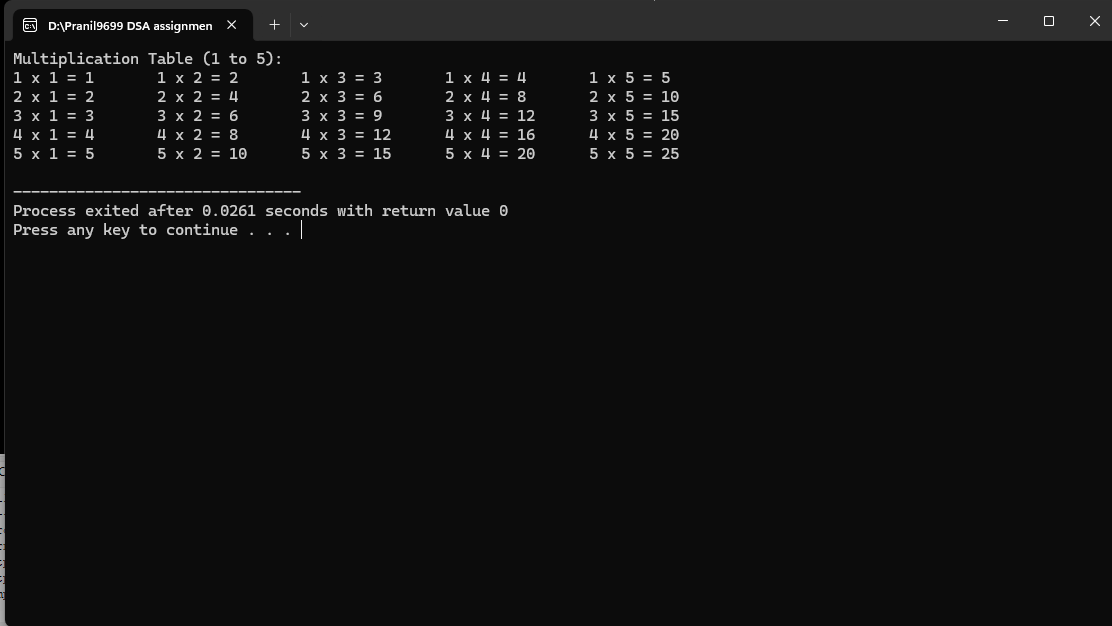
printf("\n");

}

return 0;

}

#Output:



F. Data Manipulation (Array Traversal)

#Algorithm:

#Code:

#include <stdio.h>

int main() {

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

int size = sizeof(numbers) / sizeof(numbers[0]);

printf("Elements of the array:\n");

int i;

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

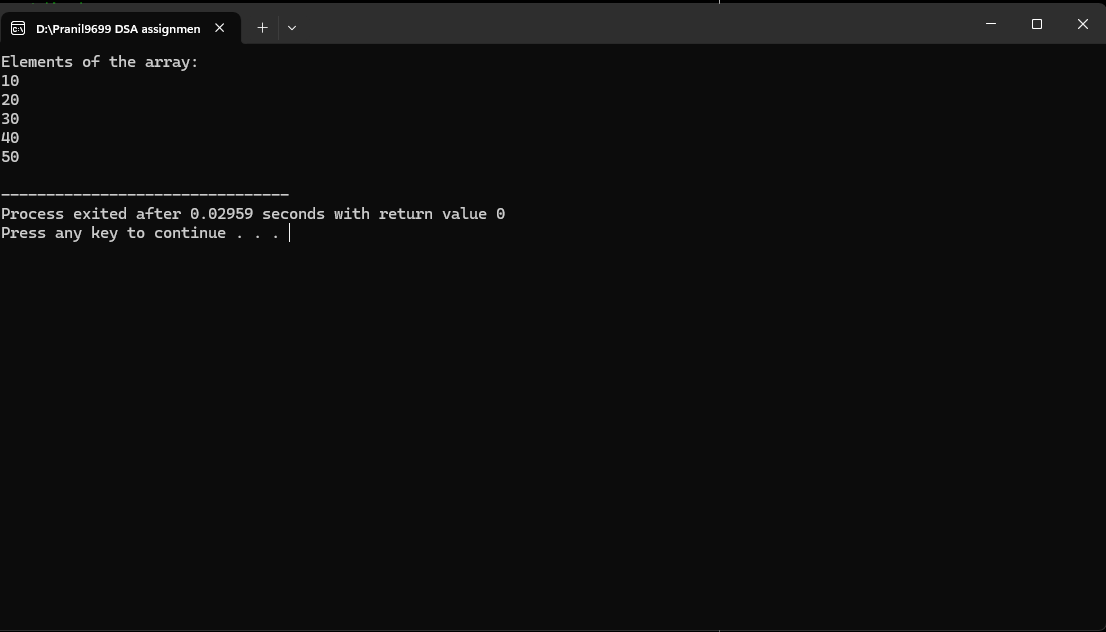
printf("%d\n", numbers[i]);

