Ambekeshwar Group Of Institutions



Technology & Management, Lucknow

**Session: 2023-24**

Practical File

# **Branch:- CSE 2nd Year | 3rd Sem**

**Subject:- Data Structure Using C**

# **Name:- Suraj Arya**

# **Date:-**

Name Signature

Subject Teacher Miss. Richa Mishra …………………………

Class Co-Ordinator Miss. Fiza Hussain ……………………..…

Head of Department Miss. Richa Mishra …………………………

Academic Co-Ordinator Mr. Shivanshu Singh …………………………

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. no.** | **Name of Exp.** | **Date** | **Subject**  **Teacher**  **Sign** | **Academic**  **Co-Ordinator**  **Sign** |
| 1. | Addition of two matrices using functions. | 13/Sep/2023 |  |  |
| 2. | Multiplication of two matrices | 20/Sep/2023 |  |  |
| 3. | Push and POP operation in stack | 11/Oct/2023 |  |  |
| 4. | Insertion and deleting elements in queue. | 17/ Oct/2023 |  |  |
| 5. | The binary search procedures to search on element in a given list | 25/Oct/2023 |  |  |
| 6. | The Selection sort technique. | 7/Nov/2023 |  |  |
| 7. | The Factorial of given number with recursion. | 15/Nov/2023 |  |  |

**Practical No: 1**

**AIM : Addition of two matrices using functions.**

#include <stdio.h>

int main() {

    int r, c, a[100][100], b[100][100], sum[100][100], i, j;

    printf("Enter the number of rows (between 1 and 100): ");

    scanf("%d", &r);

    printf("Enter the number of columns (between 1 and 100): ");

    scanf("%d", &c);

    printf("\nEnter elements of the 1st Matrix:\n");

    for (i = 0; i < r; ++i) {

        for (j = 0; j < c; ++j) {

            printf("Enter element a%d%d: ", i + 1, j + 1);

            scanf("%d", &a[i][j]);

        }

    }

    printf("\nEnter elements of the 2nd Matrix:\n");

    for (i = 0; i < r; ++i) {

        for (j = 0; j < c; ++j) {

            printf("Enter element b%d%d: ", i + 1, j + 1);

            scanf("%d", &b[i][j]);

        }

    }

    // Adding two matrices

    for (i = 0; i < r; ++i) {

        for (j = 0; j < c; ++j) {

            sum[i][j] = a[i][j] + b[i][j];

        }

    }

    // Printing the result

    printf("\nSum of two matrices:\n");

