|  |
| --- |
| **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 1 : 24.07.2024**

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

1. **Reversing a 32 bit signed intergers**

**Answer:**

**#include <stdio.h>**

**#include <limits.h> // For INT\_MAX and INT\_MIN**

**// Function to reverse the bits of a 32-bit signed integer**

**int reverseBits(int num) {**

**unsigned int result = 0;**

**unsigned int mask = 1;**

**// Iterate over each bit in the 32-bit integer**

**for (int i = 0; i < 32; i++) {**

**// Extract the current bit from num**

**int bit = (num >> i) & mask;**

**// Shift the bit to the appropriate position in result**

**result |= (bit << (31 - i));**

**}**

**// Handle overflow by checking if the reversed number is within the 32-bit signed integer range**

**if (result > INT\_MAX) {**

**return 0; // or handle overflow case as needed**

**}**

**return (int)result;**

**}**

**// Function to print the integer in binary format**

**void printBinary(int num) {**

**for (int i = 31; i >= 0; i--) {**

**printf("%d", (num >> i) & 1);**

**}**

**printf("\**

1. **Check for a valid String**

**Answer:**

**#include <stdio.h>**

**#include <ctype.h> // For isprint function**

**// Function to check if the string is valid (non-empty and contains only printable characters)**

**int isValidString(const char \*str) {**

**// Check if the string is non-empty**

**if (str == NULL || \*str == '\0') {**

**return 0; // Invalid: NULL or empty string**

**}**

**// Check if all characters are printable**

**while (\*str) {**

**if (!isprint((unsigned char)\*str)) {**

**return 0; /**

1. **Merging two Arrays**

**Answer:**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Function to merge two arrays into a single array**

**int\* mergeArrays(int\* arr1, int size1, int\* arr2, int size2) {**

**// Allocate memory for the merged array**

**int\* mergedArray = (int\*)malloc((size1 + size2) \* sizeof(int));**

**if (mergedArray == NULL) {**

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

**exit(1);**

**}**

**// Copy elements from the first array**

**for (int i = 0; i < size1; i++) {**

**mergedArray[i] = arr1[i];**

**}**

**// Copy elements from the second array**

**for (int i = 0; i < size2; i++) {**

**mergedArray[size1 + i] = arr2[i];**

**}**

**return mergedArray;**

**}**

**// Function to print an array**

**void printArray(int\* array, int size) {**

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

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

**}**

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

**}**

**// Main function to test merging two arrays**

**int main() {**

**// Define and initialize the first array**

**int arr1[] = {1, 3, 5, 7};**

**int size1 = sizeof(arr1) / sizeof(arr1[0]);**

**// Define and initialize the second array**

**int arr2[] = {2, 4, 6, 8};**

**int size2 = sizeof(arr2) / sizeof(arr2[0]);**

**// Merge the arrays**

**int\* mergedArray = mergeArrays(arr1, size1, arr2, size2);**

**// Print the merged array**

**printf("Merged array: ");**

**printArray(mergedArray, size1 + size2);**

**// Free the allocated memory for the merged array**

**free(mergedArray);**

**return 0;**

**}**

1. **Given an array finding duplication values**

**Answer:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_SIZE 1000 // Define maximum possible value for elements in the array**

