1. **Write a C Program to demonstrate Stack operations using arrays.**

#include<stdio.h>

#define Max 5

int Push(int[],int);

int Pop(int[],int);

void Show(int[],int);

int main(){

int ch,stack[Max],top=-1;

for(;;)

{

printf("1.Push\n2.Pop\n3.Show\n4.Exit\nEnter your choice");

scanf("%d",&ch);

switch(ch)

{

case 1: top=Push(stack,top);

break;

case 2: top=Pop(stack,top);

break;

case 3: Show(stack,top);

break;

case 4: return 0;

default:printf("Invalid choice\nPlease enter valid choice\n");

}//enf of switch

}//end of for

}//end of main

//Push operation

int Push(int s[Max], int t)

{

int element;

if(t==Max-1)

printf("Stack is full....!\n");

else

{

printf("Enter the elements");

scanf("%d",&element);

t+=1;

s[t]=element;

}

return t;

}

//Pop Operation

int Pop(int s[Max],int Top)

{

int element;

if(Top==-1)

printf("Stack is empty");

else

{

element=s[Top];

printf("%d is popped",element);

Top-=1;

}

return(Top);

}

//Display

void Show(int s[Max],int T)

{

int i;

if(T==-1)

printf("stack is empty");

else

{

for(i=0;i<=T;i++)

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

}

}

Output

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 1

Enter the elements: 10

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 1

Enter the elements: 20

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 3

10

20

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 2

20 is popped

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 3

10

1.Push

2.Pop

3.Show

4.Exit

Enter your choice: 4

**2. Write a C Program to evaluate postfix expression, postfix expression contains single digit integers and the operators +,-,\*and /.**

#include <stdio.h>

#define max 10

int push(int[], int, int);

int pop(int[], int\*);

int evaluatePostfix(char postfix[]) {

int stack[max];

int top = -1; // Initialize stack top

int i = 0, op1, op2, result;

char symb;

while (postfix[i] != '\0') {

symb = postfix[i];

if (symb >= '0' && symb <= '9') {

top = push(stack, top, symb - '0'); // Push converted integer

} else if (symb == '+' || symb == '-' || symb == '\*' || symb == '/') {

op2 = pop(stack, &top); // Get second operand

op1 = pop(stack, &top); // Get first operand

// Perform the operation

if (symb == '+')

result = op1 + op2;

else if (symb == '-')

result = op1 - op2;

else if (symb == '\*')

result = op1 \* op2;

else if (symb == '/')

result = op1 / op2;

top = push(stack, top, result); // Push the result onto the stack

}

i++;

}

return pop(stack, &top); // Final result

}

int push(int s[max], int t, int ele) {

if (t == max - 1) {

printf("Stack is full\n");

} else {

s[++t] = ele; // Increment top and push element

}

return t;

}

int pop(int s[max], int \*top) {

if (\*top == -1) {

printf("Stack is empty\n");

return -1; // Indicate error

} else {

return s[(\*top)--]; // Return element and decrement top

}

}

int main() {

char postfix[25];

printf("Enter the postfix expression: ");

scanf("%s", postfix);

int result = evaluatePostfix(postfix);

printf("The result of postfix expression %s is: %d\n", postfix, result);

return 0;

}

output:

1.

Enter the postfix expression: 23+5\*

The result of postfix expression 23+5\* is: 25

2.

Enter the postfix expression: 678+\*

The result of postfix expression 678+\* is: 69

3.

Enter the postfix expression: 9108\*+

The result of postfix expression 9108\*+ is: 88

**3.Write a C Program to convert infix to postfix expression**.

#include <stdio.h>

#include <string.h>

#define MAX 100

// Function to push an element onto the stack

int push(int s[MAX], int t, int ele) {

if (t == MAX - 1) {

printf("Stack is full\n");

} else {

s[++t] = ele; // Increment top and push element

}

return t;

}

// Function to pop an element from the stack

int pop(int s[MAX], int \*top) {

if (\*top == -1) {

printf("Stack is empty\n");

return -1; // Indicate error

} else {

return s[(\*top)--]; // Return element and decrement top

}

}

// Function to check precedence of operators

int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

// Function to convert infix to postfix

void infixToPostfix(char\* infix, char\* postfix) {

int stack[MAX];

int top = -1; // Initialize stack top

int k = 0;