    for (i = 0; i < r; ++i) {

        for (j = 0; j < c; ++j) {

            printf("%d\t", sum[i][j]); // Add a tab (\t) for spacing

        }

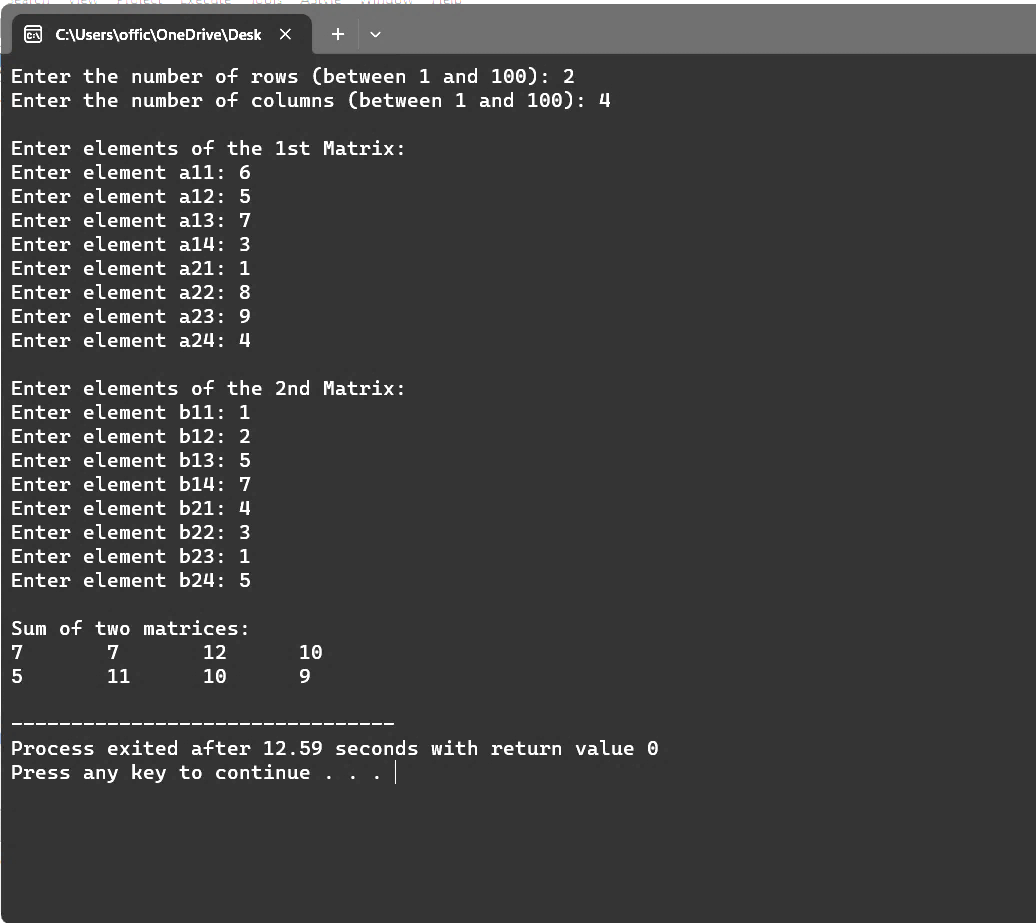
        printf("\n"); // Move to the next row

    }

 return 0;

}

**Output:**



**Practical No: 2**

AIM : Multiplication of two matrices

#include <stdio.h>

int main() {

    int a[10][10], b[10][10], mul[10][10];

    int r1, c1, r2, c2, i, j, k;

    printf("Enter the number of rows for the first matrix: ");

    scanf("%d", &r1);

    printf("Enter the number of columns for the first matrix: ");

    scanf("%d", &c1);

    printf("Enter the number of rows for the second matrix: ");

    scanf("%d", &r2);

    printf("Enter the number of columns for the second matrix: ");

    scanf("%d", &c2);

    if (c1 != r2) {

        printf("Matrix multiplication is not possible.\n");

        return 1;

    }

    printf("Enter the elements of the first matrix:\n");

    for (i = 0; i < r1; i++) {

        for (j = 0; j < c1; j++) {

            scanf("%d", &a[i][j]);

        }

    }

    printf("Enter the elements of the second matrix:\n");

    for (i = 0; i < r2; i++) {

        for (j = 0; j < c2; j++) {

            scanf("%d", &b[i][j]);

        }

    }

    // Multiplication of matrices

    for (i = 0; i < r1; i++) {

        for (j = 0; j < c2; j++) {

            mul[i][j] = 0;

            for (k = 0; k < c1; k++) {

                mul[i][j] += a[i][k] \* b[k][j];

            }

        }

    }

    // Printing the result

    printf("Multiplication of matrices:\n");

    for (i = 0; i < r1; i++) {

        for (j = 0; j < c2; j++) {

            printf("%d\t", mul[i][j]);

