1.

#include <stdio.h>

#define MAX\_SIZE 10

void matrixMultiplication(int mat1[][MAX\_SIZE], int mat2[][MAX\_SIZE], int result[][MAX\_SIZE], int rows1, int cols1, int cols2) {

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols2; j++) {

result[i][j] = 0;

for (int k = 0; k < cols1; k++) {

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

}

}

}

}

void displayMatrix(int matrix[][MAX\_SIZE], int rows, int cols) {

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

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

printf("%d\t", matrix[i][j]);

}

printf("\n");

}

}

int main() {

int mat1[MAX\_SIZE][MAX\_SIZE], mat2[MAX\_SIZE][MAX\_SIZE], result[MAX\_SIZE][MAX\_SIZE];

int rows1, cols1, rows2, cols2;

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

scanf("%d %d", &rows1, &cols1);

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

for (int i = 0; i < rows1; i++) {

for (int j = 0; j < cols1; j++) {

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

}

}

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

scanf("%d %d", &rows2, &cols2);

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

for (int i = 0; i < rows2; i++) {

for (int j = 0; j < cols2; j++) {

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

}

}

if (cols1 != rows2) {

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

} else {

matrixMultiplication(mat1, mat2, result, rows1, cols1, cols2);

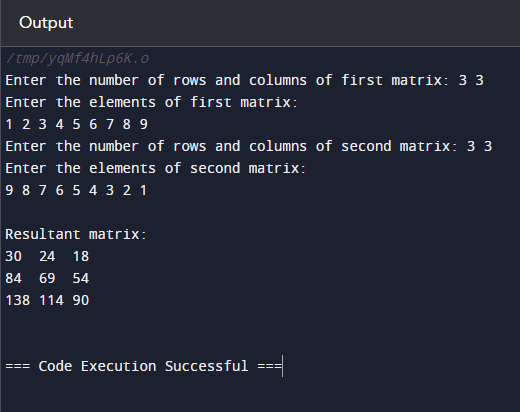
printf("\nResultant matrix:\n");

displayMatrix(result, rows1, cols2);

}

return 0;

}



2.

#include <stdio.h>

int main() {

int num;

char choice;

do {

printf("Enter a number: ");

scanf("%d", &num);

if (num % 2 == 0) {

printf("%d is even.\n", num);

} else {

printf("%d is odd.\n", num);

}

printf("Do you want to check another number? (y/n): ");

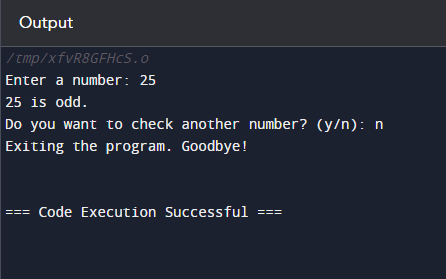
scanf(" %c", &choice); // Note the space before %c to consume the newline character

} while (choice == 'y' || choice == 'Y');

printf("Exiting the program. Goodbye!\n");

return 0;

}



3.

#include <stdio.h>

unsigned long long factorial(int number) {

unsigned long long fact = 1;

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

fact \*= i;

}

return fact;

}

int main() {

int num;

printf("Enter a number to find factorial: ");

scanf("%d", &num);

if (num < 0) {

printf("Factorial of negative number doesn't exist.\n");

} else {

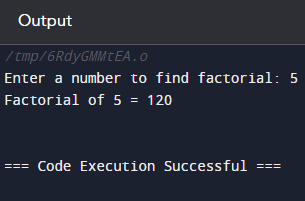
unsigned long long result = factorial(num);

printf("Factorial of %d = %llu\n", num, result);

}

return 0;

}



4.

#include<stdio.h>

int main(){

int n,f1=0,f2=1,f,i;

printf("Enter the limit:");

scanf("%d",&n);

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

{

if(i<=1)

f=i;

else

{

f=f1+f2;

f1=f2;

f2=f;

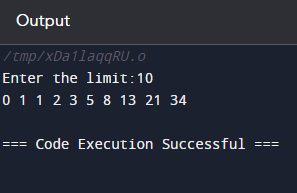
}

printf("%d ",f);

}

return 0;

}



5.