}

return 0;

}

#Output:



G. Data Manipulation (String Handling)

#Algorithm:

#Code:

#include <stdio.h>

#include <string.h>

int main() {

char str1[] = "Pranil";

char str2[] = "Takawane";

char result[20];

strcpy(result, str1);

strcat(result, " ");

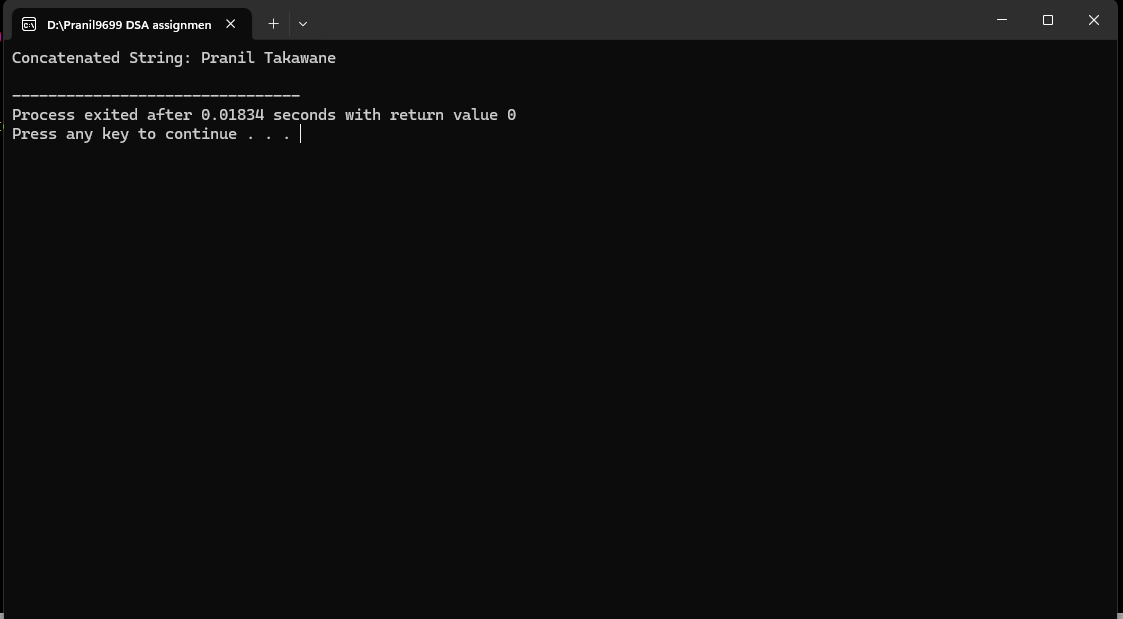
strcat(result, str2);

printf("Concatenated String: %s\n", result);

return 0;

}

#Output:



Q2. Implement insertion ,deletion ,and searching in single-dimensional and multidimensional.

