|  |
| --- |
| **SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES** |
| **COMPUTER SCIENCE AND ENGINEERING PROGRAMME** |

**SUB CODE: CSA0392 SUB NAME: Data Structures for Hashing Techniques**

**LIST OF PROGRAMS**

**DAY 2 : 25.07.2024**

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

1. **Sum of row and column – Array**

**Answer:**

**#include <stdio.h>**

**#define ROWS 3**

**#define COLS 4**

**// Function to calculate and print the sum of each row and column**

**void sumRowsAndColumns(int arr[ROWS][COLS], int rows, int cols) {**

**// Array to store the sum of each row**

**int rowSum[ROWS] = {0};**

**// Array to store the sum of each column**

**int colSum[COLS] = {0};**

**// Calculate row sums and column sums**

**for (int i = 0; i < rows; i++) {**

**for (int j = 0; j < cols; j++) {**

**rowSum[i] += arr[i][j];**

**colSum[j] += arr[i][j];**

**}**

**}**

**// Print the sum of each row**

**printf("Sum of each row:\n");**

**for (int i = 0; i < rows; i++) {**

**printf("Row %d: %d\n", i, rowSum[i]);**

**}**

**// Print the sum of each column**

**printf("Sum of each column:\n");**

**for (int j = 0; j < cols; j++) {**

**printf("Column %d: %d\n", j, colSum[j]);**

**}**

**}**

**// Main function to test the sumRowsAndColumns function**

**int main() {**

**// Example 2D array**

**int arr[ROWS][COLS] = {**

**{1, 2, 3, 4},**

**{5, 6, 7, 8},**

**{9, 10, 11, 12}**

**};**

**// Calculate and print the sums**

**sumRowsAndColumns(arr, ROWS, COLS);**

**return 0;**

**}**

1. **Elements repeated twice – Array**

**Answer:**

**#include <stdio.h>**

**#include <stdbool.h>**

**#define MAX\_SIZE 100**

**#define MAX\_VAL 1000**

**// Function to count occurrences of elements**

**void countOccurrences(int arr[], int size, int count[]) {**

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

**count[arr[i]]++;**

**}**

**}**

**// Function to print elements repeated exactly twice**

**void printElementsRepeatedTwice(int arr[], int size) {**

**// Array to store counts of elements**

**int count[MAX\_VAL] = {0};**

**// Count occurrences of each element**

**countOccurrences(arr, size, count);**

**// Print elements with exactly two occurrences**

**printf("Elements repeated exactly twice:\n");**

**bool found = false;**

**for (int i = 0; i < MAX\_VAL; i++) {**

**if (count[i] == 2) {**

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

**found = true;**

**}**

**}**

**if (!found) {**

**printf("None");**

**}**

**printf("\n");**

**}**

**// Main function to test the printElementsRepeatedTwice function**

**int main() {**

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

**int size = sizeof(arr) / sizeof(arr[0]);**

**printElementsRepeatedTwice(arr, size);**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**#define MAX 10 // Maximum size for matrices, can be adjusted as needed**

**// Function to multiply two matrices**

**void multiplyMatrices(int mat1[MAX][MAX], int mat2[MAX][MAX], int result[MAX][MAX], int row1, int col1, int row2, int col2) {**

**// Check if multiplication is possible**

**if (col1 != row2) {**

**printf("Matrix multiplication is not possible. Number of columns of the first matrix must be equal to the number of rows of the second matrix.\n");**

**return;**

**}**

**// Initialize result matrix to 0**

**for (int i = 0; i < row1; i++) {**

**for (int j = 0; j < col2; j++) {**

**result[i][j] = 0;**

**}**

**}**

**// Perform matrix multiplication**

**for (int i = 0; i < row1; i++) {**

**for (int j = 0; j < col2; j++) {**

**for (int k = 0; k < col1; k++) {**

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

**}**

**}**

**}**

**}**

**// Function to print a matrix**

**void printMatrix(int mat[MAX][MAX], int row, int col) {**

**for (int i = 0; i < row; i++) {**

**for (int j = 0; j < col; j++) {**

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

**}**

**printf("\n");**

**}**

**}**

**// Main function to test matrix multiplication**

**int main() {**

**int mat1[MAX][MAX], mat2[MAX][MAX], result[MAX][MAX];**

**int row1, col1, row2, col2;**

**// Input dimensions and matrices**

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

**scanf("%d %d", &row1, &col1);**

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

**for (int i = 0; i < row1; i++) {**

**for (int j = 0; j < col1; j++) {**

**scanf("%d", &mat1[i][j]);**

**}**

**}**

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

**scanf("%d %d", &row2, &col2);**

**// Check if multiplication is possible**

**if (col1 != row2) {**

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

**return 1;**

**}**

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

**for (int i = 0; i < row2; i++) {**

**for (int j = 0; j < col2; j++) {**

**scanf("%d", &mat2[i][j]);**

**}**

**}**

**// Multiply matrices**

**multiplyMatrices(mat1, mat2, result, row1, col1, row2, col2);**

**// Print result matrix**

**printf("Resultant Matrix:\n");**

**printMatrix(result, row1, col2);**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**// Function to compute factorial iteratively**