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

char symb = infix[i];

// Directly check if the symbol is an operand (A-Z or 0-9)

if ((symb >= 'A' && symb <= 'Z') || (symb >= '0' && symb <= '9')) {

postfix[k++] = symb;

}

// If the symbol is '(', push it to the stack

else if (symb == '(') {

top = push(stack, top, symb);

}

// If the symbol is ')', pop until '(' is found

else if (symb == ')') {

while (top != -1 && stack[top] != '(') {

postfix[k++] = pop(stack, &top);

}

pop(stack, &top); // Pop '('

}

// If the symbol is an operator

else {

while (top != -1 && precedence(stack[top]) >= precedence(symb)) {

postfix[k++] = pop(stack, &top);

}

top = push(stack, top, symb);

}

}

// Pop all remaining operators from the stack

while (top != -1) {

postfix[k++] = pop(stack, &top);

}

postfix[k] = '\0'; // Null-terminate the postfix expression

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter an infix expression: ");

scanf("%s", infix);

infixToPostfix(infix, postfix);

printf("Postfix expression: %s\n", postfix);

return 0;

}

Output

1.

Enter an infix expression: (A+B)\*C

Postfix expression: AB+C\*

2.

Enter an infix expression: A+B\*C

Postfix expression: ABC\*+

3.

Enter an infix expression: A+(B\*C-(D/E^F)\*G)\*H

Postfix expression: ABC\*DEF^/G\*-H\*+

1. **Write a C Program to demonstrate Queue operations using arrays.**

#include <stdio.h>

#define Max 5 // Define the maximum size of the queue

// Function to insert an element into the queue

int Insert(int q[Max], int rear) {

int ele;

if (rear == Max - 1) {

printf("Queue is full.\n");

} else {

printf("Enter the element to be inserted: ");

scanf("%d", &ele);

rear++;

q[rear] = ele;

}

return rear;

}

// Function to delete an element from the queue

int Delete(int q[Max], int front, int rear) {

int ele;

if (front > rear) {

printf("Queue is empty.\n");

} else {

ele = q[front];

printf("%d is deleted from the queue.\n", ele);

front++;

}

return front;

}

// Function to display the elements of the queue

void Show(int q[Max], int front, int rear) {

if (front > rear) {

printf("Queue is empty.\n");

} else {

printf("Queue elements: ");

for (int i = front; i <= rear; i++) { // Loop from front to rear

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

}

printf("\n");

}

}

int main() {

int queue[Max];

int choice;

int front = 0, rear = -1;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Insert\n2. Delete\n3. Show\n4. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

rear = Insert(queue, rear);

break;

case 2:

front = Delete(queue, front, rear);

break;

case 3:

Show(queue, front, rear);

break;

case 4: // Add exit option

printf("Exiting...\n");

return 0;

default:

printf("Invalid queue choice!\n");

break;

}

}

return 0;

}

Output

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Enter the element to be inserted: 10

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Enter the element to be inserted: 20

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Enter the element to be inserted: 30

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 3

Queue elements: 10 20 30

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 2

10 is deleted from the queue.

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 3

Queue elements: 20 30

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Enter the element to be inserted: 40

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Enter the element to be inserted: 50

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 1

Queue is full.

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 3

Queue elements: 20 30 40 50

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 2

20 is deleted from the queue.

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 2

30 is deleted from the queue.

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 3

Queue elements: 40 50

Queue Operations:

1. Insert

2. Delete

3. Show

4. Exit

Enter your choice: 4

Exiting...

1. **Write a C Program to demonstrate different operations on singly linked list**.

