|  |
| --- |
| **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**

**DATE : 21.08.2024**

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

1.

You are given with an array arr which contains integer elements. Sort these elements in ascending order using insertion sort and print the 6th Iteration result.

Example:

Input:98,23,45,14,6,67,33,42

Output:6,14,23,33,45,67,98,42

Answer:

#include <stdio.h>

#define ARRAY\_SIZE 8

// Function to print the array

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

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

if (i != 0) printf(",");

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

}

printf("\n");

}

// Function to perform insertion sort and print the array after the 6th iteration

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

int iterationCount = 0;

for (int i = 1; i < size; i++) {

int key = arr[i];

int j = i - 1;

// Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

// Increment the iteration count

iterationCount++;

// Print the array after the 6th iteration

if (iterationCount == 6) {

printArray(arr, size);

}

}

}

int main() {

int arr[ARRAY\_SIZE] = {98, 23, 45, 14, 6, 67, 33, 42};

// Perform insertion sort and print the result after the 6th iteration

insertionSortAndPrintIteration(arr, ARRAY\_SIZE);

return 0;

}

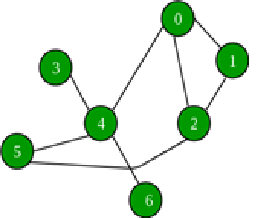
2.

You are given an undirected graph G(V, E) with N vertices and M edges. We need to

find the minimum number of edges between a given pair of vertices (u, v).

Examples:

Input: For given graph G. Find minimum number of edges between (1, 5).



Output: 2

Explanation: (1, 2) and (2, 5) are the only edges resulting into shortest path between 1 and 5.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

typedef struct {

int front, rear;

int data[MAX\_VERTICES];

} Queue;

void initQueue(Queue \*q) {

q->front = 0;

q->rear = -1;

}

bool isQueueEmpty(Queue \*q) {

return q->front > q->rear;

}

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

q->data[++q->rear] = value;

}

int dequeue(Queue \*q) {

return q->data[q->front++];

}

// Function to perform BFS and find the shortest path from u to v

int findShortestPath(int graph[MAX\_VERTICES][MAX\_VERTICES], int N, int u, int v) {

Queue q;

initQueue(&q);

int visited[MAX\_VERTICES] = {0};

int distance[MAX\_VERTICES] = {0};

enqueue(&q, u);

visited[u] = 1;

distance[u] = 0;

while (!isQueueEmpty(&q)) {

int current = dequeue(&q);

if (current == v) {

return distance[current];

}

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

if (graph[current][i] == 1 && !visited[i]) {

visited[i] = 1;

distance[i] = distance[current] + 1;

enqueue(&q, i);

}

}

}

return -1; // Return -1 if there is no path from u to v

}

int main() {

int N = 6; // Number of vertices

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

// Example graph initialization

// Graph (0-based index): 0 -- 1 -- 2 -- 3 -- 4

// | | | |

// 5 -- 6 7 8

graph[0][1] = 1; graph[1][0] = 1;

graph[1][2] = 1; graph[2][1] = 1;

graph[2][3] = 1; graph[3][2] = 1;

graph[3][4] = 1; graph[4][3] = 1;

graph[0][5] = 1; graph[5][0] = 1;

graph[1][6] = 1; graph[6][1] = 1;

graph[2][7] = 1; graph[7][2] = 1;

graph[3][8] = 1; graph[8][3] = 1;

int u = 0; // Start vertex

int v = 4; // End vertex

int minEdges = findShortestPath(graph, N, u, v);

if (minEdges != -1) {

printf("The minimum number of edges between %d and %d is %d.\n", u, v, minEdges);

} else {

printf("There is no path between %d and %d.\n", u, v);

}

return 0;

}

3.

