**Assignment No 1**

**Title**: Write a program to represent sparse matrix using array and perform simple and fast Transpose.

**Code:**

//SimpleT.c

#include<stdio.h>

void printsparse(int[20][20]);

void readsparse(int[20][20]);

void transpose(int[][3],int[][3]);

int main()

{

int b1[20][20],b2[20][20],m,n;

printf("Enter the Size of the Matrix (rows x columns):");

scanf("%d %d",&m,&n);

b1[0][0]=m;

b1[0][1]=n;

readsparse(b1);

transpose(b1,b2);

printsparse(b2);

}

void readsparse(int b[20][20])

{

int i,t;

printf("Enter the number of non-zero elements:");

scanf("%d",&t);

b[0][2]=t;

for(i=1 ;i<=t;i++)

{

printf("Enter the Three values of matrix(row,col,value):");

scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);

}

}

void printsparse(int b[20][20])

{

int i,n;

n=b[0][2];

printf("\nAfter Transpose");

printf("\nRows \tColumns \tValues");

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

{

printf("\n%d \t%d \t%d",b[i][0],b[i][1],b[i][2]);

}

}

void transpose(int b1[][3],int b2[][3])

{

int i,j,k,n;

b2[0][0]=b1[0][1];

b2[0][1]=b1[0][0];

b2[0][2]=b1[0][2];

k=1;

n=b1[0][2];

n=b1[0][2];

for(i=0;i<b1[0][1];i++)

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

{

if(i==b1[j][1])

{

b2[k][0]=i;

b2[k][1]=b1[j][0];

b2[k][2]=b1[j][2];

k++;

}

}

}

//FastT.c

#include<stdio.h>

int main(){

int A[][3]={{4,4,5},{0,0,8},{0,3,5},{1,2,30},{1,3,26},{3,1,28}};

int t=sizeof(A)/sizeof(int)/3;

int i,j;

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

for(j=0;j<3;j++){

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

}

printf("\n");

}

int limit=A[0][1],k=0;

int freq[limit],start[limit];

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

freq[i]=0;

start[i]=0;

}

while(k!=limit){

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

if(k==A[i][1]){

freq[k]+=1;

}

}

k++;

}

printf("\n--------Frequency---------\n");

for(j=0;j<limit;j++){

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

}

printf("\n---------Index------------\n");

if(freq[0]>0)

start[0]=1;

else

start[0]=0;

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

start[i]=start[i-1]+freq[i-1];

}

for(j=0;j<limit;j++){

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

}

int B[t][3];

B[0][0]=A[0][1];

B[0][1]=A[0][0];

B[0][2]=A[0][2];

int in=0;

printf("\n------Fast Transpose-------\n");

int up=1;

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

int temp=A[i][1];

B[start[temp]][0]=temp;

B[start[temp]][1]=A[i][0];

B[start[temp]][2]=A[i][2];

start[temp]+=1;

}

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

for(j=0;j<3;j

printf("%d ",B[i][j]);

}

printf("\n");

}

}

**Assignment no 2**

**Title:** Write a menu driven program to perform following operations on singly linked list: Create, Insert, Delete, reverse, search, count and Display

**Code :**

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

int info;

struct node \*next;

}NODE;

NODE\* createlist(NODE \*list);

void Display (NODE \*list);

void search (NODE \*list);

NODE\* insertbeg(NODE \*list);

NODE\* insertbetween(NODE \*list);

NODE\* insertlast(NODE \*list);

NODE\* delvalue(NODE \*list);

void reverse(NODE \*list);

NODE\* Count (NODE \*list);

//MAIN FUNCTION

void main ()

{

printf(">>Singly Linked List<<");

NODE\*list=NULL, \*temp;

int ch,n,pos;

do

{

printf("\n1. Create:");

printf("\n2. Display");

printf("\n3.Insert AtFirst");

printf("\n4.Insert AtMiddle");

printf("\n5.Insert AtLast");

printf("\n6. Delete");

printf("\n7. Search");

printf("\n8.Reverse");

printf("\n9.Count");

printf("\n10. Exit\n");

