CSA 0315-DATA STRUCTURES FOR OPTIMIZED MEMORY USAGE

Theveshrhaj S (192424120)

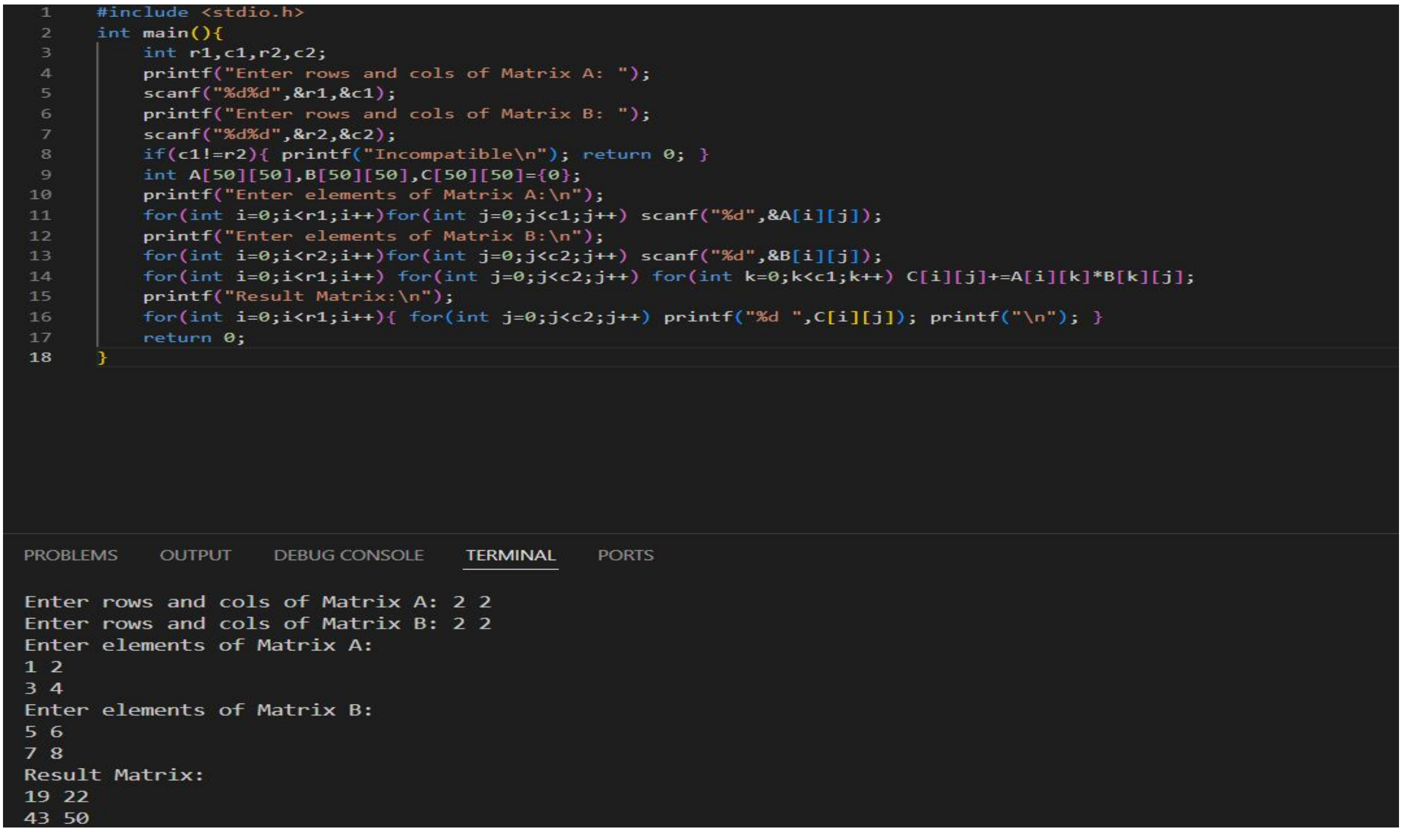
1. Write a C program to perform Matrix Multiplication

Aim: To multiply two matrices.

Algorithm:

1. Read order of two matrices.
2. If c1 != r2, multiplication not possible.
3. Read both matrices.
4. Multiply row × column and store in result matrix.
5. Display result.

Program:



```c
#include <stdio.h>
int main(){
    int r1,c1,r2,c2;
    printf("Enter rows and cols of Matrix A: ");
    scanf("%d%d",&r1,&c1);
    printf("Enter rows and cols of Matrix B: ");
    scanf("%d%d",&r2,&c2);
    if(c1!=r2){ printf("Incompatible\n"); return 0; }
    int A[50][50],B[50][50],C[50][50]={0};
    printf("Enter elements of Matrix A:\n");
    for(int i=0;i<r1;i++)for(int j=0;j<c1;j++) scanf("%d",&A[i][j]);
    printf("Enter elements of Matrix B:\n");
    for(int i=0;i<r2;i++)for(int j=0;j<c2;j++) scanf("%d",&B[i][j]);
    for(int i=0;i<r1;i++) for(int j=0;j<c2;j++) for(int k=0;k<c1;k++) C[i][j]+=A[i][k]*B[k][j];
    printf("Result Matrix:\n");
    for(int i=0;i<r1;i++){ for(int j=0;j<c2;j++) printf("%d ",C[i][j]); printf("\n"); }
    return 0;
}
```

```
Enter rows and cols of Matrix A: 2 2
Enter rows and cols of Matrix B: 2 2
Enter elements of Matrix A:
1 2
3 4
Enter elements of Matrix B:
5 6
7 8
Result Matrix:
19 22
43 50
```

Result: Successfully multiplies two matrices.

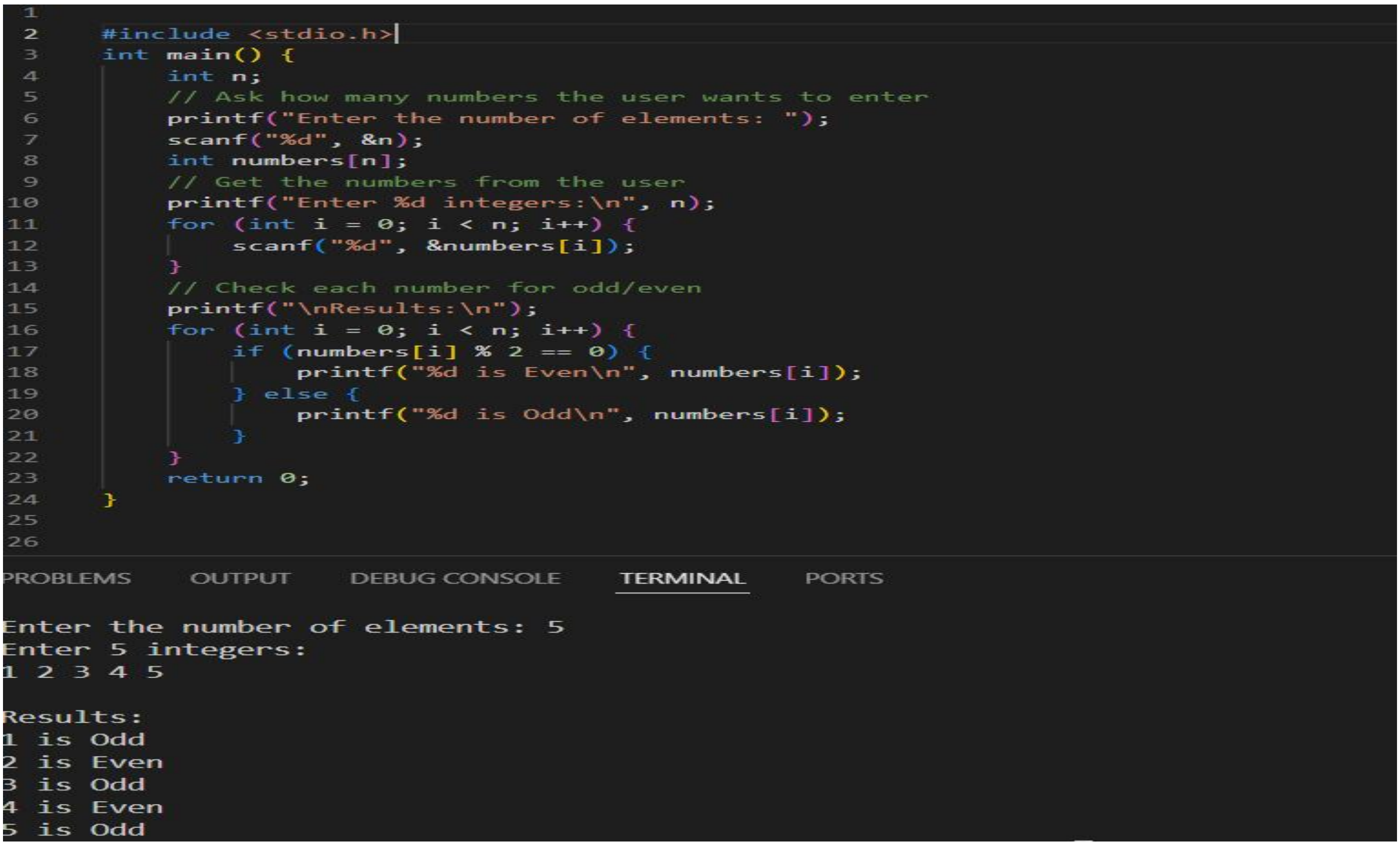
2. Write a C program to find Odd or Even number from a given set of numbers

Aim: To check odd/even for set of numbers.

Algorithm:

1. Read n from user.
2. For each number entered by user, check remainder by 2.
3. Print Odd/Even.

Program:



```c

#include <stdio.h>
int main() {
    int n;
    // Ask how many numbers the user wants to enter
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int numbers[n];
    // Get the numbers from the user
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &numbers[i]);
    }
    // Check each number for odd/even
    printf("\nResults:\n");
    for (int i = 0; i < n; i++) {
        if (numbers[i] % 2 == 0) {
            printf("%d is Even\n", numbers[i]);
        } else {
            printf("%d is Odd\n", numbers[i]);
        }
    }
    return 0;
}


```

```
Enter the number of elements: 5
Enter 5 integers:
1 2 3 4 5

Results:
1 is Odd
2 is Even
3 is Odd
4 is Even
5 is Odd
```

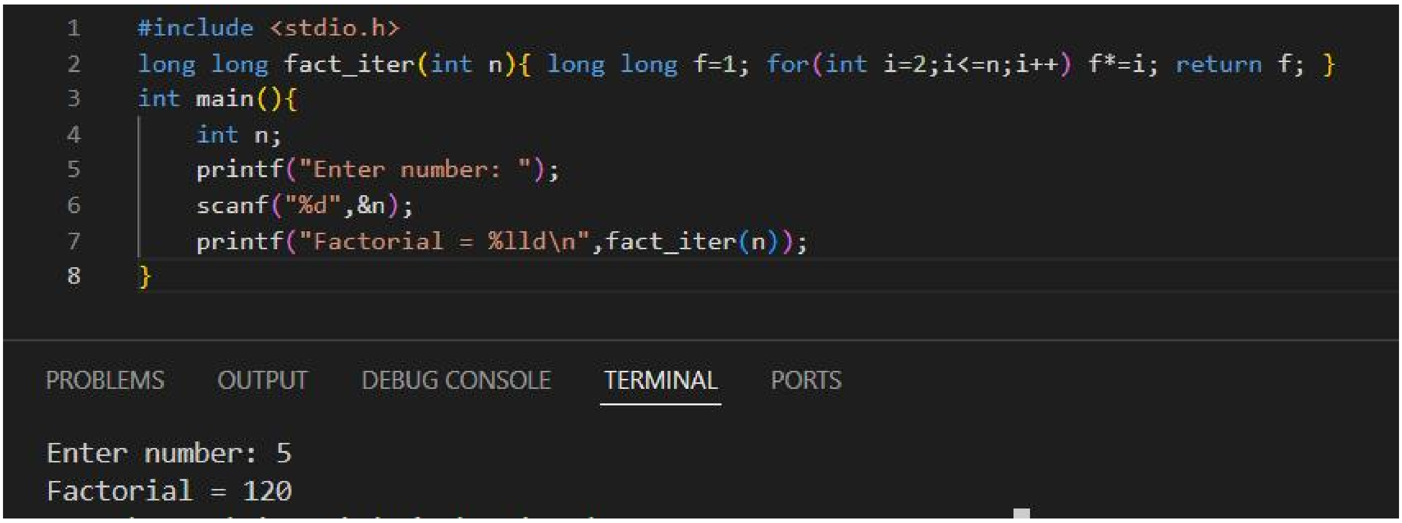
3.Write a C program to find Factorial of a given number without using Recursion

Aim: To find factorial without recursion.

Algorithm:

1. Read n from user.
2. Initialize fact=1.
3. Loop i=2 to n, fact*=i.
4. Display fact.

PROGRAM:



```c
#include <stdio.h>
long long fact_iter(int n){ long long f=1; for(int i=2;i<=n;i++) f*=i; return f; }
int main(){
    int n;
    printf("Enter number: ");
    scanf("%d",&n);
    printf("Factorial = %lld\n",fact_iter(n));
}
```

```
Enter number: 5
Factorial = 120
```

Result: Factorial displayed.

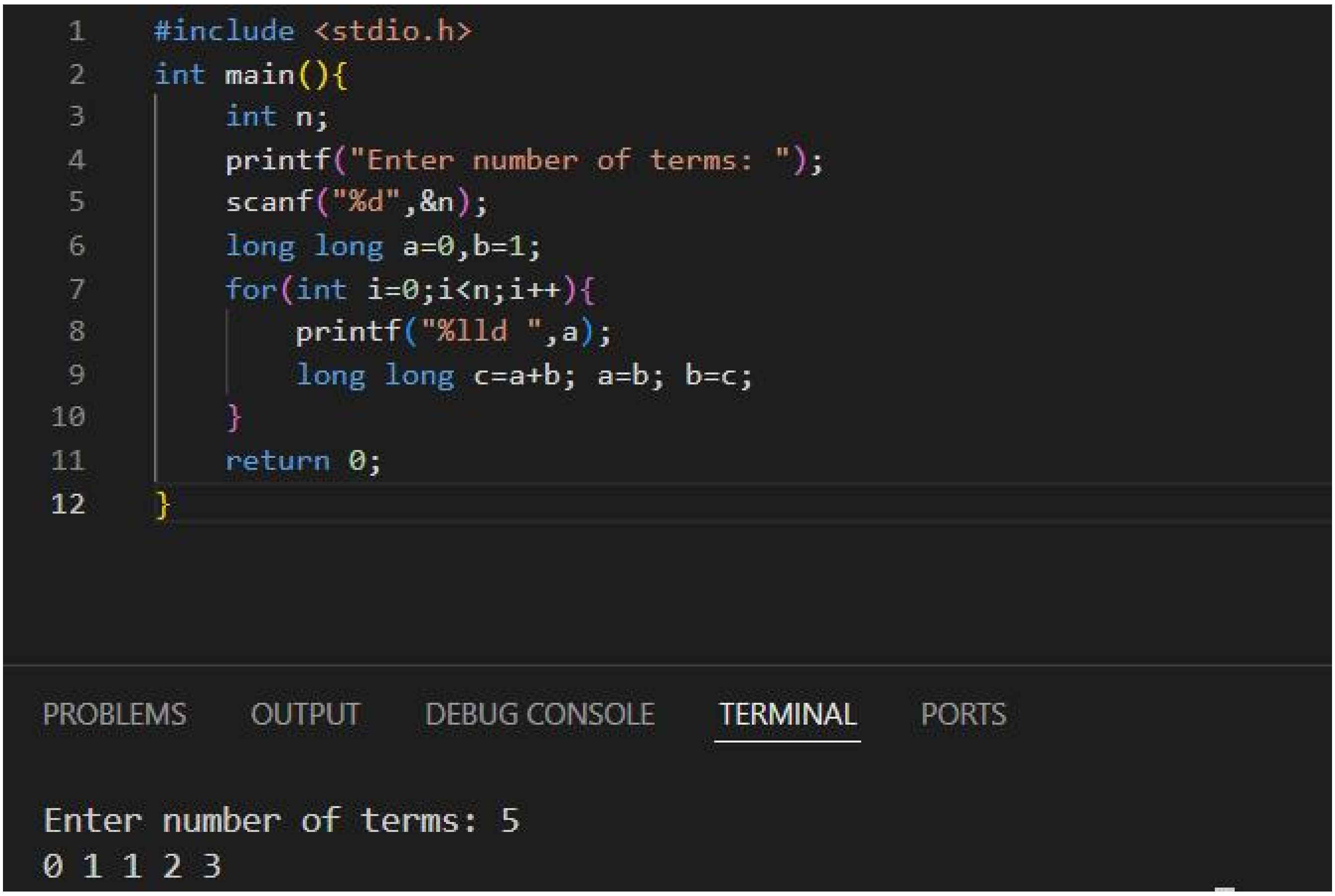
4.Write a C program to find Fibonacci series without using Recursion

Aim: To generate Fibonacci series without recursion.