Given the head of a singly linked list, return number of nodes present in a linked

Example 1:

1->2->3->5->8

Output 5

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));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to count the number of nodes in the linked list

int countNodes(Node\* head) {

int count = 0;

Node\* current = head;

while (current != NULL) {

count++;

current = current->next;

}

return count;

}

// Function to print the linked list

void printList(Node\* head) {

Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

int main() {

// Create a linked list: 1 -> 2 -> 3 -> 5 -> 8

Node\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(3);

head->next->next->next = createNode(5);

head->next->next->next->next = createNode(8);

// Print the linked list

printf("Linked list: ");

printList(head);

// Count the number of nodes

int numNodes = countNodes(head);

printf("Number of nodes: %d\n", numNodes);

// Free the allocated memory

Node\* current = head;

Node\* nextNode;

while (current != NULL) {

nextNode = current->next;

free(current);

current = nextNode;

}

return 0;

}

4.

Given a number n. the task is to print the Fibonacci series and the sum of the series using recursion.

input: n=10

output: Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum: 88

Answer:

#include <stdio.h>

// Function to compute Fibonacci number recursively

int fibonacci(int n) {

if (n <= 1) {

return n;

}

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

}

// Function to compute the sum of Fibonacci numbers up to the nth term

int fibonacciSum(int n) {

if (n <= 0) {

return 0;

}

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

}

// Function to print the Fibonacci series up to the nth term

void printFibonacciSeries(int n) {

printf("Fibonacci series:\n");

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

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

if (i < n - 1) {

printf(", ");

}

}

printf("\n");

}

int main() {

int n = 10; // Number of terms in the Fibonacci series

printFibonacciSeries(n);

// Calculate the sum of the first n Fibonacci numbers

int sum = fibonacciSum(n);

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

return 0;

}

5.

You are given an array arr in increasing order. Find the element x from arr using binary search.

Example 1: arr={ 1,5,6,7,9,10},X=6

Output : Element found at location 2

Example 2: arr={ 1,5,6,7,9,10},X=11

Output : Element not found at location 2

Answer:

#include <stdio.h>

// Function to perform binary search

int binarySearch(int arr[], int size, int x) {

int left = 0;

int right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

// Check if x is present at mid

if (arr[mid] == x) {

return mid;

}

// If x is greater, ignore the left half

if (arr[mid] < x) {

left = mid + 1;

}

// If x is smaller, ignore the right half

else {

right = mid - 1;

}

}

// Element is not present in array

return -1;

}

int main() {

int arr[] = {1, 5, 6, 7, 9, 10};

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

int x = 6; // Element to search for

int result = binarySearch(arr, size, x);

if (result != -1) {

printf("Element found at location %d\n", result);

} else {

printf("Element not found\n");

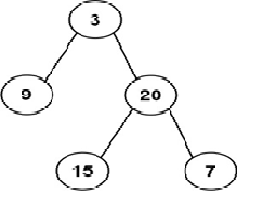
}

return 0;

}

6.

Write a program to traverse the nodes present in the following tree in inorder and postorder traversal



Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a tree node

typedef struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

// Function to create a new tree node

TreeNode\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Inorder Traversal function

void inorderTraversal(TreeNode\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

// Postorder Traversal function

void postorderTraversal(TreeNode\* root) {

if (root == NULL) {

return;

}

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

int main() {

// Creating a simple tree:

// 1

// / \

// 2 3

// / \

// 4 5

TreeNode\* root = createNode(1);

root->left = createNode(2);

root->right = createNode(3);

root->left->left = createNode(4);

root->left->right = createNode(5);

printf("Inorder Traversal:\n");

inorderTraversal(root);

printf("\n");

printf("Postorder Traversal:\n");

postorderTraversal(root);

printf("\n");

// Freeing allocated memory (not shown here for simplicity)

return 0;

}

7.