        }

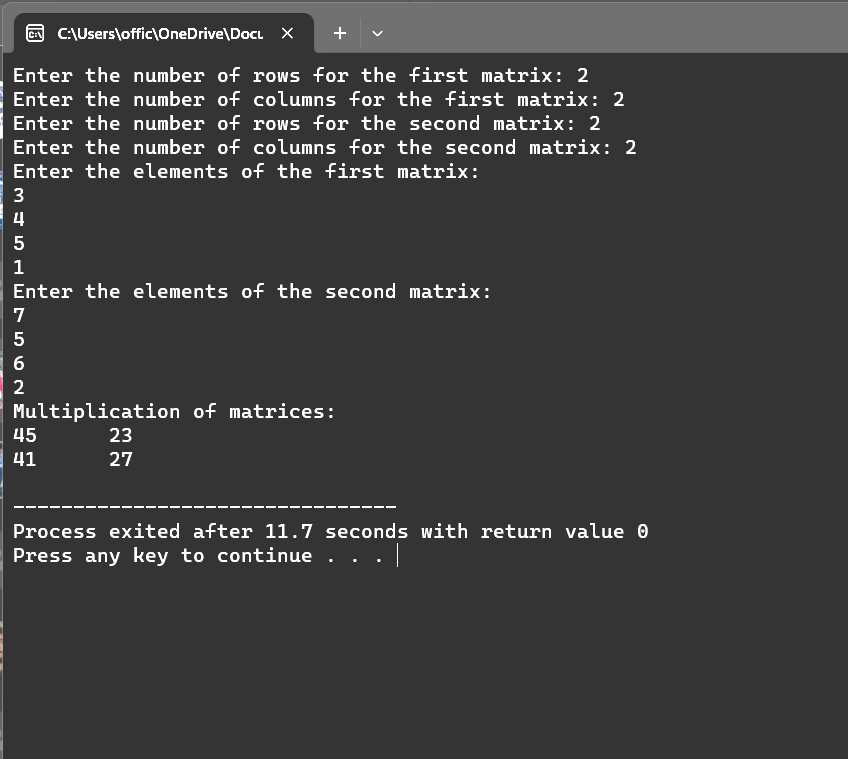
        printf("\n");

    }

return 0;

}

Output:



**Practical No:- 3**

**AIM : Push and POP operation in stack**

**PUSH Operation**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

void push(int value) {

    if (top == MAX\_SIZE - 1) {

        printf("Stack is full. Cannot push %d\n", value);

    } else {

        stack[++top] = value;

        printf("%d pushed to the stack\n", value);

    }

}

int main() {

    push(10);

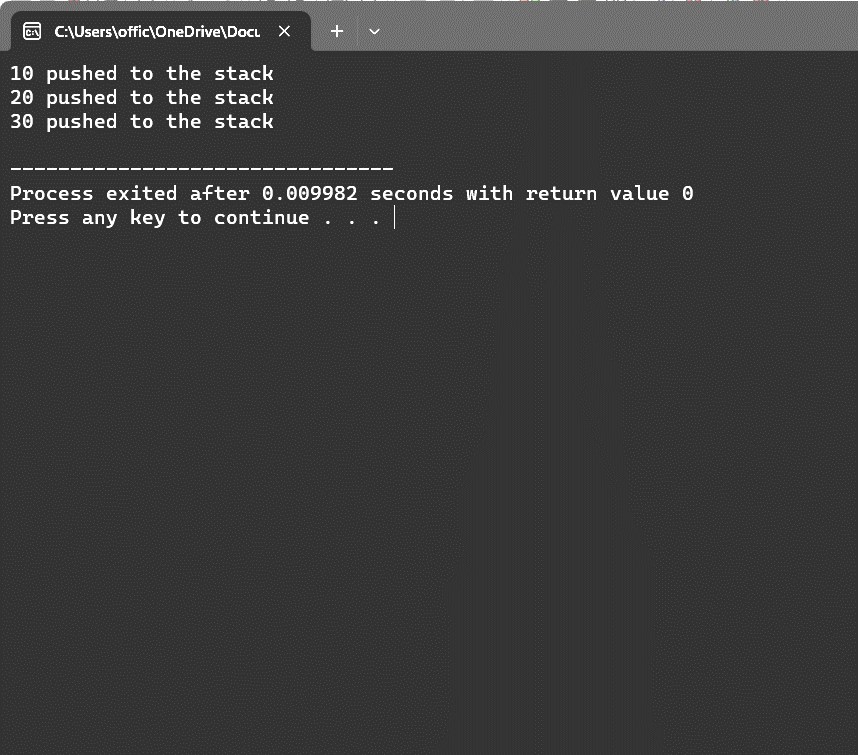
    push(20);

    push(30);

return 0;

}

Output:



**POP Operation**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int stack[MAX\_SIZE];

int top = -1;

void push(int value) {

    if (top == MAX\_SIZE - 1) {

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

    } else {

        stack[++top] = value;

    }

}

int pop() {

    if (top == -1) {

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

        return -1; // You can choose an appropriate value to indicate an error.