Algorithm:

1. Read n from user.
2. Initialize a=0, b=1.
3. Loop n times, print a, update a=b, b=a+b.

Program:



```c
#include <stdio.h>
int main(){
    int n;
    printf("Enter number of terms: ");
    scanf("%d",&n);
    long long a=0,b=1;
    for(int i=0;i<n;i++){
        printf("%lld ",a);
        long long c=a+b; a=b; b=c;
    }
    return 0;
}
```

```
Enter number of terms: 5
0 1 1 2 3
```

Result: Fibonacci series printed

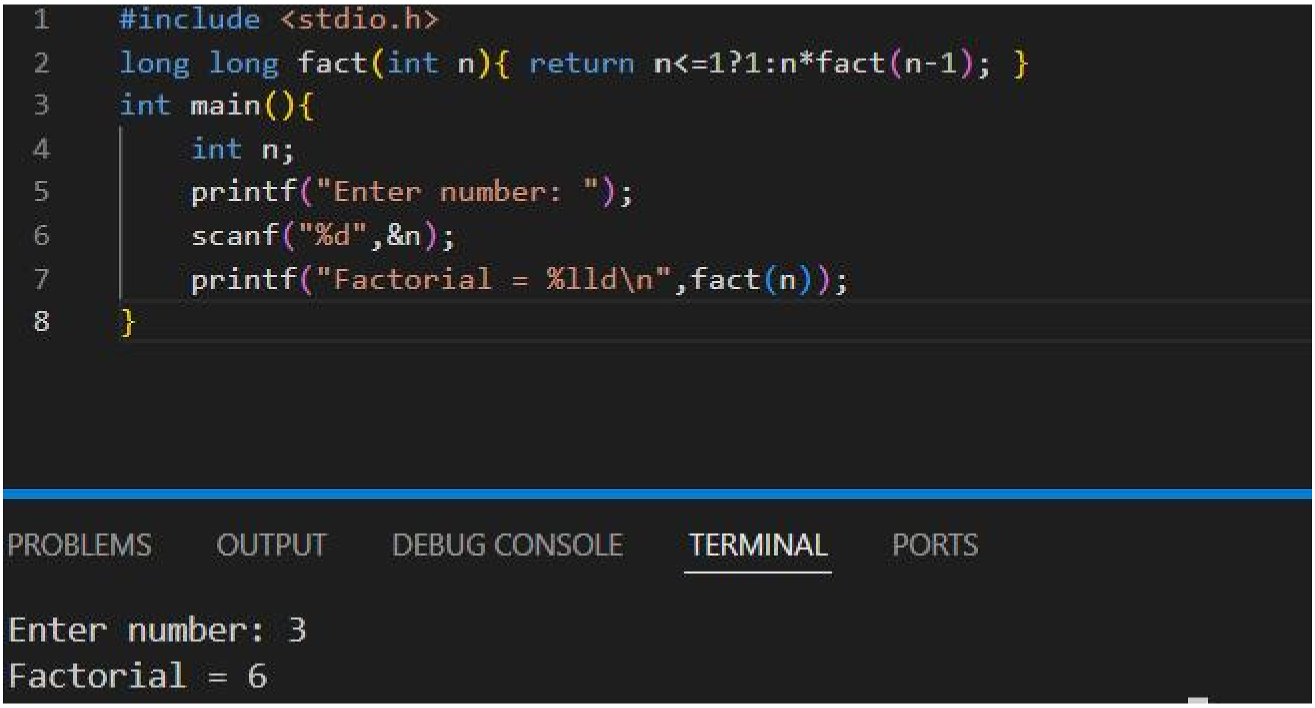
5. Write a C program to find Factorial of a given number using Recursion

Aim: To find factorial using recursion.

Algorithm:

1. Read n from user.
2. Define recursive function fact(n).
3. Base case: n<=1 return 1.
4. Recursive case: return n*fact(n-1).
5. Print result.

Program:



```c
#include <stdio.h>
long long fact(int n){ return n<=1?1:n*fact(n-1); }
int main(){
    int n;
    printf("Enter number: ");
    scanf("%d",&n);
    printf("Factorial = %lld\n",fact(n));
}
```

```
Enter number: 3
Factorial = 6
```

Result: Factorial displayed.

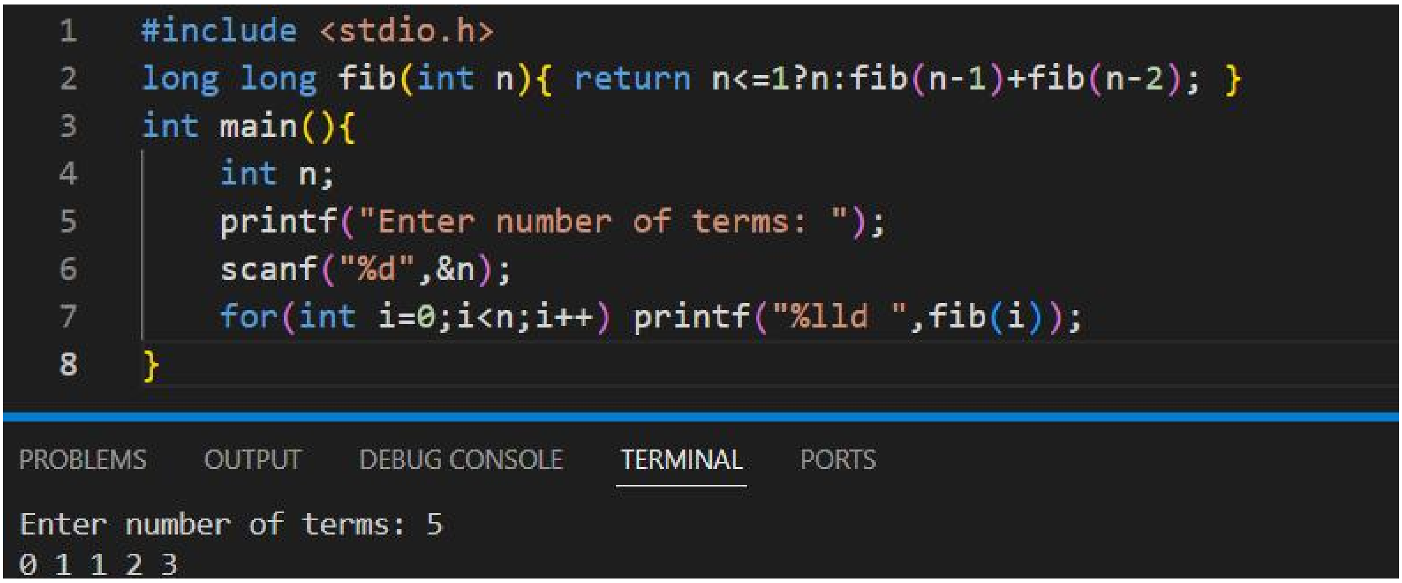
6. Write a C program to find Fibonacci series using Recursion

Aim: To generate Fibonacci series using recursion.

Algorithm:

1. Read n from user.
2. Define fib(n).
3. If n<=1 return n.
4. Else return fib(n-1)+fib(n-2).
5. Print fib(i) for i=0..n-1.

Program:



```c
#include <stdio.h>
long long fib(int n){ return n<=1?n:fib(n-1)+fib(n-2); }
int main(){
    int n;
    printf("Enter number of terms: ");
    scanf("%d",&n);
    for(int i=0;i<n;i++) printf("%lld ",fib(i));
}
```

```
Enter number of terms: 5
0 1 1 2 3
```

Result: Fibonacci printed.

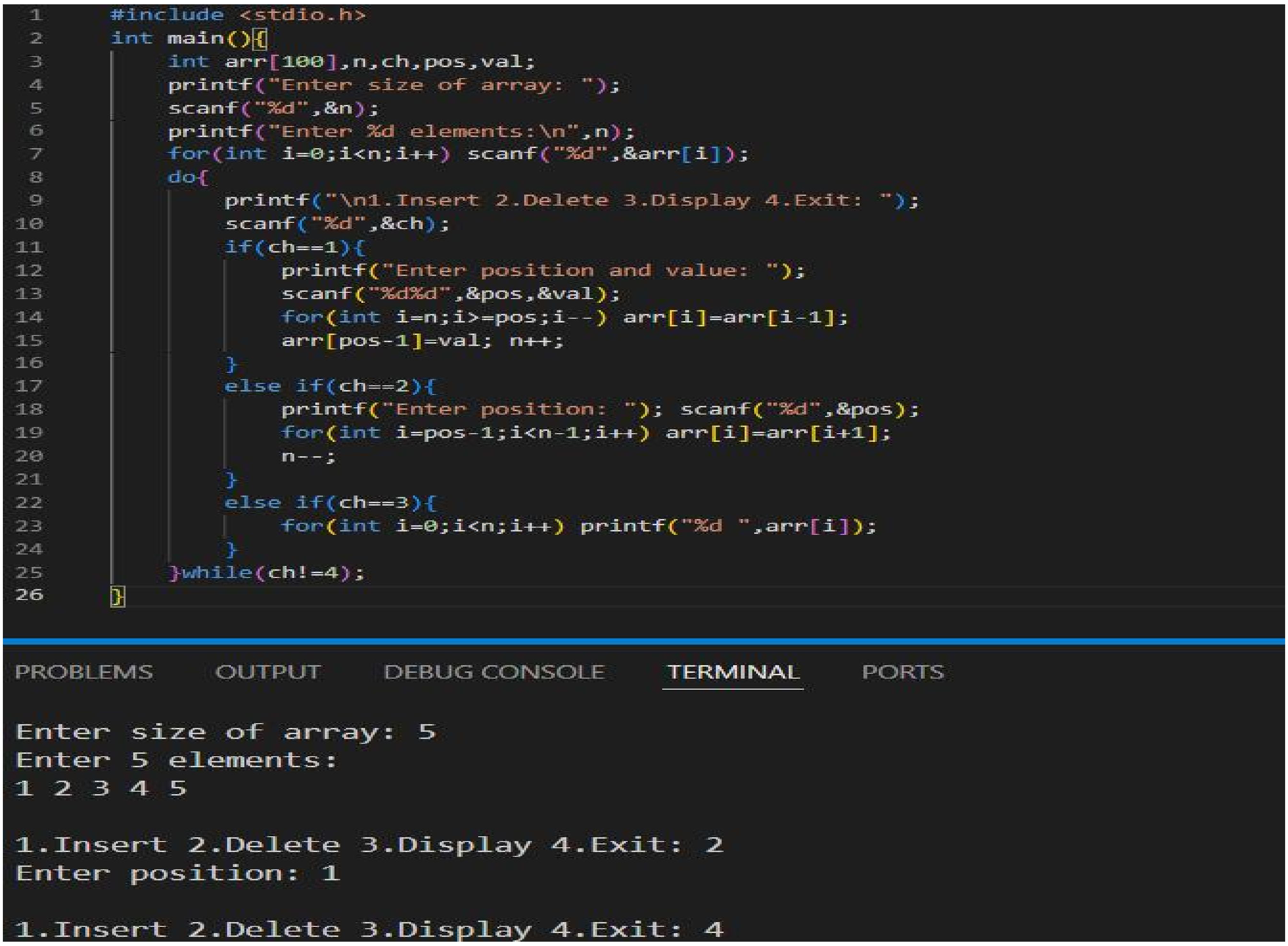
7. Write a C program to implement Array operations such as Insert, Delete and Display

Aim: To implement insert, delete, display operations on array.

Algorithm:

1. Read n and array elements.
2. Perform menu-driven operations (insert, delete, display).
3. Update array accordingly.

Program:



```c
#include <stdio.h>
int main(){
    int arr[100],n,ch,pos,val;
    printf("Enter size of array: ");
    scanf("%d",&n);
    printf("Enter %d elements:\n",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    do{
        printf("\n1.Insert 2.Delete 3.Display 4.Exit: ");
        scanf("%d",&ch);
        if(ch==1){
            printf("Enter position and value: ");
            scanf("%d%d",&pos,&val);
            for(int i=n;i>=pos;i--) arr[i]=arr[i-1];
            arr[pos-1]=val; n++;
        }
        else if(ch==2){
            printf("Enter position: "); scanf("%d",&pos);
            for(int i=pos-1;i<n-1;i++) arr[i]=arr[i+1];
            n--;
        }
        else if(ch==3){
            for(int i=0;i<n;i++) printf("%d ",arr[i]);
        }
    }while(ch!=4);
}
```

```
Enter size of array: 5
Enter 5 elements:
1 2 3 4 5

1.Insert 2.Delete 3.Display 4.Exit: 2
Enter position: 1

1.Insert 2.Delete 3.Display 4.Exit: 4
```

Result: Array operations performed.

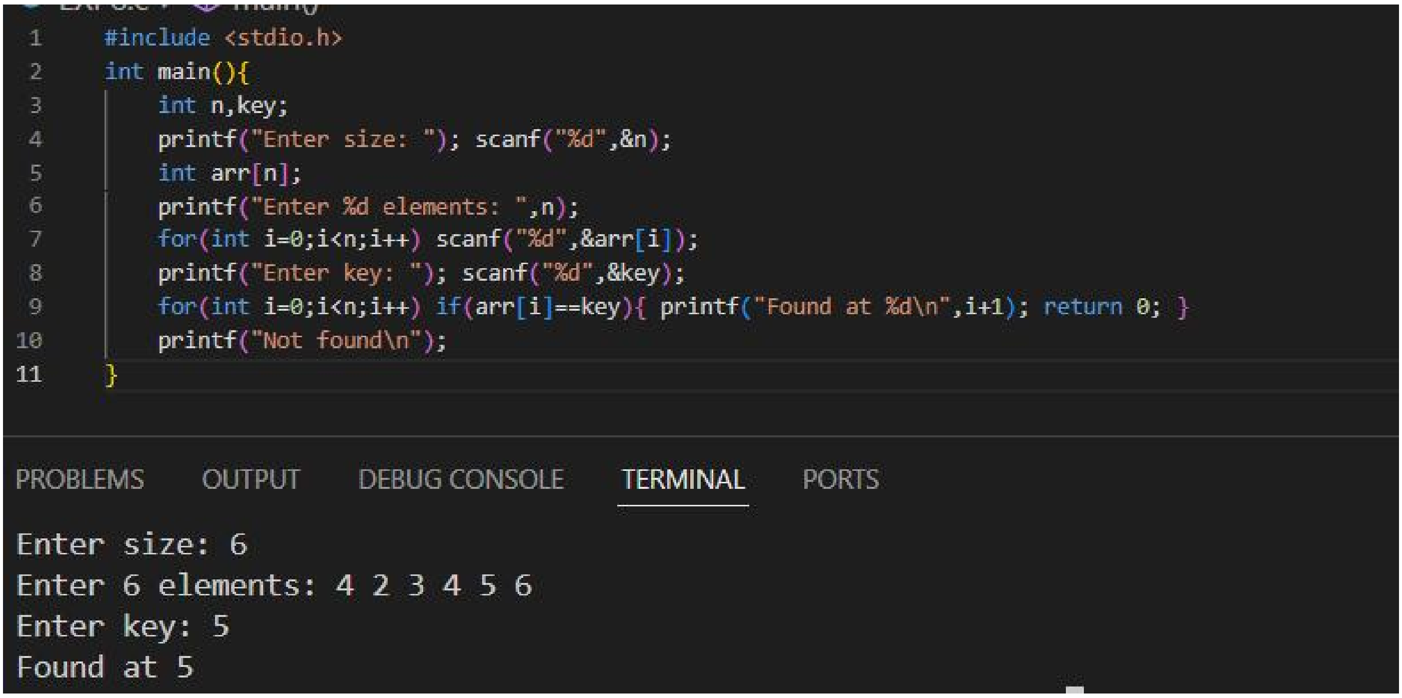
8. Write a C program to search a number using Linear Search method

Aim: To search an element using linear search.