Given a string s, sort it in ascending order and find the starting index of repeated character

Input: s = "tree"

Output: "eert", starting index 0

Input: s = "kkj"

Output: "jkk", starting index : 1

Example 2:

Input: s = "cccaaa"

Output: "aaaccc", starting index 0,3

Example 3:

Input: s = "Aabb"

Output: "bbAa",starting index 0,2

Answer:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Function to compare characters for qsort**

**int compareChars(const void \*a, const void \*b) {**

**return (\*(char \*)a - \*(char \*)b);**

**}**

**// Function to sort the string in ascending order**

**void sortString(char \*str) {**

**int length = strlen(str);**

**qsort(str, length, sizeof(char), compareChars);**

**}**

**// Function to find the starting index of repeated characters**

**int findRepeatingIndex(char \*str) {**

**int length = strlen(str);**

**for (int i = 0; i < length - 1; i++) {**

**if (str[i] == str[i + 1]) {**

**return i;**

**}**

**}**

**return -1; // Return -1 if no repetition is found**

**}**

**int main() {**

**char str[100];**

**printf("Enter a string: ");**

**scanf("%s", str);**

**// Sort the string**

**sortString(str);**

**// Find the starting index of repeated characters**

**int repeatIndex = findRepeatingIndex(str);**

**printf("Sorted string: %s\n", str);**

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

**printf("Starting index of repeated character: %d\n", repeatIndex);**

**} else {**

**printf("No repeated characters found.\n");**

**}**

**return 0;**

**}**

8.

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example 1:

Input: head = [1,2,2,1]

Output: true

Example 2:

Input: head = [1,2]

Output: false

Input: R->A->D->A->R->NULL

Output: Yes

Input: C->O->D->E->NULL

Output: No

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Definition of a linked list node

typedef struct ListNode {

int data;

struct ListNode\* next;

} ListNode;

// Function to create a new node

ListNode\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to print the linked list

void printList(ListNode\* head) {

ListNode\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

// Function to find the middle of the linked list

ListNode\* findMiddle(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

// Function to reverse a linked list

ListNode\* reverseList(ListNode\* head) {

ListNode\* prev = NULL;

ListNode\* current = head;

ListNode\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

// Function to check if the linked list is a palindrome

bool isPalindrome(ListNode\* head) {

if (head == NULL || head->next == NULL) {

return true;

}

// Find the middle of the list

ListNode\* middle = findMiddle(head);

// Reverse the second half

ListNode\* secondHalf = reverseList(middle);

// Compare the first half and the reversed second half

ListNode\* firstHalf = head;

while (secondHalf != NULL) {

if (firstHalf->data != secondHalf->data) {

return false;

}

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;

}

return true;

}

int main() {

// Create a linked list: 1 -> 2 -> 2 -> 1

ListNode\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(2);

head->next->next->next = createNode(1);

// Check if the list is a palindrome

bool result = isPalindrome(head);

printf("Is palindrome: %s\n", result ? "true" : "false");

// Free allocated memory (not shown here for simplicity)

return 0;

}

9.

Given the root of a binary search tree and K as input, find Kth smallest element in BST.

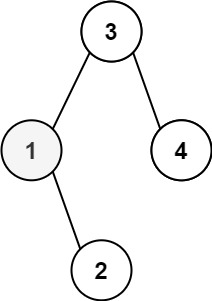
For example, in the following BST,



if k = 3, then the output should be 10, and

if k = 5, then the output should be 14.

Sample:



Input: root = [3,1,4,null,2], k = 1

Output: 1

Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3

Answer:

#include <stdio.h>

#include <stdlib.h>

// Definition of a binary tree node

typedef struct TreeNode {

int data;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

// Function to create a new tree node

TreeNode\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// In-order traversal to find Kth smallest element

void inorderTraversal(TreeNode\* root, int\* count, int k, int\* result) {

if (root == NULL) {

return;

}

// Traverse the left subtree

inorderTraversal(root->left, count, k, result);

// Visit the current node

(\*count)++;

if (\*count == k) {

\*result = root->data;

return;

}

// Traverse the right subtree

inorderTraversal(root->right, count, k, result);

}

// Function to find Kth smallest element in BST

int findKthSmallest(TreeNode\* root, int k) {

int count = 0;

int result = -1; // Initialize result to -1 or any value that indicates "not found"

inorderTraversal(root, &count, k, &result);

return result;

}

int main() {

// Create a sample BST

// 5

// / \

// 3 7

// / \ \

// 2 4 8

TreeNode\* root = createNode(5);

root->left = createNode(3);

root->right = createNode(7);

root->left->left = createNode(2);

root->left->right = createNode(4);

root->right->right = createNode(8);

int k = 3;

int kthSmallest = findKthSmallest(root, k);

if (kthSmallest != -1) {

printf("The %dth smallest element in the BST is: %d\n", k, kthSmallest);

} else {

printf("The %dth smallest element does not exist in the BST.\n", k);

}

// Free allocated memory (not shown here for simplicity)

return 0;

}

10.