**unsigned long long factorialIterative(int n) {**

**if (n < 0) {**

**printf("Factorial is not defined for negative numbers.\n");**

**return 0;**

**}**

**unsigned long long factorial = 1;**

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

**factorial \*= i;**

**}**

**return factorial;**

**}**

**// Main function to test the factorialIterative function**

**int main() {**

**int number;**

**// Input number**

**printf("Enter a positive integer: ");**

**scanf("%d", &number);**

**// Calculate factorial**

**unsigned long long result = factorialIterative(number);**

**// Print the result**

**if (number >= 0) {**

**printf("Factorial of %d is: %llu\n", number, result);**

**}**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**// Function to print Fibonacci series iteratively**

**void printFibonacci(int n) {**

**if (n <= 0) {**

**printf("Number of terms should be a positive integer.\n");**

**return;**

**}**

**// Initialize the first two terms**

**unsigned long long a = 0, b = 1;**

**// Print the first term**

**printf("Fibonacci Series:\n");**

**if (n >= 1) {**

**printf("%llu ", a);**

**}**

**if (n >= 2) {**

**printf("%llu ", b);**

**}**

**// Calculate and print the remaining terms**

**for (int i = 2; i < n; i++) {**

**unsigned long long next = a + b;**

**printf("%llu ", next);**

**a = b;**

**b = next;**

**}**

**printf("\n");**

**}**

**// Main function to test the printFibonacci function**

**int main() {**

**int terms;**

**// Input number of terms**

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

**scanf("%d", &terms);**

**// Print Fibonacci series**

**printFibonacci(terms);**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**// Recursive function to compute factorial**

**unsigned long long factorialRecursive(int n) {**

**// Base case**

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

**return 1;**

**}**

**// Recursive case**

**return n \* factorialRecursive(n - 1);**

**}**

**// Main function to test the factorialRecursive function**

**int main() {**

**int number;**

**// Input number**

**printf("Enter a positive integer: ");**

**scanf("%d", &number);**

**// Check for non-negative integer**

**if (number < 0) {**

**printf("Factorial is not defined for negative numbers.\n");**

**return 1;**

**}**

**// Calculate factorial**

**unsigned long long result = factorialRecursive(number);**

**// Print the result**

**printf("Factorial of %d is: %llu\n", number, result);**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**// Recursive function to find Fibonacci number at position n**

