**DAY 2 : 25.07.2024**

**Lab Questions to be practiced with test cases**

1. **Sum of row and column – Array:**

#include <stdio.h>

int main() {

int rows, cols, i, j;

int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (i = 0; i < 3; i++) {

int row\_sum = 0;

for (j = 0; j < 3; j++) {

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

row\_sum += arr[i][j];

}

printf("Sum of row %d: %d\n", i, row\_sum);

}

for (j = 0; j < 3; j++) {

int col\_sum = 0;

for (i = 0; i < 3; i++) {

col\_sum += arr[i][j];

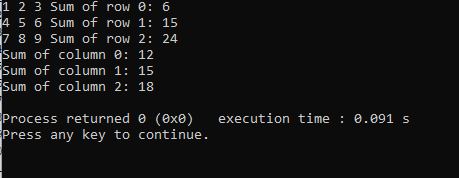
}

printf("Sum of column %d: %d\n", j, col\_sum);

}

return 0;

}

****

1. **Elements repeated twice – Array:**

#include <stdio.h>

int main() {

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

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

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

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

if (arr[i] == arr[j]) {

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

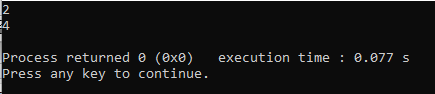
}

}

}

return 0;

}

****

1. **Write a C program to perform Matrix Multiplication:**

#include<stdio.h>

#define ROW1 3

#define COL1 3

#define ROW2 3

#define COL2 3

void multiplyMatrix(int mat1[][COL1],int mat2[][COL2],int res[][COL2]) {

int i,j,k;

for(i=0;i<ROW1;i++){

for(j=0;j<COL2;j++) {

res[i][j]=0;

for(k=0;k<COL2;k++){

res[i][j] += mat1[i][k] \* mat2[k][j];

}

}

}

}

void displayMatrix(int mat1[][COL2]) {

int i,j;

for(i=0;i<ROW1;i++) {

for (j=0;j<COL2;j++) {

printf("%d\n",mat1[i][j]);

}

printf("\n");

}

}

int main() {

int mat1[ROW1][COL1]={{1,2,3},{4,5,6},{7,8,9}};

int mat2[ROW2][COL2]={{1,2,3},{4,5,6},{7,8,9}};

int res[ROW1][COL2];

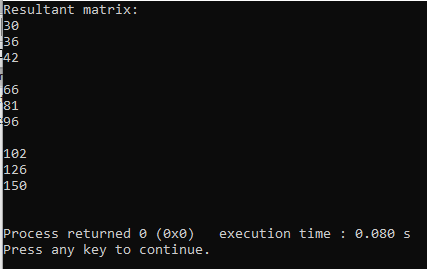
multiplyMatrix(mat1,mat2,res);

printf("Resultant matrix:\n");

displayMatrix(res);

return 0;

}

****

1. **Write a C program to find Factorial of a given number without using Recursion:**

#include<stdio.h>

int main() {

int num,i;

unsigned long long factorial=1;

printf("Enter a positive integer:");

scanf("%d",&num);

if(num<0) {

printf("Error!");

}

else {

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

factorial\*=i;

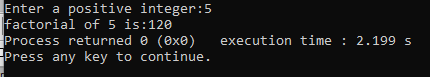
}

printf("factorial of %d is:%1lu",num,factorial);

}

return 0;

}

****

1. **Write a C program to find Fibonacci series without using Recursion:**

#include<stdio.h>

int main () {

int n,n1=0,n2=1,next;

printf("Enter no:of terms:");

scanf("%d",&n);

printf("fibonacci series:");

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

if (i<=1)

next=i;

else

{

next=n1+n2;

n1=n2;

n2=next;

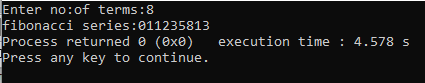
}

printf("%d",next);

}

return 0;

}

****

1. **Write a C program to find Factorial of a given number using Recursion:**

#include <stdio.h>

// Function to calculate factorial using recursion

int factorial(int n) {

if (n == 0 || n == 1) {

return 1; // base case

} else {

return n \* factorial(n - 1); // recursive call

}

}

int main() {

int num;

printf("Enter a positive integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Error: Factorial of negative numbers is not defined.\n");

} else {

int result = factorial(num);

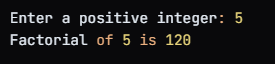
printf("Factorial of %d is %d\n", num, result);

}

return 0;

}

Output:



1. **Write a C program to find Fibonacci series using Recursion:**

#include <stdio.h>

int fibonacci(int n) {

if (n == 0 || n == 1) {

return n; // base case

} else {

return fibonacci(n - 1) + fibonacci(n - 2); // recursive call

}

}

int main() {

int n;

printf("Enter the number of terms in the Fibonacci series: ");

scanf("%d", &n);

printf("Fibonacci series: ");

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

printf("%d ", fibonacci(i));

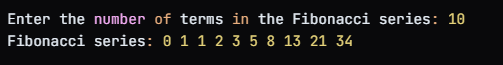
}

printf("\n");

return 0;

}

Output:



1. **Write a C program to implement Array operations such as Insert, Delete and Display:**