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

scanf("%d", &ch);

switch(ch)

{

case 1:

list=createlist (list);

break;

case 2:

Display (list);

break;

case 3:

list=insertbeg (list);

break;

case 4:

list=insertbetween (list);

break;

case 5:

list=insertlast (list);

break;

case 6:

delvalue(list);

break;

case 7:

search (list);

break;

case 8:

reverse(list);

break;

case 9:

list = Count(list);

break;

case 10:printf("\_END\_");

break;

}

}while(ch != 10);

}

//CREATE FUNCTION

NODE\* createlist (NODE \*list)

{

int n,count;

NODE \*temp, \*newnode;

printf ("\nHow many nodes you want to enter ? \n");

scanf("%d" ,&n);

for(count=1 ; count<=n; count++)

{

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->next=NULL;

printf( "Enter the node data:- " );

scanf ("%d",&newnode->info);

if (list==NULL)

{

list=temp=newnode;

}

else

{

temp->next=newnode;

temp=newnode;

}

}

return list;

}

//DISPLAY

void Display (NODE \*list)

{

NODE \*temp=list;

if(temp == NULL)

{

printf("List is Empty...\n");

}

else

{

while(temp!=NULL)

{

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

printf("-->");

temp=temp->next ;

}

printf("NULL \n");

}

}

//Insert AT BEGINING

NODE \* insertbeg(NODE \*list)

{

int n;

printf("Enter the node you want to insert at first position:");

scanf("%d", &n);

NODE \*newnode;

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->info=n;

newnode->next=list;

list=newnode;

return list;

}

//INSERT IN BETWEEN

NODE \* insertbetween(NODE \*list)

{

NODE \*newnode, \*temp=list;

int n,i,pos;

printf("Enter the node data and position you want to insert between the node: \t");

scanf("\n%d", &n);

scanf("\n%d", &pos);

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->next=NULL;

newnode->info=n;

for(i=1; i<pos-1&&temp->next!=NULL; i++)

temp=temp->next;

newnode->next=temp->next;

temp->next=newnode;

return list;

}

//INSERT AT LAST

NODE \* insertlast(NODE \*list)

{

int n;

printf("Enter the node data you want to insert at last position: \t");

scanf("%d", &n);

NODE \*newnode, \*temp;

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->info=n;

newnode->next=NULL;

for(temp=list; temp->next!=NULL; temp=temp->next);

temp->next=newnode;

return list;

}

//SEARCH

void search (NODE \*list)

{

int num, flag=0;

NODE \*temp;

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

scanf("%d", &num);

for(temp=list; temp!=NULL; temp=temp->next)

{

if(temp->info==num)

{

printf(">>%d is Found<<\n",num);

flag=1;

exit;

}

}

if(flag==0)

printf(">>%d is not found<<\n", num);

}

NODE\* delvalue(NODE \*list)

{

NODE \*temp=list, \*temp1;

int num;

printf("Enter node data to delete that node \t");

scanf("%d", &num);

if(list->info==num)

{

list=list->next;

free (temp);

return list;

exit;

}

for(temp=list; temp->next!=NULL; temp=temp->next)

if(temp->next->info==num)

{

temp1=temp->next;

temp->next=temp1->next;

free (temp1);

return list;

exit;

}

printf(">>Element is not found<<\n");

return list;

}

void reverse(NODE \*list)

{

struct node \*t1, \*t2,\*temp;

t1=t2=NULL;

if(list == NULL)

printf("List is empty");

else

{

while(list != NULL)

{

t2=list->next;

list->next = t1;

t1=list;

list=t2;

}

list=t1;

temp=list;

printf("Reversed Link List\n");

while(temp!=NULL)

{

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

printf("-->");

temp=temp->next ;

}

printf("NULL \n");

}

}

NODE\* Count (NODE \*list)

{

int cnt = 0;

NODE \*temp=list;

while(temp!=NULL)

{

temp = temp->next;

cnt++;

}

printf("Total Node in linked list is --> %d \n",cnt);

}

**Assignment No 3**

**Title:** Write a menu driven program which will maintain a list of car models,their price, name of the manufacture, engine capacity etc. as a doubly linked list. The menu should make provisions for inserting information pertaining to new car models, delete obsolete models, update data such as price besides answering queries such as listing all car models within a price range specified by the client and listing all details given a car model.