Given a string s, find the frequency of characters

Example 1:

Input: s = "tree"

Output t->1, r->1, e->2

Answer:

#include <stdio.h>

#include <string.h>

// Function to find and print the frequency of characters in a string

void findCharacterFrequency(const char \*s) {

int freq[256] = {0}; // Array to store frequency of each character

int length = strlen(s);

// Count the frequency of each character

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

freq[(unsigned char)s[i]]++;

}

// Print the frequency of each character

printf("Character frequencies:\n");

for (int i = 0; i < 256; i++) {

if (freq[i] > 0) {

printf("%c->%d\n", i, freq[i]);

}

}

}

int main() {

char s[100];

printf("Enter a string: ");

scanf("%99s", s); // Read input string

findCharacterFrequency(s);

return 0;

}

11.

Given an unsorted array arr[] with both positive and negative elements, the task

is to find the smallest positive number missing from the array.

Input: arr[] = {2, 3, 7, 6, 8, -1, -10, 15}

Output: 1

Input: arr[] = { 2, 3, -7, 6, 8, 1, -10, 15 }

Output: 4

Input: arr[] = {1, 1, 0, -1, -2}

Output: 2

Answer:

#include <stdio.h>

// Function to find the smallest positive missing number

int findSmallestMissingPositive(int arr[], int size) {

// Step 1: Replace negative numbers and zeros with a placeholder value

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

if (arr[i] <= 0 || arr[i] > size) {

arr[i] = size + 1; // Out of the range of interest

}

}

// Step 2: Mark the presence of elements

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

int num = abs(arr[i]);

if (num <= size) {

if (arr[num - 1] > 0) {

arr[num - 1] = -arr[num - 1]; // Mark as present

}

}

}

// Step 3: Find the smallest missing positive number

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

if (arr[i] > 0) {

return i + 1;

}

}

// If no missing number found in the range, return size + 1

return size + 1;

}

int main() {

int arr1[] = {2, 3, 7, 6, 8, -1, -10, 15};

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

printf("Smallest missing positive number: %d\n", findSmallestMissingPositive(arr1, size1));

int arr2[] = {2, 3, -7, 6, 8, 1, -10, 15};

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

printf("Smallest missing positive number: %d\n", findSmallestMissingPositive(arr2, size2));

int arr3[] = {1, 1, 0, -1, -2};

int size3 = sizeof(arr3) / sizeof(arr3[0]);

printf("Smallest missing positive number: %d\n", findSmallestMissingPositive(arr3, size3));

return 0;

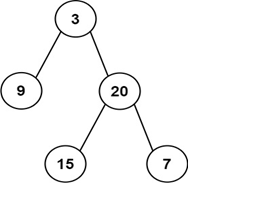
}

12.

Given two integer arrays preorder and inorder where preorder is the preorder

traversal of a binary tree and inorder is the inorder traversal of the same tree,

construct and return the binary tree.



Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

Answer:

#include <stdio.h>

#include <stdlib.h>

// Definition of a binary tree node

typedef struct TreeNode {

int val;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

// Function to create a new tree node

TreeNode\* createNode(int val) {

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

newNode->val = val;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Function to search for an element in the array

int search(int arr[], int start, int end, int value) {

for (int i = start; i <= end; i++) {

if (arr[i] == value) {

return i;

}

}

return -1;

}

// Recursive function to build the tree

TreeNode\* buildTree(int\* preorder, int\* inorder, int inorderStart, int inorderEnd, int\* preorderIndex, int size) {

if (inorderStart > inorderEnd) {

return NULL;

}

// Create the root node with the current value in preorder

TreeNode\* root = createNode(preorder[\*preorderIndex]);

(\*preorderIndex)++;

// If the tree has only one node

if (inorderStart == inorderEnd) {

return root;

}

// Find the index of this node in inorder traversal

int inorderIndex = search(inorder, inorderStart, inorderEnd, root->val);

// Build the left and right subtrees

root->left = buildTree(preorder, inorder, inorderStart, inorderIndex - 1, preorderIndex, size);

root->right = buildTree(preorder, inorder, inorderIndex + 1, inorderEnd, preorderIndex, size);

return root;

}

// Function to print the tree in level-order traversal

void printLevelOrder(TreeNode\* root) {

if (root == NULL) {

return;

}

// Use a queue to print the nodes level by level

TreeNode\* queue[100];

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

TreeNode\* current = queue[front++];

printf("%d ", current->val);

if (current->left != NULL) {

queue[rear++] = current->left;

}

if (current->right != NULL) {

queue[rear++] = current->right;

}

}

printf("\n");

}

int main() {

int preorder[] = {3, 9, 20, 15, 7};

int inorder[] = {9, 3, 15, 20, 7};

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

int preorderIndex = 0;

TreeNode\* root = buildTree(preorder, inorder, 0, size - 1, &preorderIndex, size);

printf("Level-order traversal of the constructed tree:\n");

printLevelOrder(root);

return 0;

}

13.

Write a program to create and display a linked list

Example 1:

Nodes : 6,7,8,9

Output: 6->7->8->9

Answer:

#include <stdio.h>

#include <stdlib.h>

// Definition of 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 insert a new node at the end of the list

void appendNode(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 display the linked list

void displayList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

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

if (temp->next != NULL) {

printf("->");

}

temp = temp->next;

}

printf("\n");

}

int main() {

Node\* head = NULL;

// Insert nodes into the linked list

appendNode(&head, 6);

appendNode(&head, 7);

appendNode(&head, 8);

appendNode(&head, 9);

// Display the linked list

printf("Linked List:\n");

displayList(head);

// Free the allocated memory (optional, but good practice)

Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

return 0;

}

14.

Write a program to sort the below numbers in descending order using bubble sort

Input 4,7,9,1,2

Output:9,7,4,2,1

Answer:

#include <stdio.h>

// Function to perform Bubble Sort in descending order

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

int i, j, temp;

for (i = 0; i < size - 1; i++) {

for (j = 0; j < size - i - 1; j++) {

if (arr[j] < arr[j + 1]) {

// Swap if the element found is less than the next element

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Function to print the array

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

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

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

if (i < size - 1) {

printf(",");

}

}

printf("\n");

}

int main() {

int arr[] = {4, 7, 9, 1, 2}; // Input array

int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array

// Perform Bubble Sort

bubbleSortDescending(arr, size);

// Print the sorted array

printf("Output: ");

printArray(arr, size);

return 0;

}

15.

Given an array of size N-1 such that it only contains distinct integers in the

range of 1 to N. Find the missing element.

Input:

N = 5

A[] = {1,2,3,5}

Output: 4

Input:

N = 10

A[] = {6,1,2,8,3,4,7,10,5}

Output: 9

Answer:

#include <stdio.h>

// Function to find the missing number

int findMissingNumber(int arr[], int size, int N) {

int totalSum = (N \* (N + 1)) / 2; // Sum of first N natural numbers

int arraySum = 0;

// Calculate the sum of elements in the array

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

arraySum += arr[i];

}

// The missing number

return totalSum - arraySum;

}

int main() {

int N, size;

// Example 1

N = 5;

int arr1[] = {1, 2, 3, 5};

size = sizeof(arr1) / sizeof(arr1[0]);

printf("Missing number in array [1, 2, 3, 5] with N = %d is %d\n", N, findMissingNumber(arr1, size, N));

// Example 2

N = 10;

int arr2[] = {6, 1, 2, 8, 3, 4, 7, 10, 5};

size = sizeof(arr2) / sizeof(arr2[0]);

printf("Missing number in array [6, 1, 2, 8, 3, 4, 7, 10, 5] with N = %d is %d\n", N, findMissingNumber(arr2, size, N));

return 0;

}

16.