Algorithm:

1. Read n and array.
2. Read key.
3. Traverse array, check if key matches.
4. Display result.

Program:



```c
#include <stdio.h>
int main(){
    int n,key;
    printf("Enter size: "); scanf("%d",&n);
    int arr[n];
    printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    printf("Enter key: "); scanf("%d",&key);
    for(int i=0;i<n;i++) if(arr[i]==key){ printf("Found at %d\n",i+1); return 0; }
    printf("Not found\n");
}
```

```
Enter size: 6
Enter 6 elements: 4 2 3 4 5 6
Enter key: 5
Found at 5
```

Result: Key found or not found displayed.

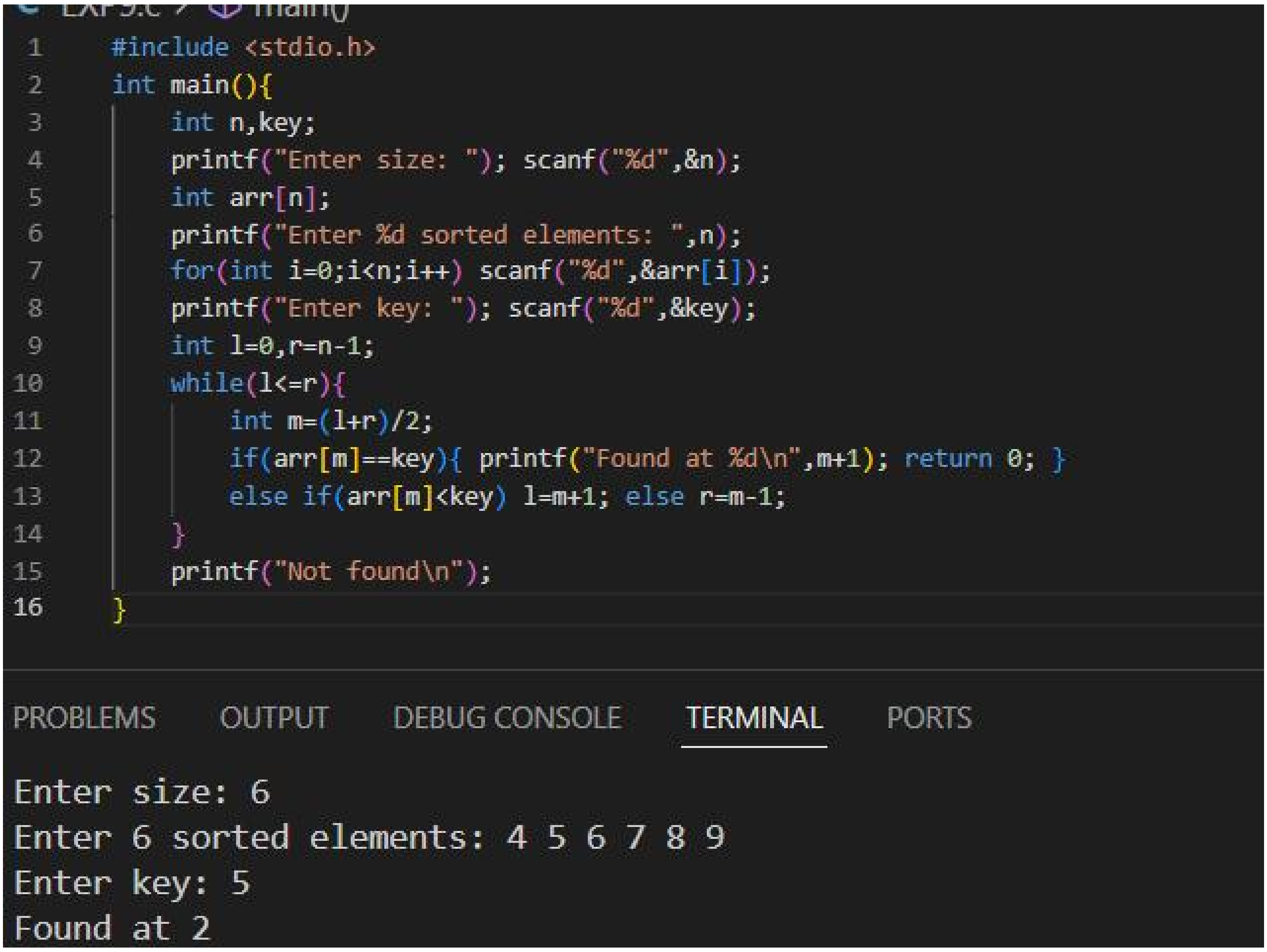
9. Write a C program to search a number using Binary Search method

Aim: To search an element using binary search.

Algorithm:

1. Read n and sorted array.
2. Read key.
3. Apply binary search mid=l+(r-l)/2.
4. Print position or not found.

Program:



```c
#include <stdio.h>
int main(){
    int n,key;
    printf("Enter size: "); scanf("%d",&n);
    int arr[n];
    printf("Enter %d sorted elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    printf("Enter key: "); scanf("%d",&key);
    int l=0,r=n-1;
    while(l<=r){
        int m=(l+r)/2;
        if(arr[m]==key){ printf("Found at %d\n",m+1); return 0; }
        else if(arr[m]<key) l=m+1; else r=m-1;
    }
    printf("Not found\n");
}
```

```
Enter size: 6
Enter 6 sorted elements: 4 5 6 7 8 9
Enter key: 5
Found at 2
```

Result: Displays position if found.

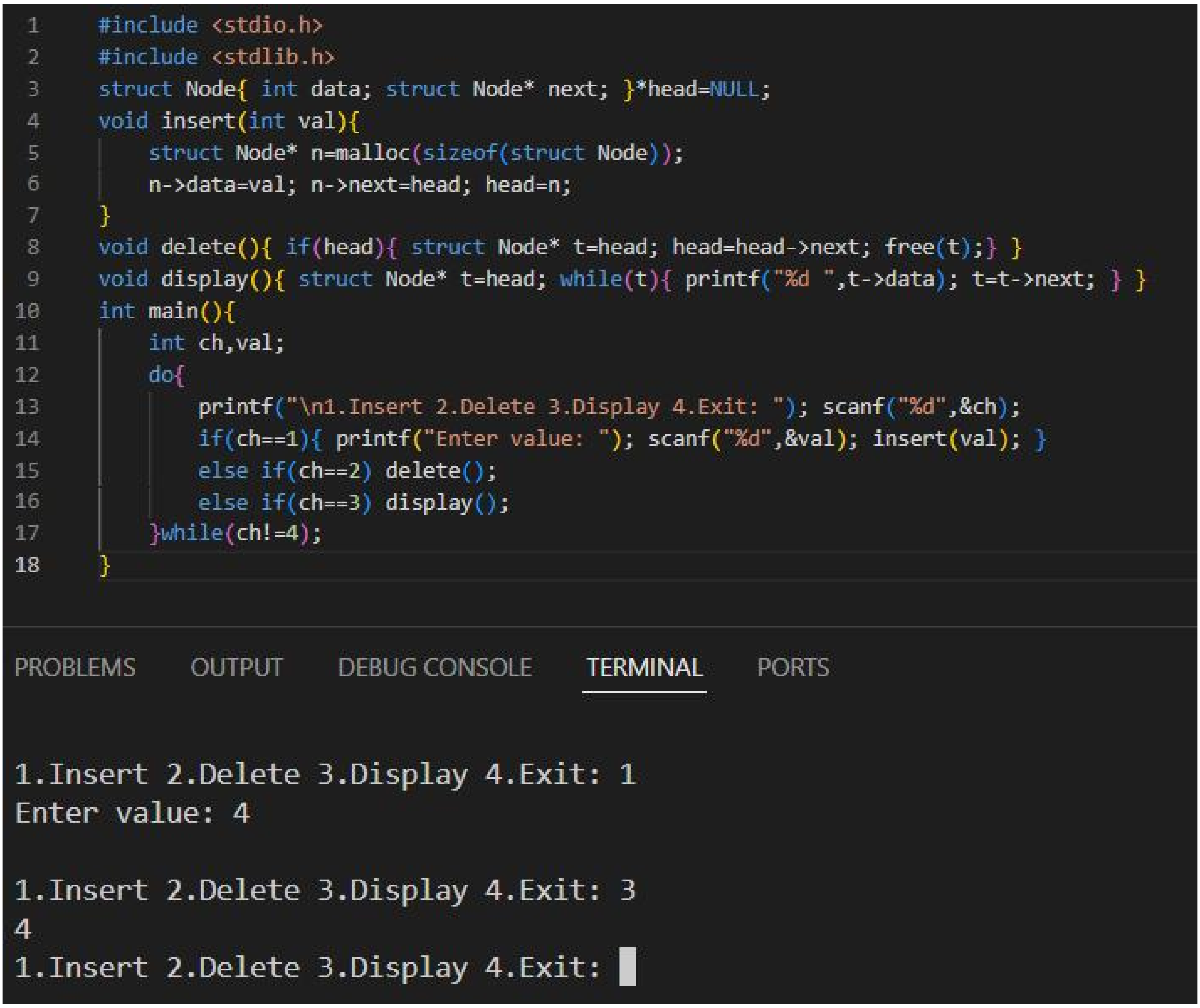
10. Write a C program to implement Linked list operations

Aim: To implement basic linked list operations.

Algorithm:

1. Use structure node {data,next}.
2. Menu-driven: insert, delete, display.
3. Update head pointer accordingly.

Program:



```c
#include <stdio.h>
#include <stdlib.h>
struct Node{ int data; struct Node* next; }*head=NULL;
void insert(int val){
    struct Node* n=malloc(sizeof(struct Node));
    n->data=val; n->next=head; head=n;
}
void delete(){ if(head){ struct Node* t=head; head=head->next; free(t);} }
void display(){ struct Node* t=head; while(t){ printf("%d ",t->data); t=t->next; } }
int main(){
    int ch,val;
    do{
        printf("\n1.Insert 2.Delete 3.Display 4.Exit: "); scanf("%d",&ch);
        if(ch==1){ printf("Enter value: "); scanf("%d",&val); insert(val); }
        else if(ch==2) delete();
        else if(ch==3) display();
    }while(ch!=4);
}
```

```
1.Insert 2.Delete 3.Display 4.Exit: 1
Enter value: 4

1.Insert 2.Delete 3.Display 4.Exit: 3
4
1.Insert 2.Delete 3.Display 4.Exit: 
```

Result: Linked list operations done.

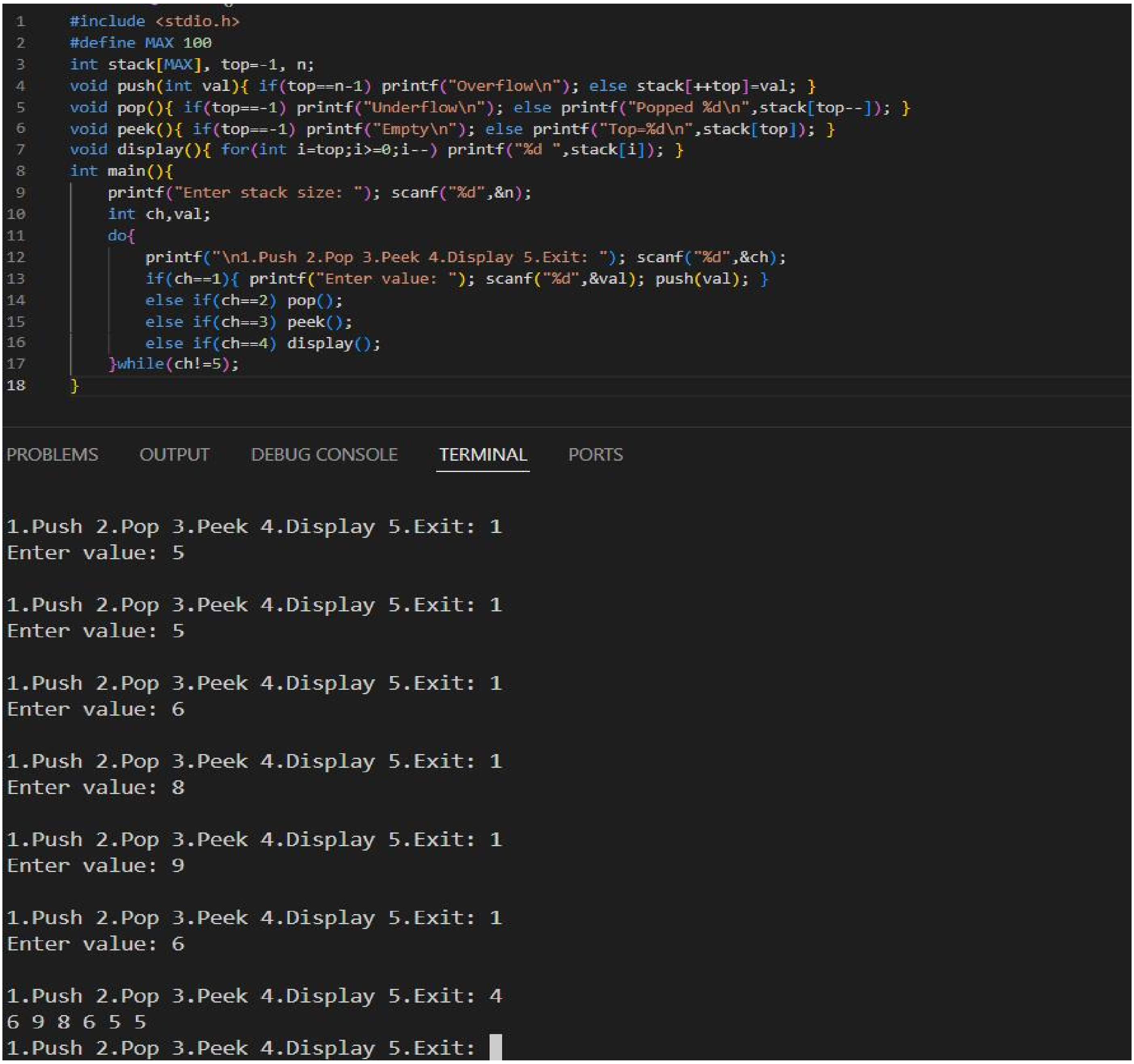
11. Write a C program to implement Stack operations such as PUSH, POP and PEEK

Aim: To implement stack operations (PUSH, POP, PEEK).

Algorithm:

1. Read stack size from user.
2. Implement push, pop, peek functions.
3. Use menu-driven approach for operations.

Program:



```c
#include <stdio.h>
#define MAX 100
int stack[MAX], top=-1, n;
void push(int val){ if(top==n-1) printf("Overflow\n"); else stack[++top]=val; }
void pop(){ if(top==-1) printf("Underflow\n"); else printf("Popped %d\n",stack[top--]); }
void peek(){ if(top==-1) printf("Empty\n"); else printf("Top=%d\n",stack[top]); }
void display(){ for(int i=top;i>=0;i--) printf("%d ",stack[i]); }
int main(){
    printf("Enter stack size: "); scanf("%d",&n);
    int ch,val;
    do{
        printf("\n1.Push 2.Pop 3.Peek 4.Display 5.Exit: "); scanf("%d",&ch);
        if(ch==1){ printf("Enter value: "); scanf("%d",&val); push(val); }
        else if(ch==2) pop();
        else if(ch==3) peek();
        else if(ch==4) display();
    }while(ch!=5);
}
```

```
1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 5

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 5

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 6

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 8

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 9

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 1
Enter value: 6

1.Push 2.Pop 3.Peek 4.Display 5.Exit: 4
6 9 8 6 5 5
1.Push 2.Pop 3.Peek 4.Display 5.Exit: 
```

Result: Stack operations executed.

