**BFS**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 50

// This struct represents a directed graph using

// adjacency list representation

typedef struct Graph\_t {

// No. of vertices

int V;

bool adj[MAX\_VERTICES][MAX\_VERTICES];

} Graph;

// Constructor

Graph\* Graph\_create(int V)

{

Graph\* g = malloc(sizeof(Graph));

g->V = V;

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

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

g->adj[i][j] = false;

}

}

return g;

}

// Destructor

void Graph\_destroy(Graph\* g) {

free(g);

}

// function to add an edge to graph

void Graph\_addEdge(Graph\* g, int v, int w)

{

// Add w to v’s list.

g->adj[v][w] = true;

}

// Prints BFS traversal from a given source s

void Graph\_BFS(Graph\* g, int s)

{

// Mark all the vertices as not visited

bool visited[MAX\_VERTICES];

for (int i = 0; i < g->V; i++) {

visited[i] = false;

}

// Create a queue for BFS

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue[rear++] = s;

while (front != rear) {

// Dequeue a vertex from queue and print it

s = queue[front++];

printf("%d ", s);

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (int adjacent = 0; adjacent < g->V;

adjacent++) {

if (g->adj[s][adjacent] && !visited[adjacent]) {

visited[adjacent] = true;

queue[rear++] = adjacent;

}

}

}

}

// Driver code

int main()

{

// Create a graph

Graph\* g = Graph\_create(4);

Graph\_addEdge(g, 0, 1);

Graph\_addEdge(g, 0, 2);

Graph\_addEdge(g, 1, 2);

Graph\_addEdge(g, 2, 0);

Graph\_addEdge(g, 2, 3);

Graph\_addEdge(g, 3, 3);

printf("Following is Breadth First Traversal "

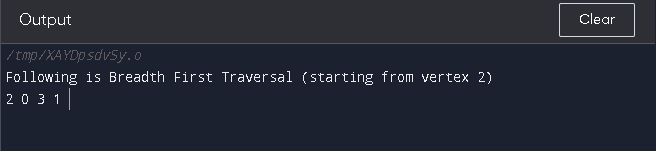
"(starting from vertex 2) \n");

Graph\_BFS(g, 2);

Graph\_destroy(g);

return 0;

}

****

**DFS**

// C code to implement above approach

#include <stdio.h>

#include <stdlib.h>

// Globally declared visited array

int vis[100];

// Graph structure to store number

// of vertices and edges and

// Adjacency matrix

struct Graph {

int V;

int E;

int\*\* Adj;

};

// Function to input data of graph

struct Graph\* adjMatrix()

{

struct Graph\* G = (struct Graph\*)

malloc(sizeof(struct Graph));

if (!G) {

printf("Memory Error\n");

return NULL;

}

G->V = 7;

G->E = 7;

G->Adj = (int\*\*)malloc((G->V) \* sizeof(int\*));

for (int k = 0; k < G->V; k++) {

G->Adj[k] = (int\*)malloc((G->V) \* sizeof(int));

}

for (int u = 0; u < G->V; u++) {

for (int v = 0; v < G->V; v++) {

G->Adj[u][v] = 0;

}

}

G->Adj[0][1] = G->Adj[1][0] = 1;

G->Adj[0][2] = G->Adj[2][0] = 1;

G->Adj[1][3] = G->Adj[3][1] = 1;

G->Adj[1][4] = G->Adj[4][1] = 1;

G->Adj[1][5] = G->Adj[5][1] = 1;

G->Adj[1][6] = G->Adj[6][1] = 1;

G->Adj[6][2] = G->Adj[2][6] = 1;

return G;

}

// DFS function to print DFS traversal of graph

void DFS(struct Graph\* G, int u)

{

vis[u] = 1;

printf("%d ", u);

for (int v = 0; v < G->V; v++) {

if (!vis[v] && G->Adj[u][v]) {

DFS(G, v);

}

}

}

// Function for DFS traversal

void DFStraversal(struct Graph\* G)

{

for (int i = 0; i < 100; i++) {

vis[i] = 0;

}

for (int i = 0; i < G->V; i++) {

if (!vis[i]) {

DFS(G, i);

}

}

}

// Driver code

void main()

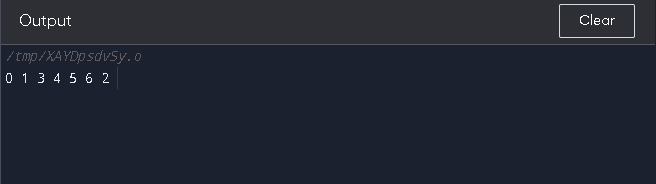
{

struct Graph\* G;