#include <stdio.h>

#define MAX\_SIZE 100

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

printf("Array elements: ");

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

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

}

printf("\n");

}

void insert(int arr[], int \*size, int element, int position) {

if (\*size == MAX\_SIZE) {

printf("Error: Array is full. Cannot insert element.\n");

return;

}

for (int i = \*size; i > position; i--) {

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

}

arr[position] = element;

(\*size)++;

}

void delete(int arr[], int \*size, int position) {

if (\*size == 0) {

printf("Error: Array is empty. Cannot delete element.\n");

return;

}

if (position >= \*size) {

printf("Error: Invalid position. Cannot delete element.\n");

return;

}

for (int i = position; i < \*size - 1; i++) {

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

}

(\*size)--;

}

int main() {

int arr[MAX\_SIZE];

int size = 0;

int choice;

while (1) {

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

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

printf("3. Display array\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

int element, position;

printf("Enter element to insert: ");

scanf("%d", &element);

printf("Enter position to insert: ");

scanf("%d", &position);

insert(arr, &size, element, position);

break;

}

case 2: {

int position;

printf("Enter position to delete: ");

scanf("%d", &position);

delete(arr, &size, position);

break;

}

case 3:

display(arr, size);

break;

case 4:

return 0;

default:

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

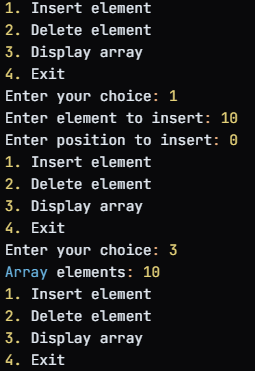
}

}

return 0;

}

Output:



1. **Write a C program to implement singly linked list:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the linked list

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the linked list

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head;

\*head = newNode;

}

}

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

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Function to delete a node from the linked list

void deleteNode(Node\*\* head, int data) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

if ((\*head)->data == data) {

Node\* temp = \*head;

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

free(temp);

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

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

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

free(nodeToDelete);

return;

}

temp = temp->next;

}

printf("Node not found\n");

}

// Function to print the linked list

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

Node\* head = NULL;

// Insert nodes at the beginning of the linked list

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtBeginning(&head, 30);

// Print the linked list

printf("Linked List: ");

printList(head);

// Insert nodes at the end of the linked list

insertAtEnd(&head, 40);

insertAtEnd(&head, 50);

// Print the linked list

printf("Linked List: ");

printList(head);

// Delete a node from the linked list

deleteNode(&head, 20);

// Print the linked list

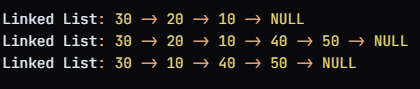
printf("Linked List: ");

printList(head);

return 0;

}

output:



1. **Write a C program to implement doubly linked list:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

struct Node\* prev;

} Node;

Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

newNode->next = \*head;

(\*head)->prev = newNode;

\*head = newNode;

}

}

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

}

void deleteNode(Node\*\* head, int data) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

if ((\*head)->data == data) {

Node\* temp = \*head;

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

if (\*head != NULL) {

(\*head)->prev = NULL;

}

free(temp);

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

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

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

if (temp->next != NULL) {

temp->next->prev = temp;

}

free(nodeToDelete);

return;

}

temp = temp->next;

}

printf("Node not found\n");

}

void printForward(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void printBackward(Node\* head) {

if (head == NULL) {

printf("List is empty\n");

return;

}

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

while (temp != NULL) {

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

temp = temp->prev;

}

printf("NULL\n");

}

int main() {

Node\* head = NULL;

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtBeginning(&head, 30);

printf("Forward: ");

printForward(head);

printf("Backward: ");

printBackward(head);

insertAtEnd(&head, 40);

insertAtEnd(&head, 50);

printf("Forward: ");

printForward(head);

printf("Backward: ");

printBackward(head);

deleteNode(&head, 20);

printf("Forward: ");

printForward(head);

printf("Backward: ");

printBackward(head);

return 0;

}

1. **Write a C program to implement circular linked list:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

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

} else {

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

newNode->next = \*head;

temp->next = newNode;

\*head = newNode;

}

}

void insertAtEnd(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

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

} else {

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

}

void deleteNode(Node\*\* head, int data) {

if (\*head == NULL) {

printf("List is empty\n");

return;

}

if ((\*head)->data == data) {

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = (\*head)->next;

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

free(temp);

return;

}

Node\* temp = \*head;

while (temp->next != \*head) {

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

Node\* nodeToDelete = temp->next;

temp->next = temp->next->next;

free(nodeToDelete);

return;

}

temp = temp->next;

}

printf("Node not found\n");

}

void printList(Node\* head) {

Node\* temp = head;

while (true) {

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

temp = temp->next;

if (temp == head) {

break;

}

}

printf("HEAD\n");

}

int main() {

Node\* head = NULL;

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtBeginning(&head, 30);

printf("Circular Linked List: ");

printList(head);

insertAtEnd(&head, 40);

insertAtEnd(&head, 50);

printf("Circular Linked List: ");

printList(head);

deleteNode(&head, 20);

printf("Circular Linked List: ");

printList(head);

return 0;

}

Output:

