**Assignment 1:-To implement addition ,multiplication and transpose of two 2D arrays.**

#include<conio.h>

#include<stdio.h>

void read(int m[][100],int row,int col)

{

  for(int i=0;i<row;i++)

  {

    for(int j=0;j<col;j++)

    {scanf("%d",&m[i][j]);}

  }

}

void addition(int mat1[][100],int mat2[][100],int sum[][100],int row,int col)

{

  for(int i=0;i<row;i++)

  {

    for(int j=0;j<col;j++)

    {sum[i][j]=mat1[i][j]+mat2[i][j];}

  }

}

void transpose(int mat1[][100],int tra[][100],int row,int col)

{

  for(int i=0;i<row;i++)

  {

   for(int j=0;j<col;j++)

   {tra[j][i]=mat1[i][j];}

  }

}

void multiplication(int mat1[][100],int mat2[][100],int mul[][100],int row1,int col,int row2)

{

   for(int i=0;i<row1;i++)

   {

     for(int j=0;j<col;j++)

     {mul[i][j]=0;

        for(int k=0;k<row2;k++)

        {mul[i][j]+=mat1[i][k]\*mat2[k][j];}

     }

   }

}

void display(int result[][100],int row,int col)

{

    for(int i=0;i<row;i++)

    {

      for(int j=0;j<col;j++)

      {printf("%d\t",result[i][j]);}

       printf("\n");

    }

}

void main()

{

int mat1[100][100],mat2[100][100],sum[100][100],mul[100][100],tra[100][100];

int r1,c1,r2,c2;

printf("enter the size of first matrix=");

scanf("%d %d",&r1,&c1);

printf("enter the size of second matrix=");

scanf("%d %d",&r2,&c2);

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

read(mat1,r1,c1);

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

read(mat2,r2,c2);

if(r1==r2 && c1==c2)

{

   addition(mat1,mat2,sum,r1,c1);

   printf("sum matrix:\n");

   display(sum,r1,c1);

}

else

{  printf("\nsum of matrix is not possible ");}

if(c1==r2)

{

   multiplication(mat1,mat2,mul,r1,c2,r2);

   printf("multiplication:\n");

   display(mul,r1,c2);

}

else

{  printf("multiplication is not possible");}

transpose(mat1,tra,r1,c1);

printf("transpose of first matrix:\n");

display(tra,c1,r1);

transpose(mat2,tra,r2,c2);

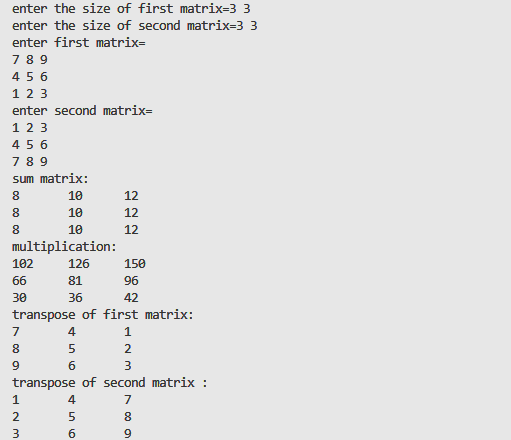
printf("transpose of second matrix :\n");

display(tra,c2,r2);

getch();

}

**Output-**

****

#include<stdio.h>

Assignment 2 **To implement linear search**

#include<stdio.h>

#include<conio.h>

void read(int A[],int size)

{

int i;

printf("enter the element of array:\n");

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

{

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

}

}

void display(int A[],int size)

{

int i;

printf("element of array:\n ");

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

{

printf("%5d",A[i]);

}

}

int b\_search(int A[],int size,int k)

{

int i,mid,beg=0,end=size-1;

while(beg<=end)

{

mid=(beg+end)/2;

if(A[mid]==k)

{

return mid;

}

else if(A[mid]<k)

{

beg=mid+1;

}

else

{

end=mid-1;

}

}

return -1;

}

int main()

{

int A[10],k,n,loc;

printf("\nenter the size of array=");

scanf("%d",&n);

read(A,n);

display(A,n);

printf("\n\nenter the element for searching=");

scanf("%d",&k);

loc=b\_search(A,n,k);

if(loc==-1)

{

printf("element is not present");

}

else

{

printf("\nelement %d is present at tha location=%d",A[loc],loc);

}

return 0;

}

**Output-**

**2b:- To implement Binary search**

#include<stdio.h>

#include<conio.h>

void read(int A[],int size)

{

int i;

printf("enter the element of array:\n");

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

{

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

}

}

void display(int A[],int size)

{

int i;

printf("element of array:\n ");

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

{

printf("%5d",A[i]);

}

}

int b\_search(int A[],int size,int k)

{

int i,mid,beg=0,end=size-1;

while(beg<=end)

{

mid=(beg+end)/2;

if(A[mid]==k)

{

return mid;

}

else if(A[mid]<k)

{

beg=mid+1;

}

else

{

end=mid-1;

}

}

return -1;

}

int main()

{

int A[10],k,n,loc;

printf("\nenter the size of array=");

scanf("%d",&n);

read(A,n);

display(A,n);

printf("\n\nenter the element for searching=");

scanf("%d",&k);

loc=b\_search(A,n,k);

if(loc==-1)

{

printf("element is not present");

}

else

{

printf("\nelement %d is present at tha location=%d",A[loc],loc);

}

return 0;

}

**Output-**

**Assignment 3:- To implement stack, queue and circular queue using array.**

**Assignment-3A**

#include<stdio.h>

int stack[10],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

void main()

{

top=-1;

printf("\n Enter the size of STACK:");

scanf("%d",&n);

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter your Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t You have Exited the program ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

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

printf("\n Press Next Choice");

}

else

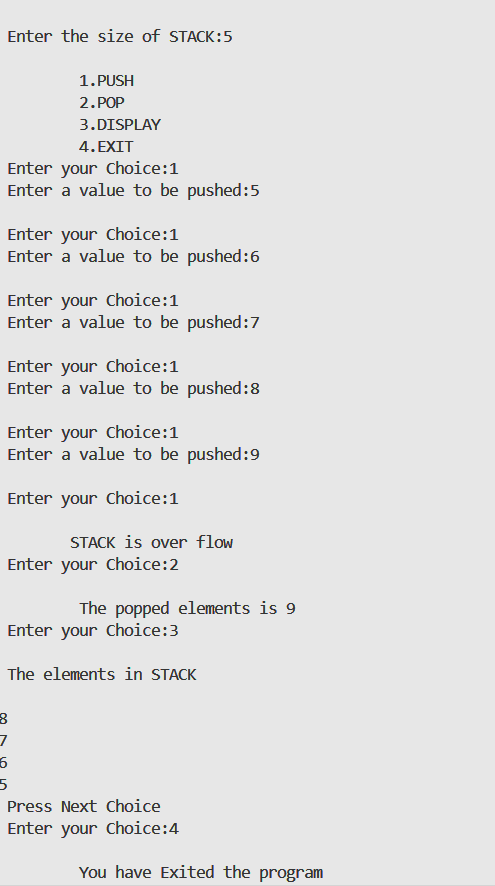
{

printf("\n The STACK is empty");

}

}

**OUTPUT**

****

**Assignment-3B to implement queue**

#include <stdio.h>

#define MAX\_SIZE 100

int queue[MAX\_SIZE];

int front = -1;

int rear = -1;

void enqueue(int value)

{

if (rear == MAX\_SIZE - 1)

{

printf("Queue is full. Cannot enqueue any more elements.\n");

return;

}

if (front == -1)

{

front = 0;

}

rear++;

queue[rear] = value;

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

}

void dequeue()

{

if (front == -1 || front > rear)

{

printf("Queue is empty. Cannot dequeue any element.\n");

return;

}

printf("%d dequeued successfully.\n", queue[front]);

front++;

}

void display()

{

if (front == -1 || front > rear)