G = adjMatrix();

DFStraversal(G);

}



**Matrix addition**

#include <stdio.h>

int main() {

    int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

    int b[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

    int c[3][3];

    int i, j;

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            c[i][j] = a[i][j] + b[i][j];

        }

    }

    printf("Result of addition: \n");

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            printf("%d ", c[i][j]);

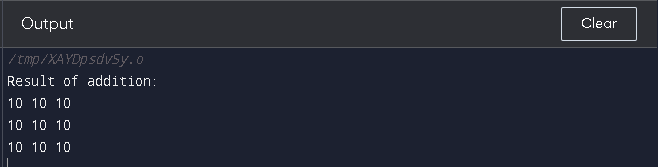
        }

        printf("\n");

    }

    return 0;

}



**Bubble sort**

#include<stdio.h>

void print(int a[], int n) //function to print array elements

{

int i;

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

{

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

}

}

void bubble(int a[], int n) // function to implement bubble sort

{

int i, j, temp;

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

{

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

{

if(a[j] < a[i])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

void main ()

{

int i, j,temp;

int a[5] = { 10, 35, 32, 13, 26};

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

printf("Before sorting array elements are - \n");

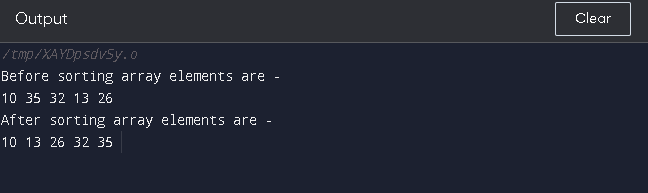
print(a, n);

bubble(a, n);

printf("\nAfter sorting array elements are - \n");

print(a, n);

}

****

**Merge sort**

/\* C program for Merge Sort \*/

#include <stdio.h>

#include <stdlib.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

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

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

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

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

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

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

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

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

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

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

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

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

printf("\n");

}

/\* Driver code \*/

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

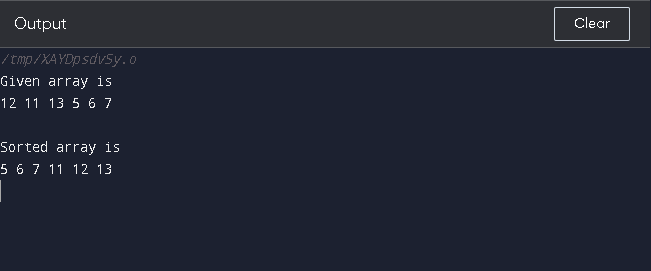
mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}



**Finding min max number sum avg in one dimension**

#include <stdio.h>

int main()

{

int a[8],i,s=0,g,l;

float avg;

printf("Enter 8 Numbers:\n");

for(i=0;i<8;i++)

{

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

s=s+a[i];

avg=s/8.0;

}

printf("Sum of Array Elements = %d\n",s);

printf("Average of Elements = %.2f\n",avg);

g=a[0];

for(i=0;i<8;i++)

if(a[i]>g)

g=a[i];

printf(" Maximum Element = %d\n",g);

l=a[0];

for(i=0;i<8;i++)

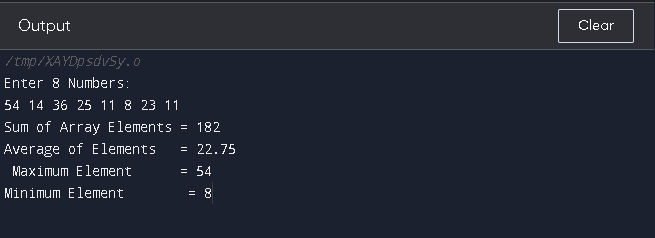
if(a[i]<l)

l=a[i];

printf("Minimum Element = %d",l);

return 0;

}

****

**Matrix multiplication**

#include<stdio.h>

#include<stdlib.h>

int main(){

int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;

system("cls");

printf("enter the number of row=");

scanf("%d",&r);

printf("enter the number of column=");

scanf("%d",&c);

printf("enter the first matrix element=\n");

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

{

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

{

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

}

}

printf("enter the second matrix element=\n");

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

{

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

{

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

}

}

printf("multiply of the matrix=\n");

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

{

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

{

mul[i][j]=0;

for(k=0;k<c;k++)

{

mul[i][j]+=a[i][k]\*b[k][j];

}

}

}

//for printing result

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

{

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

{

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

}

printf("\n");

}

return 0;

}