#include<stdio.h>

#include<stdlib.h>

struct Node{

int info;

struct Node\* link;

};

typedef struct Node\* NODE;

//Function declarations

NODE Insert\_Front(int,NODE);

NODE Insert\_End(int,NODE);

NODE Delete\_Front(NODE);

NODE Delete\_End(NODE);

void display(NODE);

//Function to insert the element at front of linked list

NODE Insert\_Front(int e, NODE First){

NODE New = (NODE)malloc(sizeof(struct Node));

New->info=e;

New->link=NULL;

//chech if list empty and newly created is first node

if(First==NULL){

return New;

}

New->link=First;

printf("Insert successfully\n");

return New;

}

//Function to insert the element at End of linked list

NODE Insert\_End(int e, NODE First){

NODE temp=First,New;

New = (NODE)malloc(sizeof(struct Node));

New->info=e;

New->link=NULL;

//chech if list empty and newly created is first node

if(First==NULL){

return New;

}

while(temp->link!=NULL){

temp=temp->link;

}

temp->link=New;

printf("Insert successfully\n");

return First;

}

// Function to delete from the front

NODE Delete\_Front(NODE First) {

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// If the list has one item

if (First->link == NULL) {

printf("%d deleted element\n", First->info);

free(First);

return NULL; // The list will now be empty

}

NODE temp = First;

// More than one item in the list

First = First->link; // Move head to the next node

printf("%d deleted element\n", temp->info);

free(temp); // Free the old head

return First; // Return the new head of the list

}

// Function to delete from the front rear

NODE Delete\_End(NODE First){

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// If the list has one item

if (First->link == NULL) {

printf("%d deleted element\n", First->info);

free(First);

return NULL;

}

NODE Next=First,prev=NULL;

while(Next->link!=NULL){

prev=Next;

Next=Next->link;

}

prev->link=NULL;

printf("%d deleted element\n", Next->info);

free(Next);

return First;

}

void display(NODE First) {

NODE temp = First;

// If the list is empty

if (First == NULL) {

printf("List is empty\n");

return;

}

printf("\n");

printf("Linked List: \n");

while (temp != NULL) {

printf("%d\t", temp->info);

temp = temp->link;

}

printf("\n");

}

// Main function

int main() {

NODE First = NULL;

int choice, ele;

while (1) {

printf("1. Insert at Front\n2. Insert at End\n3. Delete from Front\n4. Delete from End\n5. Display List\n6. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &ele);

First = Insert\_Front(ele, First);

break;

case 2:

printf("Enter value to insert at end: ");

scanf("%d", &ele);

First = Insert\_End(ele, First);

break;

case 3:

First = Delete\_Front(First);

break;

case 4:

First = Delete\_End(First);

break;

case 5:

display(First);

break;

case 6:

exit(0);

default:

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

}

}

return 0;

}

Output

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 1

Enter value to insert at front: 10

Insert successfully

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 2

Enter value to insert at end: 20

Insert successfully

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 5

Linked List:

10 20

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 3

10 deleted element

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 5

Linked List:

20

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 4

20 deleted element

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 5

List is empty

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 6

**6. Write a C Program to demonstrate different operations on circular doubly linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int info;

struct Node\* prev;

struct Node\* next;

};

typedef struct Node\* NODE;

//Function declarations

NODE Insert\_Front(int, NODE);

NODE Insert\_End(int, NODE);

NODE Delete\_Front(NODE);

NODE Delete\_End(NODE);

void display(NODE);

//Function to insert the element at the front of the linked list

NODE Insert\_Front(int e, NODE First){

NODE New, temp;

New = (NODE)malloc(sizeof(struct Node));

New->info = e;

New->prev = NULL;

New->next = NULL;

// Check if list is empty

if (First == NULL) {

New->next = New;

New->prev = New;

return New;

}

// Non-empty list, find the last node

temp = First;

while (temp->next != First) {

temp = temp->next;

}

// Insert New node at the front

New->next = First;

New->prev = temp;

First->prev = New;

temp->next = New;

printf("Inserted %d at the front successfully\n", e);

return New;

}

// Function to insert element at end of linked list

NODE Insert\_End(int e, NODE First){

NODE New, temp;

New = (NODE)malloc(sizeof(struct Node));

New->info = e;

New->prev = NULL;

New->next = NULL;

// Check if list is empty

if (First == NULL) {

New->next = New;

New->prev = New;

return New;

}

// Find last node

temp = First;

while (temp->next != First) {

temp = temp->next;

}

// Insert new node at the end

New->prev = temp;

New->next = First;

temp->next = New;

First->prev = New;

printf("Inserted %d at the end successfully\n", e);

return First;