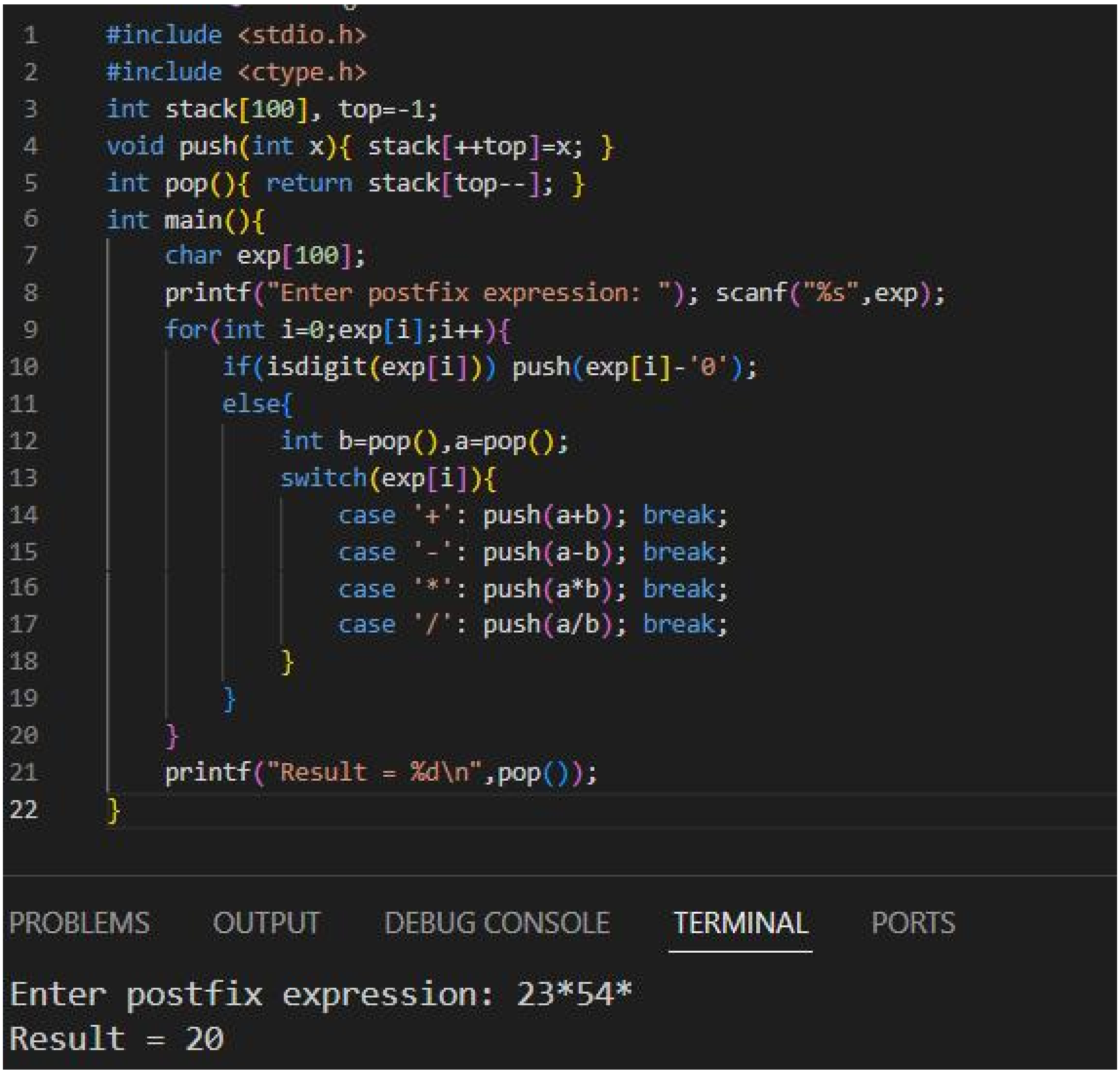
12. Write a C program to implement the application of Stack (Notations)

Aim: To evaluate postfix expression using stack.

Algorithm:

1. Read postfix expression.
2. Traverse char by char.
3. If operand, push.
4. If operator, pop two, evaluate, push result.
5. Final stack top is result.

Program:



```c
#include <stdio.h>
#include <ctype.h>
int stack[100], top=-1;
void push(int x){ stack[++top]=x; }
int pop(){ return stack[top--]; }
int main(){
    char exp[100];
    printf("Enter postfix expression: "); scanf("%s",exp);
    for(int i=0;exp[i];i++){
        if(isdigit(exp[i])) push(exp[i]-'0');
        else{
            int b=pop(),a=pop();
            switch(exp[i]){
                case '+': push(a+b); break;
                case '-': push(a-b); break;
                case '*': push(a*b); break;
                case '/': push(a/b); break;
            }
        }
    }
    printf("Result = %d\n",pop());
}
```

```
Enter postfix expression: 23*54*
Result = 20
```

Result: Postfix evaluated.

13. Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE

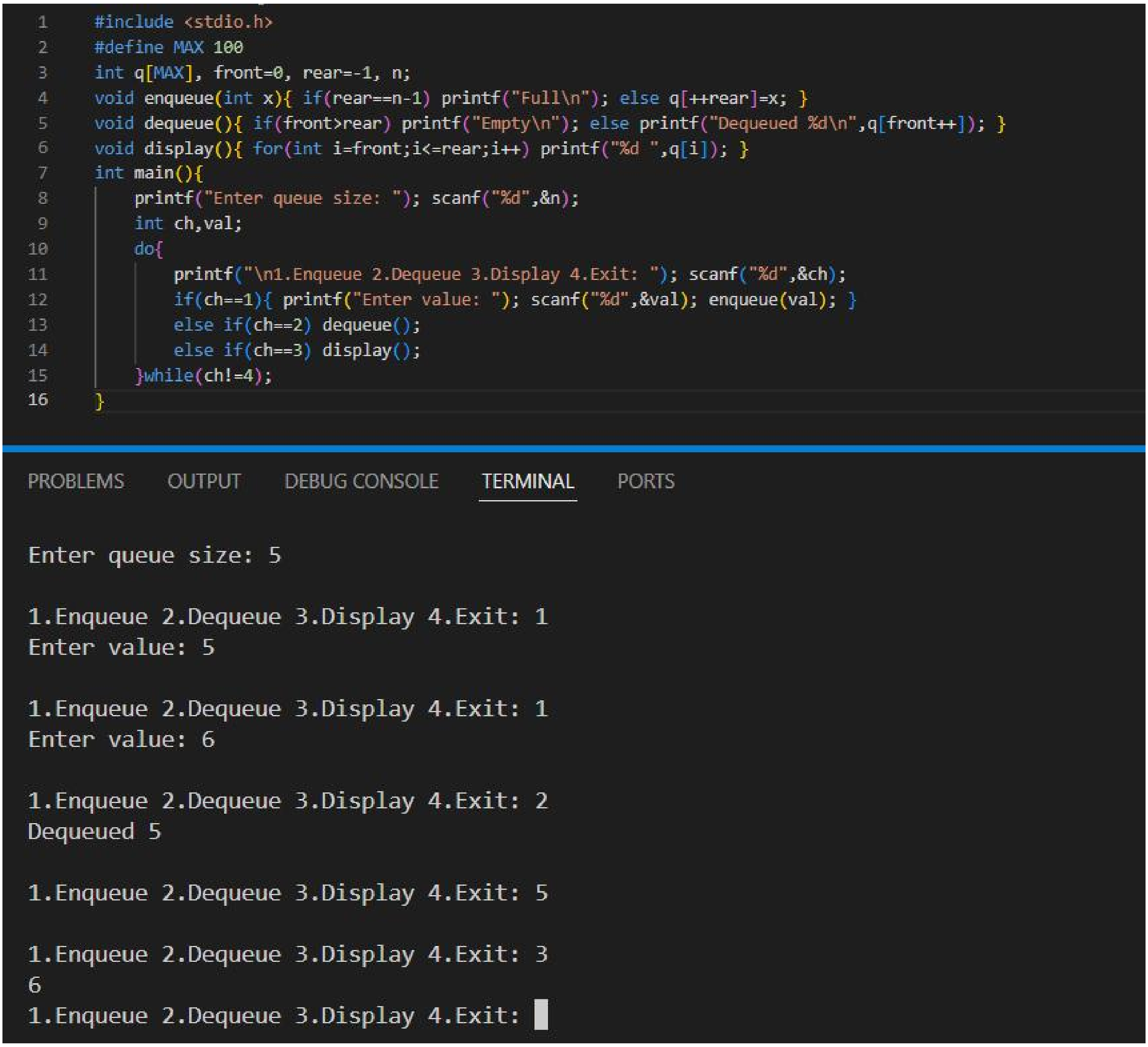
and Display

Aim: To implement queue operations.

Algorithm:

1. Use array with front, rear.
2. Menu: enqueue, dequeue, display.

Program:



```c
#include <stdio.h>
#define MAX 100
int q[MAX], front=0, rear=-1, n;
void enqueue(int x){ if(rear==n-1) printf("Full\n"); else q[++rear]=x; }
void dequeue(){ if(front>rear) printf("Empty\n"); else printf("Dequeued %d\n",q[front++]); }
void display(){ for(int i=front;i<=rear;i++) printf("%d ",q[i]); }
int main(){
    printf("Enter queue size: "); scanf("%d",&n);
    int ch,val;
    do{
        printf("\n1.Enqueue 2.Dequeue 3.Display 4.Exit: "); scanf("%d",&ch);
        if(ch==1){ printf("Enter value: "); scanf("%d",&val); enqueue(val); }
        else if(ch==2) dequeue();
        else if(ch==3) display();
    }while(ch!=4);
}
```

```
Enter queue size: 5

1.Enqueue 2.Dequeue 3.Display 4.Exit: 1
Enter value: 5

1.Enqueue 2.Dequeue 3.Display 4.Exit: 1
Enter value: 6

1.Enqueue 2.Dequeue 3.Display 4.Exit: 2
Dequeued 5

1.Enqueue 2.Dequeue 3.Display 4.Exit: 5

1.Enqueue 2.Dequeue 3.Display 4.Exit: 3
6
1.Enqueue 2.Dequeue 3.Display 4.Exit: 
```

Result: Queue operations performed.

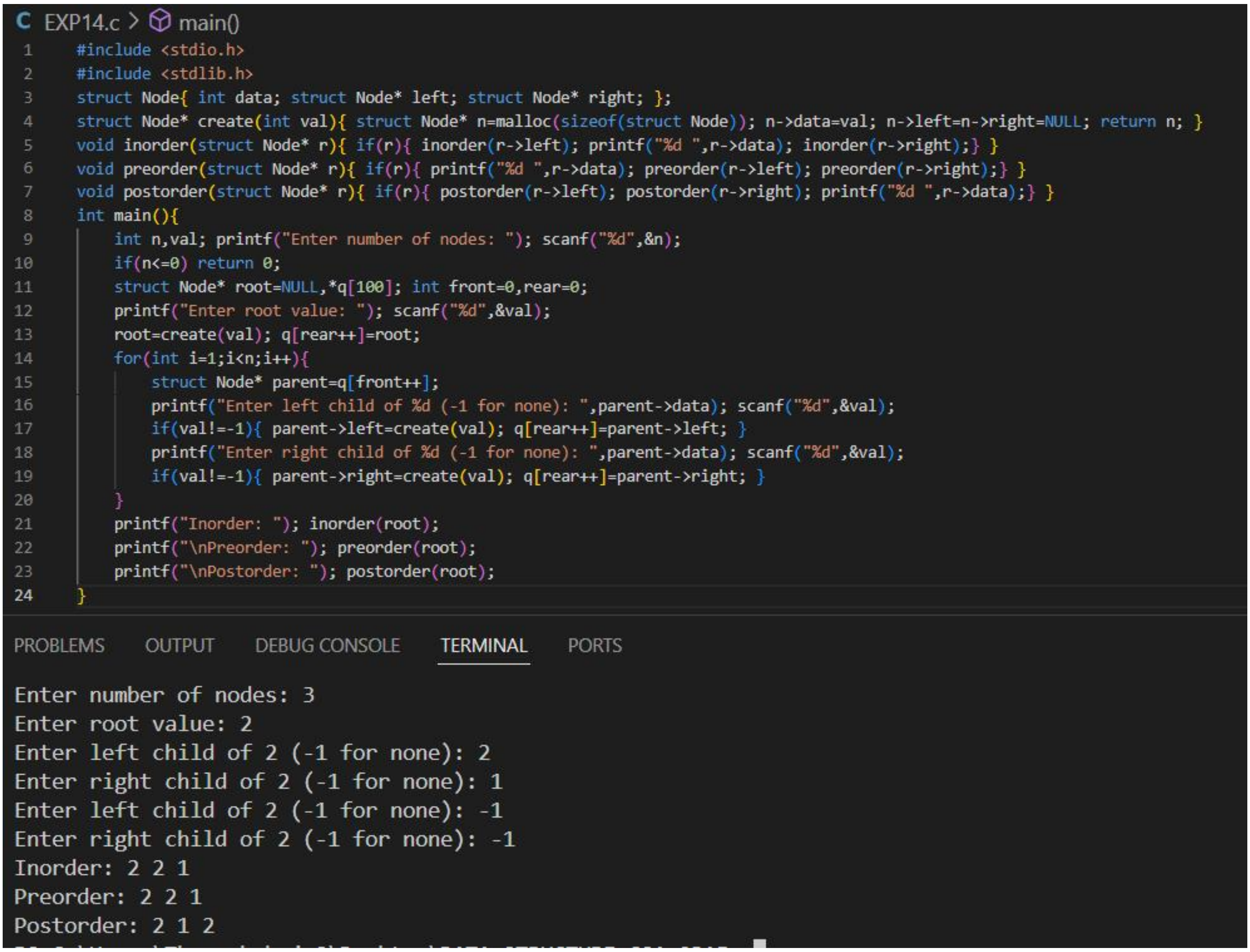
14. Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder)

Aim: To implement inorder, preorder, postorder traversals.

Algorithm:

1. Create binary tree.
2. Perform recursive traversals.

Program:



```c
#include <stdio.h>
#include <stdlib.h>
struct Node{ int data; struct Node* left; struct Node* right; };
struct Node* create(int val){ struct Node* n=malloc(sizeof(struct Node)); n->data=val; n->left=n->right=NULL; return n; }
void inorder(struct Node* r){ if(r){ inorder(r->left); printf("%d ",r->data); inorder(r->right);} }
void preorder(struct Node* r){ if(r){ printf("%d ",r->data); preorder(r->left); preorder(r->right);} }
void postorder(struct Node* r){ if(r){ postorder(r->left); postorder(r->right); printf("%d ",r->data);} }
int main(){
    int n,val; printf("Enter number of nodes: "); scanf("%d",&n);
    if(n<=0) return 0;
    struct Node* root=NULL,*q[100]; int front=0,rear=0;
    printf("Enter root value: "); scanf("%d",&val);
    root=create(val); q[rear++]=root;
    for(int i=1;i<n;i++){
        struct Node* parent=q[front++];
        printf("Enter left child of %d (-1 for none): ",parent->data); scanf("%d",&val);
        if(val!=-1){ parent->left=create(val); q[rear++]=parent->left; }
        printf("Enter right child of %d (-1 for none): ",parent->data); scanf("%d",&val);
        if(val!=-1){ parent->right=create(val); q[rear++]=parent->right; }
    }
    printf("Inorder: "); inorder(root);
    printf("\nPreorder: "); preorder(root);
    printf("\nPostorder: "); postorder(root);
}
```

```
Enter number of nodes: 3
Enter root value: 2
Enter left child of 2 (-1 for none): 2
Enter right child of 2 (-1 for none): 1
Enter left child of 2 (-1 for none): -1
Enter right child of 2 (-1 for none): -1
Inorder: 2 2 1
Preorder: 2 2 1
Postorder: 2 1 2
```

Result: Tree traversals displayed.