#include <stdio.h>

int factorial(int n);

int main() {

int num;

printf("Enter a non-negative integer to calculate its factorial: ");

scanf("%d", &num);

if (num < 0) {

printf("Error: Factorial of a negative number is not defined.\n");

} else {

printf("Factorial of %d = %d\n", num, factorial(num));

}

return 0;

}

int factorial(int n) {

if (n == 0) {

return 1;

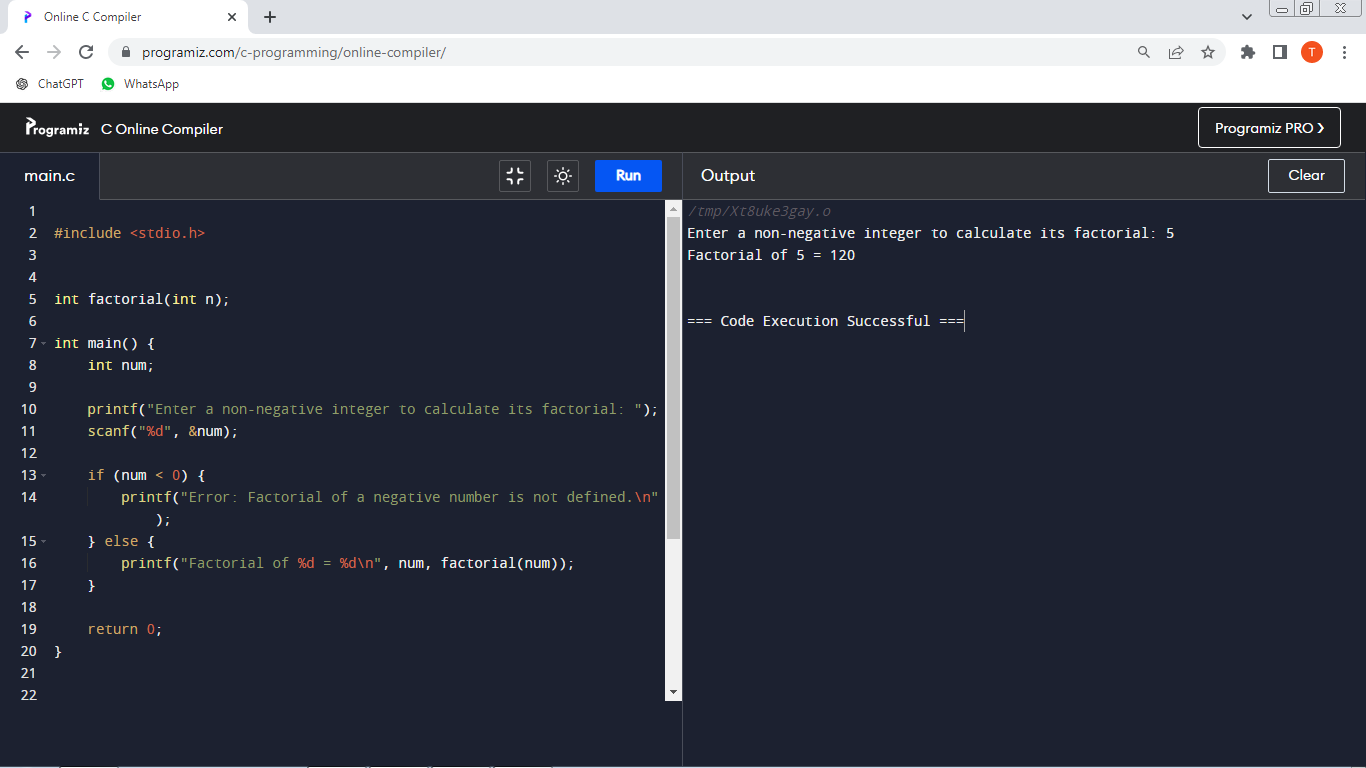
}

else {

return n \* factorial(n - 1);

}

}



6.

#include <stdio.h>

int fibonacci(int n);

int main() {

int numTerms, i;

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

scanf("%d", &numTerms);

printf("Fibonacci Series:\n");

for (i = 0; i < numTerms; i++) {

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

}

printf("\n");

return 0;

}

int fibonacci(int n) {

if (n == 0)

return 0;

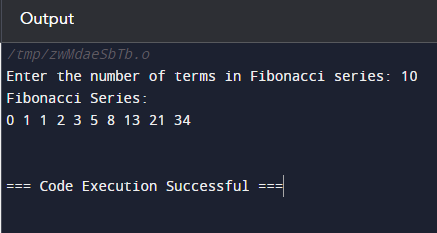
else if (n == 1)

return 1;

else

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

}



7.

#include <stdio.h>

void displayArray(int arr[], int size);

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

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

int main() {

int array[100];

int size = 0;

int choice, element, position;

while (1) {

printf("\nArray Operations Menu:\n");

printf("1. Insert element\n");

printf("2. Delete element\n");

printf("3. Display array\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to insert: ");

scanf("%d", &element);

printf("Enter position to insert (0-indexed): ");

scanf("%d", &position);

insertElement(array, &size, element, position);

break;

case 2:

printf("Enter position to delete (0-indexed): ");

scanf("%d", &position);

deleteElement(array, &size, position);

break;

case 3:

printf("Array elements:\n");

displayArray(array, size);

break;

case 4:

printf("Exiting program.\n");

return 0;

default:

printf("Invalid choice! Please enter a valid menu option.\n");

}

}

return 0;

}

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

if (size == 0) {

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

} else {

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

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

}

printf("\n");

}

}

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

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

printf("Invalid position! Insertion failed.\n");

} else {

for (int i = \*size - 1; i >= position; i--) {

arr[i + 1] = arr[i];

}

arr[position] = element;

(\*size)++;

printf("Element %d inserted successfully at position %d.\n", element, position);

}

}

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

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

printf("Invalid position! Deletion failed.\n");

} else {

for (int i = position; i < \*size - 1; i++) {

arr[i] = arr[i + 1];

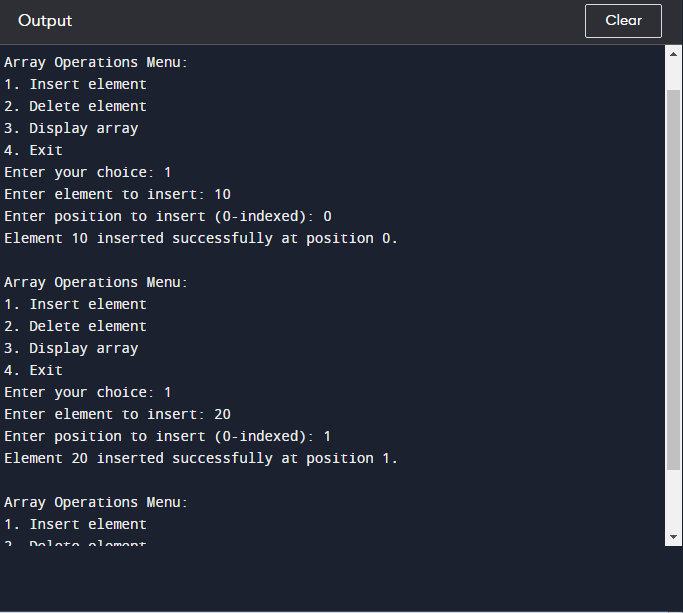
}

(\*size)--;

printf("Element at position %d deleted successfully.\n", position);

}

}



8.

#include <stdio.h>

void linearSearch(int arr[], int size, int key) {

int found = 0;

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

if (arr[i] == key) {

printf("Element %d found at index %d\n", key, i);

found = 1;

break;

}

}

if (!found) {

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

}

}

int main() {

int size, key;

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size];

printf("Enter %d elements of the array:\n", size);

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

printf("Element %d: ", i + 1);

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

}

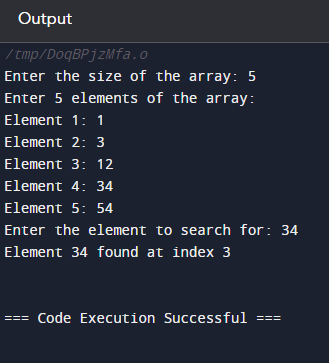
printf("Enter the element to search for: ");

scanf("%d", &key);

linearSearch(arr, size, key);

return 0;

}



9.

#include <stdio.h>

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

int low = 0, high = size - 1;

while (low <= high) {

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

if (arr[mid] == key) {

return mid;

}

if (arr[mid] < key) {

low = mid + 1;

}

else {

high = mid - 1;

}

}

return -1;

}

int main() {

int size, key, result;

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size];

printf("Enter %d elements of the array in sorted order:\n", size);

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

printf("Element %d: ", i + 1);

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

}

printf("Enter the element to search for: ");

scanf("%d", &key);

result = binarySearch(arr, size, key);

if (result != -1) {

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

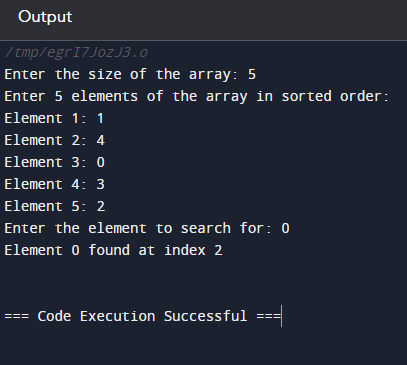
} else {

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

}

return 0;

}



10.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void insertAtBeginning(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void insertAtEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

struct Node\* last = \*head\_ref;

new\_node->data = new\_data;

new\_node->next = NULL;

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

\*head\_ref = new\_node;

return;

}

while (last->next != NULL) {

last = last->next;

}

last->next = new\_node;

return;

}

void deleteNode(struct Node\*\* head\_ref, int key) {

struct Node\* temp = \*head\_ref, \*prev;

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

\*head\_ref = temp->next;

free(temp);

return;

}

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

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

void displayList(struct Node \*node) {

while (node != NULL) {

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

node = node->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, value;

while (1) {

printf("\nMenu:\n");

printf("1. Insert at beginning\n");

printf("2. Insert at end\n");

printf("3. Delete by value\n");

printf("4. Display list\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

insertAtBeginning(&head, value);

break;

case 2:

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

scanf("%d", &value);

insertAtEnd(&head, value);

break;

case 3:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

break;

case 4:

printf("Linked list: ");

displayList(head);

break;

case 5:

exit(0);

default:

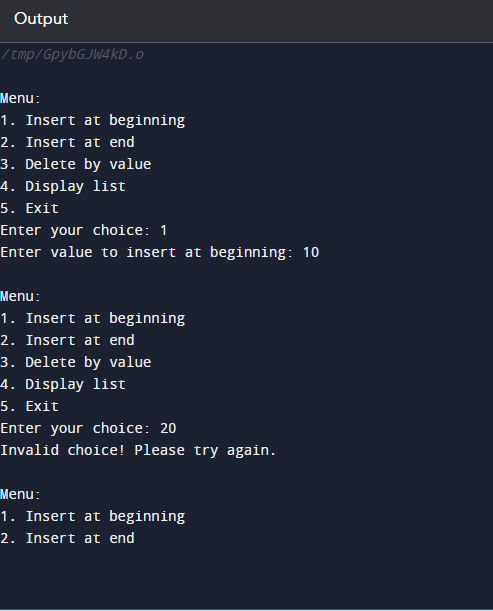
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



11.

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Stack {

int items[MAX];

int top;

};

void initialize(struct Stack\* stack) {

stack->top = -1;

}

int isFull(struct Stack\* stack) {

return stack->top == MAX - 1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void push(struct Stack\* stack, int item) {

if (isFull(stack)) {

printf("Stack overflow! Cannot push %d\n", item);

return;

}

stack->items[++stack->top] = item;

printf("%d pushed to stack\n", item);

}

int pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack underflow! Cannot pop\n");

return -1;

}

return stack->items[stack->top--];

}

int peek(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty! Cannot peek\n");

return -1;

}

return stack->items[stack->top];

}

void display(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack is empty!\n");

return;

}

printf("Stack elements: ");

for (int i = 0; i <= stack->top; i++) {

printf("%d ", stack->items[i]);

}

printf("\n");

}

int main() {

struct Stack stack;

int choice, value;

initialize(&stack);

while (1) {

printf("\nMenu:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Display stack\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

value = pop(&stack);

if (value != -1) {

printf("Popped value: %d\n", value);

}

break;

case 3:

value = peek(&stack);

if (value != -1) {

printf("Top value: %d\n", value);

}

break;

case 4:

display(&stack);

break;

case 5:

exit(0);

default:

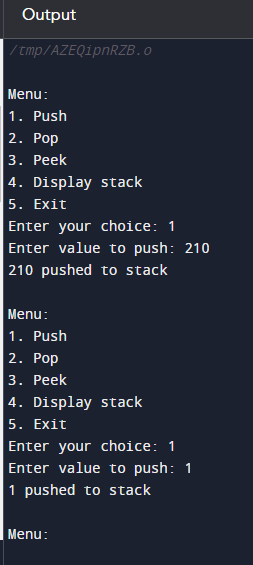
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



12.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

struct Stack {

int top;

int items[MAX];

};

void initialize(struct Stack\* stack) {

stack->top = -1;

}

int isFull(struct Stack\* stack) {

return stack->top == MAX - 1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void push(struct Stack\* stack, char item) {

if (isFull(stack)) {

printf("Stack overflow!\n");

return;

}

stack->items[++stack->top] = item;

}

char pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack underflow!\n");

return -1;

}

return stack->items[stack->top--];

}

char peek(struct Stack\* stack) {

if (isEmpty(stack)) {

return -1;

}

return stack->items[stack->top];

}

int isOperator(char ch) {

return ch == '+' || ch == '-' || ch == '\*' || ch == '/';

}

int precedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '(':

case ')':

return 0;

}

return -1;

}

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

struct Stack stack;

initialize(&stack);

int i = 0, j = 0;

char ch;

while ((ch = infix[i++]) != '\0') {

if (isdigit(ch) || isalpha(ch)) {

postfix[j++] = ch;

} else if (ch == '(') {

push(&stack, ch);

} else if (ch == ')') {

while (!isEmpty(&stack) && peek(&stack) != '(') {

postfix[j++] = pop(&stack);

}

if (!isEmpty(&stack) && peek(&stack) != '(') {

printf("Invalid expression\n");

return;

} else {

pop(&stack);

}

} else if (isOperator(ch)) {

while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(ch)) {

postfix[j++] = pop(&stack);

}

push(&stack, ch);

}

}

while (!isEmpty(&stack)) {

postfix[j++] = pop(&stack);

}

postfix[j] = '\0';

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter infix expression: ");

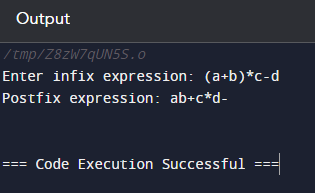
scanf("%s", infix);

infixToPostfix(infix, postfix);

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

return 0;

}



13.

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Queue {

int items[MAX];

int front;

int rear;

};

void initialize(struct Queue\* queue) {

queue->front = -1;

queue->rear = -1;

}

int isFull(struct Queue\* queue) {

return queue->rear == MAX - 1;

}

int isEmpty(struct Queue\* queue) {

return queue->front == -1 || queue->front > queue->rear;

}

void enqueue(struct Queue\* queue, int item) {

if (isFull(queue)) {

printf("Queue overflow! Cannot enqueue %d\n", item);

return;

}

if (isEmpty(queue)) {

queue->front = 0;

}

queue->items[++queue->rear] = item;

printf("%d enqueued to queue\n", item);

}

int dequeue(struct Queue\* queue) {

if (isEmpty(queue)) {

printf("Queue underflow! Cannot dequeue\n");

return -1;

}

int item = queue->items[queue->front++];

if (queue->front > queue->rear) {

initialize(queue); // Reset the queue if it becomes empty

}

return item;

}

void display(struct Queue\* queue) {

if (isEmpty(queue)) {

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

return;

}

printf("Queue elements: ");

for (int i = queue->front; i <= queue->rear; i++) {

printf("%d ", queue->items[i]);

}

printf("\n");

}

int main() {

struct Queue queue;

int choice, value;

initialize(&queue);

while (1) {

printf("\nMenu:\n");

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display queue\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(&queue, value);

break;

case 2:

value = dequeue(&queue);

if (value != -1) {

printf("Dequeued value: %d\n", value);

}

break;

case 3:

display(&queue);

break;

case 4:

exit(0);

default:

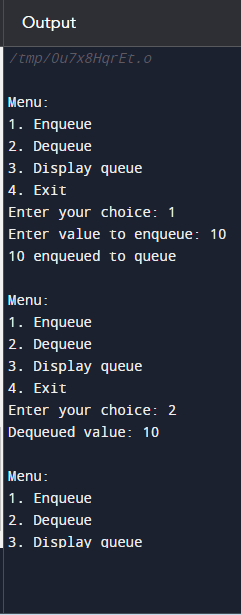
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



14.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

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

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void inorder(struct Node\* node) {

if (node == NULL) return;

inorder(node->left);

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

inorder(node->right);

}

void preorder(struct Node\* node) {

if (node == NULL) return;