}

// Function to delete the front node

NODE Delete\_Front(NODE First) {

NODE temp = First, temp1 = First;

// Check if list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// Check if there is only one node

if (First->next == First) {

printf("%d deleted\n", First->info);

free(First);

return NULL;

}

// Find the last node

while (temp->next != First) {

temp = temp->next;

}

First = First->next;

printf("%d deleted\n", temp1->info); // Corrected to print the front node's info

First->prev = temp;

temp->next = First;

free(temp1);

return First;

}

// Function to delete the last node

NODE Delete\_End(NODE First) {

NODE temp = First, temp1 = NULL;

// Check if list is empty

if (First == NULL) {

printf("List is empty\n");

return NULL;

}

// Check if there is only one node

if (First->next == First) {

printf("%d deleted\n", First->info);

free(First);

return NULL;

}

// Find the last node

while (temp->next != First) {

temp = temp->next;

}

temp1 = temp->prev;

printf("%d deleted\n", temp->info);

temp1->next = First;

First->prev = temp1;

free(temp);

return First;

}

// Function to display the circular doubly linked list

void display(NODE First) {

NODE temp;

if (First == NULL) {

printf("List is empty\n");

return;

}

temp = First;

do {

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

temp = temp->next;

} while (temp != First);

printf("\n");

}

// Main function

int main() {

NODE First = NULL;

int choice, ele;

while (1) {

printf("1. Insert at Front\n2. Insert at End\n3. Delete from Front\n4. Delete from End\n5. Display List\n6. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &ele);

First = Insert\_Front(ele, First);

break;

case 2:

printf("Enter value to insert at end: ");

scanf("%d", &ele);

First = Insert\_End(ele, First);

break;

case 3:

First = Delete\_Front(First);

break;

case 4:

First = Delete\_End(First);

break;

case 5:

display(First);

break;

case 6:

exit(0);

default:

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

}

}

return 0;

}

Output

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 1

Enter value to insert at front: 10

Inserted 10 at the front successfully

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 1

Enter value to insert at front: 20

Inserted 20 at the front successfully

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 2

Enter value to insert at end: 30

Inserted 30 at the end successfully

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 5

20 10 30

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 3

20 deleted

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 4

30 deleted

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 5

10

1. Insert at Front

2. Insert at End

3. Delete from Front

4. Delete from End

5. Display List

6. Exit

Enter your choice: 6

**7.Write a C program to implement the following operation on binary treeusing array:**

**i. Insert**

**ii. Delete**

**iii. Tree traversal**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

// Node structure for the binary search tree

struct node {

int info;

int left, right;

};

// Array-based tree representation

struct node tree[MAX];

int avail = 0; // Availability list for free nodes

// Function to initialize the tree and availability list

void initialize() {

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

tree[i].left = i + 1;

}

tree[MAX - 1].left = -1; // End of availability list

avail = 0; // Initially, all nodes are available

}

// Insert function to add an element to the tree

int Insert(int root, int ele, struct node t[], int \*avail) {

int New = \*avail, temp1 = root, temp2 = root;

\*avail = t[\*avail].left; // Update avail to the next available node

t[New].info = ele;

t[New].left = t[New].right = -1;

// If the tree is empty, return the new node as the root

if (root == -1) {

return New;

}

// Traverse the tree to find the correct position for the new element

while (temp2 != -1) {

temp1 = temp2;

if (ele < t[temp2].info) {

temp2 = t[temp2].left;

} else if (ele > t[temp2].info) {

temp2 = t[temp2].right;

} else {

// Element already exists, return the root without inserting

return root;

}

}

// Insert the new node at the correct position

if (ele < t[temp1].info) {

t[temp1].left = New;

} else {

t[temp1].right = New;

}

return root; // Return the root of the tree

}

// Delete function to remove an element from the tree

int Delete(int root, int ele, struct node tree[], int \*avail) {

int temp1 = root, temp2 = root, parent = -1;

// Check if the tree is empty

if (root == -1) {

printf("Tree is empty\n");

return -1;

}

// Search for the node to delete

while (temp2 != -1 && tree[temp2].info != ele) {

parent = temp2;

if (ele < tree[temp2].info) {

temp2 = tree[temp2].left;