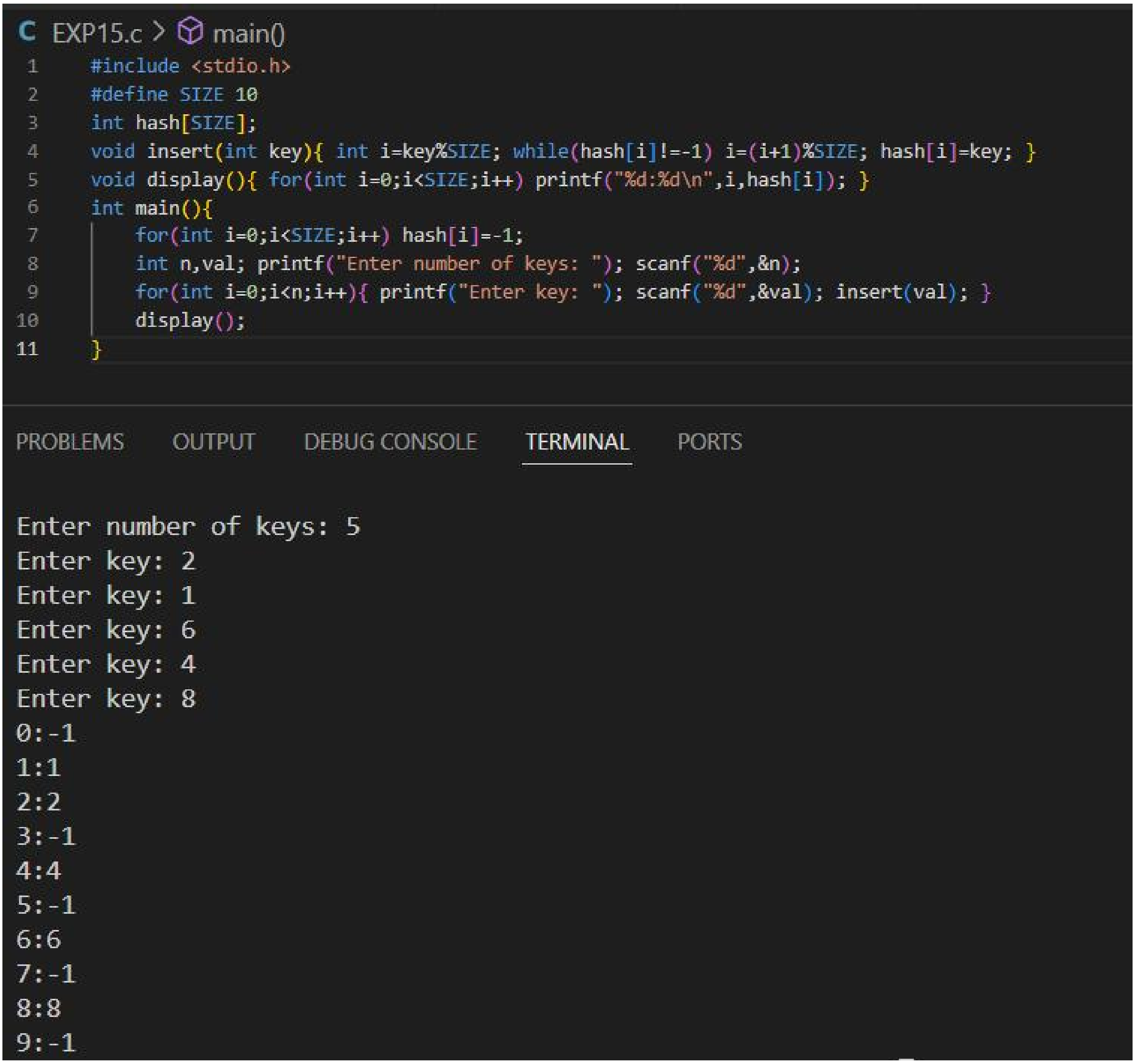
15. Write a C program to implement hashing using Linear Probing method

Aim: To implement hashing using linear probing.

Algorithm:

1. Read table size and elements.
2. Compute index=key%size.
3. If occupied, probe next.

Program:



```c
#include <stdio.h>
#define SIZE 10
int hash[SIZE];
void insert(int key){ int i=key%SIZE; while(hash[i]!=-1) i=(i+1)%SIZE; hash[i]=key; }
void display(){ for(int i=0;i<SIZE;i++) printf("%d:%d\n",i,hash[i]); }
int main(){
    for(int i=0;i<SIZE;i++) hash[i]=-1;
    int n,val; printf("Enter number of keys: "); scanf("%d",&n);
    for(int i=0;i<n;i++){ printf("Enter key: "); scanf("%d",&val); insert(val); }
    display();
}
```

```
Enter number of keys: 5
Enter key: 2
Enter key: 1
Enter key: 6
Enter key: 4
Enter key: 8
0:-1
1:1
2:2
3:-1
4:4
5:-1
6:6
7:-1
8:8
9:-1
```

Result: Hash table displayed.

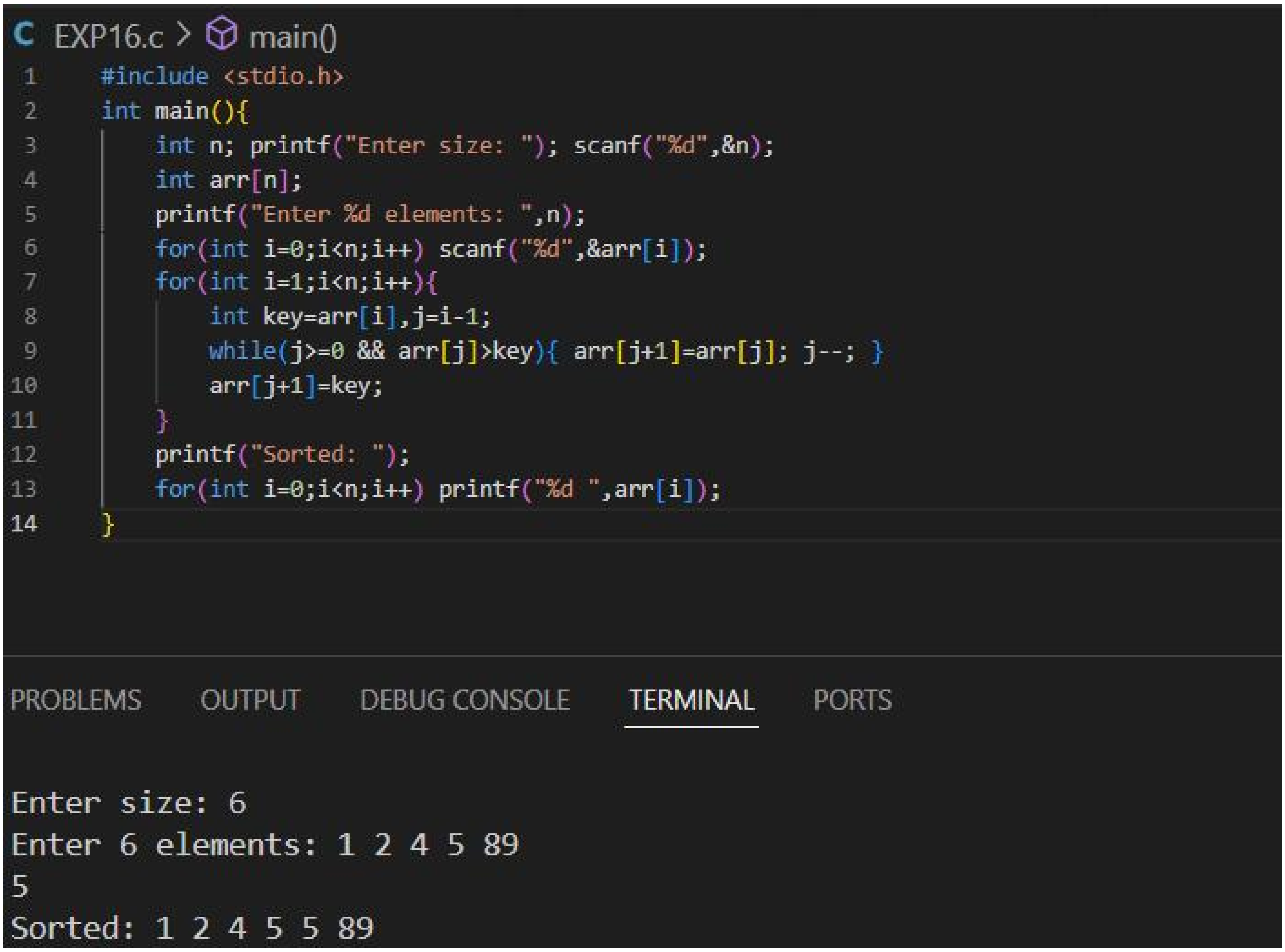
16. Write a C program to arrange a series of numbers using Insertion Sort

Aim: To sort array using insertion sort.

Algorithm:

1. Read n, array.
2. For i=1 to n-1, insert element in sorted part.
3. Print sorted array.

Program:



```c
#include <stdio.h>
int main(){
    int n; printf("Enter size: "); scanf("%d",&n);
    int arr[n];
    printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    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;
    }
    printf("Sorted: ");
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
}
```

```
Enter size: 6
Enter 6 elements: 1 2 4 5 89
5
Sorted: 1 2 4 5 5 89
```

Result: Sorted array displayed.

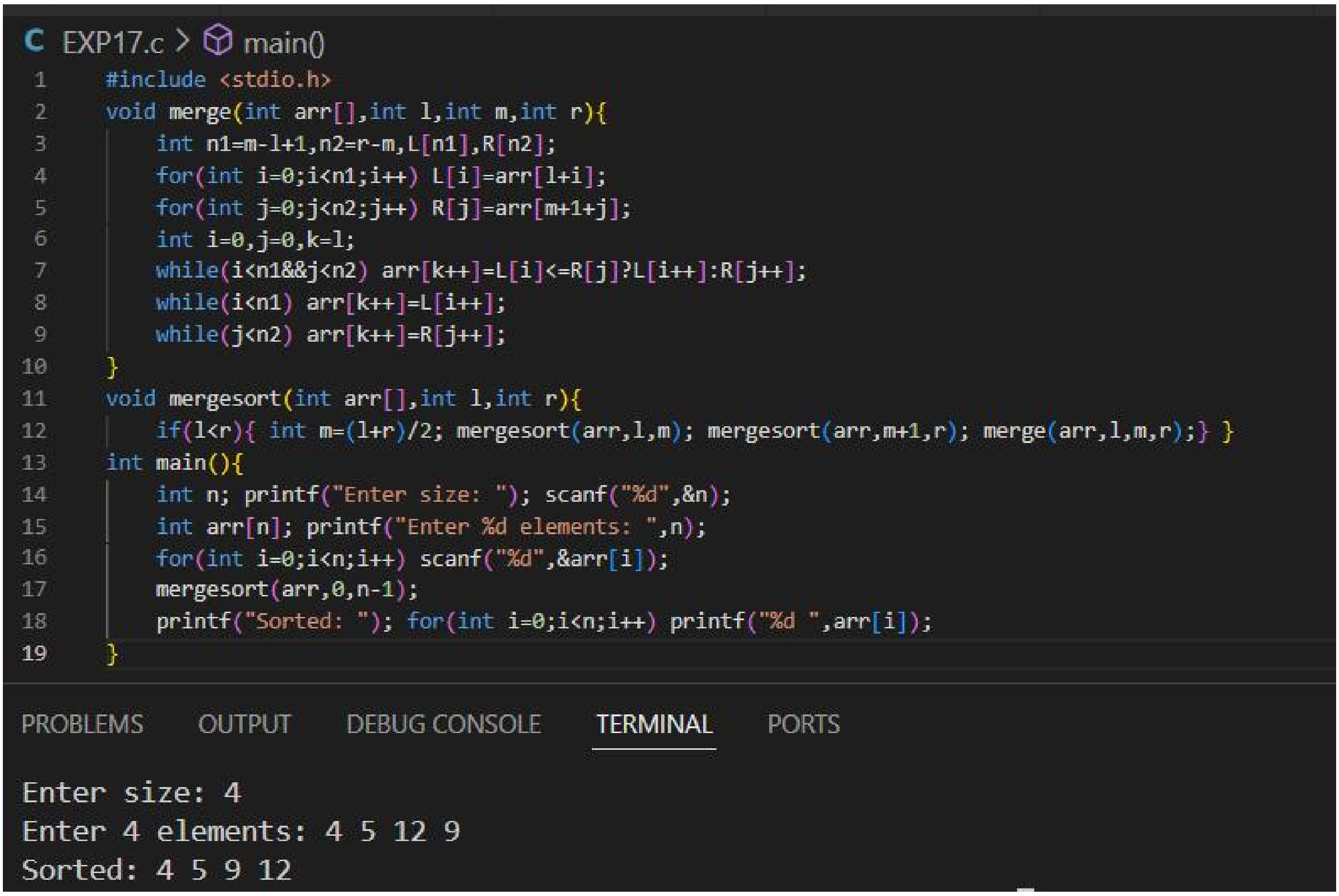
17. Write a C program to arrange a series of numbers using Merge Sort

Aim: To sort array using merge sort.

Algorithm:

1. Divide array into halves.
2. Recursively sort.
3. Merge sorted halves.

Program:



```c
#include <stdio.h>
void merge(int arr[],int l,int m,int r){
    int n1=m-l+1,n2=r-m,L[n1],R[n2];
    for(int i=0;i<n1;i++) L[i]=arr[l+i];
    for(int j=0;j<n2;j++) R[j]=arr[m+1+j];
    int i=0,j=0,k=l;
    while(i<n1&&j<n2) arr[k++]=L[i]<=R[j]?L[i++]:R[j++];
    while(i<n1) arr[k++]=L[i++];
    while(j<n2) arr[k++]=R[j++];
}
void mergesort(int arr[],int l,int r){
    if(l<r){ int m=(l+r)/2; mergesort(arr,l,m); mergesort(arr,m+1,r); merge(arr,l,m,r);} }
int main(){
    int n; printf("Enter size: "); scanf("%d",&n);
    int arr[n]; printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    mergesort(arr,0,n-1);
    printf("Sorted: "); for(int i=0;i<n;i++) printf("%d ",arr[i]);
}
```

```
Enter size: 4
Enter 4 elements: 4 5 12 9
Sorted: 4 5 9 12
```

Result: Sorted array displayed.