**Code :**

#include<stdio.h>

#include<stdlib.h>

typedef struct node{

char cname[10];

char cmodel[10];

int year;

int cprice;

struct node \*llink, \*rlink;

}node;

node \*create();

void display(node \*);

void insert();

void update(node \*);

node\* head=NULL;

void removeNode();

void find\_within\_range();

int main()

{

int ch;

do{

printf("-------Car\_Dekho.com-----");

printf("\n1.Create \n2.Insert \n3.Display \n4.Update Price \n5.Find Within Range \n6.Delete \n7.Exit\n\n");

printf("Enter Your Choice :");

scanf("%d",&ch);

switch(ch)

{

case 1:

head = create();

break;

case 2:

insert();

break;

case 3:

display(head);

break;

case 4:

update(head);

break;

case 5:

find\_within\_range(head);

break;

case 6:

removeNode(head);

break;

case 7:

break;

default:

printf("Enter Valid Choice!");

break;

}

}while(ch!=7);

return 0;

}

node\* create()

{

node \*head=NULL, \*prev, \*newnode;

char ch;

do{

newnode = (node\*)malloc(sizeof(node));

if(newnode==NULL)

{

printf("Can't Allocate Memory");

return 0;

}

printf("\nEnter Car Company Name :");

scanf("%s",&(newnode->cname));

printf("Enter Car Model :");

scanf("%s",&(newnode->cmodel));

printf("Enter Car Price :");

scanf("%d",&(newnode->cprice));

printf("Enter Car Year :");

scanf("%d",&(newnode->year));

newnode->llink = NULL;

newnode->rlink = NULL;

if(head==NULL)

{

head = newnode;

prev = head;

}

else

{

prev->rlink = newnode;

newnode->llink = prev;

prev = newnode;

}

printf("\nWhether To Add Next Node (Y/N) :");

scanf("%s",&ch);

}while(ch=='y');

}

void insert()

{

node \*curr = head, \*newnode;

int pos,cnt=1;

newnode = (node\*)malloc(sizeof(node));

printf("\nEnter Car Company Name :");

scanf("%s",&(newnode->cname));

printf("Enter Car Model :");

scanf("%s",&(newnode->cmodel));

printf("Enter Car Price :");

scanf("%d",&(newnode->cprice));

printf("Enter Car Year :");

scanf("%d",&(newnode->year));

newnode->llink = NULL;

newnode->rlink = NULL;

printf("\nEnter Position for Node :");

scanf("%d",&pos);

if(pos==1)

{

newnode->rlink = head;

head->llink = newnode;

head = newnode;

}

else

{

while(cnt<pos-1)

{

curr = curr->rlink;

cnt++;

}

if(curr->rlink!=NULL)

{

(curr->rlink)->llink = newnode;

}

newnode->rlink = curr->rlink;

newnode->llink = curr;

curr->rlink = newnode;

}

}

void display(node \*head)

{

node \*curr = head;

//printf("\nYour Entered Linked List\n");

printf("\n\nCompany \tModel \tPrice \tYear\n");

printf("----------------------------------\n");

while(curr!=NULL)

{

printf("%s\t%s\t%d\t%d\n",curr->cname,curr->cmodel,curr->cprice,curr->year);

curr = curr->llink;

}

}

void removeNode(node \*head)

{

node \*curr = head;

int cnt=1;

char model[20];

printf("Enter Car Model to be Deleted :");

scanf("%s",&model);

while(curr!=NULL)

{

if(strcmp(curr->cmodel,model)==0)

