***Day14 Lab:***

**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.**

**Code:**

#include <stdio.h>

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

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

int key = arr[i], j = i - 1;

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

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

j--;

}

arr[j + 1] = key;

if (i == 6) {

for (int k = 0; k < n; k++) printf("%d%s", arr[k], k == n-1 ? "" : ",");

printf("\n");

}

}

}

int main() {

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

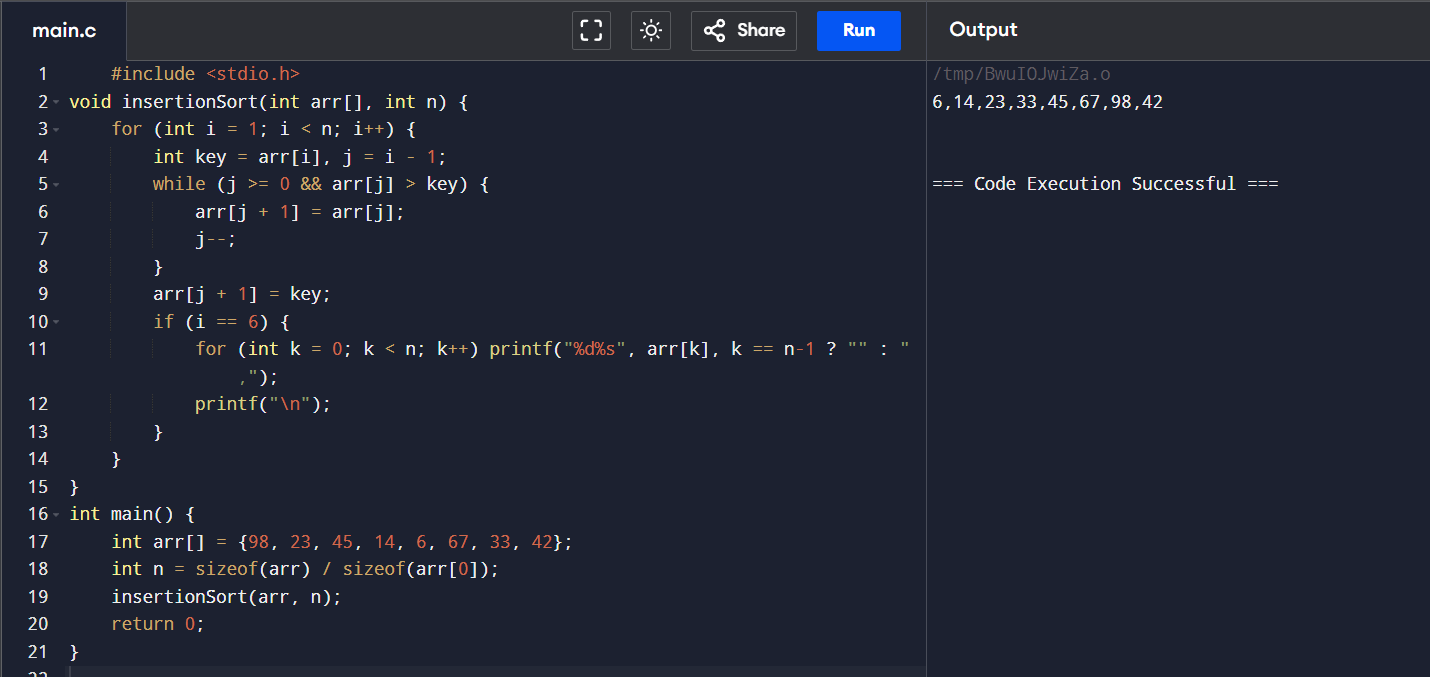
int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

return 0;

}

**Output screenshot:**

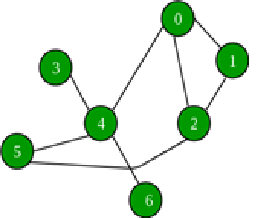
****

**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.**

**Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

typedef struct {

int vertices[MAX\_VERTICES];

int size;

} Queue;

void enqueue(Queue\* q, int v) { q->vertices[q->size++] = v; }

int dequeue(Queue\* q) { return q->vertices[--q->size]; }

bool isEmpty(Queue\* q) { return q->size == 0; }

void bfs(int graph[][MAX\_VERTICES], int start, int end, int num\_vertices) {

bool visited[MAX\_VERTICES];

int distance[MAX\_VERTICES];

Queue q;

for (int i = 0; i < num\_vertices; i++) {

visited[i] = false;

distance[i] = -1;

}

q.size = 0;

enqueue(&q, start);

visited[start] = true;

distance[start] = 0;

while (!isEmpty(&q)) {

int cv = dequeue(&q);

if (cv == end) {

printf("Min edges between %d and %d is %d\n", start, end, distance[end]);

return;

}

for (int i = 0; i < num\_vertices; i++) {

if (graph[cv][i] && !visited[i]) {

enqueue(&q, i);

visited[i] = true;

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

}

}

}

printf("No path between %d and %d\n", start, end);

}

int main() {

int num\_vertices, num\_edges;

printf("Enter num vertices: ");

scanf("%d", &num\_vertices);

printf("Enter num edges: ");

scanf("%d", &num\_edges);

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

printf("Enter edges:\n");

for (int i = 0; i < num\_edges; i++) {

int u, v;

scanf("%d %d", &u, &v);

graph[u][v] = graph[v][u] = 1;

}

int start, end;

printf("Enter start vertex: ");

scanf("%d", &start);