A.Single-Dimensional Array

i) Inertion:

#Algorithm:

#Code:

#include <stdio.h>

int main() {

int arr[100] = {1, 2, 3, 4, 5};

int n = 5; // Initial size of the array

int pos = 3, value = 10; // Insert value 10 at position 3

printf("Original Array:\n");

int i;

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

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

}

printf("\n");

// Shift elements to the right

for ( i = n; i >= pos; i--) {

arr[i] = arr[i - 1];

}

arr[pos - 1] = value; // Insert value

n++; // Increase size

printf("Array after insertion:\n");

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

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

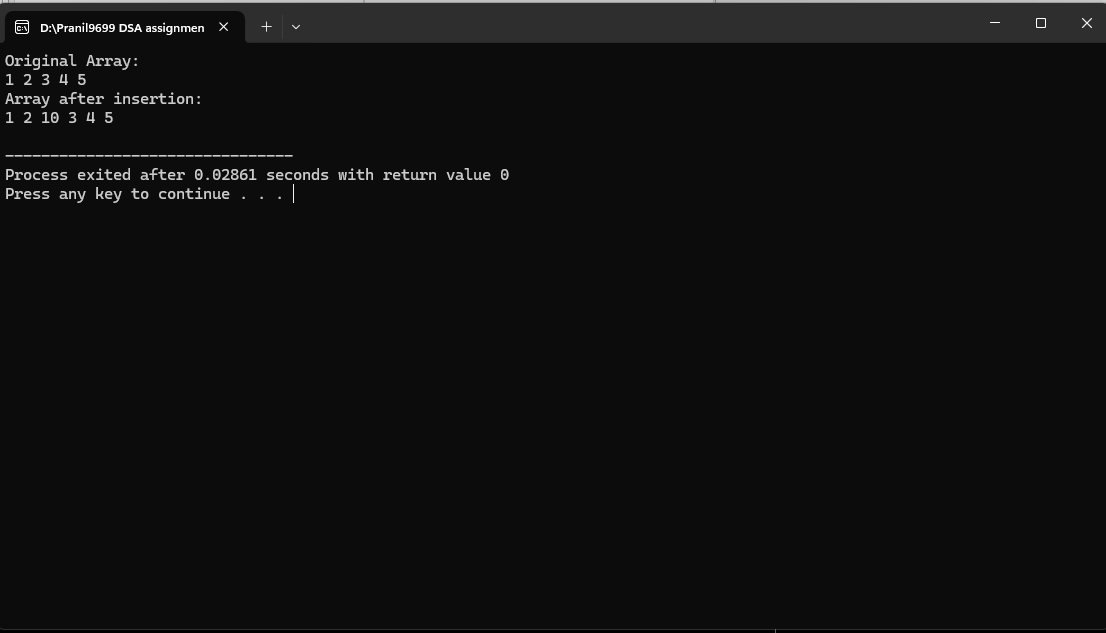
}

printf("\n");

return 0;

}

#Output:



ii) Deletion:

#Algorithm:

#Code:

#include <stdio.h>

int main() {

int arr[100] = {1, 2, 3, 4, 5};

int n = 5; // Initial size of the array

int pos = 3; // Delete element at position 3

printf("Original Array:\n");

int i;

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

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

}

printf("\n");

// Shift elements to the left

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

arr[i] = arr[i + 1];

}

n--; // Decrease size

printf("Array after deletion:\n");