    } else {

        int value = stack[top--];

        return value;

    }

}

int main() {

    push(10);

    push(20);

    push(30);

    int poppedValue = pop();

    if (poppedValue != -1) {

        printf("Popped value: %d\n", poppedValue);

    }

    return 0;

}

Output:



**PRACTICAL: - 4**

**AIM : Insertion and deleting elements in queue.**

* **Enqueue operation in Queue**

#include <stdio.h>

#include <stdlib.h>

struct QueueNode {

    int data;

    struct QueueNode\* next;

};

struct Queue {

    struct QueueNode\* front;

    struct QueueNode\* rear;

};

struct QueueNode\* createNode(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

struct Queue\* createQueue() {

    struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

    queue->front = queue->rear = NULL;

    return queue;

}

void enqueue(struct Queue\* queue, int data) {

    struct QueueNode\* newNode = createNode(data);

    if (queue->rear == NULL) {

        queue->front = queue->rear = newNode;

    } else {

        queue->rear->next = newNode;

        queue->rear = newNode;

    }

    printf("%d enqueued to the queue\n", data);

}

int main() {

    struct Queue\* queue = createQueue();

    enqueue(queue, 29);

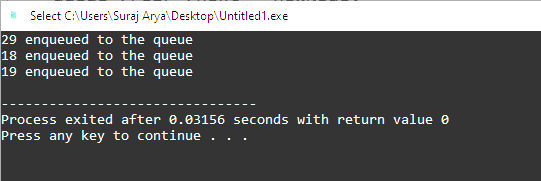
    enqueue(queue, 18);

    enqueue(queue, 19);

return 0;

}

**Output**



**Dequeue operation in Queue**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a queue node

struct QueueNode {

    int data;

    struct QueueNode\* next;

};

// Define the structure for the queue

struct Queue {

    struct QueueNode\* front;

    struct QueueNode\* rear;

};

// Function to create a new node

struct QueueNode\* createNode(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

// Function to create an empty queue

struct Queue\* createQueue() {

    struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

    queue->front = queue->rear = NULL;

    return queue;

}

// Function to enqueue (insert) an element into the queue

void enqueue(struct Queue\* queue, int data) {

    struct QueueNode\* newNode = createNode(data);

    if (queue->rear == NULL) {

        queue->front = queue->rear = newNode;

    } else {

        queue->rear->next = newNode;

        queue->rear = newNode;

    }

    printf("%d enqueued to the queue\n", data);

}

// Function to dequeue (delete) an element from the queue

int dequeue(struct Queue\* queue) {

    if (queue->front == NULL) {

        printf("Queue is empty, cannot dequeue\n");

        return -1; // Return a sentinel value to indicate an empty queue

    }

    int data = queue->front->data;

    struct QueueNode\* temp = queue->front;

    queue->front = queue->front->next;

    free(temp);

    if (queue->front == NULL) {

        queue->rear = NULL;

    }

    printf("%d dequeued from the queue\n", data);

    return data;

}

int main() {

    struct Queue\* queue = createQueue();

    // Enqueue some elements

    enqueue(queue, 29);

    enqueue(queue, 18);

    enqueue(queue, 19);

    // Dequeue elements

    int dequeued = dequeue(queue);

    if (dequeued != -1) {

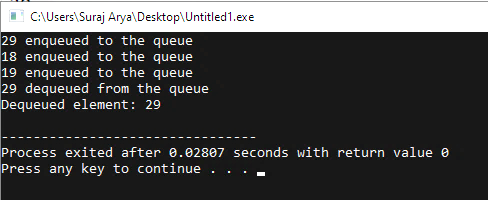
        printf("Dequeued element: %d\n", dequeued);

    }

return 0;