{

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

return;

}

printf("Queue elements: ");

for (int i = front; i <= rear; i++)

{

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

}

printf("\n");

}

int getValue()

{

int value;

printf("Enter a value: ");

scanf("%d", &value);

return value;

}

int main()

{

int value;

value = getValue();

enqueue(value);

value = getValue();

enqueue(value);

value = getValue();

enqueue(value);

display();

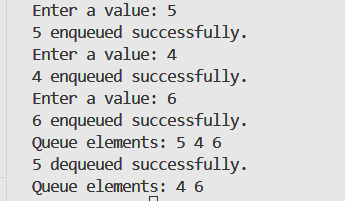
dequeue();

display();

return 0;

}

**OUTPUT**



**Assignment-3C circular queue**

#include <stdio.h>

#define MAX\_SIZE 10

int queue[MAX\_SIZE];

int front = -1;

int rear = -1;

int isFull()

{

if ((front == 0 && rear == MAX\_SIZE - 1) || (rear == front - 1))

return 1;

else

return 0;

}

int isEmpty()

{

if (front == -1)

return 1;

else

return 0;

}

void enqueue(int data)

{

if (isFull())

{

printf("Queue is full. Cannot enqueue element.\n");

}

else

{

if (front == -1)

front = 0;

rear = (rear + 1) % MAX\_SIZE;

queue[rear] = data;

printf("%d enqueued to the queue.\n", data);

}

}

void dequeue()

{

if (isEmpty())

{

printf("Queue is empty. Cannot dequeue element.\n");

}

else

{

printf("%d dequeued from the queue.\n", queue[front]);

if (front == rear)

front = rear = -1;

else

front = (front + 1) % MAX\_SIZE;

}

}

void display()

{

if (isEmpty())

{

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

}

else

{

int i = front;

printf("Queue elements: ");

while (i != rear)

{

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

i = (i + 1) % MAX\_SIZE;

}

printf("%d\n", queue[rear]);

}

}

int main()

{

int data;

printf("Enter elements to enqueue (enter 0 to stop):\n");

while (1)

{

scanf("%d", &data);

if (data == 0)

break;

enqueue(data);

}

display();

int i,d;

printf("How many elements you want to dequeue?");

scanf("%d",&d);

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

dequeue();

}

if(d>MAX\_SIZE){

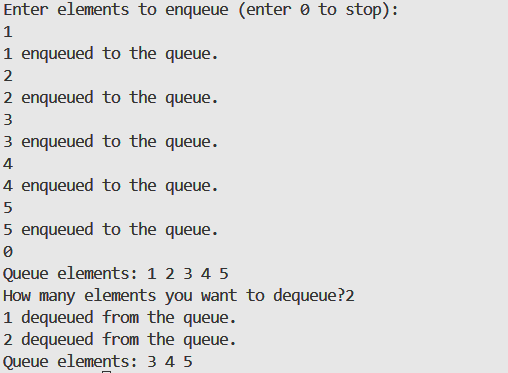
printf("the elements does not exist");

}

display();

return 0;

}



**Assignment 4:- To implement insert and deletion operations in linked list.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

return newNode;

}

void insertAtBeginning(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

newNode->next = \*head;

\*head = newNode;

}

void insertAtEnd(struct Node\*\* head, int value) {

struct Node\* newNode = createNode(value);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

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

if (\*head == NULL) {

printf("List is empty. Cannot delete.\n");

return;

}

struct Node\* current = \*head;

struct Node\* prev = NULL;

while (current != NULL && current->data != value) {

prev = current;

current = current->next;

}

if (current == NULL) {

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

return;

}

if (prev == NULL) {

\*head = current->next;

} else {

prev->next = current->next;

}

free(current);

}

void display(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

void main() {

struct Node\* head = NULL;

int choice, value;

do {

printf("Linked List Menu:\n");

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

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

printf("3. Delete a node\n");

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

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%d", &value);

insertAtBeginning(&head, value);

break;

case 2:

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

scanf("%d", &value);

insertAtEnd(&head, value);

break;

case 3:

printf("Enter a value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

break;

case 4:

printf("Linked list contents: ");

display(head);

break;

case 5:

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

break;

default:

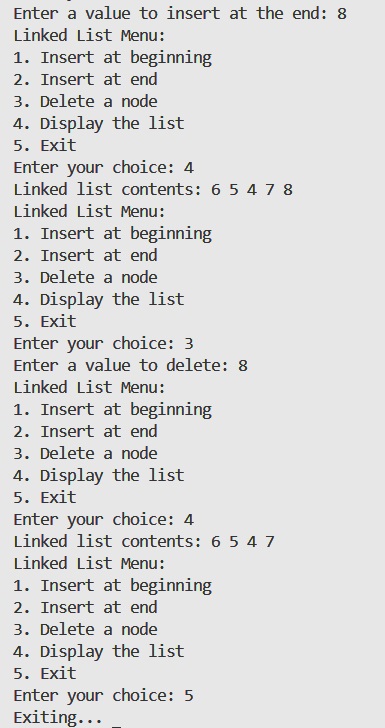
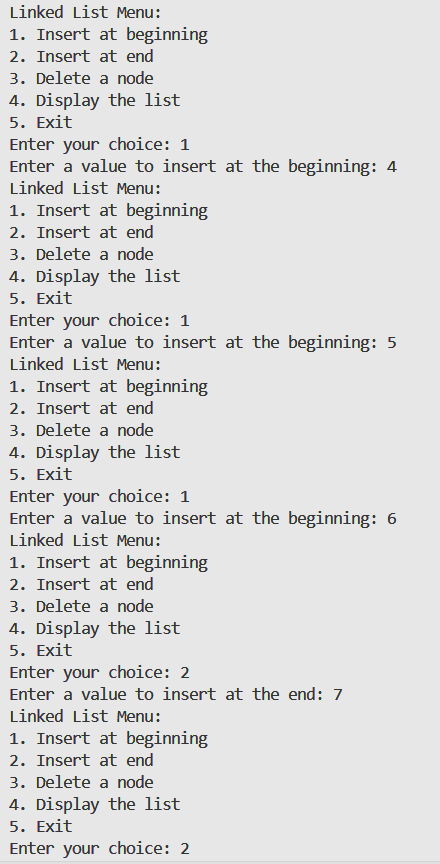
printf("Invalid choice. Try again.\n");

}

}

}

**OUTPUT**



**Assignment 5:-To implement stack and queue using linked list. Assignment 5A:- To implement stack using linked list**  **INPUT CODE**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int value) {

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

if (!newNode) {

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

exit(1);

}

newNode->data = value;

newNode->next = NULL;

return newNode;

}

void push(struct Node\*\* top, int value) {

struct Node\* newNode = createNode(value);

newNode->next = \*top;

\*top = newNode;

}

int pop(struct Node\*\* top) {

if (\*top == NULL) {

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

exit(1);

}

struct Node\* temp = \*top;

int poppedValue = temp->data;

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

return poppedValue;

}

void display(struct Node\* top) {

if (top == NULL) {

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

return;

}

printf("Stack elements:\n");

while (top != NULL) {

printf("%d\n", top->data);

top = top->next;

}

}

int main() {

struct Node\* stackTop = NULL;

int choice, value;

while (1) {

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

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stackTop, value);

break;

case 2:

printf("Popped element: %d\n", pop(&stackTop));

break;

case 3:

display(stackTop);

break;

case 4:

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice. Try again.\n");