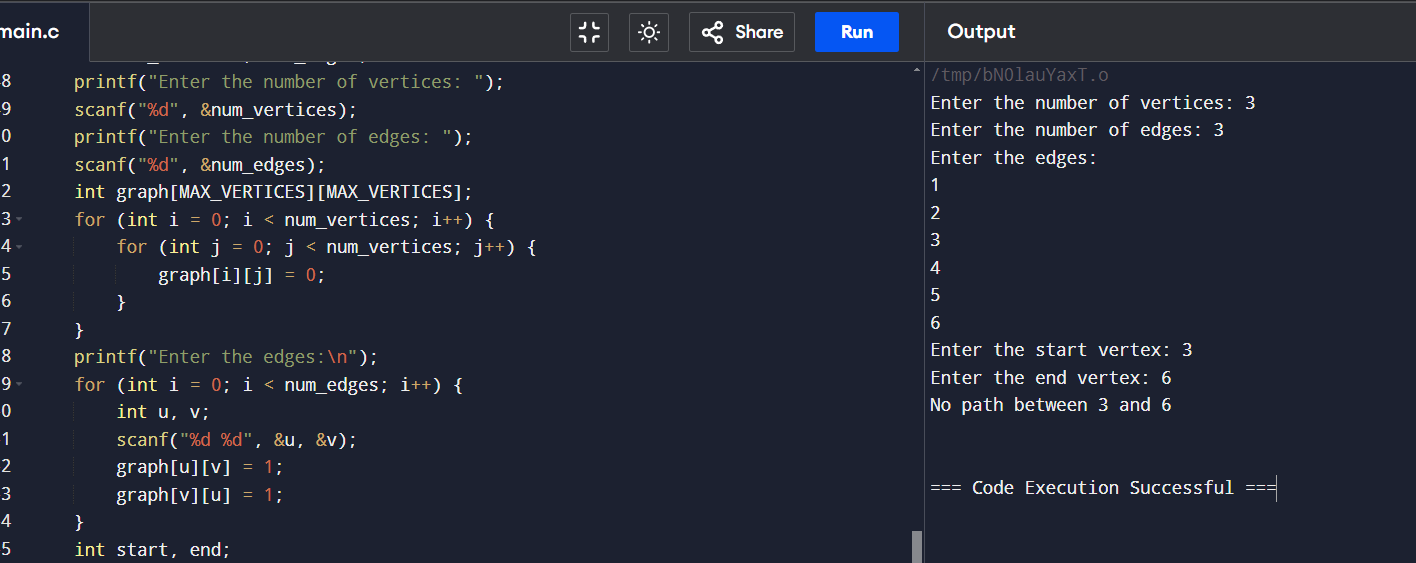
printf("Enter end vertex: ");

scanf("%d", &end);

bfs(graph, start, end, num\_vertices);

return 0;

}**Output screenshot:**

****

**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**

**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

int countNodes(Node\* head) {

int count = 0;

Node\* current = head;

while (current != NULL) {

count++;

current = current->next;

}

return count;

}

int main() {

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

head->data = 1;

head->next = (Node\*)malloc(sizeof(Node));

head->next->data = 2;

head->next->next = (Node\*)malloc(sizeof(Node));

head->next->next->data = 3;

head->next->next->next = (Node\*)malloc(sizeof(Node));

head->next->next->next->data = 5;

head->next->next->next->next = (Node\*)malloc(sizeof(Node));

head->next->next->next->next->data = 8;

head->next->next->next->next->next = NULL;

int numNodes = countNodes(head);

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

}

**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**

**Code:**

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1) {

return n;

}

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

}

void printFibonacciSeries(int n) {

int sum = 0;

printf("Fibonacci series: ");

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

int fib = fibonacci(i);

printf("%d, ", fib);

sum += fib;

}

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

}

int main() {

int n;

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

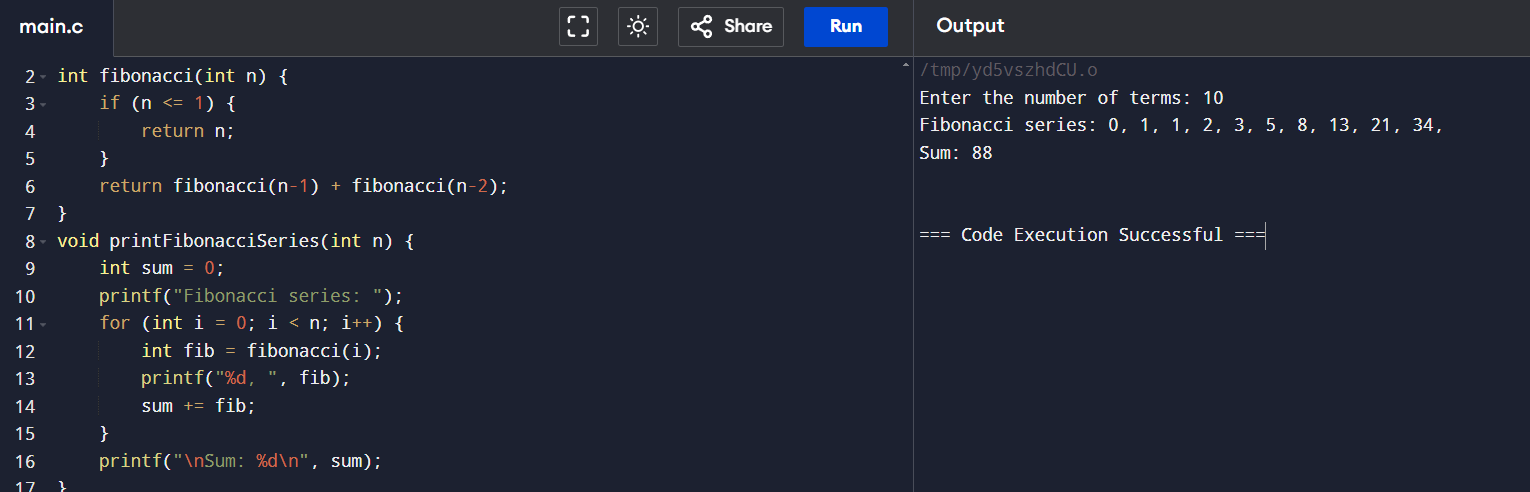
scanf("%d", &n);

printFibonacciSeries(n);

return 0;

}

**Output screenshot:**

****

**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**

**Code:**

#include <stdio.h>

int binarySearch(int arr[], int x, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, x, low, mid - 1);

return binarySearch(arr, x, mid + 1, high);

}

return -1;

}

int main() {

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

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

int x = 6;

int result = binarySearch(arr, x, 0, n - 1);

if (result == -1)

printf("Element is not present in array");

else

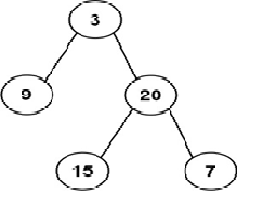
printf("Element is present at index %d", result);

}

**Output screenshot:**

****

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



**Code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* left;

struct Node\* right;