**unsigned long long fibonacci(int n) {**

**// Base cases**

**if (n == 0) {**

**return 0;**

**}**

**if (n == 1) {**

**return 1;**

**}**

**// Recursive case**

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

**}**

**// Function to print Fibonacci series up to a given number of terms**

**void printFibonacciSeries(int terms) {**

**if (terms <= 0) {**

**printf("Number of terms should be a positive integer.\n");**

**return;**

**}**

**printf("Fibonacci Series:\n");**

**for (int i = 0; i < terms; i++) {**

**printf("%llu ", fibonacci(i));**

**}**

**printf("\n");**

**}**

**// Main function to test the printFibonacciSeries function**

**int main() {**

**int terms;**

**// Input number of terms**

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

**scanf("%d", &terms);**

**// Print Fibonacci series**

**printFibonacciSeries(terms);**

**return 0;**

**}**

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

**Answer:**

**#include <stdio.h>**

**#define MAX\_SIZE 100 // Define the maximum size of the array**

**// Function to display the array**

**void displayArray(int arr[], int size) {**

**printf("Array elements:\n");**

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

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

**}**

**printf("\n");**

**}**

**// Function to insert an element into the array**

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

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

**printf("Array is full. Cannot insert new element.\n");**

**return;**

**}**

**if (position < 0 || position > \*size) {**

**printf("Invalid position. Cannot insert element.\n");**

**return;**

**}**

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

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

**}**

**arr[position] = element;**

**(\*size)++;**

**}**

**// Function to delete an element from the array**

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

**if (\*size <= 0) {**

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

**return;**

**}**

**if (position < 0**

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

**Answer:**

**#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 allocation failed.\n");**

**exit(1);**

**}**

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

**// Function to insert a node at the beginning**

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

**Node\* newNode = createNode(data);**

**newNode->next = \*head;**

**\*head = newNode;**

**}**

**// Function to insert a node at the end**

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

**Node\* newNode = createNode(data);**

**if (\*head == NULL) {**

**\*head = newNode;**

**return;**

**}**

**Node\* temp = \*head;**

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

**temp = temp->next;**

**}**

**temp->next = newNode;**

**}**

**// Function to delete a node by value**

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

**Node\* temp = \*head;**

**Node\* prev = NULL;**

**// Check if the node to be deleted is the head**

**if (temp != NULL && temp->data == value) {**

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

**free(temp);**

**return;**

**}**

**// Search for the node to be deleted**

**while (temp != NULL && temp->data != value) {**

**prev = temp;**

**temp = temp->next;**

**}**

**// Node not found**

**if (temp == NULL) {**

**printf("Value %d not found in the list.\n", value);**

**return;**

**}**

**// Unlink the node from the list**

**prev->next = temp->next;**

**free(temp);**

**}**

**// Function to di**

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

**Answer:**

**#include <stdio.h>**

**#include <stdlib.h>**

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

**typedef struct Node {**

**int data;**

**struct Node\* next;**

**struct Node\* prev;**

**} Node;**

**// Function to create a new node**

**Node\* createNode(int data) {**

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

**if (!newNode) {**

**printf("Memory allocation failed.\n");**

**exit(1);**

**}**

**newNode->data = data;**

**newNode->next = NULL;**

**newNode->prev = NULL;**

**return newNode;**

**}**

**// Function to insert a node at the beginning**

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

**Node\* newNode = createNode(data);**

**newNode->next = \*head;**

**if (\*head != NULL) {**

**(\*head)->prev = newNode;**

**}**

**\*head = newNode;**

**}**

**// Function to insert a node at the end**

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

**Node\* newNode = createNode(data);**

**if (\*head == NULL) {**

**\*head = newNode;**

**return;**

**}**

**Node\* temp = \*head;**

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

**temp = temp->next;**

**}**

**temp->next = newNode;**

**newNode->prev = temp;**

**}**

**// Function to delete a node by value**

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

**Node\* temp = \*head;**

**// Find the node to delete**

**while (temp != NULL && temp->data != value) {**

**temp = temp->next;**

**}**

**// Node not found**

**if (t**

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

**Answer:**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Define the structure for a node in the circular 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 allocation failed.\n");**

**exit(1);**

**}**

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

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

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

**Node\* newNode = createNode(data);**

**if (\*head == NULL) {**

**// If the list is empty, create a new circular link**

**\*head = newNode;**

**newNode->next = \*head;**

**} else {**

**Node\* temp = \*head;**

**// Traverse to the last node**

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

**temp = temp->next;**

**}**

**// Insert the new node at the end and update links**

**temp->next = newNode;**

**newNode->next = \*head;**

**}**

**}**

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

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

**Node\* newNode = createNode(data);**

**if (\*head == NULL) {**

**// If the list is empty, create a new circular link**

**\*head = newNode;**

**newNode->next = \*head;**

**} else {**

**Node\* temp = \*head;**

**// Traverse to the last node**

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

**temp = temp->next;**

**}**

**// Insert the new node at the beginning and update links**

**newNode->next = \*head;**

**\*head = newNode;**

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

**}**

**}**

**// Function to delete a node by value from the circular linked list**

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

**if (\*head == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**Node\* temp = \*head;**

**Node\* prev = NULL;**

**// Check if the node to be deleted is the head**

**if (temp->data == value) {**

**if (temp->next == \*head) {**

**// Only one node in the list**

**free(temp);**

**\*head = NULL;**

**return;**

**}**

**// Traverse to the last node**

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

**temp = temp->next;**

**}**

**// Update head and last node's next pointer**

**Node\* toDelete = \*head;**

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

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

**free(toDelete);**

**return;**

**}**

**// Search for the node to delete**

**prev = temp;**

**temp = temp->next;**

**while (temp != \*head && temp->data != value) {**

**prev = temp;**

**temp = temp->next;**

**}**

**// Node not found**

**if (temp == \*head) {**

**printf("Value %d not found in the list.\n", value);**

**return;**

**}**

**// Unlink the node from the list**

**prev->next = temp->next;**

**free(temp);**

**}**

**// Function to display the circular linked list**

**void displayList(Node\* head) {**

**if (head == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**Node\* temp = head;**

**printf("Circular Linked List:\n");**

**do {**

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

**temp = temp->next;**

**} while (temp != head);**

**printf("(head)\n");**

**}**

**// Main function to demonstrate circular linked list operations**

**int main() {**

**Node\* head = NULL; // Initialize the head of the list as NULL**

**int choice, data;**

**while (1) {**

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

**printf("1. Insert at Beginning\n");**

**printf("2. Insert at End\n");**

**printf("3. Delete Node\n");**

**printf("4. Display List\n");**

**printf("5. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter data to insert at the beginning: ");**

**scanf("%d", &data);**

**insertAtBeginning(&head, data);**

**break;**

**case 2:**

**printf("Enter data to insert at the end: ");**

**scanf("%d", &data);**

**insertAtEnd(&head, data);**

**break;**

**case 3:**

**printf("Enter value to delete: ");**

**scanf("%d", &data);**

**deleteNode(&head, data);**

**break;**

**case 4:**

**displayList(head);**

**break;**

**case 5:**

**// Free all allocated memory**