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

preorder(node->left);

preorder(node->right);

}

void postorder(struct Node\* node) {

if (node == NULL) return;

postorder(node->left);

postorder(node->right);

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

}

struct Node\* insertNode(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

// Using a queue to perform level-order traversal

struct Node\*\* queue = (struct Node\*\*) malloc(sizeof(struct Node\*));

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

struct Node\* temp = queue[front++];

if (temp->left == NULL) {

temp->left = createNode(data);

free(queue);

return root;

} else {

queue = (struct Node\*\*) realloc(queue, (rear + 1) \* sizeof(struct Node\*));

if (queue == NULL) {

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

exit(EXIT\_FAILURE);

}

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

}

if (temp->right == NULL) {

temp->right = createNode(data);

free(queue);

return root;

} else {

queue = (struct Node\*\*) realloc(queue, (rear + 1) \* sizeof(struct Node\*));

if (queue == NULL) {

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

exit(EXIT\_FAILURE);

}

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

}

}

free(queue);

return root;

}

int main() {

struct Node\* root = NULL;

int choice, value;

while (1) {

printf("\nMenu:\n");

printf("1. Insert\n");

printf("2. Inorder traversal\n");

printf("3. Preorder traversal\n");

printf("4. Postorder traversal\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insertNode(root, value);

break;

case 2:

printf("Inorder traversal: ");

inorder(root);

printf("\n");

break;

case 3:

printf("Preorder traversal: ");

preorder(root);

printf("\n");

break;

case 4:

printf("Postorder traversal: ");

postorder(root);

printf("\n");

break;

case 5:

exit(EXIT\_SUCCESS);

default:

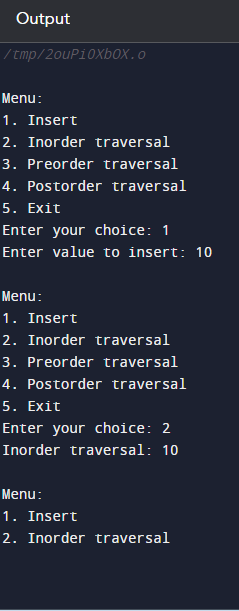
printf("Invalid choice! Please try again.\n");

}

}

return 0;

}



15.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

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

return newNode;

}

struct Node\* insertNode(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insertNode(root->left, data);

} else if (data > root->data) {

root->right = insertNode(root->right, data);

}

return root;

}

int search(struct Node\* root, int key) {

if (root == NULL) {

return 0;

}

if (root->data == key) {

return 1;

}

if (key < root->data) {

return search(root->left, key);

} else {

return search(root->right, key);

}

}

int findMin(struct Node\* root) {

struct Node\* current = root;

while (current && current->left != NULL) {

current = current->left;

}

return current->data;

}

int findMax(struct Node\* root) {

struct Node\* current = root;

while (current && current->right != NULL) {

current = current->right;

}

return current->data;

}

int main() {

struct Node\* root = NULL;

int choice, value;

while (1) {

printf("\n1. Insert a number\n");

printf("2. Search for a number\n");

printf("3. Find minimum value\n");

printf("4. Find maximum value\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the number to insert: ");

scanf("%d", &value);

root = insertNode(root, value);

printf("%d inserted.\n", value);

break;

case 2:

printf("Enter the number to search: ");

scanf("%d", &value);

if (search(root, value)) {

printf("%d is found in the BST.\n", value);

} else {

printf("%d is not found in the BST.\n", value);

}

break;

case 3:

if (root == NULL) {

printf("The BST is empty.\n");

} else {

printf("The minimum value in the BST is %d.\n", findMin(root));

}

break;

case 4:

if (root == NULL) {

printf("The BST is empty.\n");

} else {

printf("The maximum value in the BST is %d.\n", findMax(root));

}

break;

case 5:

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

exit(0);

break;

default:

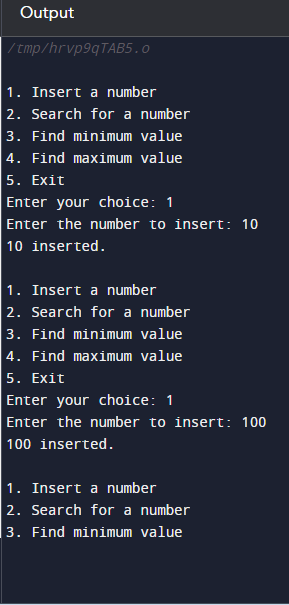
printf("Invalid choice. Please try again.\n");

}

}

return 0;

}



16.

#include <stdio.h>

#include <stdlib.h>

#define SIZE 10

int hashFunction(int key) {

return key % SIZE;

}

void insert(int hashTable[], int key) {

int index = hashFunction(key);

while (hashTable[index] != -1) {

index = (index + 1) % SIZE;

}

hashTable[index] = key;

printf("%d inserted at index %d\n", key, index);

}

int search(int hashTable[], int key) {

int index = hashFunction(key);

while (hashTable[index] != key) {

index = (index + 1) % SIZE;

if (index == hashFunction(key)) {

return -1;

}

}

return index;

}

void display(int hashTable[]) {

printf("\nHash Table:\n");

printf("Index\tKey\n");

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

printf("%d\t", i);

if (hashTable[i] != -1) {

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

}

printf("\n");

}

}

int main() {

int hashTable[SIZE];

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

hashTable[i] = -1;

}

int choice, key, index;

while (1) {

printf("\n1. Insert a key\n");

printf("2. Search for a key\n");

printf("3. Display hash table\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the key to insert: ");

scanf("%d", &key);

insert(hashTable, key);

break;

case 2:

printf("Enter the key to search: ");

scanf("%d", &key);

index = search(hashTable, key);

if (index == -1) {

printf("%d not found in the hash table.\n", key);

} else {

printf("%d found at index %d in the hash table.\n", key, index);

}

break;

case 3:

display(hashTable);

break;

case 4:

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

exit(0);

break;

default:

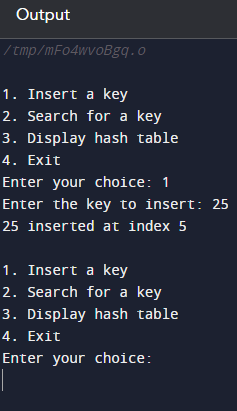
printf("Invalid choice. Please try again.\n");

}

}

return 0;

}



17.

#include <stdio.h>

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

int i, j, key;

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

key = arr[i];

j = i - 1;

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

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

j = j - 1;

}

arr[j + 1] = key;

}

}

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

printf("Sorted array: ");

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

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

}

printf("\n");

}

int main() {

int arr[100], n, i;

printf("Enter number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

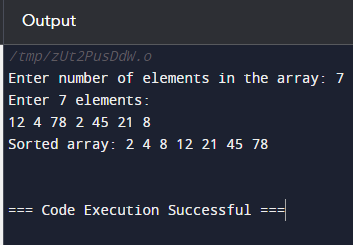
}

insertionSort(arr, n);

displayArray(arr, n);

return 0;

}



18.

#include <stdio.h>

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

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

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

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

printf("Sorted array: ");

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

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

}

printf("\n");

}

int main() {

int arr[100], n, i;

printf("Enter number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

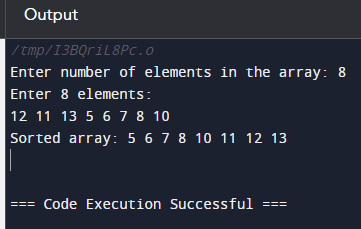
}

mergeSort(arr, 0, n - 1);

displayArray(arr, n);

return 0;

}



19.

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

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

printf("Sorted array: ");

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

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

}

printf("\n");

}

int main() {

int arr[100], n, i;

printf("Enter number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

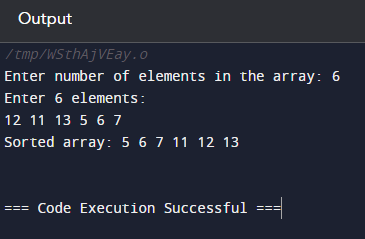
}

quickSort(arr, 0, n - 1);

displayArray(arr, n);

return 0;

}



20.