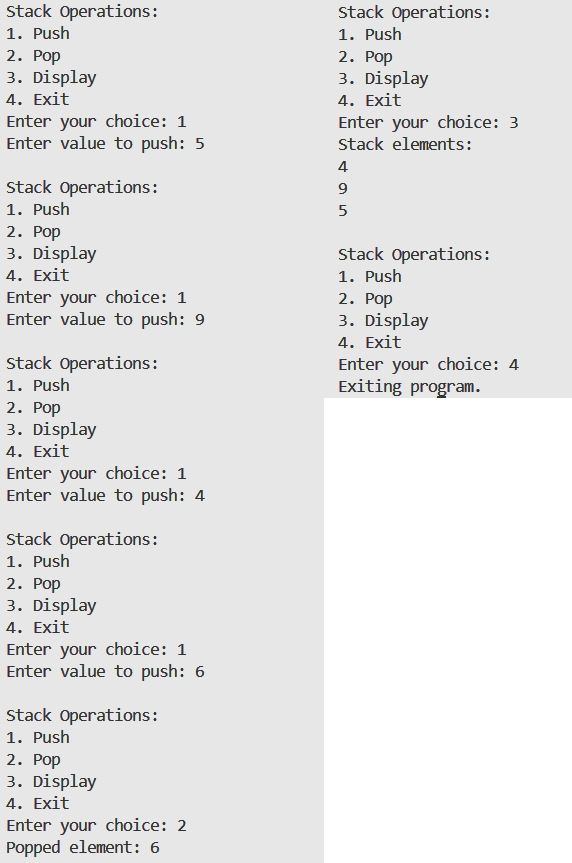
}

}

return 0;

}

**OUTPUT**

****

**Assignment 5:-To implement stack and queue using linked list. Assignment 5B:- To implement Queue using linked list**  **INPUT CODE**

#include <stdio.h>

#include <stdlib.h>

struct QNode {

int data;

struct QNode\* next;

};

struct QNode\* createNode(int value) {

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

if (!newNode) {

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

exit(1);

}

newNode->data = value;

newNode->next = NULL;

return newNode;

}

struct Queue {

struct QNode\* front;

struct QNode\* rear;

};

void enQueue(struct Queue\* q, int value) {

struct QNode\* temp = createNode(value);

if (q->rear == NULL) {

q->front = q->rear = temp;

return;

}

q->rear->next = temp;

q->rear = temp;

}

int deQueue(struct Queue\* q) {

if (q->front == NULL) {

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

exit(1);

}

struct QNode\* temp = q->front;

int dequeuedValue = temp->data;

q->front = q->front->next;

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

return dequeuedValue;

}

void display(struct Queue\* q) {

if (q->front == NULL) {

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

return;

}

struct QNode\* current = q->front;

printf("Queue elements:\n");

while (current != NULL) {

printf("%d\n", current->data);

current = current->next;

}

}

void main() {

struct Queue q;

q.front = q.rear = NULL;

int choice, value;

while (1) {

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

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\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(&q, value);

break;

case 2:

if (q.front != NULL) {

int dequeuedValue = deQueue(&q);

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

} else {

printf("Queue is empty. Cannot dequeue.\n");

}

break;

case 3:

display(&q);

break;

case 4:

printf("Exiting program.\n");

exit(0);

default:

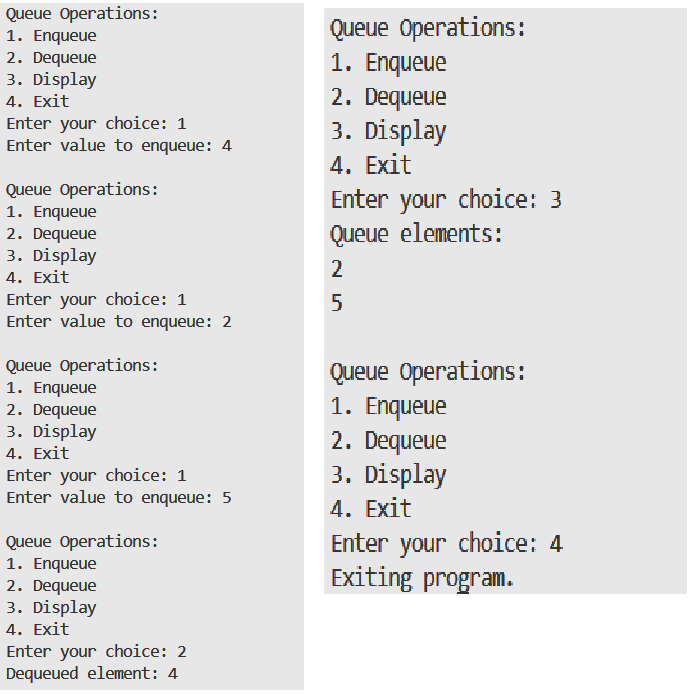
printf("Invalid choice. Try again.\n");