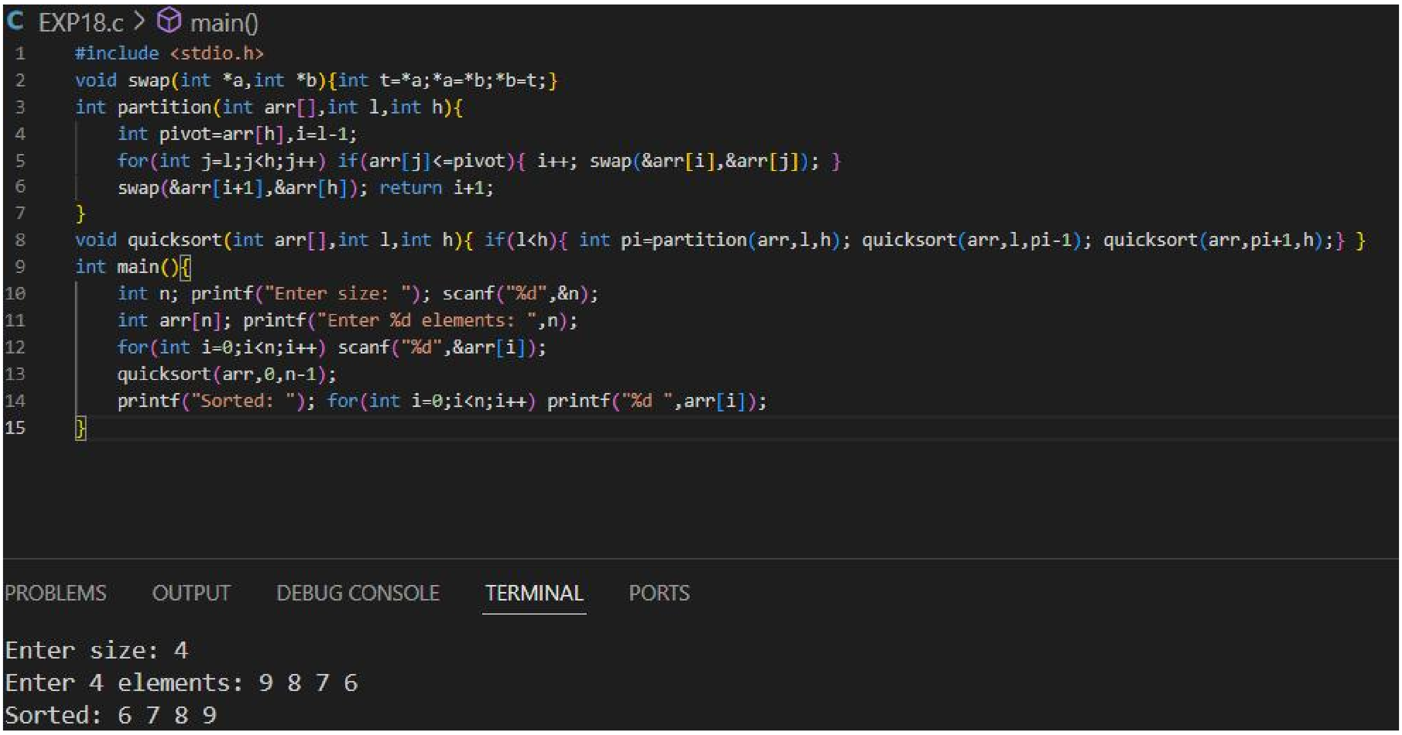
18. Write a C program to arrange a series of numbers using Quick Sort

Aim: To sort array using quick sort.

Algorithm:

1. Choose pivot.
2. Partition array.
3. Recursively quicksort.

Program:



```c
#include <stdio.h>
void swap(int *a,int *b){int t=*a;*a=*b;*b=t;}
int partition(int arr[],int l,int h){
    int pivot=arr[h],i=l-1;
    for(int j=l;j<h;j++) if(arr[j]<=pivot){ i++; swap(&arr[i],&arr[j]); }
    swap(&arr[i+1],&arr[h]); return i+1;
}
void quicksort(int arr[],int l,int h){ if(l<h){ int pi=partition(arr,l,h); quicksort(arr,l,pi-1); quicksort(arr,pi+1,h);} }
int main(){
    int n; printf("Enter size: "); scanf("%d",&n);
    int arr[n]; printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    quicksort(arr,0,n-1);
    printf("Sorted: "); for(int i=0;i<n;i++) printf("%d ",arr[i]);
}
```

```
Enter size: 4
Enter 4 elements: 9 8 7 6
Sorted: 6 7 8 9
```

Result: Sorted array displayed.

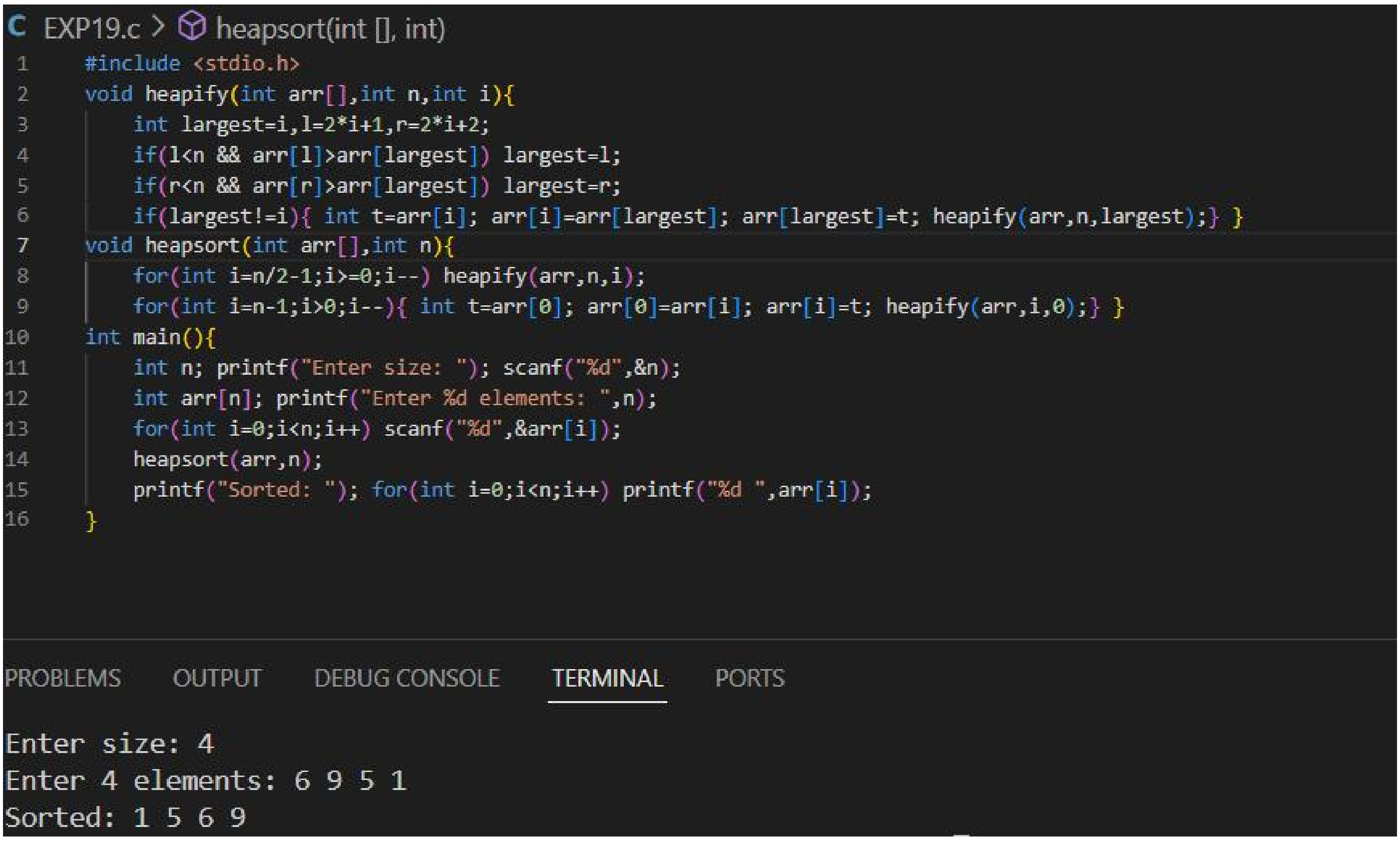
19. Write a C program to implement Heap sort

Aim: To sort array using heap sort.

Algorithm:

1. Build max heap.
2. Repeatedly swap root with last element.
3. Heapify.

Program:



```c
#include <stdio.h>
void heapify(int arr[],int n,int i){
    int largest=i,l=2*i+1,r=2*i+2;
    if(l<n && arr[l]>arr[largest]) largest=l;
    if(r<n && arr[r]>arr[largest]) largest=r;
    if(largest!=i){ int t=arr[i]; arr[i]=arr[largest]; arr[largest]=t; 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--){ int t=arr[0]; arr[0]=arr[i]; arr[i]=t; heapify(arr,i,0);} }
int main(){
    int n; printf("Enter size: "); scanf("%d",&n);
    int arr[n]; printf("Enter %d elements: ",n);
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    heapsort(arr,n);
    printf("Sorted: "); for(int i=0;i<n;i++) printf("%d ",arr[i]);
}
```

```
Enter size: 4
Enter 4 elements: 6 9 5 1
Sorted: 1 5 6 9
```

Result: Sorted array displayed.

20. Write a program to perform the following operations:

a) Insert an element into a AVL tree

b) Delete an element from a AVL tree

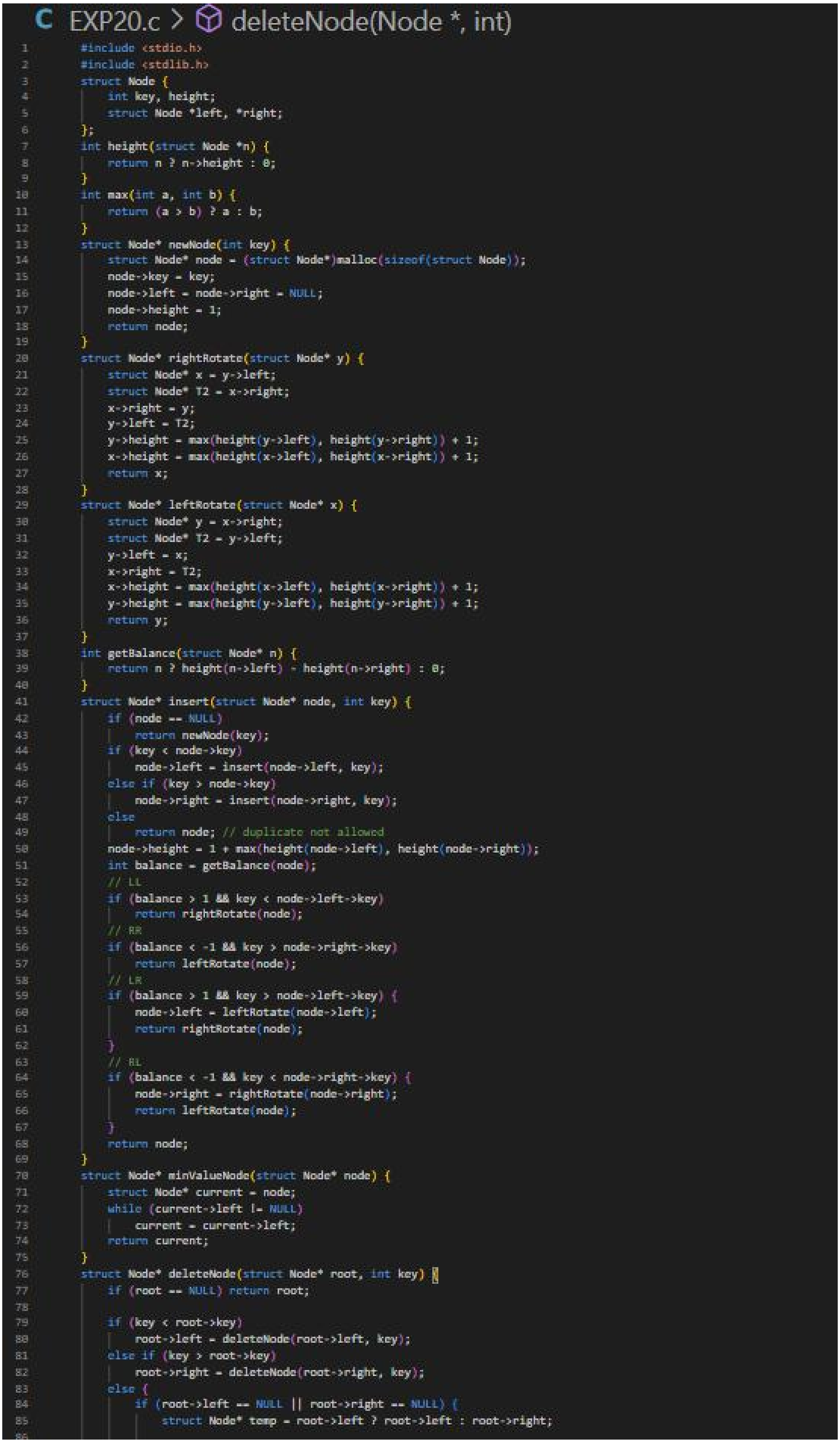
c) Search for a key element in a AVL tree

Aim:
To perform insertion, deletion, and search in an AVL tree.

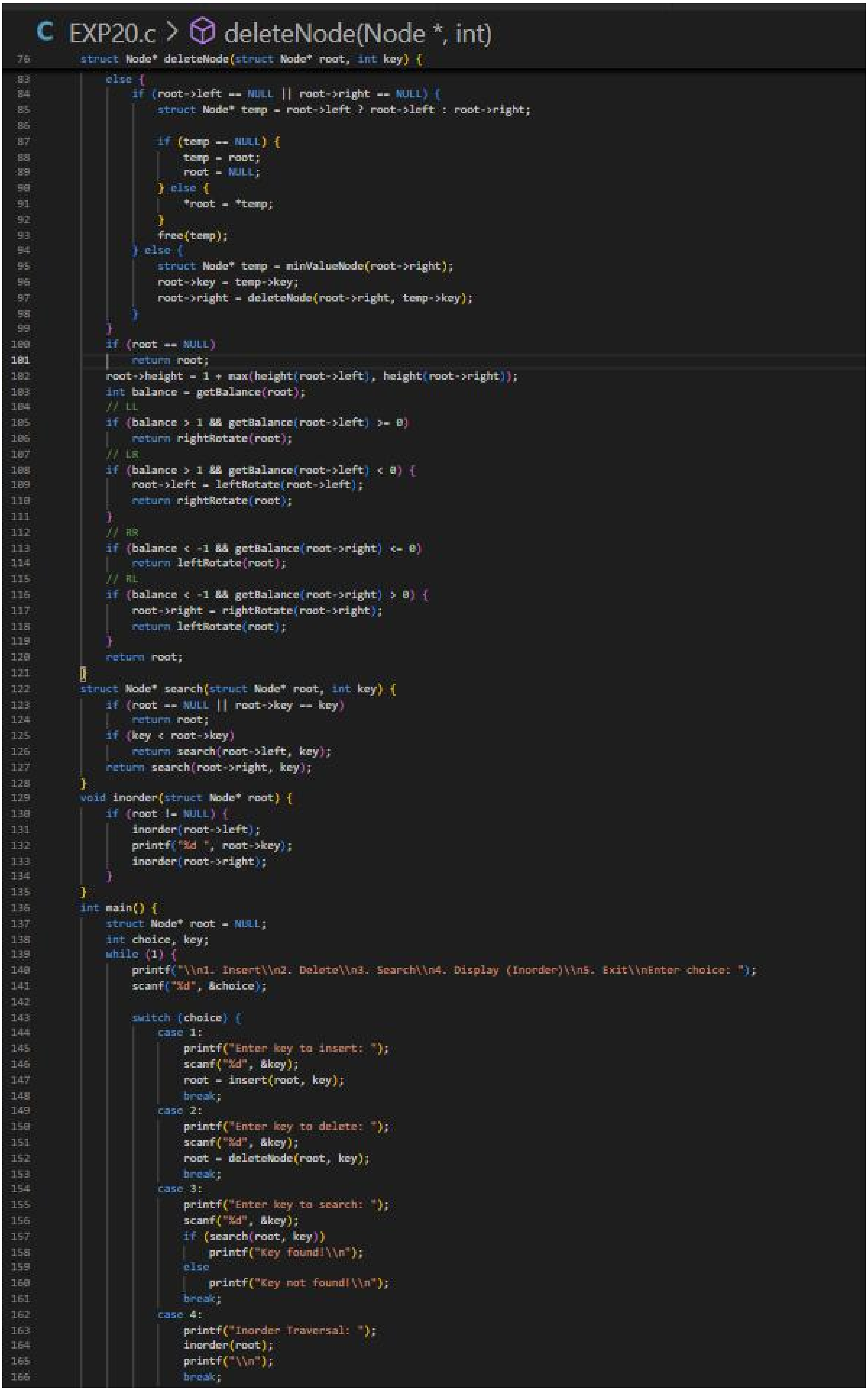
Algorithm:

1. Define a node structure with data, height, left and right pointers.
2. For insertion:
    - Perform normal BST insert.
    - Update height and balance factor.
    - Perform rotations (LL, RR, LR, RL) if unbalanced.
3. For deletion:
    - Perform normal BST delete.
    - Update height and balance factor.
    - Perform rotations if needed.
4. For search:
    - Traverse tree like BST until key found or NULL.
5. Provide menu to user: Insert, Delete, Search, Display (inorder).