} else {

temp2 = tree[temp2].right;

}

}

// If the element is not found

if (temp2 == -1) {

printf("%d is not found\n", ele);

return root;

}

// Node to be deleted is found

if (tree[temp2].left == -1 && tree[temp2].right == -1) {

// Node is a leaf

printf("%d is deleted\n", tree[temp2].info);

tree[temp2].left = \*avail; // Return node to avail list

\*avail = temp2; // Update the available list to include this node

} else {

printf("Can't delete the node because it is not a leaf node (it has children).\n");

}

return root; // Return the root if no deletion occurs

}

// In-order traversal of the binary search tree

void inorder(int index) {

if (index == -1) return;

inorder(tree[index].left); // Left child

printf("%d ", tree[index].info);

inorder(tree[index].right); // Right child

}

// Pre-order traversal of the binary search tree

void preorder(int index) {

if (index == -1) return;

printf("%d ", tree[index].info);

preorder(tree[index].left); // Left child

preorder(tree[index].right); // Right child

}

// Post-order traversal of the binary search tree

void postorder(int index) {

if (index == -1) return;

postorder(tree[index].left); // Left child

postorder(tree[index].right); // Right child

printf("%d ", tree[index].info);

}

// Main function to demonstrate the operations

int main() {

int root = -1;

initialize();

int choice, ele;

while (1) {

printf("\n1. Insert\n2. Delete\n3. In-order Display\n4. Pre-order Display\n5. Post-order Display\n6. Exit\nEnter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to insert: ");

scanf("%d", &ele);

root = Insert(root, ele, tree, &avail);

break;

case 2:

printf("Enter element to delete: ");

scanf("%d", &ele);

root = Delete(root, ele, tree, &avail);

break;

case 3:

printf("Tree elements (in-order): ");

if (root == -1) {

printf("Tree is empty");

} else {

inorder(root);

}

printf("\n");

break;

case 4:

printf("Tree elements (pre-order): ");

if (root == -1) {

printf("Tree is empty");

} else {

preorder(root);

}

printf("\n");

break;

case 5:

printf("Tree elements (post-order): ");

if (root == -1) {

printf("Tree is empty");

} else {

postorder(root);

}

printf("\n");

break;

case 6:

exit(0);

default:

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

}

}

return 0;

}

Output:

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 1

Enter element to insert: 10

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 1

Enter element to insert: 5

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 1

Enter element to insert: 15

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 3

Tree elements (in-order): 5 10 15

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 4

Tree elements (pre-order): 10 5 15

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 5

Tree elements (post-order): 5 15 10

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 2

Enter element to delete: 15

15 is deleted

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 3

Tree elements (in-order): 5 10

1. Insert

2. Delete

3. In-order Display

4. Pre-order Display

5. Post-order Display

6. Exit

Enter choice: 6

8. Write a C program to demonstrate binary search using recursion.

#include <stdio.h>

// Function prototype for binary search

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

int main() {

int a[10], n, i, j, ele, temp;

printf("How many elements? (max 10)\n");

scanf("%d", &n);

printf("Enter the elements:\n");

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

scanf("%d", &a[i]);

// Sorting the elements using selection sort

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

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

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

// Swap elements

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

printf("Enter the element to be searched: ");

scanf("%d", &ele);

// Perform binary search

int result = binarySearch(a, 0, n - 1, ele);

// Display the result

if (result == -1)

printf("Element is not present in the array.\n");

else

printf("Element is present at index %d (0-based index).\n", result);

return 0;

}

// Recursive binary search function

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

if (high >= low) {

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

// Check if the middle element is the one we're searching for

if (arr[mid] == x)

return mid;

// Search in the lower half if the element is smaller than mid

if (arr[mid] > x)

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

// Search in the upper half otherwise

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

}

// Element not found

return -1;

}

Output:

1.

How many elements? (max 10)

5

Enter the elements:

12

3

7

19

1

Enter the element to be searched: 7

Element is present at index 2.

2.

How many elements? (max 10)

6

Enter the elements:

15

8

6

11

3

20

Enter the element to be searched: 10

Element is not present in the array.