} Node;

Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

void inorderTraversal(Node\* root) {

if (root) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

void postorderTraversal(Node\* root) {

if (root) {

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

}

int main() {

Node\* root = createNode(3);

root->left = createNode(9);

root->right = createNode(20);

root->right->left = createNode(15);

root->right->right = createNode(7);

printf("Inorder Traversal: ");

inorderTraversal(root);

printf("\n");

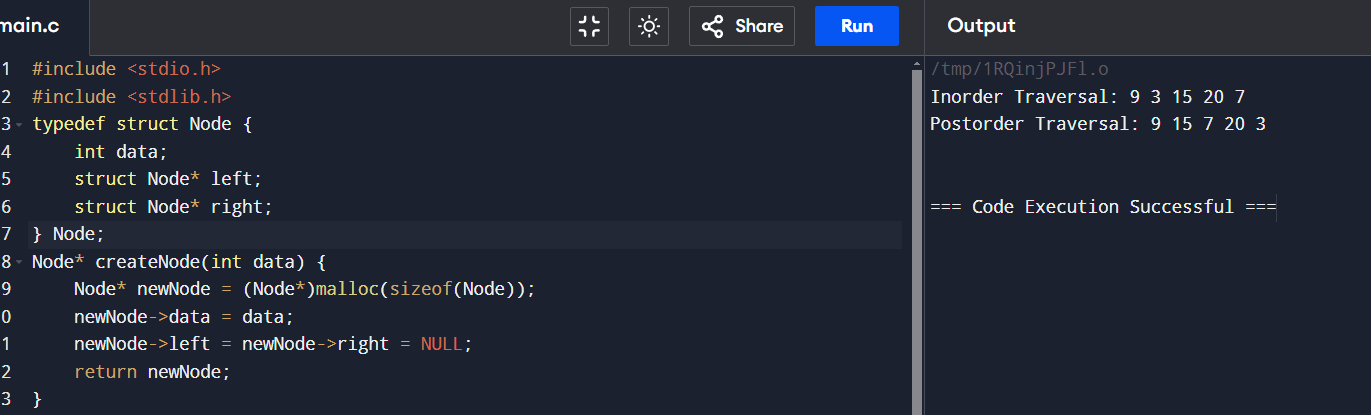
printf("Postorder Traversal: ");

postorderTraversal(root);

printf("\n");

}

**Output screenshot:**

****

**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**

**Code:**

#include <stdio.h>

#include <string.h>

void sort\_string(char \*s) {

int len = strlen(s);

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

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

if (s[i] > s[j]) {

char temp = s[i];

s[i] = s[j];

s[j] = temp;

}

}

}

}

void find\_repeated\_chars(char \*s) {

int len = strlen(s);

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

if (s[i] == s[i + 1]) {

printf("Starting index of repeated character '%c': %d\n", s[i], i);

}

}

}

int main() {

char s[100];

printf("Enter a string: ");

scanf("%s", s);

sort\_string(s);

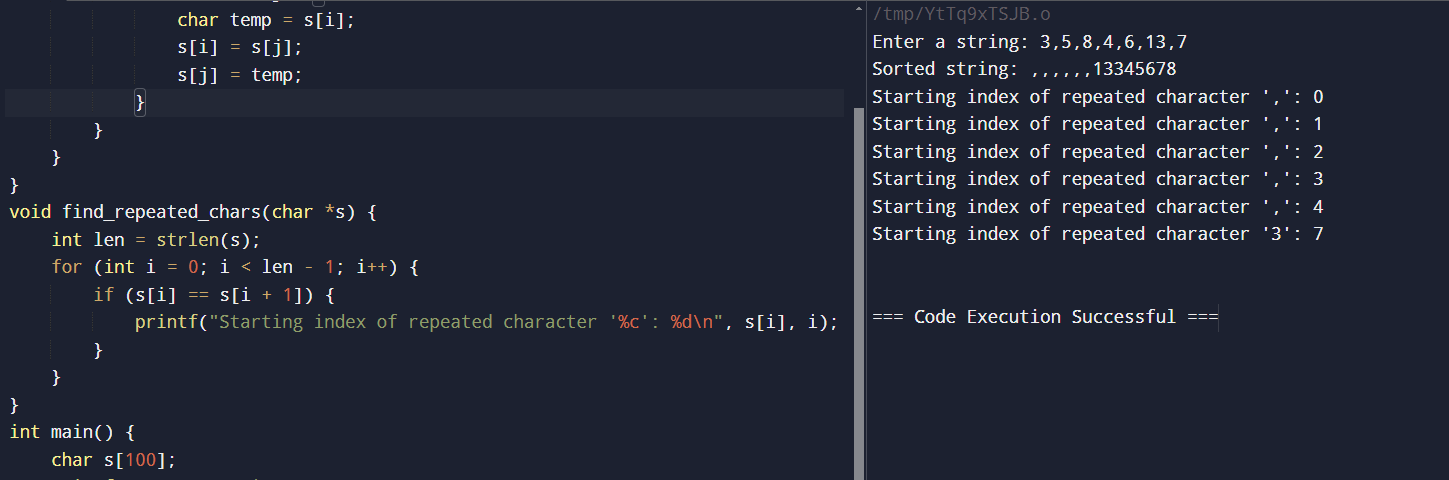
printf("Sorted string: %s\n", s);

find\_repeated\_chars(s);

return 0;

}

Screenshot output:



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

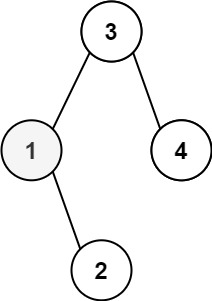
**For example, in the following BST,**

****

**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**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* left;

struct Node\* right;

} Node;

Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory error\n");

return NULL;

}

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}void inorder(Node\* root, int\* count, int k) {

if (root == NULL || \*count >= k)

return;

inorder(root->left, count, k);

(\*count)++;

if (\*count == k) {

printf("The %d-th smallest element is: %d\n", k, root->data);

return;

}

inorder(root->right, count, k);

}void kthSmallest(Node\* root, int k) {

int count = 0;

inorder(root, &count, k);

}int main() {

Node\* root = createNode(3);

root->left = createNode(1);