{

if(cnt==1)

{

head = head->rlink;

head->llink = NULL;

free(curr);

}

else{

if(curr->rlink!=NULL)

{

(curr->rlink)->llink = curr->llink;

}

(curr->llink)->rlink = curr->rlink;

free(curr);

}

}

cnt++;

curr = curr->rlink;

}

}

void update(node \*head)

{

node \*curr = head;

int price;

char model[20];

printf("\nEnter Car Model Where You Want To Change Car Price :");

scanf("%s",&model);

while(curr!=NULL)

{

if(strcmp(curr->cmodel,model)==0)

{

printf("\nEnter New Car Price :");

scanf("%d",&price);

curr->cprice = price;

break;

}

curr=curr->rlink;

}

}

void find\_within\_range(node \*head)

{

node \*curr = head;

int lprice,bprice;

printf("\nEnter Lowest Price :");

scanf("%d",&lprice);

printf("\nEnter Budget Price :");

scanf("%d",&bprice);

printf("\nAll Cars Within Entered Range :\n");

printf("\nCompany\tModel\tPrice\tYear\n");

while(curr!=NULL)

{

if(curr->cprice>=lprice && curr->cprice<=bprice)

{

printf("%s\t%s\t%d\t%d\n",curr->cname,curr->cmodel,curr->cprice,curr->year);

}

curr=curr->rlink;

}

}

**Assignment no 4**

**Title**: Write a program to implement stack as an ADT. Use this ADT to perform expression conversion and evaluation. (Infix – Postfix)

**Code** :

#include<stdio.h>

#define MAXSIZE 20

typedef struct

{

int data[MAXSIZE];

int top;

}STACK;

void initstack(STACK \*ps)

{

ps->top=-1;

}

int isempty(STACK \*ps)

{

return(ps->top==-1);

}

int isfull(STACK \*ps)

{

return(ps->top==MAXSIZE-1);

}

int pop(STACK \*ps)

{

return(ps->data[ps->top--]);

}

int push(STACK \*ps,int n)

{

ps->data[++ps->top]=n;

}

int peek(STACK \*ps)

{

return ps->data[ps->top];

}

int priority (char ch)

{

switch(ch)

{

case '(': return 0;

case '+':

case '-':return 1;

case '\*':

case '/':

case '%':return 2;

case '^':

case '$':return 3;

}

return 0;

}

void convert(char infix[20], char post[20])

{

int i,j=0;

char ch,ch1;

STACK s;

initstack(&s);

for(i=0;infix[i]!='\0';i++)

{

ch=infix[i];

switch(ch)

{

case '(': push(&s,ch);

break;

case '+':

case '-':

case '\*':

case '/':

case '%':

case '^':

case '$':

while(!isempty(&s) && (priority(peek(&s))>=priority(ch)))post[j++]=pop(&s);

push(&s,ch);

break;

case ')':

while ((ch1=pop(&s))!='(')

post[j++]=ch1;

break;

default:post[j++]=ch;

}

}

while(!isempty(&s))

post[j++]=pop(&s);

post[j]='\0';

}

main()

{

char infix[20],postfix[20];

printf("\nEnter the infix expression:");

scanf("%s",infix);

convert(infix,postfix);

printf("\n The postfix string is %s",postfix);

}

**Assignment no 5**

**Title:** Write a program to implement circular queue using arrays.

**Code :**

#include <stdio.h>

#define MAX\_SIZE 5

struct CircularQueue {

int items[MAX\_SIZE];

int front, rear;

};

void initializeQueue(struct CircularQueue\* queue) {

queue->front = -1;

queue->rear = -1;

}

int isEmpty(struct CircularQueue\* queue) {

return (queue->front == -1 && queue->rear == -1);

}

int isFull(struct CircularQueue\* queue) {

return (queue->front == (queue->rear + 1) % MAX\_SIZE);

}

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

if (isFull(queue)) {

printf("Queue is full. Cannot enqueue %d\n", value);

return;