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

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

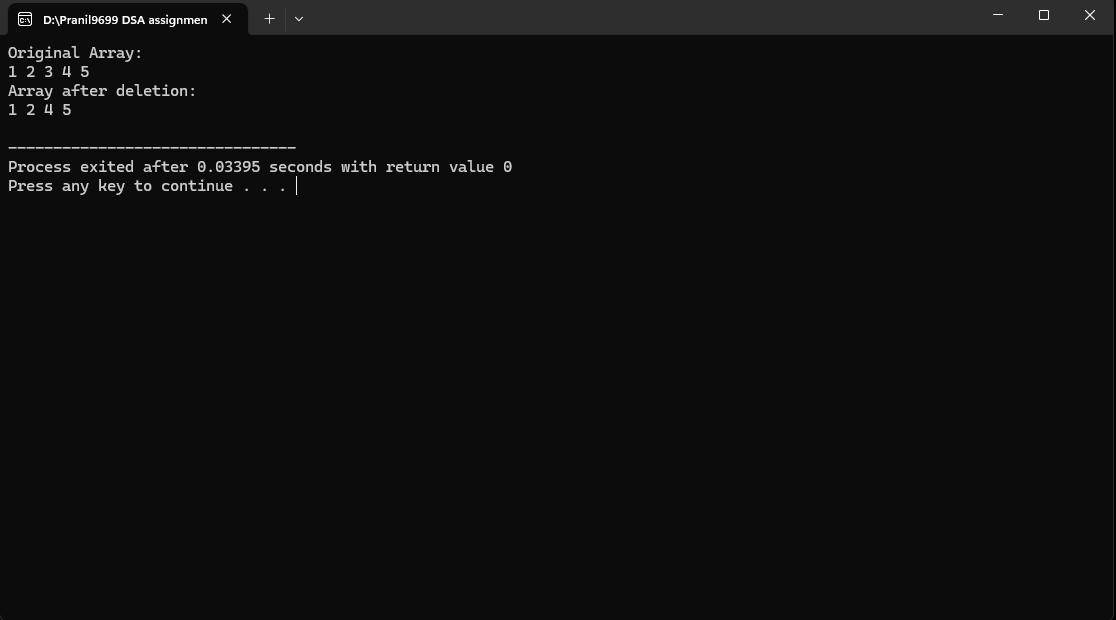
}

printf("\n");

return 0;

}

#Output:



ii) Searching :

#Algorithm:

#Code :

#include <stdio.h>

int main() {

int arr[] = {1, 2, 3, 4, 5};

int n = 5;

int key = 3; // Element to search for

int found = 0;

int i;

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

if (arr[i] == key) {

printf("Element %d found at position %d.\n", key, i + 1);

found = 1;

break;

}

}

if (!found) {

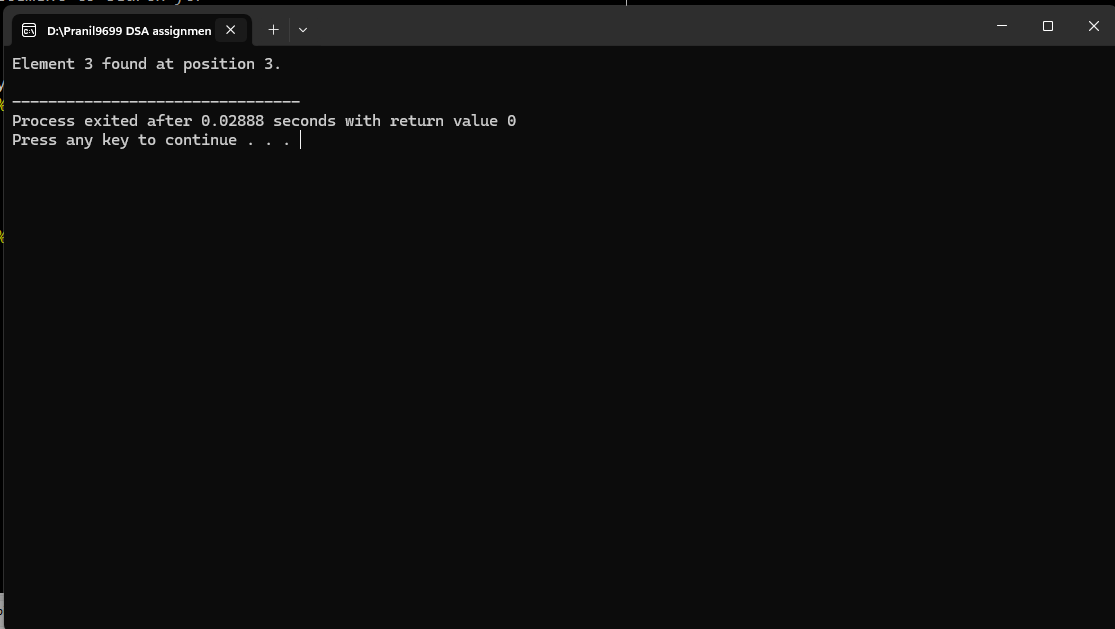
printf("Element %d not found in the array.\n", key);

}

return 0;

}

#Output:



Q5. Implement stack operations (push, pop, peek) using both arrays and linked lists.