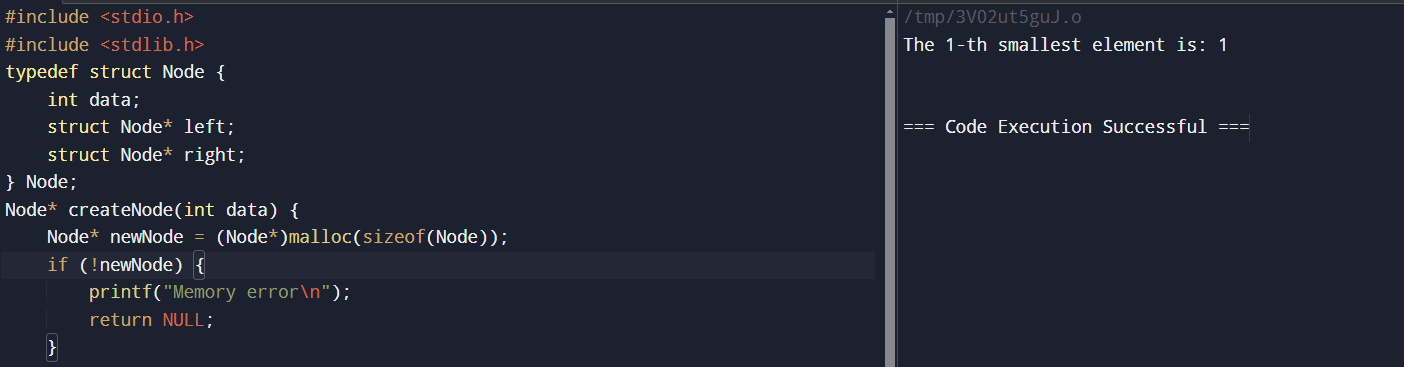
root->right = createNode(4);

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

int k = 1;

kthSmallest(root, k);

}



**10.Given a string s, find the frequency of characters**

**Example 1:**

**Input: s = "tree"**

#include <stdio.h>

#include <string.h>

void frequency(char\* s) {

int freq[256] = {0};

int len = strlen(s);

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

freq[s[i]]++;

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

if (freq[i] > 0) {

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

}

}

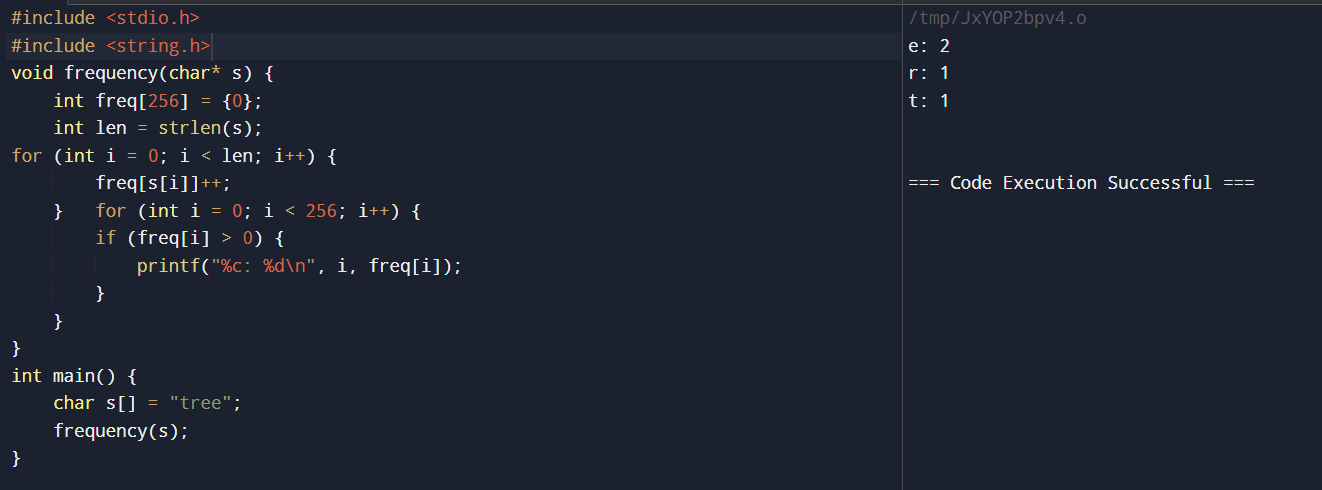
}

int main() {

char s[] = "tree";

frequency(s);

}



**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.**

#include <stdio.h>

int findSmallestPositive(int arr[], int n) {

int i;

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

int j;

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

if (arr[j] == i) {

break;

}

}

if (j == n) {

return i;

}

}

}

int main() {

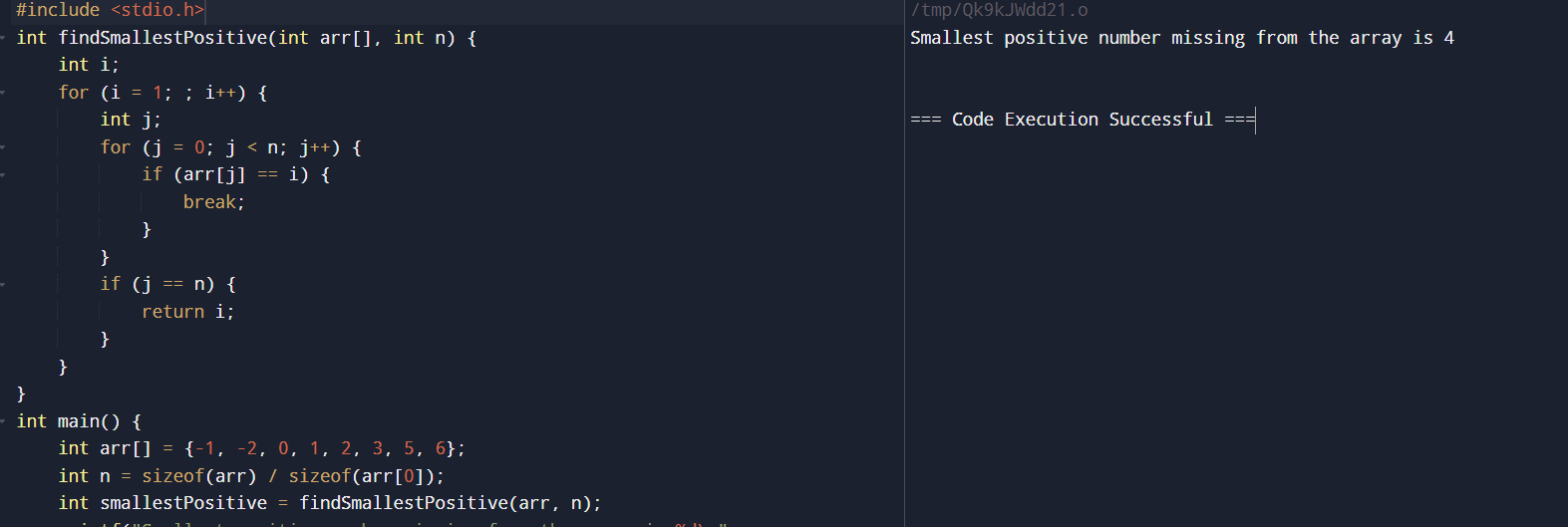
int arr[] = {-1, -2, 0, 1, 2, 3, 5, 6};