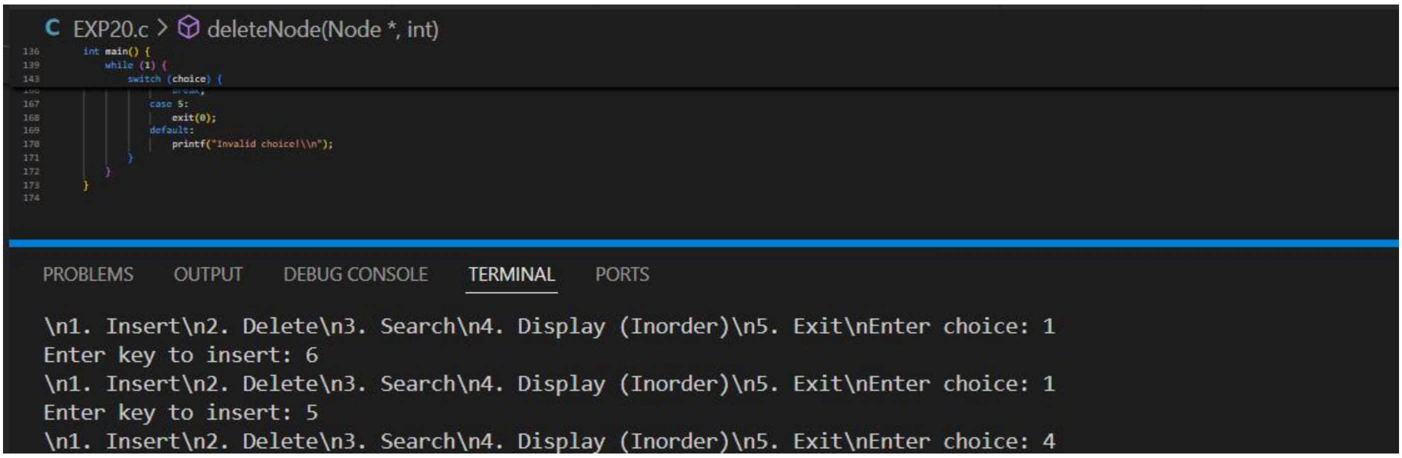
Program:



```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key, height;
    struct Node *left, *right;
};
int height(struct Node *n) {
    return n ? n->height : 0;
}
int max(int a, int b) {
    return (a > b) ? a : b;
}
struct Node* newNode(int key) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->key = key;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}
struct Node* rightRotate(struct Node* y) {
    struct Node* x = y->left;
    struct Node* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}
struct Node* leftRotate(struct Node* x) {
    struct Node* y = x->right;
    struct Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}
int getBalance(struct Node* n) {
    return n ? height(n->left) - height(n->right) : 0;
}
struct Node* insert(struct Node* node, int key) {
    if (node == NULL)
        return newNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node; // duplicate not allowed
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    // LL
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    // RR
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    // LR
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    // RL
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}
struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL || root->right == NULL) {
            struct Node* temp = root->left ? root->left : root->right;
```



```c
struct Node* deleteNode(struct Node* root, int key) {
    else {
        if (root->left == NULL || root->right == NULL) {
            struct Node* temp = root->left ? root->left : root->right;

            if (temp == NULL) {
                temp = root;
                root = NULL;
            } else {
                *root = *temp;
            }
            free(temp);
        } else {
            struct Node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    if (root == NULL)
        return root;
    root->height = 1 + max(height(root->left), height(root->right));
    int balance = getBalance(root);
    // LL
    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);
    // LR
    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    // RR
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);
    // RL
    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->key == key)
        return root;
    if (key < root->key)
        return search(root->left, key);
    return search(root->right, key);
}
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}
int main() {
    struct Node* root = NULL;
    int choice, key;
    while (1) {
        printf("\\n1. Insert\\n2. Delete\\n3. Search\\n4. Display (Inorder)\\n5. Exit\\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                root = insert(root, key);
                break;
            case 2:
                printf("Enter key to delete: ");
                scanf("%d", &key);
                root = deleteNode(root, key);
                break;
            case 3:
                printf("Enter key to search: ");
                scanf("%d", &key);
                if (search(root, key))
                    printf("Key found!\\n");
                else
                    printf("Key not found!\\n");
                break;
            case 4:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\\n");
                break;
```



```c
int main() {
    while (1) {
        switch (choice) {
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice!\\n");
        }
    }
}
```

```
\n1. Insert\n2. Delete\n3. Search\n4. Display (Inorder)\n5. Exit\nEnter choice: 1
Enter key to insert: 6
\n1. Insert\n2. Delete\n3. Search\n4. Display (Inorder)\n5. Exit\nEnter choice: 1
Enter key to insert: 5
\n1. Insert\n2. Delete\n3. Search\n4. Display (Inorder)\n5. Exit\nEnter choice: 4
```

Result:

AVL tree supports Insert, Delete, Search, Display with automatic balancing.

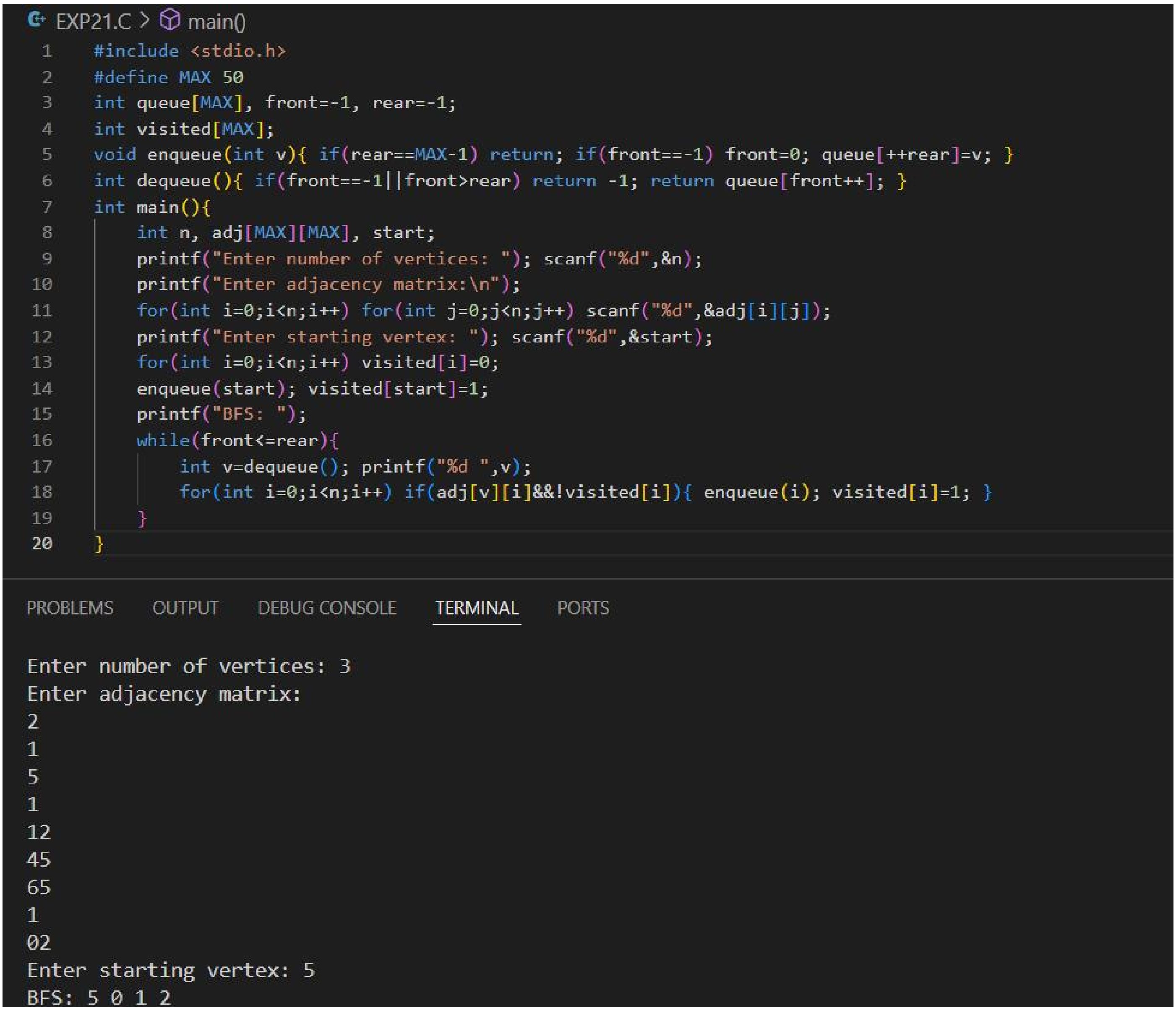
21. Write a C program to Graph traversal using Breadth First Search

Aim: To traverse a graph using Breadth First Search.

Algorithm:

1. Read number of vertices.
2. Input adjacency matrix.
3. Read starting vertex.
4. Use queue to explore neighbors level by level.
5. Mark visited and print order.

Program:



```c
#include <stdio.h>
#define MAX 50
int queue[MAX], front=-1, rear=-1;
int visited[MAX];
void enqueue(int v){ if(rear==MAX-1) return; if(front==-1) front=0; queue[++rear]=v; }
int dequeue(){ if(front==-1||front>rear) return -1; return queue[front++]; }
int main(){
    int n, adj[MAX][MAX], start;
    printf("Enter number of vertices: "); scanf("%d",&n);
    printf("Enter adjacency matrix:\n");
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&adj[i][j]);
    printf("Enter starting vertex: "); scanf("%d",&start);
    for(int i=0;i<n;i++) visited[i]=0;
    enqueue(start); visited[start]=1;
    printf("BFS: ");
    while(front<=rear){
        int v=dequeue(); printf("%d ",v);
        for(int i=0;i<n;i++) if(adj[v][i]&&!visited[i]){ enqueue(i); visited[i]=1; }
    }
}
```

```
Enter number of vertices: 3
Enter adjacency matrix:
2
1
5
1
12
45
65
1
02
Enter starting vertex: 5
BFS: 5 0 1 2
```

Result: BFS traversal order displayed.

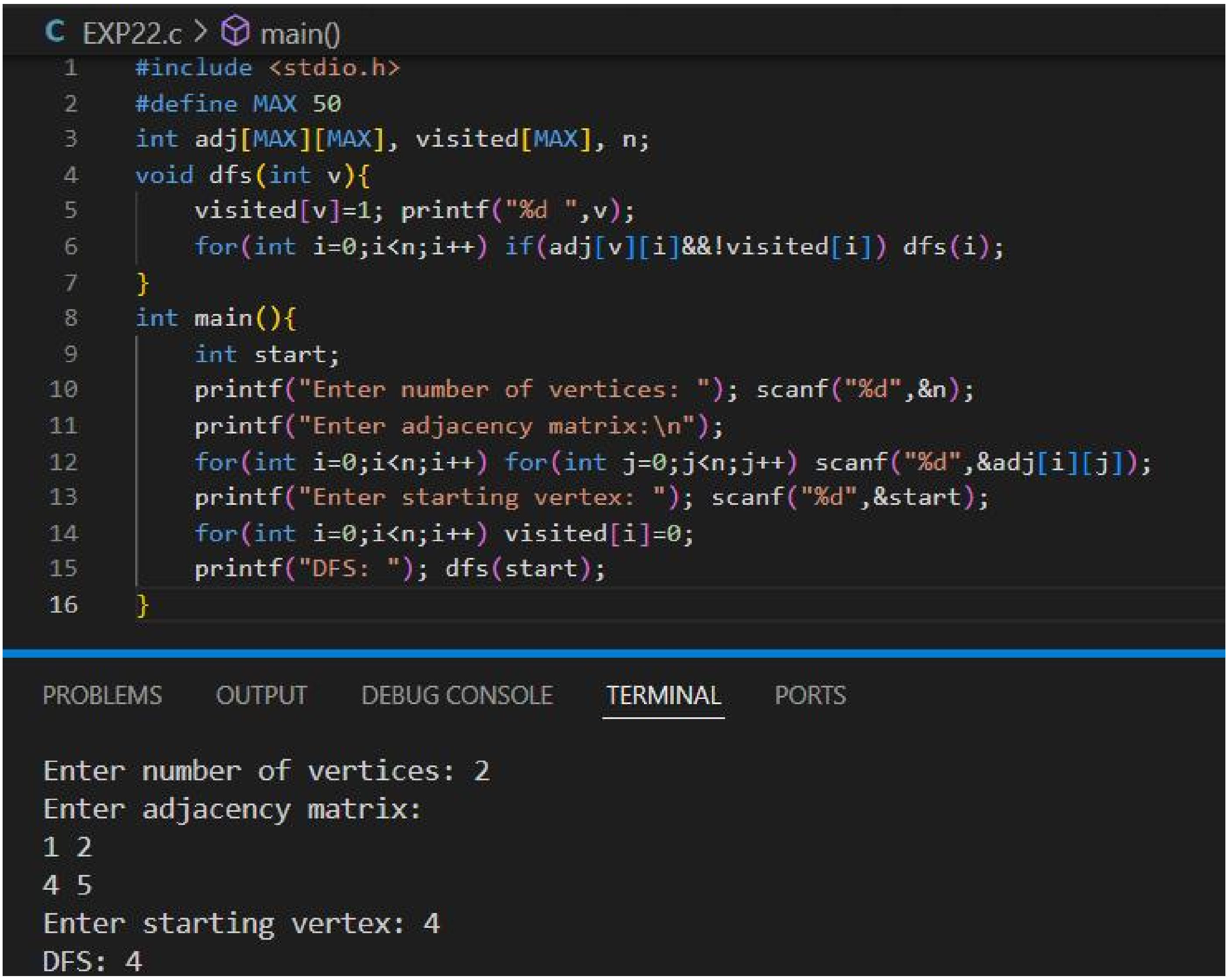
22. Write a C program to Graph traversal using Depth First Search

Aim: To traverse a graph using Depth First Search.

Algorithm:

1. Read number of vertices.
2. Input adjacency matrix.
3. Read starting vertex.
4. Recursively explore unvisited neighbors.

Program:



```c
#include <stdio.h>
#define MAX 50
int adj[MAX][MAX], visited[MAX], n;
void dfs(int v){
    visited[v]=1; printf("%d ",v);
    for(int i=0;i<n;i++) if(adj[v][i]&&!visited[i]) dfs(i);
}
int main(){
    int start;
    printf("Enter number of vertices: "); scanf("%d",&n);
    printf("Enter adjacency matrix:\n");
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&adj[i][j]);
    printf("Enter starting vertex: "); scanf("%d",&start);
    for(int i=0;i<n;i++) visited[i]=0;
    printf("DFS: "); dfs(start);
}
```

```
Enter number of vertices: 2
Enter adjacency matrix:
1 2
4 5
Enter starting vertex: 4
DFS: 4
```

Result: DFS traversal order displayed.