}

**Output**

****

**PRACTICAL:- 5**

**AIM : The binary search procedures to search on element in a given list.**

#include <stdio.h>

int binary\_search(int arr[], int size, int target) {

    int low = 0, high = size - 1;

    while (low <= high) {

        int mid = (low + high) / 2;

        int mid\_element = arr[mid];

        if (mid\_element == target) {

            return mid;  // Element found, return its index

        } else if (mid\_element < target) {

            low = mid + 1;  // Search the right half

        } else {

            high = mid - 1;  // Search the left half

        }

    }

    return -1;  // Element not found in the array

}

int main() {

    int my\_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    int size = sizeof(my\_array) / sizeof(my\_array[0]);

    int target\_element = 7;

    int result = binary\_search(my\_array, size, target\_element);

    if (result != -1) {

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

    } else {

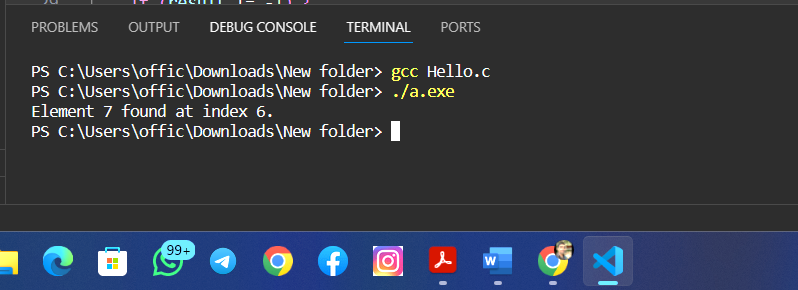
        printf("Element %d not found in the array.\n", target\_element);

    }

    return 0;

}

**Output:-**



**PRACTICAL:- 6**

**AIM : The Selection sort technique.**

#include <stdio.h>

void swap(int \*xp, int \*yp) {

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

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

    int i, j, minIndex;

    // Traverse the array

    for (i = 0; i < n-1; i++) {

        // Find the minimum element in unsorted array

        minIndex = i;

        for (j = i+1; j < n; j++) {

            if (arr[j] < arr[minIndex]) {

                minIndex = j;

            }

        }

        // Swap the found minimum element with the first element

        swap(&arr[minIndex], &arr[i]);

    }

}

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

    int i;

    for (i=0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    int arr[] = {64, 25, 12, 22, 11};

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

    printf("Original array: \n");

    printArray(arr, n);

    // Perform selection sort

    selectionSort(arr, n);

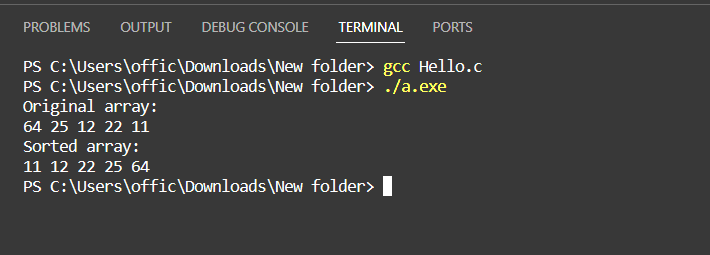
    printf("Sorted array: \n");

    printArray(arr, n);

    return 0;

}

**Output:-**

****

**PRACTICAL:- 7**

**AIM : The Fibonacci Series with recursion.**

#include <stdio.h>

// Function to calculate Fibonacci number using recursion

int fibonacci(int n) {

    if (n <= 1) {

        return n;

    } else {

        return fibonacci(n - 1) + fibonacci(n - 2);

    }

}

int main() {

    int n, i;

    printf("Enter the number of terms: ");

    scanf("%d", &n);

    printf("Fibonacci Series: ");

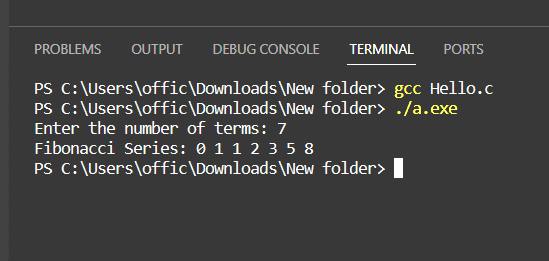
    for (i = 0; i < n; i++) {

        printf("%d ", fibonacci(i));

    }

    return 0;

}

****