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

int smallestPositive = findSmallestPositive(arr, n);

printf("Smallest positive number missing from the array is %d\n", smallestPositive);

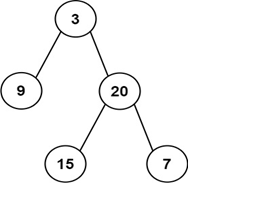
}



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]**

#include <stdio.h>

#include <stdlib.h>

typedef struct TreeNode {

int val;

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

TreeNode\* buildTree(int\* preorder, int preorderSize, int\* inorder, int inorderSize) {

if (preorderSize == 0) {

return NULL;

}

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

root->val = preorder[0];

root->left = root->right = NULL;

int idx = 0;

while (inorder[idx] != root->val) {

idx++;

}

root->left = buildTree(preorder + 1, idx, inorder, idx);

root->right = buildTree(preorder + idx + 1, preorderSize - idx - 1, inorder + idx + 1, inorderSize - idx - 1);

return root;

}

void printInorder(TreeNode\* root) {

if (root == NULL) {

return;

}

printInorder(root->left);

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

printInorder(root->right);

}

int main() {

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

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

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

int inorderSize = sizeof(inorder) / sizeof(inorder[0]);

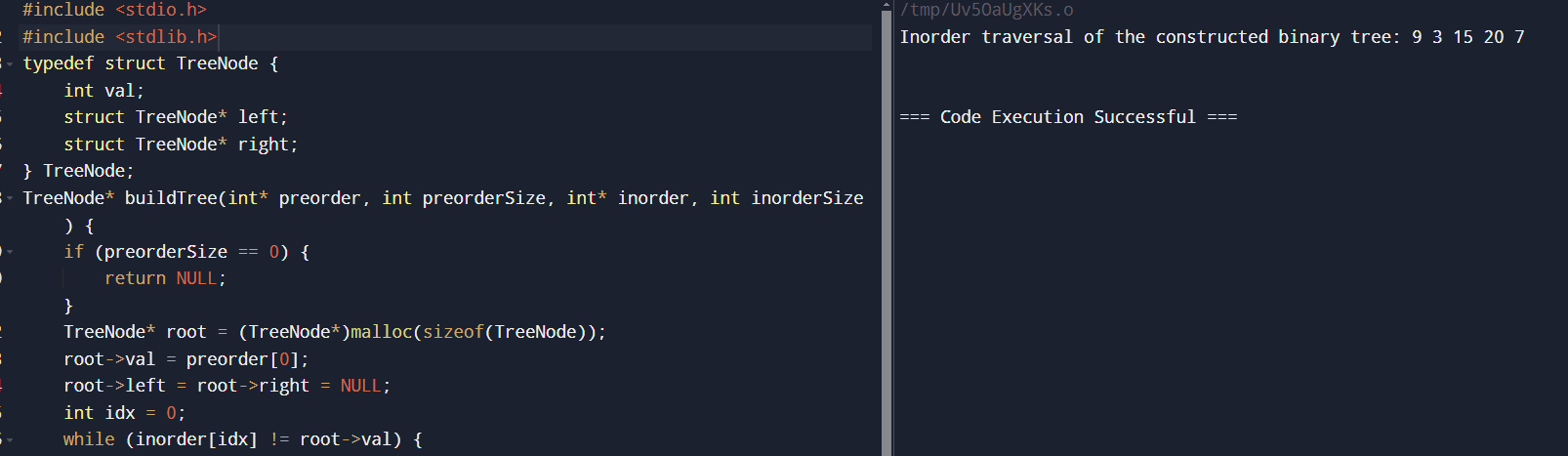
TreeNode\* root = buildTree(preorder, preorderSize, inorder, inorderSize);

printf("Inorder traversal of the constructed binary tree: ");

printInorder(root);

printf("\n");

}



**13.Write a program to create and display a 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 insertNode(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* lastNode = \*head;

while (lastNode->next) {

lastNode = lastNode->next;

}

lastNode->next = newNode;

}

void displayList(Node\* head) {

while (head) {

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

head = head->next;

}

printf("\n");

}

int main() {

Node\* head = NULL;

insertNode(&head, 1);

insertNode(&head, 2);

insertNode(&head, 3);

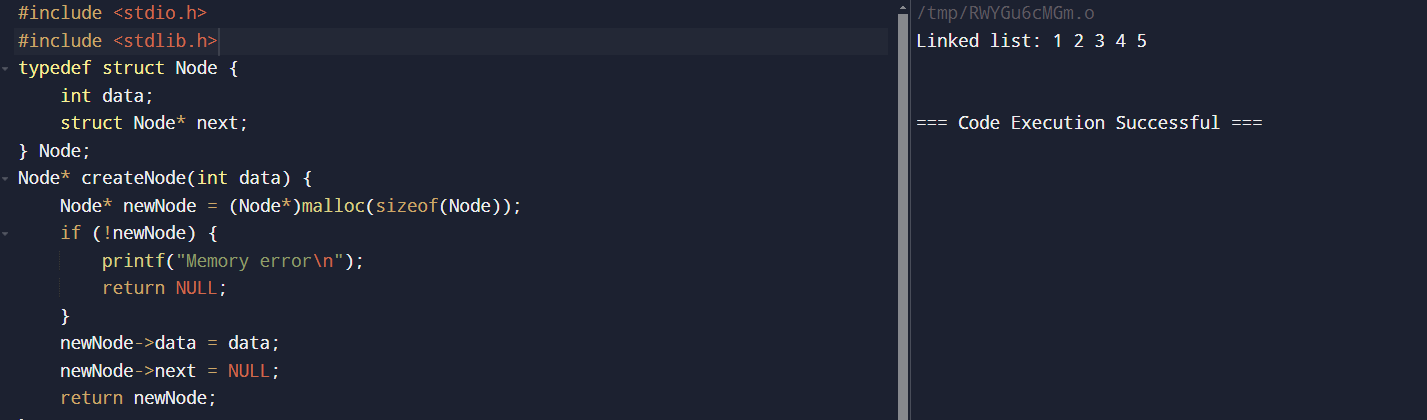
insertNode(&head, 4);