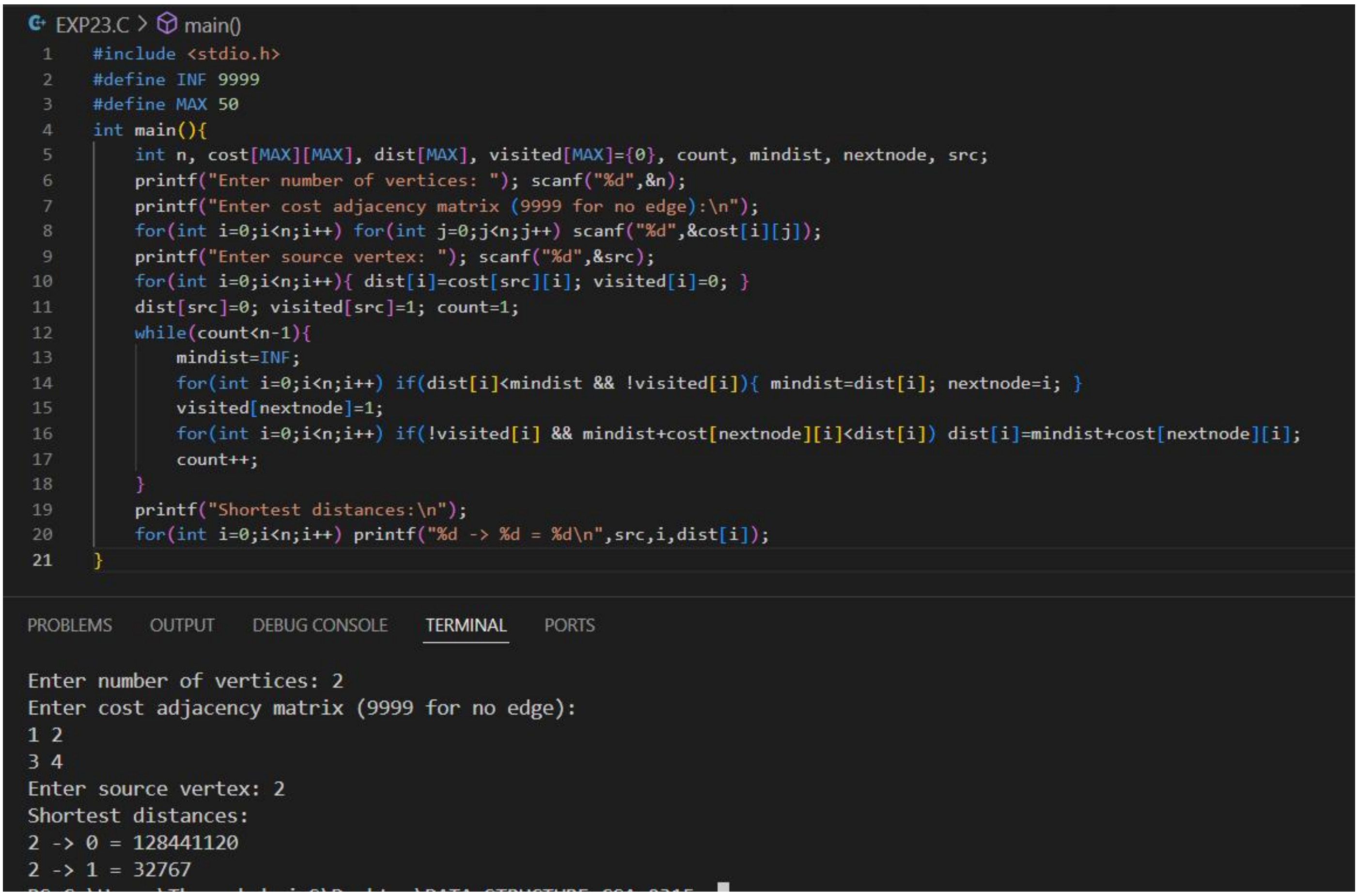
23. Implementation of Shortest Path Algorithms using Dijkstra's Algorithm

Aim: To find shortest path from a source using Dijkstra's Algorithm.

Algorithm:

1. Read number of vertices.
2. Input adjacency matrix (use large value for no edge).
3. Read source vertex.
4. Use Dijkstra's algorithm to compute shortest distances.
5. Display distances.

Program:



```c
#include <stdio.h>
#define INF 9999
#define MAX 50
int main(){
    int n, cost[MAX][MAX], dist[MAX], visited[MAX]={0}, count, mindist, nextnode, src;
    printf("Enter number of vertices: "); scanf("%d",&n);
    printf("Enter cost adjacency matrix (9999 for no edge):\n");
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&cost[i][j]);
    printf("Enter source vertex: "); scanf("%d",&src);
    for(int i=0;i<n;i++){ dist[i]=cost[src][i]; visited[i]=0; }
    dist[src]=0; visited[src]=1; count=1;
    while(count<n-1){
        mindist=INF;
        for(int i=0;i<n;i++) if(dist[i]<mindist && !visited[i]){ mindist=dist[i]; nextnode=i; }
        visited[nextnode]=1;
        for(int i=0;i<n;i++) if(!visited[i] && mindist+cost[nextnode][i]<dist[i]) dist[i]=mindist+cost[nextnode][i];
        count++;
    }
    printf("Shortest distances:\n");
    for(int i=0;i<n;i++) printf("%d -> %d = %d\n",src,i,dist[i]);
}
```

```
Enter number of vertices: 2
Enter cost adjacency matrix (9999 for no edge):
1 2
3 4
Enter source vertex: 2
Shortest distances:
2 -> 0 = 128441120
2 -> 1 = 32767
```

Result: Displays shortest distances.

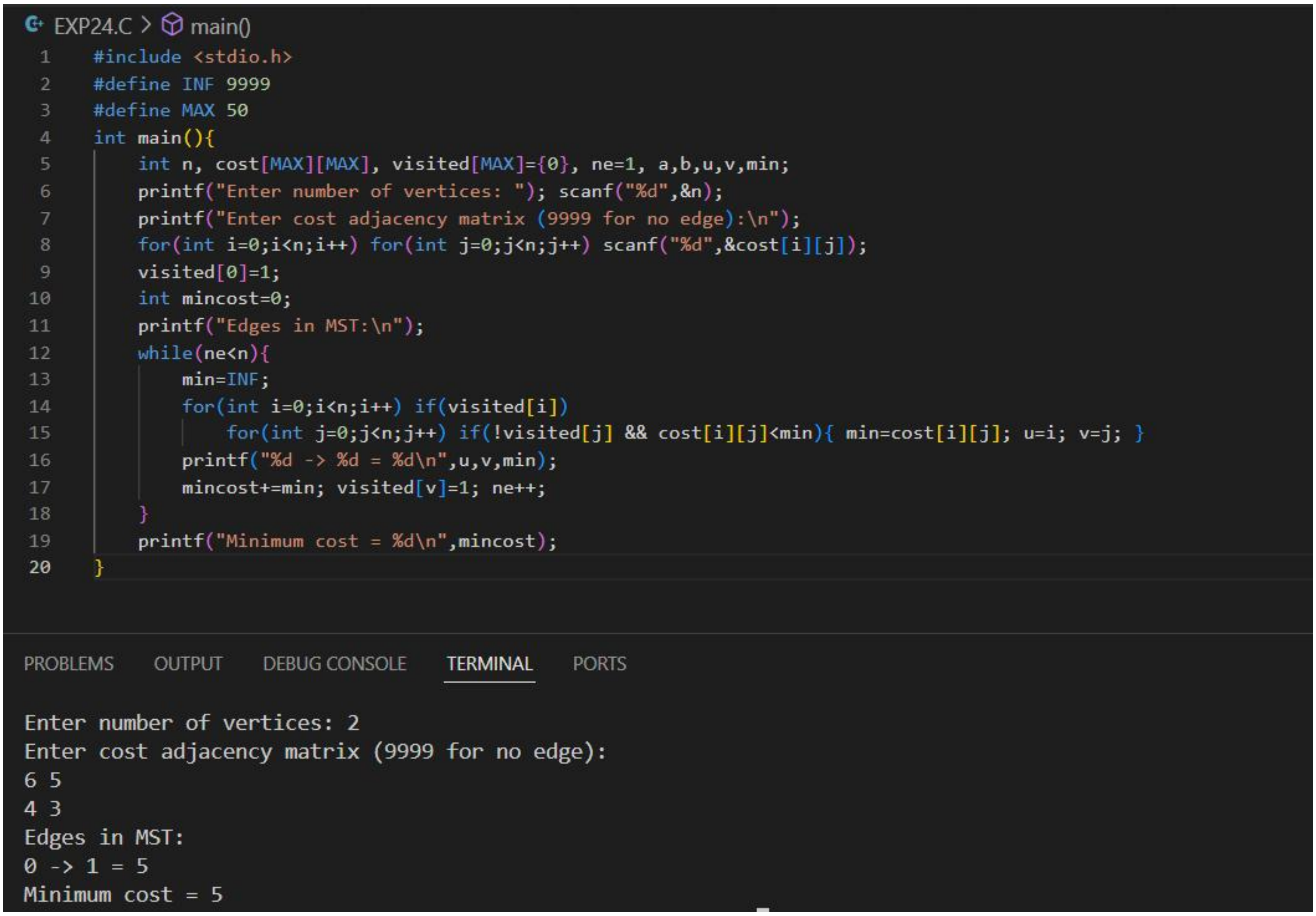
24. Implementation of Minimum Spanning Tree using Prim’s Algorithm

Aim: To find Minimum Spanning Tree using Prim’s Algorithm.

Algorithm:

1. Read number of vertices.
2. Input cost adjacency matrix.
3. Start from vertex 0.
4. At each step, add minimum edge connecting tree to new vertex.
5. Repeat until all vertices included.

Program:



```
#include <stdio.h>
#define INF 9999
#define MAX 50
int main(){
    int n, cost[MAX][MAX], visited[MAX]={0}, ne=1, a,b,u,v,min;
    printf("Enter number of vertices: "); scanf("%d",&n);
    printf("Enter cost adjacency matrix (9999 for no edge):\n");
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) scanf("%d",&cost[i][j]);
    visited[0]=1;
    int mincost=0;
    printf("Edges in MST:\n");
    while(ne<n){
        min=INF;
        for(int i=0;i<n;i++) if(visited[i])
            for(int j=0;j<n;j++) if(!visited[j] && cost[i][j]<min){ min=cost[i][j]; u=i; v=j; }
        printf("%d -> %d = %d\n",u,v,min);
        mincost+=min; visited[v]=1; ne++;
    }
    printf("Minimum cost = %d\n",mincost);
}
```

```
Enter number of vertices: 2
Enter cost adjacency matrix (9999 for no edge):
6 5
4 3
Edges in MST:
0 -> 1 = 5
Minimum cost = 5
```

Result: MST edges and cost displayed.

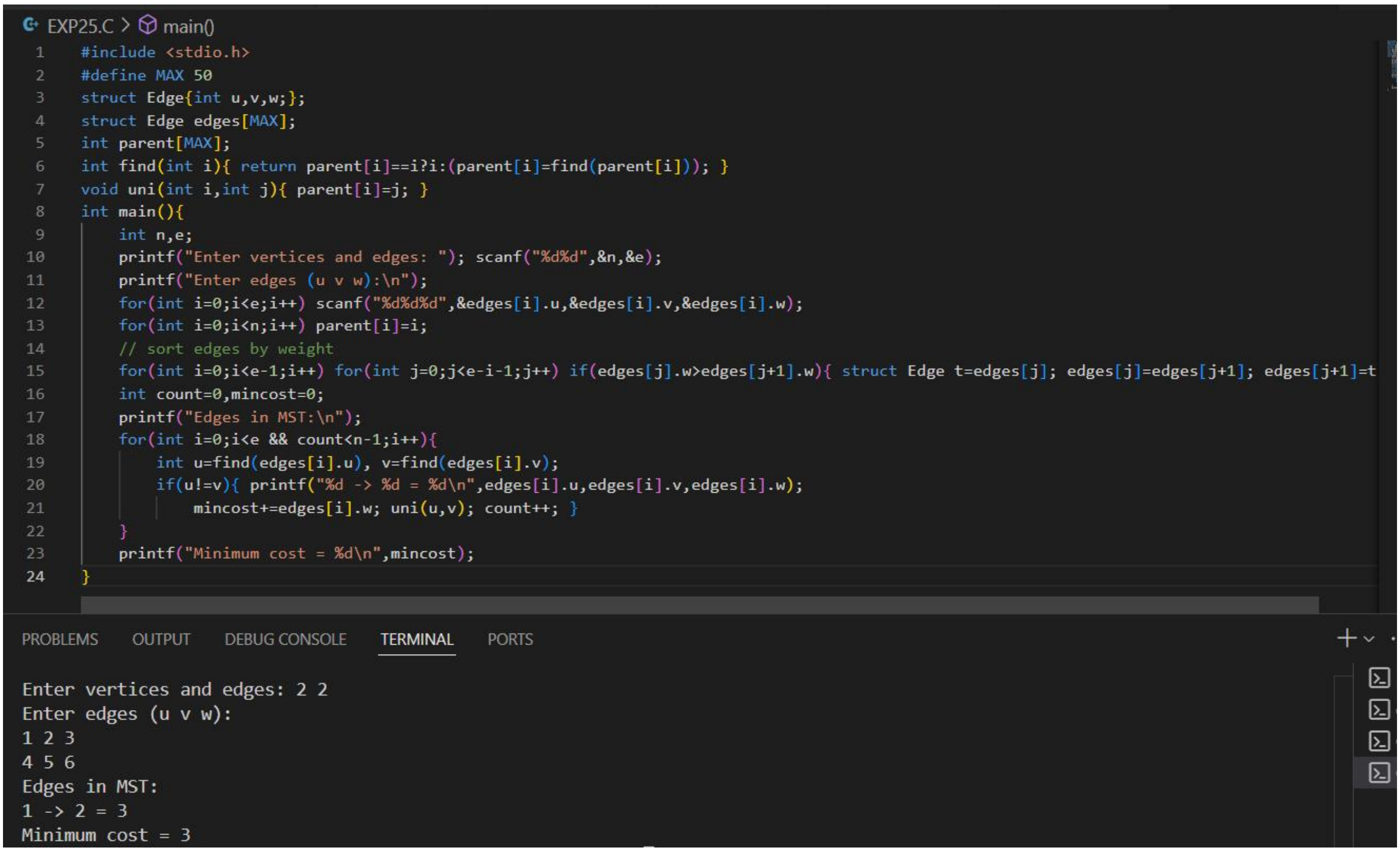
25. Implementation of Minimum Spanning Tree using Kruskal Algorithm

Aim: To find Minimum Spanning Tree using Kruskal’s Algorithm.

Algorithm:

1. Read number of vertices and edges.
2. Sort edges by weight.
3. Use union-find to avoid cycles.
4. Add edges until MST complete.

Program:



```c
#include <stdio.h>
#define MAX 50
struct Edge{int u,v,w;};
struct Edge edges[MAX];
int parent[MAX];
int find(int i){ return parent[i]==i?i:(parent[i]=find(parent[i])); }
void uni(int i,int j){ parent[i]=j; }
int main(){
    int n,e;
    printf("Enter vertices and edges: "); scanf("%d%d",&n,&e);
    printf("Enter edges (u v w):\n");
    for(int i=0;i<e;i++) scanf("%d%d%d",&edges[i].u,&edges[i].v,&edges[i].w);
    for(int i=0;i<n;i++) parent[i]=i;
    // sort edges by weight
    for(int i=0;i<e-1;i++) for(int j=0;j<e-i-1;j++) if(edges[j].w>edges[j+1].w){ struct Edge t=edges[j]; edges[j]=edges[j+1]; edges[j+1]=t
    int count=0,mincost=0;
    printf("Edges in MST:\n");
    for(int i=0;i<e && count<n-1;i++){
        int u=find(edges[i].u), v=find(edges[i].v);
        if(u!=v){ printf("%d -> %d = %d\n",edges[i].u,edges[i].v,edges[i].w);
            mincost+=edges[i].w; uni(u,v); count++; }
    }
    printf("Minimum cost = %d\n",mincost);
}
```

```
Enter vertices and edges: 2 2
Enter edges (u v w):
1 2 3
4 5 6
Edges in MST:
1 -> 2 = 3
Minimum cost = 3
```

Result: Displays MST edges and cost using Kruskal.