Write a program to find odd number present in the data part of a node

Example Linked List 1->2->3->7

Output: 1,3,7

Answer:

#include <stdio.h>

#include <stdlib.h>

// Definition of 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 append a new node to the end of the list

void appendNode(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 print odd numbers from the linked list

void printOddNumbers(Node\* head) {

Node\* temp = head;

int first = 1; // To handle formatting for the first element

while (temp != NULL) {

if (temp->data % 2 != 0) { // Check if the data is odd

if (!first) {

printf(",");

}

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

first = 0;

}

temp = temp->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);

}

}

int main() {

Node\* head = NULL;

// Insert nodes into the linked list

appendNode(&head, 1);

appendNode(&head, 2);

appendNode(&head, 3);

appendNode(&head, 7);

// Print odd numbers

printf("Odd numbers in the linked list:\n");

printOddNumbers(head);

// Free the allocated memory

freeList(head);

return 0;

}

17.

Write a program to perform insert and delete operations in a queue

Example : 12,34,56,78

After insertion of 60 content of the queue is 12,34,56,78,60

After deletion of 12 , the contents of the queue : 34,56,78,60

Answer:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 // Define maximum size of the queue

// Define the queue structure

typedef struct Queue {

int front, rear, size;

int items[MAX];

} Queue;

// Function to initialize the queue

void initializeQueue(Queue\* q) {

q->front = 0;

q->rear = -1;

q->size = 0;

}

// Function to check if the queue is empty

int isEmpty(Queue\* q) {

return (q->size == 0);

}

// Function to check if the queue is full

int isFull(Queue\* q) {

return (q->size == MAX);

}

// Function to add an element to the queue

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

if (isFull(q)) {

printf("Queue is full. Cannot insert %d\n", value);

return;

}

q->rear = (q->rear + 1) % MAX;

q->items[q->rear] = value;

q->size++;

}

// Function to remove an element from the queue

int dequeue(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty. Cannot delete\n");

return -1;

}

int item = q->items[q->front];

q->front = (q->front + 1) % MAX;

q->size--;

return item;

}

// Function to display the queue

void displayQueue(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

printf("Queue contents: ");

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

int index = (q->front + i) % MAX;

printf("%d", q->items[index]);

if (i < q->size - 1) {

printf(", ");

}

}

printf("\n");

}

int main() {

Queue q;

initializeQueue(&q);

// Insert elements into the queue

enqueue(&q, 12);

enqueue(&q, 34);

enqueue(&q, 56);

enqueue(&q, 78);

printf("After insertion of 60\n");

displayQueue(&q);

// Insert an element

enqueue(&q, 60);

printf("Content of the queue after inserting 60:\n");

displayQueue(&q);

// Delete an element

int deletedElement = dequeue(&q);

printf("After deletion of %d, the contents of the queue:\n", deletedElement);

displayQueue(&q);

return 0;

}

18.

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.

2. Open brackets must be closed in the correct order.

Input: s = "()"

Output: true

Input: s = "()[]{}"

Output: true

Input: s = "(]"

Output: false

Input: s = "([)]"

Output: false

Input: s = "{[]}"

Output: true

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 100 // Define maximum size for the stack

// Define a structure for the stack

typedef struct Stack {

int top;

char items[MAX];

} Stack;

// Function to initialize the stack

void initStack(Stack\* s) {

s->top = -1;

}

// Function to push an item onto the stack

void push(Stack\* s, char item) {

if (s->top < MAX - 1) {

s->items[++(s->top)] = item;

} else {

printf("Stack overflow\n");

exit(EXIT\_FAILURE);

}

}

// Function to pop an item from the stack

char pop(Stack\* s) {

if (s->top >= 0) {

return s->items[(s->top)--];