insertNode(&head, 5);

printf("Linked list: ");

displayList(head);

}



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

**Input 4,7,9,1,2**

#include <stdio.h>

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

int i, j, temp;

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

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

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

temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

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

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

printf("Original array: ");

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

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

}

printf("\n");

bubbleSort(arr, n);

printf("Sorted array in descending order: ");

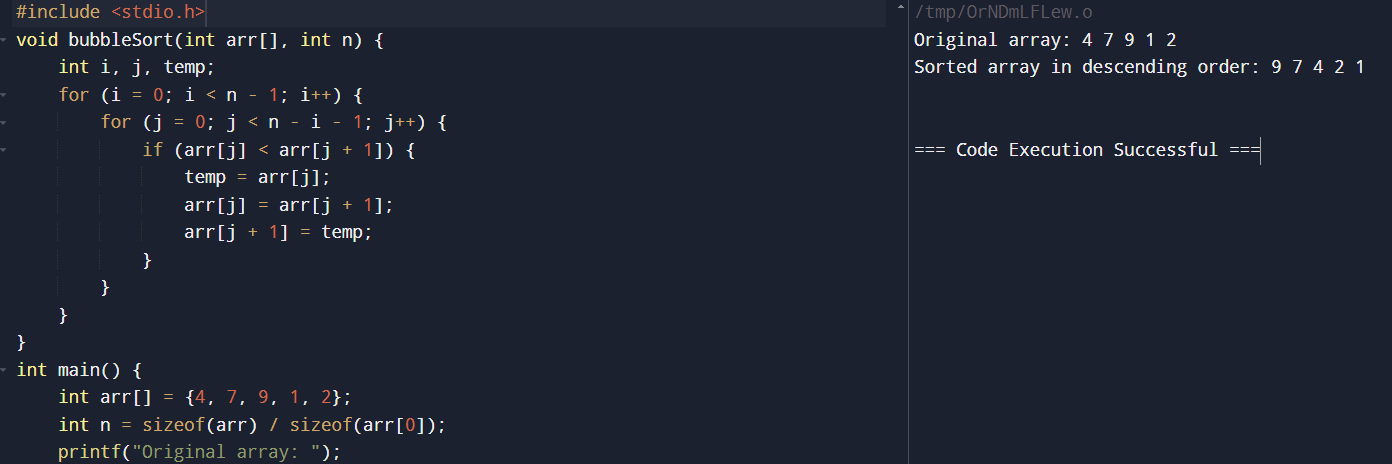
for (int i = 0; i < n; i++) {

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

}

printf("\n");

}



**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.**

#include <stdio.h>

int findMissingElement(int arr[], int n) {

int sum = (n \* (n + 1)) / 2;

int arrSum = 0;

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

arrSum += arr[i];

}

return sum - arrSum;

}

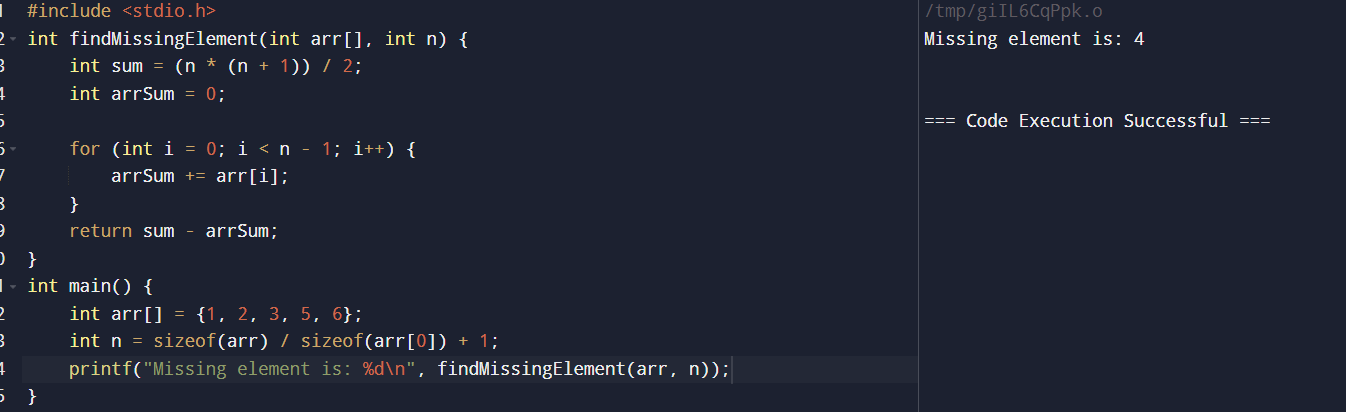
int main() {

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

int n = sizeof(arr) / sizeof(arr[0]) + 1;

printf("Missing element is: %d\n", findMissingElement(arr, n));

}



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

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

#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));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertNode(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;

}

}

void findOddNumbers(Node\* head) {

while (head != NULL) {

if (head->data % 2 != 0) {

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

}

head = head->next;

}

printf("\n");

}

int main() {

Node\* head = NULL;

insertNode(&head, 1);

insertNode(&head, 2);