**// Function to find and print duplicate values in an array**

**void findDuplicates(int\* arr, int size) {**

**// Create an auxiliary array to track occurrences**

**int\* hashTable = (int\*)calloc(MAX\_SIZE, sizeof(int));**

**if (hashTable == NULL) {**

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

**exit(1);**

**}**

**// Iterate through the array and mark occurrences**

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

**int value = arr[i];**

**// Check for valid range and mark occurrence**

**if (value >= 0 && value < MAX\_SIZE) {**

**hashTable[value]++;**

**}**

**}**

**// Print duplicates**

**printf("Duplicates in the array: ");**

**int foundDuplicate = 0;**

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

**if (hashTable[i] > 1) {**

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

**foundDuplicate = 1;**

**}**

**}**

**if (!foundDuplicate) {**

**printf("None");**

**}**

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

**// Free allocated memory**

**free(hashTable);**

**}**

**// Main function to test the findDuplicates function**

**int main() {**

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

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

**findDuplicates(arr, size);**

**return 0;**

**}**

1. **Merging of list**

**Answer:**

**#include <stdio.h>**

**#include <stdlib.h>**

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

**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 == NULL) {**

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

**exit(1);**

**}**

**newNode->data = data;**

**newNode->next = NULL;**

**return newNode;**

**}**

**// Function to merge two sorted linked lists**

**Node\* mergeLists(Node\* l1, Node\* l2) {**

**// Create a dummy node to serve as the starting point of the merged list**

**Node dummy;**

**Node\* tail = &dummy;**

**dummy.next = NULL;**

**// Merge the two lists**

**while (l1 != NULL && l2 != NULL) {**

**if (l1->data < l2->data) {**

**tail->next = l1;**

**l1 = l1->next;**

**} else {**

**tail->next = l2;**

**l2 = l2->next;**

**}**

**tail = tail->next;**

**}**

**// Attach the remaining elements**

**if (l1 != NULL) {**

**tail->next = l1;**

**} else {**

**tail->next = l2;**

**}**

**return dummy.next;**

**}**

**// Function to print the linked list**

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

**while (head != NULL) {**

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

**head = head->next;**

**}**

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

**}**

**// Function to free the linked list**

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

**Node\* temp;**

**while (head != NULL) {**

**temp = head;**

**head = head->next;**

**free(temp);**

**}**

**}**

**// Main function to test the mergeLists function**

**int main() {**

**// Create first sorted list: 1 -> 3 -> 5**

**Node\* l1 = createNode(1);**

**l1->next = createNode(3);**

**l1->next->next = createNode(5);**

**// Create second sorted list: 2 -> 4 -> 6**

**Node\* l2 = createNode(2);**

**l2->next = createNode(4);**

**l2->next->next = createNode(6);**

**// Merge the two lists**

**Node\* mergedList = mergeLists(l1, l2);**

**// Print the merged list**

**printf("Merged list: ");**

**printList(mergedList);**

**// Free allocated memory**

**freeList(mergedList)**

**return 0;**

**}**

1. **Given array of reg nos need to search for particular reg no**

**Answer:**

**#include <stdio.h>**

**#include <string.h>**

**// Function to perform a linear search in the array**

**int linearSearch(char regNos[][20], int size, const char\* target) {**

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

**if (strcmp(regNos[i], target) == 0) {**

**return i; // Return index if found**

**}**

**}**

**return -1; // Return -1 if not found**

**}**

**// Main function to test the linear search**

**int main() {**

**// Array of registration numbers**

**char regNos[][20] = {"ABC123", "XYZ789", "LMN456", "DEF012"};**

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

**// Target registration number to search for**

**const char\* target = "LMN456";**

**// Perform the search**

**int index = linearSearch(regNos, size, target);**

**if (index != -1) {**

**printf("Registration number '%s' found at index %d\n", target, index);**

**} else {**

**printf("Registration number '%s' not found\n", target);**

**}**

**return 0;**

**}**

1. **Identify location of element in given array**

**Answer:**

**#include <stdio.h>**

**// Function to perform linear search in an array**

**int linearSearch(int arr[], int size, int target) {**

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

**if (arr[i] == target) {**

**return i; // Return index if found**

**}**

**}**

**return -1; // Return -1 if not found**

**}**

**// Main function to test the linear search**

**int main() {**

**int arr[] = {5, 3, 7, 1, 9, 2};**

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

**int target = 9;**

**int index = linearSearch(arr, size, target);**

**if (index != -1) {**

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

**} else {**

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

**}**

**return 0;**

**}**

1. **Given array print odd and even values**

**Answer:**

**#include <stdio.h>**

**// Function to print odd and even values from an array**

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

**printf("Even values:\n");**

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

**if (arr[i] % 2 == 0) {**

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

**}**

**}**

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

**printf("Odd values:\n");**

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

**if (arr[i] % 2 != 0) {**

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

**}**

**}**

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

**}**

**// Main function to test the printOddAndEven function**

**int main() {**

**int arr[] = {12, 7, 9, 10, 15, 22, 31};**

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

**printOddAndEven(arr, size);**

**return 0;**

**}**

1. **Sum of Fibonacci Series**

**Answer:**

**#include <stdio.h>**

**// Function to compute the sum of the first n Fibonacci numbers**

**int fibonacciSum(int n) {**

**if (n <= 0) return 0;**

**if (n == 1) return 1;**

**int a = 0; // F(0)**

**int b = 1; // F(1)**

**int sum = a + b; // Sum of first two Fibonacci numbers**

**// Compute Fibonacci numbers and their sum**

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

**int next = a + b;**

**sum += next;**

**a = b;**

**b = next;**

**}**

**return sum;**

**}**

**// Main function to test the fibonacciSum function**

**int main() {**

**int n = 5; // Change this value to compute sum for different number of Fibonacci terms**

**int sum = fibonacciSum(n);**

**printf("The sum of the first %d Fibonacci numbers is: %d\n", n, sum);**

**return 0;**

**}**

**10. Finding factorial of a number**

**Answer:**

**#include <stdio.h>**

**// Function to compute factorial iteratively**

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

**if (n < 0) return 0; // Factorial is not defined for negative numbers**

**unsigned long long result = 1;**

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

**result \*= i;**

**}**

**return result;**

**}**

**// Main function to test the iterative factorial**

**int main() {**

**int n = 5; // Change this value to compute factorial for different numbers**

**unsigned long long fact = factorialIterative(n);**

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

**return 0;**

**}**