} else {

printf("Stack underflow\n");

exit(EXIT\_FAILURE);

}

}

// Function to check if the stack is empty

bool isEmpty(Stack\* s) {

return s->top == -1;

}

// Function to check if the given character is an opening bracket

bool isOpeningBracket(char c) {

return c == '(' || c == '{' || c == '[';

}

// Function to check if the given character is a closing bracket

bool isClosingBracket(char c) {

return c == ')' || c == '}' || c == ']';

}

// Function to check if two brackets are matching pairs

bool isMatchingPair(char opening, char closing) {

return (opening == '(' && closing == ')') ||

(opening == '{' && closing == '}') ||

(opening == '[' && closing == ']');

}

// Function to check if the string of brackets is valid

bool isValid(char\* s) {

Stack stack;

initStack(&stack);

for (int i = 0; s[i] != '\0'; i++) {

char current = s[i];

if (isOpeningBracket(current)) {

push(&stack, current);

} else if (isClosingBracket(current)) {

if (isEmpty(&stack)) {

return false;

}

char top = pop(&stack);

if (!isMatchingPair(top, current)) {

return false;

}

} else {

printf("Invalid character in string\n");

return false;

}

}

return isEmpty(&stack);

}

int main() {

char s1[] = "()";

char s2[] = "()[]{}";

char s3[] = "(]";

char s4[] = "([)]";

char s5[] = "{[]}";

printf("Input: %s\nOutput: %s\n", s1, isValid(s1) ? "true" : "false");

printf("Input: %s\nOutput: %s\n", s2, isValid(s2) ? "true" : "false");

printf("Input: %s\nOutput: %s\n", s3, isValid(s3) ? "true" : "false");

printf("Input: %s\nOutput: %s\n", s4, isValid(s4) ? "true" : "false");

printf("Input: %s\nOutput: %s\n", s5, isValid(s5) ? "true" : "false");

return 0;

}

19.

Given a number n, the task is to print the Fibonacci series and the sum of the series using Iterative procedure.

input n=10

output

Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Sum: 88

Answer:

#include <stdio.h>

int main() {

int n = 10; // You can change this to any positive integer value

int a = 0, b = 1, c;

int sum = 0;

// Print the Fibonacci series

printf("Fibonacci series:\n");

// Print the first Fibonacci number

if (n > 0) {

printf("%d", a);

sum += a;

}

// Print the second Fibonacci number

if (n > 1) {

printf(", %d", b);

sum += b;

}

// Compute and print the rest of the Fibonacci series

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

c = a + b;

printf(", %d", c);

sum += c;

a = b;

b = c;

}

// Print the sum of the Fibonacci series

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

return 0;

}

20.

Given two strings needle and haystack, return the index of the first occurrence

of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "sadbutsad", needle = "sad"

Output: 0

Explanation: "sad" occurs at index 0 and 6.

The first occurrence is at index 0, so we return 0.

Input: haystack = "leetcode", needle = "leeto"

Output: -1

Explanation: "leeto" did not occur in "leetcode", so we return -1.

Answer:

#include <stdio.h>

#include <string.h>

// Function to find the index of the first occurrence of needle in haystack

int strStr(const char \*haystack, const char \*needle) {

// Use strstr to find the first occurrence of needle in haystack

char \*ptr = strstr(haystack, needle);

// If needle is not found, return -1

if (ptr == NULL) {

return -1;

}

// Calculate the index of the first occurrence

return ptr - haystack;

}

int main() {

// Example 1

char haystack1[] = "sadbutsad";

char needle1[] = "sad";

printf("Input: haystack = \"%s\", needle = \"%s\"\n", haystack1, needle1);

printf("Output: %d\n", strStr(haystack1, needle1)); // Output: 0

// Example 2

char haystack2[] = "leetcode";

char needle2[] = "leeto";

printf("Input: haystack = \"%s\", needle = \"%s\"\n", haystack2, needle2);

printf("Output: %d\n", strStr(haystack2, needle2)); // Output: -1

return 0;

}