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int arr[], int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

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

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

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

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

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

printf("Sorted array: ");

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

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

}

printf("\n");

}

int main() {

int arr[100], n, i;

printf("Enter number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

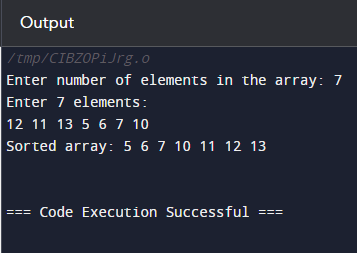
}

heapSort(arr, n);

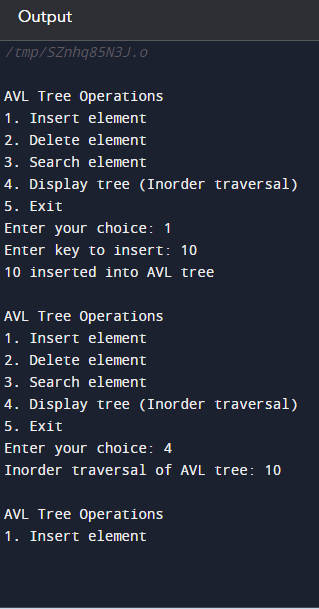
displayArray(arr, n);

return 0;

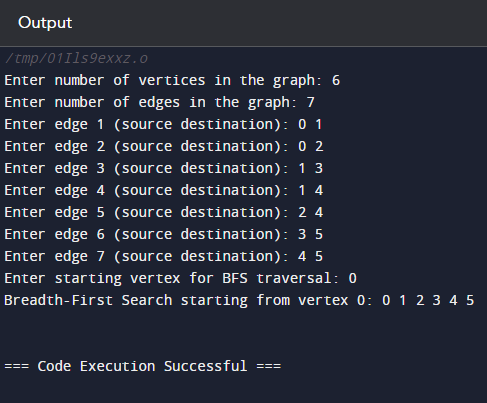
}



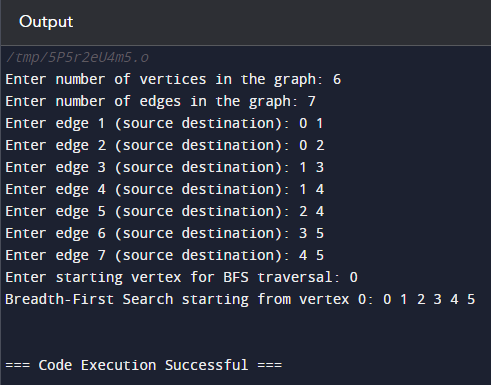
21.



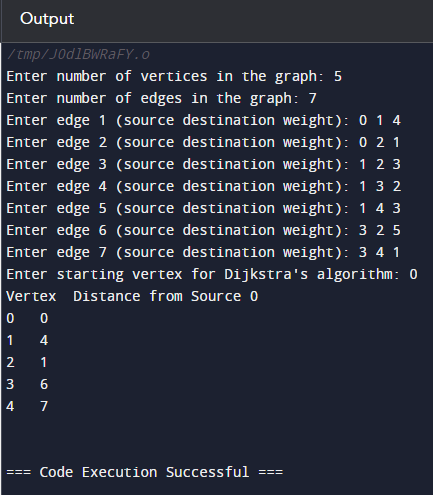
22.



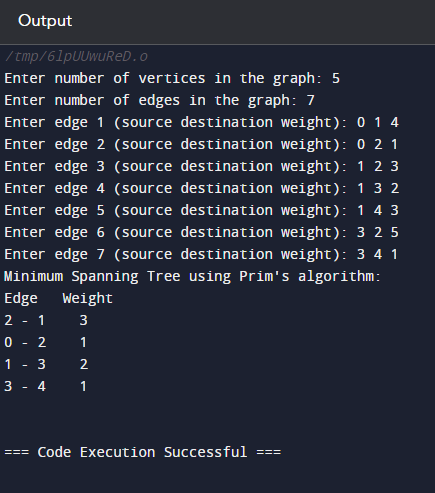
23.



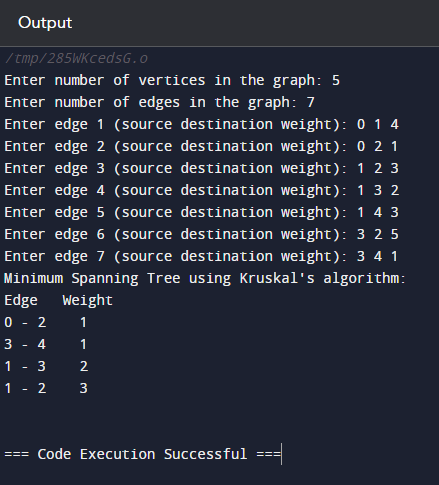
24.



25.



26.

.