A. Stack Implementation Using Arrays:

#Algorithm:

#Code:

#include <stdio.h>

#define MAX 100

// Stack structure using array

typedef struct {

int arr[MAX];

int top;

} Stack;

// Initialize stack

void initializeStack(Stack\* stack) {

stack->top = -1;

}

// Push operation

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

if (stack->top == MAX - 1) {

printf("Stack Overflow! Cannot push %d\n", value);

} else {

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

printf("Pushed %d onto the stack\n", value);

}

}

// Pop operation

int pop(Stack\* stack) {

if (stack->top == -1) {

printf("Stack Underflow! Cannot pop\n");

return -1;

} else {

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

}

}

// Peek operation

int peek(Stack\* stack) {

if (stack->top == -1) {

printf("Stack is empty\n");

return -1;

} else {

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

}

}

// Display stack

void display(Stack\* stack) {

if (stack->top == -1) {

printf("Stack is empty\n");

} else {

printf("Stack elements: ");

int i;

for ( i = 0; i <= stack->top; i++) {

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

}

printf("\n");

}

}

int main() {

Stack stack;

initializeStack(&stack);

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

display(&stack);

printf("Popped: %d\n", pop(&stack));

display(&stack);

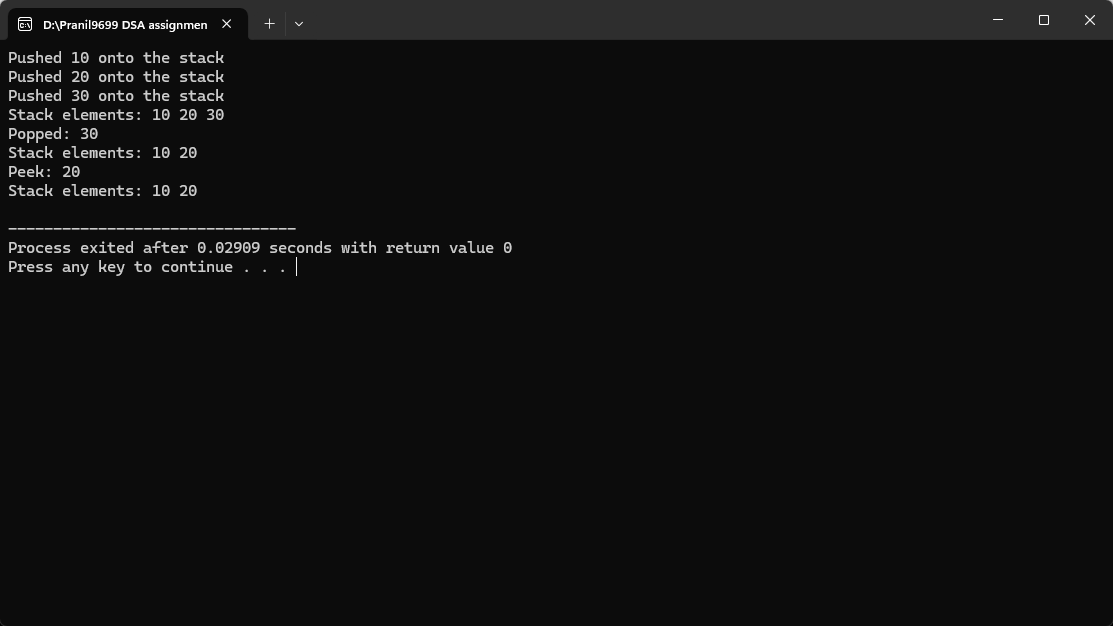
printf("Peek: %d\n", peek(&stack));

display(&stack);

return 0;

}

#Output:



B. Stack Implementation Using Linked Lists

#Algorithm:

#Code:

#include <stdio.h>

#include <stdlib.h>

// Node structure for stack

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Push operation

void push(Node\*\* top, int value) {

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

if (!newNode) {

printf("Stack Overflow! Cannot push %d\n", value);

return;

}

newNode->data = value;

newNode->next = \*top;