}

}

}

**OUTPUT**

****

**Assignment -6(a)**

**To implement Bubble Sort , Insertion sort and Selection sort**

#include <stdio.h>

#include <stdlib.h>

void Bubble\_sort(int a[], int n)

{

    int temp;

    for (int i = 0; i < n - 1; i++)

    {

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

        {

            if (a[j] > a[j + 1])

            {

                temp = a[j];

                a[j] = a[j + 1];

                a[j + 1] = temp;

            }

        }

        printf("\niteration %d:-", i + 1);

        display(a, n);

    }

}

void Insertion\_sort(int a[], int n)

{

    int key;

    int j;

    for (int i = 1; i <= n - 1; i++)

    {

        key = a[i];

        j = i - 1;

        while (j >= 0 && a[j] > key)

        {

            a[j + 1] = a[j];

            j--;

        }

        a[j + 1] = key;

        printf("\niteration %d:-", i + 1);

        display(a, n);

    }

}

void Selection\_sort(int a[], int n)

{

    for (int i = 0; i < n - 1; i++)

    {

        int temp, minind = i;

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

        {

            if (a[j] < a[minind])

            {

                minind = j;

            }

        }

        temp = a[i];

        a[i] = a[minind];

        a[minind] = temp;

        printf("\niteration %d:-", i + 1);

        display(a, n);

    }

}

void display(int a[], int n)

{

    for (int i = 0; i < n; i++)

    {

        printf("\t%d", a[i]);

    }

}

int main()

{

    int a[5] = {13, 3, 11, 8, 2};

    int b[5] = {9, 45, 23, 21, 25};

    int c[5] = {11, 14, 13, 19, 17};

    int n = 5;

    printf("\n\nunsorted array:-");

    display(a, n);

    printf("\nBubble sort:");

    Bubble\_sort(a, n);

    printf("\n\nsorted array:-");

    display(a, n);

    printf("\n\nunsorted array:-");

    display(b, n);

    printf("\nInsertion sort:");

    Insertion\_sort(b, n);

    printf("\n\nsorted array:-");

    display(b, n);

    printf("\n\nunsorted array:-");

    display(c, n);

    printf("\nSelection sort:");

    Selection\_sort(c, n);

    printf("\n\nsorted array:-");

    display(c, n);

    return 0;

}

**Output:-**

**Assignment – 7**

**To implement graph**

#include<stdio.h>

#include<stdlib.h>

struct adj

{

    int nodeno;

    struct adj \*next;

};

void insert(struct adj \*G[],int,int);

void display(struct adj \*G[],int);

void main()

{

    struct adj \*G[10];

    int n,nb,i,j;

    printf("\n How many vertices are there in graph:");

    scanf("%d",&n);

    for(i=0;i<n;i++)

    {

        G[i]=NULL;

        printf("\n How many neighbours are there of node:%d=",i);

        scanf("%d",&nb);

        if(nb>0)

        insert(G,i,nb);

    }

    display(G,n);

}

void insert(struct adj \*newlist[10],int node,int nb)

{

    int a,i=1;

    struct adj \*temp,\*temp1;

    temp=(struct adj\*)malloc(sizeof(struct adj));

    printf("\n enter neighbour no=");

    scanf("%d",&a);

    temp->nodeno=a;

    temp->next=NULL;

    newlist[node]=temp;

    for(i=1;i<nb;i++)