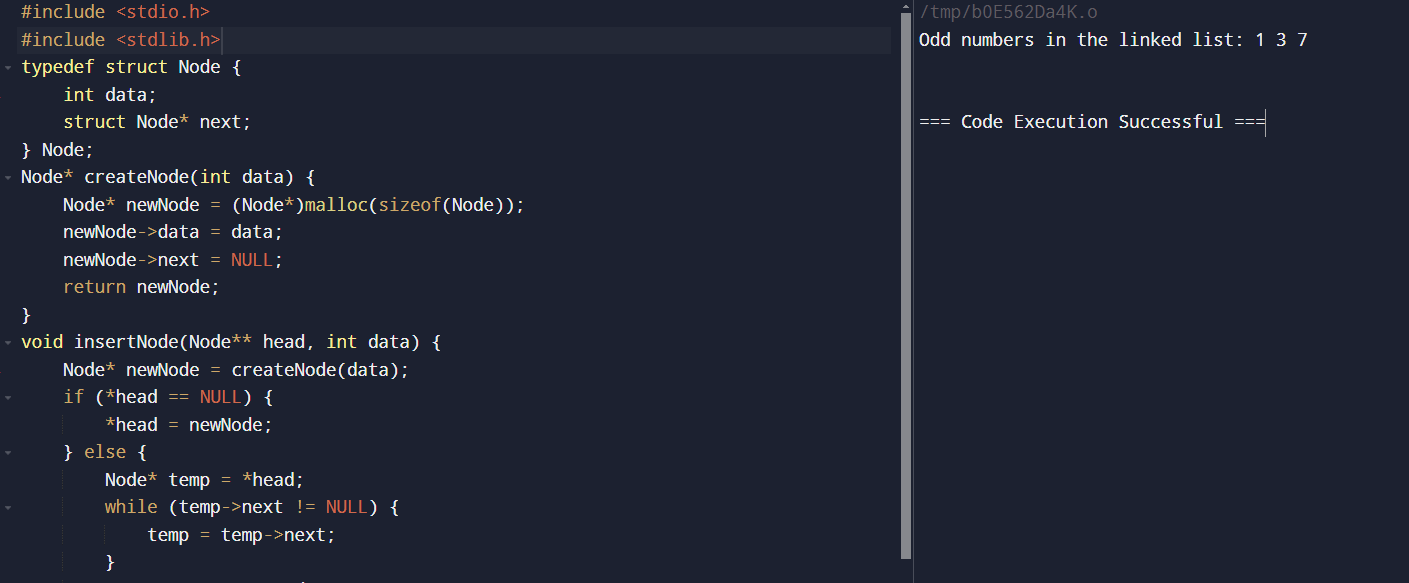
insertNode(&head, 3);

insertNode(&head, 7);

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

findOddNumbers(head);

}



**17. Write a program to perform insert and delete operations in a queue**

**Example : 12,34,56,78**

#include <stdio.h>

#include <stdlib.h>

typedef struct Queue {

int data;

struct Queue\* next;

} Queue;

Queue\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void enqueue(Queue\*\* rear, Queue\*\* front, int data) {

Queue\* newNode = createNode(data);

if (\*front == NULL) {

\*front = newNode;

\*rear = newNode;

} else {

(\*rear)->next = newNode;

\*rear = newNode;

}

}

int dequeue(Queue\*\* front, Queue\*\* rear) {

if (\*front == NULL) {

printf("Queue is empty\n");

return -1;

}

int data = (\*front)->data;

Queue\* temp = \*front;

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

if (\*front == NULL) {

\*rear = NULL;

}

free(temp);

return data;

}

void displayQueue(Queue\* front) {

while (front != NULL) {

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

front = front->next;

}

printf("\n");

}

int main() {

Queue\* front = NULL;

Queue\* rear = NULL;

enqueue(&rear, &front, 12);

enqueue(&rear, &front, 34);

enqueue(&rear, &front, 56);

enqueue(&rear, &front, 78);

printf("Queue: ");

displayQueue(front);

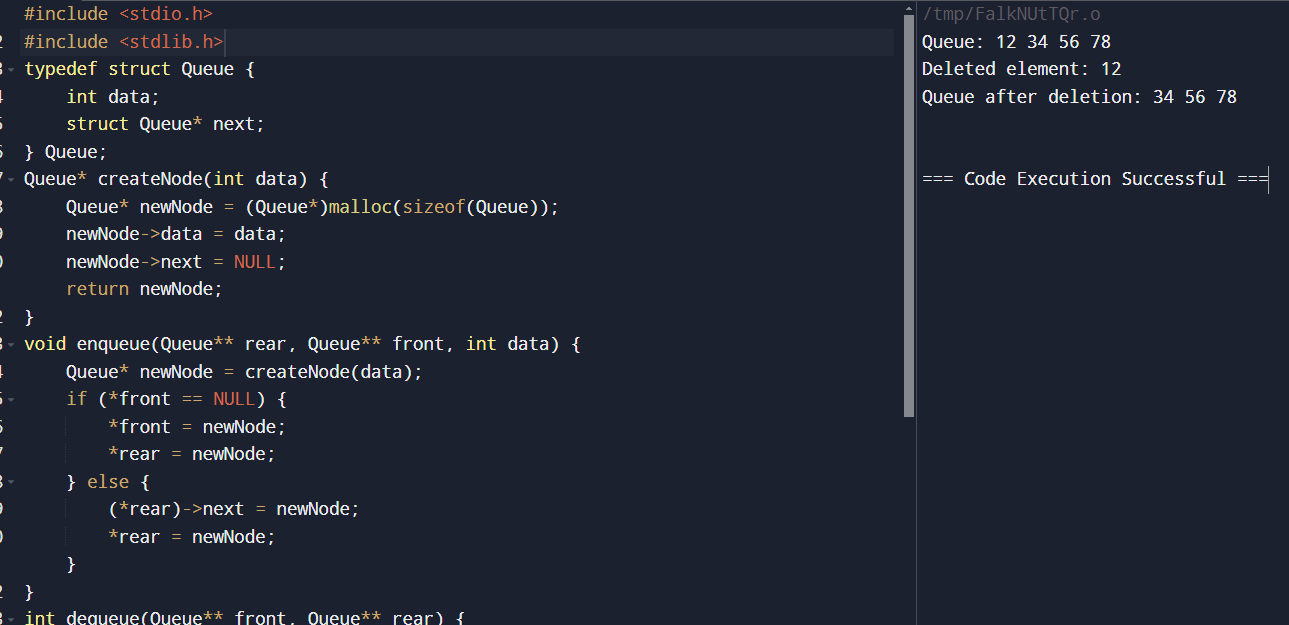
int deleted = dequeue(&front, &rear);

printf("Deleted element: %d\n", deleted);

printf("Queue after deletion: ");

displayQueue(front);