}

if (isEmpty(queue)) {

queue->front = queue->rear = 0;

} else {

queue->rear = (queue->rear + 1) % MAX\_SIZE;

}

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

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

}

void dequeue(struct CircularQueue\* queue) {

if (isEmpty(queue)) {

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

return;

}

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

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

// Queue has only one element, reset front and rear

initializeQueue(queue);

} else {

queue->front = (queue->front + 1) % MAX\_SIZE;

}

}

void display(struct CircularQueue\* queue) {

if (isEmpty(queue)) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

int i = queue->front;

do {

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

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

} while (i != (queue->rear + 1) % MAX\_SIZE);

printf("\n");

}

int main() {

struct CircularQueue queue;

initializeQueue(&queue);

enqueue(&queue, 1);

enqueue(&queue, 2);

enqueue(&queue, 3);

enqueue(&queue, 4);

display(&queue);

dequeue(&queue);

display(&queue);

enqueue(&queue, 5);

enqueue(&queue, 6);

display(&queue);

return 0;

}

**Assignment no 6**

**Tittle :** Write a program to create binary tree. Find height of the tree and print leaf nodes. Find mirror image, print original and mirror image using level-wise printing

**code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int key;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int key) {

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

newNode->key = key;

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

return newNode;

}

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

if (root == NULL) {

return createNode(key);

} else {

if (key < root->key) {

root->left = insert(root->left, key);

} else {

root->right = insert(root->right, key);

}

}

return root;

}

int height(struct Node\* root) {

if (root == NULL) {

return 0;

} else {

int leftHeight = height(root->left);

int rightHeight = height(root->right);

return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);

}

}

void printLeafNodes(struct Node\* root) {

if (root != NULL) {

if (root->left == NULL && root->right == NULL) {

printf("%d ", root->key);

} else {

printLeafNodes(root->left);

printLeafNodes(root->right);

}

}

}

void mirrorTree(struct Node\* root) {

if (root != NULL) {

struct Node\* temp = root->left;

root->left = root->right;

root->right = temp;

mirrorTree(root->left);

mirrorTree(root->right);

}

}

void printLevelOrder(struct Node\* root) {

if (root == NULL) {

return;

}

struct Node\* queue[100];

int front = -1, rear = -1;

queue[++rear] = root;

while (front != rear) {

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

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

if (current->left != NULL) {

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

}

if (current->right != NULL) {

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

}

}

}

int main() {

struct Node\* root = NULL;

int keys[] = {8, 3, 10, 1, 6, 9, 14, 4, 7, 13};

int i;

for (i = 0; i < 10; i++) {

root = insert(root, keys[i]);

}

printf("Original tree:\n");

printLevelOrder(root);

printf("\nHeight of the tree: %d\n", height(root));

printf("Leaf nodes: ");

printLeafNodes(root);

mirrorTree(root);

printf("\n\nMirror image:\n");

printLevelOrder(root);

return 0;

}

**Assignment no: 7**

**Title:** Write a program that reads a list of names and telephone numbers from user and insert into a BST tree. Once the tree has been built, present the user with a menu that allows him to search the list for a specified name, insert new name, delete an existing name or print the entire phone list.

**Code**:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure to represent a node in the BST

struct Node {

char name[50];

char phoneNumber[15];

struct Node \*left, \*right;

};

// Function to create a new node

struct Node\* createNode(char name[], char phoneNumber[]) {

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

strcpy(newNode->name, name);

strcpy(newNode->phoneNumber, phoneNumber);

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

return newNode;

}

// Function to insert a new node into the BST

struct Node\* insert(struct Node\* root, char name[], char phoneNumber[]) {

if (root == NULL) {

return createNode(name, phoneNumber);

}

if (strcmp(name, root->name) < 0) {

root->left = insert(root->left, name, phoneNumber);

} else if (strcmp(name, root->name) > 0) {

root->right = insert(root->right, name, phoneNumber);

}

return root;