    {

        temp1=(struct adj\*)malloc(sizeof(struct adj));

        printf("\n enter neighbour no=");

        scanf("%d",&a);

        temp1->nodeno=a;

        temp1->next=NULL;

temp->next=temp1;

        temp=temp1;

    }

}

void display(struct adj \*l[10],int n)

{

    int i,j;

    struct adj\* temp;

    printf("\n the graph is:");

    for(i=0;i<n;i++)

    {

        printf("\n\n The current node=%d",i);

        if(l[i]==NULL)

        printf("\n This node has no neighbour");

        else

        {

            temp=l[i];

            printf("\n The neighbours are:");

            j=0;

            while(temp!=NULL)

            {

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

                j++;

                temp=temp->next;

            }

            printf("\n It has %d neighbours",j);

        }

    }

}

**Assignment 8:- To implement BFS and DFS using linked list.**

**Implementing BFS**

#include <stdio.h>

#include <stdlib.h>

#define SIZE 40

struct queue {

int items[SIZE];

int front;

int rear;

};

struct queue\* createQueue();

void enqueue(struct queue\* q, int);

int dequeue(struct queue\* q);

void display(struct queue\* q);

int isEmpty(struct queue\* q);

void printQueue(struct queue\* q);

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int);

struct Graph {

int numVertices;

struct node\*\* adjLists;

int\* visited;

};

void bfs(struct Graph\* graph, int startVertex) {

struct queue\* q = createQueue();

graph->visited[startVertex] = 1;

enqueue(q, startVertex);

while (!isEmpty(q)) {

printQueue(q);

int currentVertex = dequeue(q);

printf("Visited %d\n", currentVertex);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (graph->visited[adjVertex] == 0) {

graph->visited[adjVertex] = 1;

enqueue(q, adjVertex);

}

temp = temp->next;

}

}

}

struct node\* createNode(int v) {

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

struct queue\* createQueue() {

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

q->front = -1;

q->rear = -1;

return q;

}

int isEmpty(struct queue\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

void enqueue(struct queue\* q, int value) {

if (q->rear == SIZE - 1)

printf("\nQueue is Full!!");

else {

if (q->front == -1)

q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct queue\* q) {

int item;

if (isEmpty(q)) {

printf("Queue is empty");

item = -1;

} else {

item = q->items[q->front];

q->front++;

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

printf("Resetting queue ");

q->front = q->rear = -1;

}

}

return item;

}

void printQueue(struct queue\* q) {

int i = q->front;

if (isEmpty(q)) {

printf("Queue is empty");

} else {

printf("\nQueue contains \n");

for (i = q->front; i < q->rear + 1; i++) {

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

}

}

}

int main() {

struct Graph\* graph = createGraph(6);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 1, 4);

addEdge(graph, 1, 3);

addEdge(graph, 2, 4);

addEdge(graph, 3, 4);

bfs(graph, 0);

return 0;

}

**A screenshot of a computer

Description automatically generated**

**Assignment 8:- To implement BFS and DFS using linked list.**

**Implementing DFS**

#include <stdio.h>

#include <stdlib.h>

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

struct node\*\* adjLists;

};

void DFS(struct Graph\* graph, int vertex) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

printf("Visited %d \n", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0) {

DFS(graph, connectedVertex);

}

temp = temp->next;

}

}

struct node\* createNode(int v) {

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph) {

int v;

for (v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

printf("\n Adjacency list of vertex %d\n ", v);

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

return 0;

}

**A screenshot of a computer program

Description automatically generated OUTPUT**

**Assignment 9:- To implement merge sort.**

#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

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

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

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

int i = 0, j = 0, k = left;

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

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

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

else

arr[k++] = R[j++];

}

while (i < n1)

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

while (j < n2)

arr[k++] = R[j++];

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

printf("Intermediate array: ");

for (int i = left; i <= right; i++)

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

printf("\n");

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

mergeSort(arr, 0, n - 1);

printf("Sorted array:\n");

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

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

return 0;

}

**OUTPUT**

**A screenshot of a computer program

Description automatically generated**

**To Implement Heap Sort:-**

#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);

printf("Intermediate steps:\n");

for (int i = 0; i < N; i++)