}



**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

#include <stdio.h>

#include <string.h>

int isValid(char \*s) {

int stack[1000];

int top = -1;

char mapping[256];

mapping[')'] = '(';

mapping[']'] = '[';

mapping['}'] = '{';

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

if (s[i] == '(' || s[i] == '[' || s[i] == '{') {

stack[++top] = s[i];

} else if (s[i] == ')' || s[i] == ']' || s[i] == '}') {

if (top == -1 || stack[top] != mapping[s[i]]) {

return 0;

}

top--;

}

}

return top == -1;

}

int main() {

char s1[] = "()";

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

char s3[] = "(]";

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

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

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

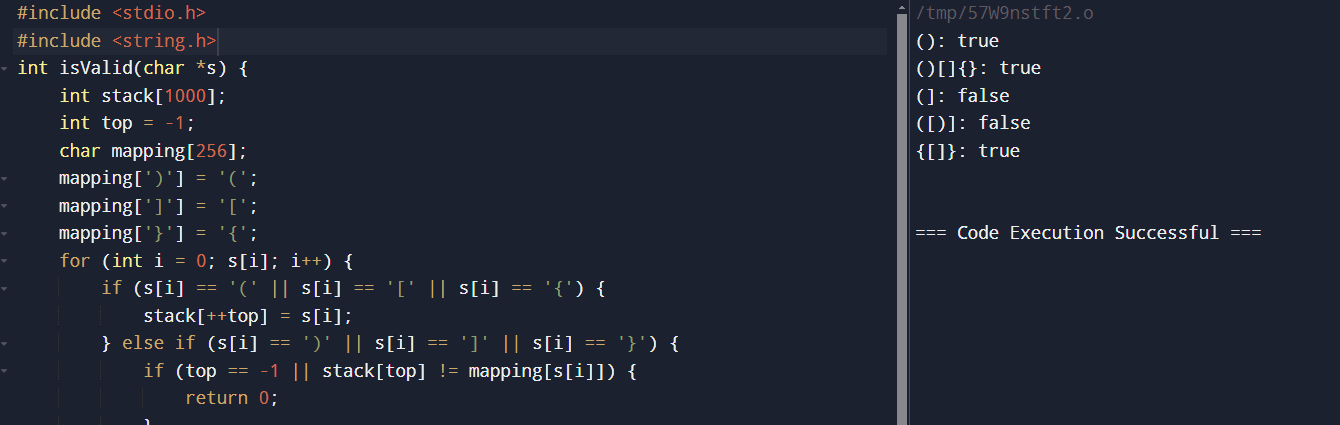
printf("%s: %s\n", s2, isValid(s2) ? "true" : "false");

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

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

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

}



**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**

#include <stdio.h>

void printFibonacci(int n) {

int t1 = 0, t2 = 1, nextTerm = 0, sum = 1;

printf("Fibonacci series:\n");

printf("%d, %d, ", t1, t2);

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

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

sum += nextTerm;

printf("%d, ", nextTerm);

}

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

}

int main() {

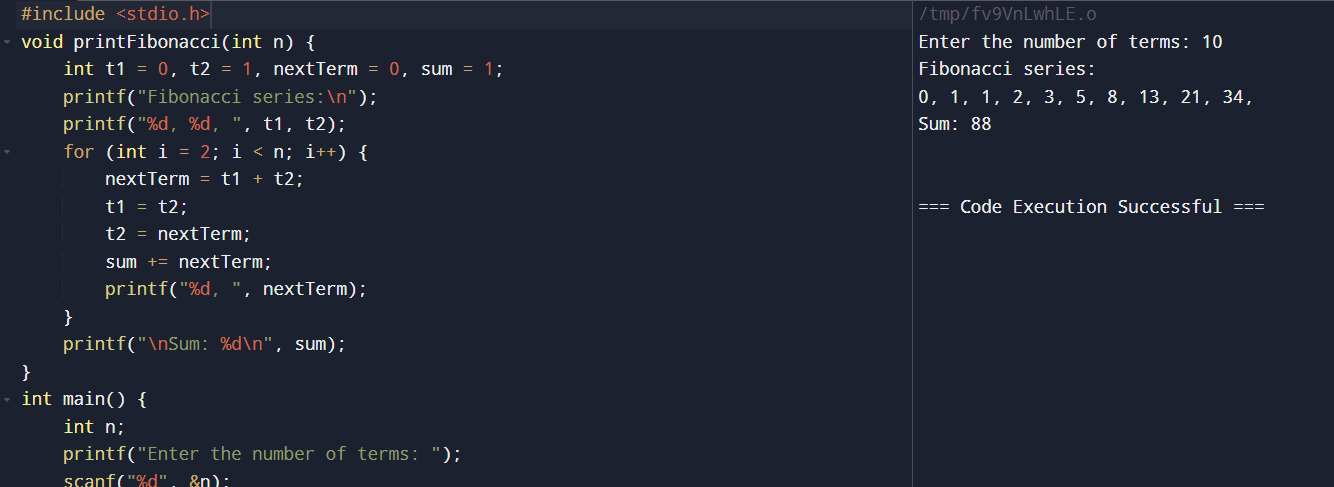
int n;

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

scanf("%d", &n);

printFibonacci(n);

}



**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.**

#include <stdio.h>

#include <string.h>

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

int haystackLen = strlen(haystack);

int needleLen = strlen(needle);

if (needleLen == 0) {

return 0;

}

for (int i = 0; i <= haystackLen - needleLen; i++) {

int j;

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

if (haystack[i + j] != needle[j]) {

break;

}

}

if (j == needleLen) {

return i;

}

}

return -1;

}

int main() {

char haystack1[] = "sadbutsad";

char needle1[] = "sad";

char haystack2[] = "leetcode";

char needle2[] = "leeto";

printf("%s in %s: %d\n", needle1, haystack1, strStr(haystack1, needle1));

printf("%s in %s: %d\n", needle2, haystack2, strStr(haystack2, needle2));

}