}

// Function to search for a name in the BST

struct Node\* search(struct Node\* root, char name[]) {

if (root == NULL || strcmp(root->name, name) == 0) {

return root;

}

if (strcmp(name, root->name) < 0) {

return search(root->left, name);

} else {

return search(root->right, name);

}

}

// Function to find the minimum value node in a BST

struct Node\* minValueNode(struct Node\* node) {

struct Node\* current = node;

while (current->left != NULL) {

current = current->left;

}

return current;

}

// Function to delete a node from the BST

struct Node\* deleteNode(struct Node\* root, char name[]) {

if (root == NULL) {

return root;

}

if (strcmp(name, root->name) < 0) {

root->left = deleteNode(root->left, name);

} else if (strcmp(name, root->name) > 0) {

root->right = deleteNode(root->right, name);

} else {

// Node with only one child or no child

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

// Node with two children, get the inorder successor

struct Node\* temp = minValueNode(root->right);

// Copy the inorder successor's content to this node

strcpy(root->name, temp->name);

strcpy(root->phoneNumber, temp->phoneNumber);

// Delete the inorder successor

root->right = deleteNode(root->right, temp->name);

}

return root;

}

// Function to print the phone list using inorder traversal

void printPhoneList(struct Node\* root) {

if (root != NULL) {

printPhoneList(root->left);

printf("Name: %s, Phone Number: %s\n", root->name, root->phoneNumber);

printPhoneList(root->right);

}

}

int main() {

struct Node\* root = NULL;

int choice;

char name[50], phoneNumber[15];

do {

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

printf("1. Search for a name\n");

printf("2. Insert a new name\n");

printf("3. Delete an existing name\n");

printf("4. Print the entire phone list\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

scanf("%s", name);

struct Node\* searchResult = search(root, name);

if (searchResult != NULL) {

printf("Name: %s, Phone Number: %s\n", searchResult->name, searchResult->phoneNumber);

} else {

printf("Name not found.\n");

}

break;

case 2:

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

scanf("%s", name);

printf("Enter the phone number: ");

scanf("%s", phoneNumber);

root = insert(root, name, phoneNumber);

printf("Name inserted successfully.\n");

break;

case 3:

printf("Enter the name to delete: ");

scanf("%s", name);

root = deleteNode(root, name);

printf("Name deleted successfully.\n");

break;

case 4:

printf("Phone List:\n");

printPhoneList(root);

break;

case 5:

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

break;

default:

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

}

} while (choice != 5);

return 0;

}

**Assignment no: 8**

**Title:** Write a program to create graph, use the map of any city as the graph. Represent graph using adjacency list/adjacency matrix and perform Depth First Search and Breadth First Search.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

//Stack Structure

typedef struct{

int items[MAX];

int top;

}Stack;

void initialize(Stack \*s)

{

s->top=-1;

}

int isEmpty(Stack \*s)

{

return s->top == -1;

}

void push(Stack \*s, int value)

{

if(s->top==MAX-1)

{

printf("Stack Overflow\n");

exit(EXIT\_FAILURE);

}

s->items[++(s->top)]=value;

}

int pop(Stack \*s)

{

if(isEmpty(s))

{

printf("Stack Underflow\n");

exit(EXIT\_FAILURE);

}

return s->items[(s->top)--];

}

//Stack structure end...

//DFS function

void DFS(int graph[MAX][MAX],int vertices,int startVertex)

{

Stack stack;

initialize(&stack);

int visited[MAX]={0};

push(&stack,startVertex);

visited[startVertex] = 1;

printf("DFS traversal starting from vertex %d : ",startVertex);

while(!isEmpty(&stack))

{

int i;

int currentVertex = pop(&stack);

printf("%d ",currentVertex);

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

{

if(graph[currentVertex][i]==1 && !visited[i])

{

push(&stack,i);

visited[i]=1;

}

}

}

printf("\n");

}

int main()

{

int i,j;

int vertices;

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

scanf("%d",&vertices);

int graph[MAX][MAX];

printf("Enter the adjacency Matrix :\n");

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

{

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

{

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

}

}

int startVertex;

printf("Enter the starting vertex for DFS: ");

scanf("%d",&startVertex);

DFS(graph,vertices,startVertex);

return 0;

}

**Assignment no: 9**

**Title**: Write a program to represent a graph of any city using adjacency matrix /adjacency list. Nodes should represent the various areas in the city and links should represent the distance between them. Find the shortest path of your college from your home using Dijkstra's algorithm.