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

printf("\n");

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

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

heapify(arr, i, 0);

printf("Intermediate steps:\n");

for (int j = 0; j <= i; j++)

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

printf("\n");

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

heapSort(arr, n);

printf("Sorted array:\n");

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

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

return 0;

}

**OUTPUT**

A screenshot of a computer

Description automatically generated

**Assignment 10:- To Implement Matrix Multiplication By Strassen’s Algorithm.**

**INPUT CODE**

#include<stdio.h>

int main(){

int a[2][2],b[2][2],c[2][2],i,j;

int m1,m2,m3,m4,m5,m6,m7;

printf("Enter the 4 elements of first matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("Enter the 4 elements of second matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("\nThe first matrix is\n");

for(i=0;i<2;i++){

printf("\n");

for(j=0;j<2;j++)

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

}

printf("\nThe second matrix is\n");

for(i=0;i<2;i++){

printf("\n");

for(j=0;j<2;j++)

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

}

m1= (a[0][0] + a[1][1])\*(b[0][0]+b[1][1]);

m2= (a[1][0]+a[1][1])\*b[0][0];

m3= a[0][0]\*(b[0][1]-b[1][1]);

m4= a[1][1]\*(b[1][0]-b[0][0]);

m5= (a[0][0]+a[0][1])\*b[1][1];

m6= (a[1][0]-a[0][0])\*(b[0][0]+b[0][1]);

m7= (a[0][1]-a[1][1])\*(b[1][0]+b[1][1]);

c[0][0]=m1+m4-m5+m7;

c[0][1]=m3+m5;

c[1][0]=m2+m4;

c[1][1]=m1-m2+m3+m6;

printf("\n\n\n Multiplication result using Strassen's algorithm: \n");

for(i=0;i<2;i++){

printf("\n");

for(j=0;j<2;j++)

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

}

return 0;

}

**OUTPUT**

**A white paper with black text

Description automatically generated**

**Assignment 11:-To implement kruskal’s algorithm for minimum spanning tree.**

#include <stdio.h>

struct edge

{

int u, v, w;

};

void kruskal(int g[][10], int n);

void unionset(int belongs[], int n, int c1, int c2);

void sort(struct edge e[], int n);

void display(struct edge mst[], int n);

int main() {

int n, i, j, g[10][10];

printf("\nEnter the number of vertices: ");

scanf("%d", &n);

printf("\nEnter the adjacency matrix:\n");

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

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

{

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

}

}

kruskal(g, n);

return 0;

}

void sort(struct edge e[], int n)

{

int i, j;

struct edge temp;

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

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

{

if (e[j].w > e[j + 1].w)

{

temp = e[j];

e[j] = e[j + 1];

e[j + 1] = temp;

}

}

}

}

void unionset(int belongs[], int n, int c1, int c2)

{

int i;

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

{

if (belongs[i] == c2)

{

belongs[i] = c1;

}

}

}

void kruskal(int g[][10], int n)

{

int belongs[10], i, j, k = 0;

struct edge e[100], mst[10];

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

{

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

{

if (g[i][j] != 0)

{

e[k].u = i;

e[k].v = j;

e[k].w = g[i][j];

k++;

}

}

}

sort(e, k);

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

{

belongs[i] = i;

}

int mstSize = 0;

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

{

if (belongs[e[i].u] != belongs[e[i].v])

{

mst[mstSize++] = e[i];

unionset(belongs, n, belongs[e[i].u], belongs[e[i].v]);

}

}

display(mst, mstSize);

}

void display(struct edge mst[], int n) {

int i, cost = 0;

printf("\nMinimum Spanning Tree edges:\n");

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

printf("%d -- %d == %d\n", mst[i].u, mst[i].v, mst[i].w);

cost += mst[i].w;

}

printf("\nTotal cost of the MST: %d\n", cost);

}

**OUTPUT**

**A screenshot of a computer

Description automatically generated**