\*top = newNode;

printf("Pushed %d onto the stack\n", value);

}

// Pop operation

int pop(Node\*\* top) {

if (\*top == NULL) {

printf("Stack Underflow! Cannot pop\n");

return -1;

}

Node\* temp = \*top;

int value = temp->data;

\*top = (\*top)->next;

free(temp);

return value;

}

// Peek operation

int peek(Node\* top) {

if (top == NULL) {

printf("Stack is empty\n");

return -1;

}

return top->data;

}

// Display stack

void display(Node\* top) {

if (top == NULL) {

printf("Stack is empty\n");

return;

}

printf("Stack elements: ");

Node\* temp = top;

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

Node\* stack = NULL;

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

display(stack);

printf("Popped: %d\n", pop(&stack));

display(stack);

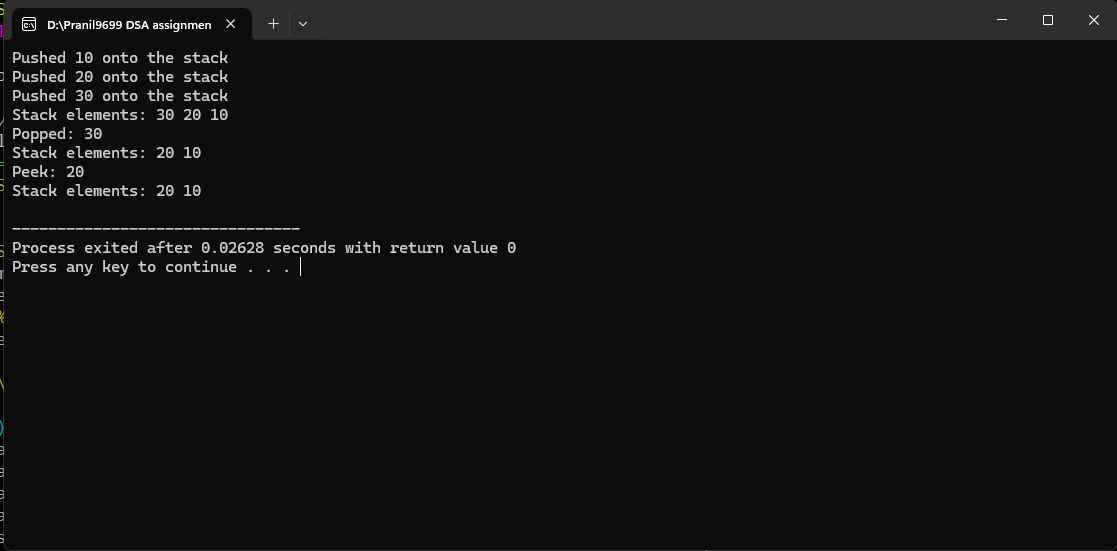
printf("Peek: %d\n", peek(stack));

display(stack);

return 0;

}

#Output:



Q6. Implement queue operations (enqueue, dequeue) using arrays and

linked lists.

A. Queue Implementation Using Arrays

#Algorithm:

#Code:

#include <stdio.h>

#define MAX 100

typedef struct {

int arr[MAX];

int front;

int rear;

} Queue;

// Initialize the queue

void initializeQueue(Queue\* queue) {

queue->front = -1;

queue->rear = -1;

}

// Enqueue operation

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

if (queue->rear == MAX - 1) {

printf("Queue Overflow! Cannot enqueue %d\n", value);

return;

}

if (queue->front == -1) queue->front = 0; // Initialize front for first element

queue->arr[++queue->rear] = value;

printf("Enqueued %d into the queue\n", value);

}

// Dequeue operation

int dequeue(Queue\* queue) {

if (queue->front == -1 || queue->front > queue->rear) {

printf("Queue Underflow! Cannot dequeue\n");

return -1;

}

return queue->arr[queue->front++];

}

// Display queue

void display(Queue\* queue) {

if (queue->front == -1 || queue->front > queue->rear) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

int i;

for ( i = queue->front; i <= queue->rear; i++) {

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

}

printf("\n");