**Code**:

**Assignment no: 10**

**Title:** Write a program to create student database. Database contains different fields of student like Roll No, Name and percentage. Search a particular student according to roll number using binary search.

**Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct Student{

int rollNo;

char name[50];

float percentage;

};

int binarySearch(struct Student students[],int n,int targetRollNo)

{

int low=0,high=n-1;

while(low<=high)

{

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

if(students[mid].rollNo==targetRollNo)

{

return mid;

}

else if(students[mid].rollNo<targetRollNo)

{

low=mid+1;

}

else

{

high = mid-1;

}

}

return -1;

}

void displayStudent(struct Student student)

{

printf("Roll no: %d\n",student.rollNo);

printf("Name: %s\n",student.name);

printf("Percentage: %.2f\n",student.percentage);

}

int main()

{

int i,n;

printf("Enter the no of students :");

scanf("%d",&n);

struct Student\* students = (struct Students\*)malloc(n\* sizeof(struct Student));

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

{

printf("\nEnter the details for student %d:\n",i+1);

printf("Rol no:");

scanf("%d",&students[i].rollNo);

printf("Name:");

scanf("%s",students[i].name);

printf("Percentage :");

scanf("%f",&students[i].percentage);

}

int targetRollNo;

printf("\nEnter the Roll no to Search: ");

scanf("%d",&targetRollNo);

int index = binarySearch(students, n, targetRollNo);

if(index !=-1)

{

printf("\nStudent found:\n");

displayStudent(students[index]);

}

else

{

printf("\nStudent not found...\n");

}

free(students);

return 0;

}

**Assignment no: 11**

**Title** : Write a program to arrange list of students to find out first ten toppers from a class using Bubble sort. (refer the student database given in assignment 10).

**Code**:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure to represent a student

struct Student {

char name[50];

float percentage;

};

// Function to perform Bubble Sort on the student array based on Percentage

void bubbleSort(struct Student students[], int n) {

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

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

if (students[j].percentage < students[j + 1].percentage) {

// Swap the students

struct Student temp = students[j];

students[j] = students[j + 1];

students[j + 1] = temp;

}

}

}

}

int main() {

int n;

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

scanf("%d", &n);

// Allocate memory for the student array

struct Student \*students = (struct Student \*)malloc(n \* sizeof(struct Student));

// Input details of each student

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

printf("\nEnter details for student %d:\n", i + 1);

printf("Name: ");

scanf("%s", students[i].name);

printf("Percentage: ");

scanf("%f", &students[i].percentage);

}

// Perform Bubble Sort to arrange students in descending order of Percentage

bubbleSort(students, n);

// Display the sorted student list

printf("\nTop Ten Toppers:\n");

printf("Rank\tName\t\tPercentage\n");

for (int i = 0; i < (n < 10 ? n : 10); ++i) {

printf("%d\t%s\t\t%.2f\n", i + 1, students[i].name, students[i].percentage);

}

// Free allocated memory

free(students);

return 0;

}

**Assignment no: 12**

**Title:** Write a program to implement Merge sort / Quick sort method

**Code:**

**//Merge\_Sort.c**

#include <stdio.h>

#include <stdlib.h>

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

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

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

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

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

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

// Merge the temporary arrays back into arr[left..right]

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = left; // 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++;

}

}

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

if (left < right) {

// Same as (left+right)/2, but avoids overflow

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

// Sort first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

// Perform Merge Sort

mergeSort(arr, 0, n - 1);

// Print the sorted array

printf("Sorted array:\n");

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

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

}

return 0;

}