}

int main() {

Queue queue;

initializeQueue(&queue);

enqueue(&queue, 10);

enqueue(&queue, 20);

enqueue(&queue, 30);

display(&queue);

printf("Dequeued: %d\n", dequeue(&queue));

display(&queue);

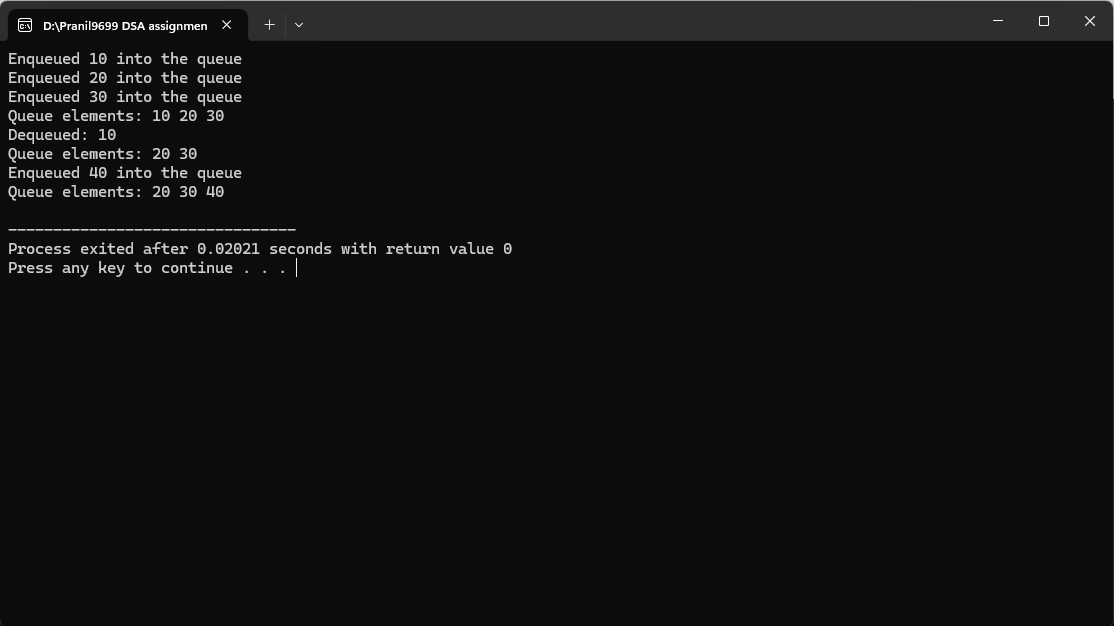
enqueue(&queue, 40);

display(&queue);

return 0;

}

#Output:



B. Queue Implementation Using Linked List

#Algorithm:

#Code:

#include <stdio.h>

#include <stdlib.h>

// Node structure for the queue

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Enqueue operation

void enqueue(Node\*\* front, Node\*\* rear, int value) {

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

if (!newNode) {

printf("Queue Overflow! Cannot enqueue %d\n", value);

return;

}

newNode->data = value;

newNode->next = NULL;

if (\*rear == NULL) { // First element in the queue

\*front = \*rear = newNode;

} else {

(\*rear)->next = newNode;

\*rear = newNode;

}

printf("Enqueued %d into the queue\n", value);

}

// Dequeue operation

int dequeue(Node\*\* front, Node\*\* rear) {

if (\*front == NULL) {

printf("Queue Underflow! Cannot dequeue\n");

return -1;

}

Node\* temp = \*front;

int value = temp->data;

\*front = (\*front)->next;

if (\*front == NULL) \*rear = NULL; // Queue is empty now

free(temp);

return value;

}

// Display queue

void display(Node\* front) {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

while (front != NULL) {

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

front = front->next;

}

printf("\n");

}

int main() {

Node\* front = NULL;

Node\* rear = NULL;

enqueue(&front, &rear, 10);

enqueue(&front, &rear, 20);

enqueue(&front, &rear, 30);

display(front);

printf("Dequeued: %d\n", dequeue(&front, &rear));

display(front);

enqueue(&front, &rear, 40);

display(front);

return 0;

}

#Output:

