

**BHARATI VIDYAPEETH’S**

**INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT**

(Affiliated to Guru Gobind Singh Indraprastha University,Approved by AICTE, New Delhi)

**Data and File Structures Lab**

**(MCA-162)**

**Practical File**

**Submitted To: Submitted By:**

Dr Saumya Bansal Abhijeet Rana

(01811604422)

(Associate Professor) MCA 2nd Sem, Sec 2

|  |  |
| --- | --- |
| **S. No** | **Assignment Set: A (Searching and Sorting in Array)** |
| AP1 | Write a program which takes an array of n integers and displays the frequency of each element present in the array. |
| AP2 | Write a program which takes an array of n integers and performs searching of an element by implementing linear search and binary search techniques. |
| AP3 | Write a program which takes an array of n integers and sorts the integers in descending order using bubble sort and selection sort techniques. |
| AP4 | Write a program which takes an array of n integers and sorts the integers in ascending order using insertion sort technique. |
| AP5 | Write a program which takes an array of n integers and sorts the integers in ascending order using quick sort technique. |
|  | **Assignment Set: B (Linked List)** |
| BP1 | Write a menu-driven program which implements a linear linked list with following operations:   1. Insertion of an element at beginning of the list 2. Insertion of an element at specific location of the list 3. Insertion of an element at end of the list 4. Deletion of an element from the beginning of the list 5. Deletion of an element from specific location of the list 6. Deletion of an element from the end of the list 7. Display all elements of the list h) Search a specific element in the list |
| BP2 | A polynomial is composed of different terms where each of them holds a coefficient and an exponent. Write a program to represent the following polynomials: 4x4 + 4x3 -2x2 + x and 11x3 + 7x2 - 4x with linear linked list, and then perform addition of the given polynomials. |
|  | **Assignment Set: C (Stacks and Queue)** |

|  |  |
| --- | --- |
| CP1 | Write a menu-driven program which implements a stack (using one dimensional array) with following operations:   1. Push (insert an element) 2. Pop (delete an element) 3. Display (print all the elements of stack) |
| CP2 | Write a menu-driven program which implements a linear queue (using one- dimensional array) with following operations:   1. Enqueue (insert an element) 2. Dequeue (delete an element) 3. Display (print all the elements of queue) |
| CP3 | Write a menu-driven program which implements a circular queue (using one- dimensional array) with following operations:   1. Enqueue (insert an element) 2. Dequeue (delete an element) 3. Display (print all the elements of queue) |
|  | **Assignment Set: D (Tree)** |
| DP1 | Write a menu-driven program which implements a binary tree (using linked list) with following operations:   1. Insertion of a node 2. Deletion of a node 3. Preorder traversal 4. Inorder traversal 5. Postorder traversal 6. Determine total numer of leaf nodes |
| DP2 | Write a menu-driven program which implements a heap (using onedimensional array) with following operations:   1. Insertion of a node 2. Deletion of a node 3. Display (print all the elements of heap) |
| DP3 | Write a menu-driven program which implements a binary search tree (using linked list) with following operations:   1. Insertion of a node 2. Deletion of a node 3. Preorder traversal |

|  |  |
| --- | --- |
|  | 1. Inorder traversal 2. Postorder traversal |
| DP4 | Write a program which takes an array of n integers and sorts the integers in ascending order using heap sort technique |
|  | **Assignment Set: E (Graph)** |
| EP1 | Write a menu-driven program which implements a graph (using adjacency matrix) with following operations:   1. Insertion of a vertex 2. Insertion of an edge 3. Deletion of a vertex 4. Deletion of an edge 5. Calculation of degree of each vertex 6. Calculation of number of self-loops in the graph |
| EP2 | Write a program to traverse the following graph using breadth first search and depth first search techniques. |
| EP3 | Write a program to determine shortest path from a to f (using Dijkstra‟s algorithm) in the following graph. |
| EP4 | Write a program to determine shortest paths between every pair of vertices |

|  |  |
| --- | --- |
|  | (using Floyd Warshell‟s algorithm) in the following graph. |
|  | **Assignment Set: F (File Handling)** |
| FP1 | Write a program that generates n random integers and stores them in a text file, named as “All.txt”. Then, retrieve the stored integers from this file and copy to “Odd.txt” and „Even.txt‟ based upon the type of number, i.e. if the retrieved integer is odd number then store in “Odd.txt” file or if the retrieved integer is even then store in “Even.txt” file. Finally, display the contents of all three files. |
| FP2 | A text file contains student‟s grade, followed by student‟s name. Sample data is following:  8.3 Gautam  9.4 Jasleen  6.7 Gaurav  9.4 Naman  5.7 Ishika  7.5 Rakesh  Write a program to find the the highest grade, and list all the students who have highest grade. Also, list the details of students‟ having 3rd highest grade. |
| FP3 | Write a program to implement 20 integers (with some duplicate integers) in a hash table (with separate chaining for collision resolution). |

# AP1

Write a program which takes an array of n integers and displays the frequency of each element present in the array.

#include<stdio.h>

int main()

{

int n,i,j,count=1;

printf("Enter the size of array :"); scanf("%d",&n);

int arr[n]; //creation of array printf("\nEnter elements in the array : "); for(i=0;i<n;i++)

{

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

}

int freq[n];

int visited = -1;

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

{

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

{

if(arr[i]==arr[j])

{

count++;

freq[j] = visited;

}

}

if(freq[i] != visited)

freq[i] = count;

}

printf("\nFrequencies of Elements are :\n"); for(i=0;i<n;i++)

{

if(freq[i]!=visited)

{

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

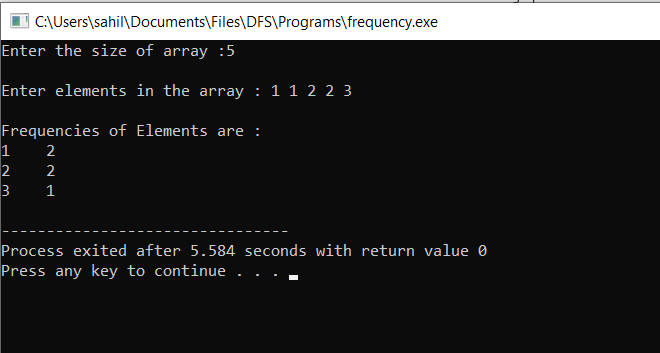
}

}

return 0;

}

# Output :



**AP2**

Write a program which takes an array of n integers and performs searching of an element by implementing linear search and binary search techniques.

# Linear Search

#include<stdio.h> int main()

{

int n,i,item,flag=0;

printf("Enter the size of array :"); scanf("%d",&n);

int arr[n]; //creation of array printf("\nEnter elements in the array : "); for(i=0;i<n;i++)

{

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

}

printf("Enter the element to be searched : "); scanf("%d",&item);

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

{

if(arr[i]==item)

{

flag = 1; break;

}

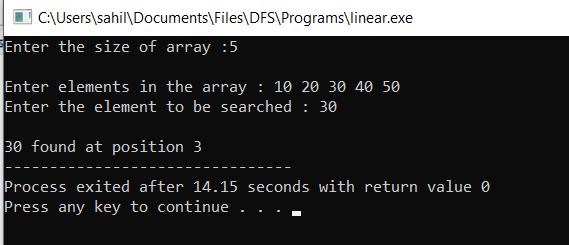
}

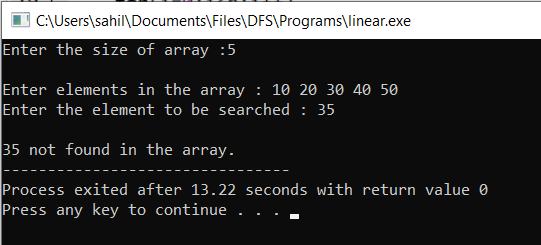
if(flag) else

printf("\n%d found at position %d",item,i+1); printf("\n%d not found in the array.",item);

return 0;

}





# Binary Search

#include<stdio.h> int main()

{

int n,i,item,mid,flag=0;

printf("Enter the size of array :"); scanf("%d",&n);

int arr[n]; //creation of array printf("\nEnter elements in the array : "); for(i=0;i<n;i++)

{

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

}

printf("Enter the element to be searched : "); scanf("%d",&item);

int low = 0, high = n-1; while(low<=high)

{

mid = (low+high)/2; if(arr[mid]==item)

{

flag=1; break;}

else if(arr[mid]>item)

high = mid-1;

else

}

low = mid+1;

if(flag==1)

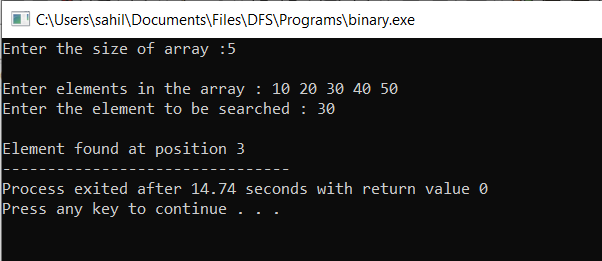
printf("\nElement found at position %d",mid+1);

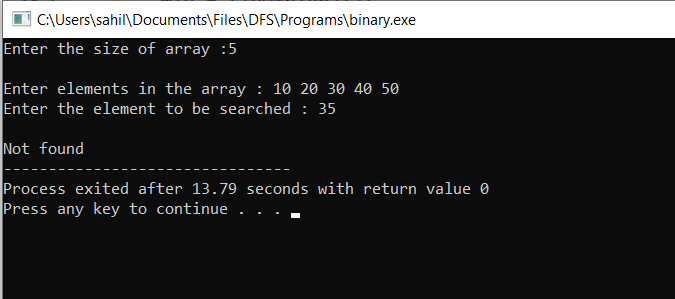
else

printf("\nNot found");

return 0;

}





# AP3

Write a program which takes an array of n integers and sorts the integers in descending order using bubble sort and selection sort techniques.

#include<stdio.h>

void bubblesort(int arr[],int n)

{

int i,j,temp;

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

if(arr[j] < arr[j + 1]){ temp = arr[j]; arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("\nArray after Bubble sort : "); for(i=0;i<n;i++)

{

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

}

}

void selectionsort(int arr[],int n)

{

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

for(j = i+1; j < n; j++){ if(arr[i] < arr[j]){ temp = arr[i];

arr[i] = arr[j]; arr[j] = temp;

}

}

}

printf("\nArray after Selection sort : "); for(i=0;i<n;i++)

{

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

}

}

int main()

{

int n,i,item,mid,flag=0;

printf("Enter the size of array :"); scanf("%d",&n);

int arr[n]; //creation of array printf("\nEnter elements in the array : "); for(i=0;i<n;i++)

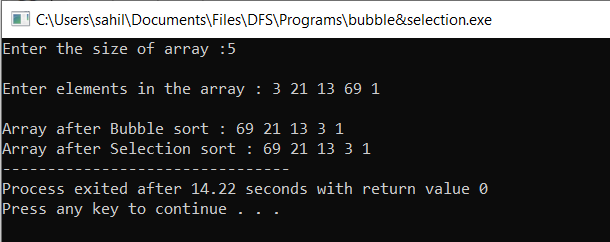
{

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

}

bubblesort(arr,n); //calling bubble sort function selectionsort(arr,n); //calling selection sort function return 0;

}



# AP4

Write a program which takes an array of n integers and sorts the integers in ascending order using insertion sort technique.

#include<stdio.h>

void insertionsort(int arr[],int n)

{

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

{

temp = arr[i]; j = i-1;

while(j>=0 && arr[j] > temp)

{

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

}

arr[j+1] = temp;

}

printf("\nArray after Insertion sort : "); for(i=0;i<n;i++)

{

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

}

}

int main()

{

int n,i,item,mid,flag=0;

printf("Enter the size of array :"); scanf("%d",&n);

int arr[n]; //creation of array printf("\nEnter elements in the array : "); for(i=0;i<n;i++)

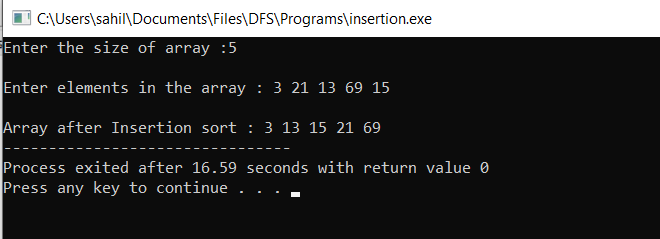
{

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

}

insertionsort(arr,n); //calling insertion sort function return 0;

}



# AP5

Write a program which takes an array of n integers and sorts the integers in ascending order using quick sort technique.

#include<stdio.h>

void quicksort(int number[25],int first,int last){ int i, j, pivot, temp;

if(first<last){ pivot=first; i=first; j=last; while(i<j){

while(number[i]<=number[pivot]&&i<last) i++;

while(number[j]>number[pivot]) j--;

if(i<j){

temp=number[i]; number[i]=number[j]; number[j]=temp;

}

}

temp=number[pivot]; number[pivot]=number[j]; number[j]=temp; quicksort(number,first,j-1); quicksort(number,j+1,last);

}

}

int main(){ int i, n;

printf("Enter the size of array : "); scanf("%d",&n);

int number[n];

printf("Enter the elements in array : "); for(i=0;i<n;i++)

{

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

}

quicksort(number,0,n-1);

printf("Array after quick sort : "); for(i=0;i<n;i++)

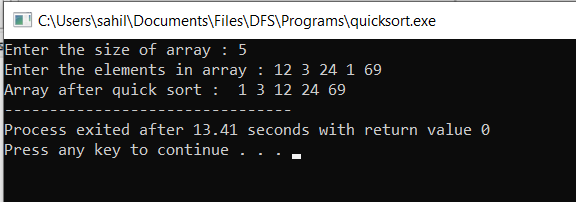
{

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

}

return 0;

}



# BP1

Write a menu-driven program which implements a linear linked list with following operations:

1. Insertion of an element at beginning of the list
2. Insertion of an element at specific location of the list
3. Insertion of an element at end of the list
4. Deletion of an element from the beginning of the list
5. Deletion of an element from specific location of the list
6. Deletion of an element from the end of the list
7. Display all elements of the list
8. Search a specific element in the list

#include<stdio.h> #include<stdlib.h> #include<malloc.h> struct node

{

int data;

struct node \*next;

};

typedef struct node NODE; NODE \*start = NULL;

void insBegining(int item)

{

NODE \*node;

node = (NODE\*)malloc(sizeof(NODE)); node->data = item;

if(start==NULL)

{

node->next = NULL;

}

else

{

node->next = start;

}

start = node; printf("\nNODE INSERTED");

}

void Lastinsert(int item)

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE)); NODE\*temp;

if(ptr == NULL )

{

printf("\n OVERFLOW");

}

else

{

ptr->data=item; if(start == NULL)

{

ptr->next=NULL; start=ptr;

printf("\nNODE INSERTED");

}

else

{

temp = start;

while(temp->next != NULL)

{

temp = temp->next;

}

temp->next=ptr; ptr->next = NULL;

printf("NODE INSERTED");

}

}

}

void displayList()

{

NODE \*temp; temp = start; if(temp == NULL)

{

printf("\n LIST IS EMPTY!");

}

else

{

while(temp != NULL)

{

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

temp = temp -> next;

}

}

}

void delbeg()

{

NODE \*ptr; if(start == NULL)

{

printf("\n List is Empty");

}

else

{

ptr=start;

start= ptr -> next; free(ptr);

printf("\n Node deleted from the begining...");

}

}

void enddel()

{

NODE \*ptr,\*ptr1; if(start == NULL)

{

printf("\nList is empty");

}

else if(start -> next ==NULL)

{

start=NULL; free(start);

printf("\nOnly node of the list deleted");

}

else

{

ptr=start;

while(ptr->next != NULL)

{

ptr1=ptr; ptr=ptr->next;

}

ptr->next= NULL; free(ptr1);

printf("\n Deleted node from the last");

}

}

void specificins(int item)

{

NODE \*ptr=(NODE \*)malloc(sizeof(NODE)); NODE \*temp;

int i,loc;

if(ptr == NULL)

{

printf("\n Overflow");

}

else

{

printf("\n Enter the location:"); scanf("%d",&loc);

ptr->data=item; temp=start; for(i=0;i<loc;i++)

{

temp=temp->next; if(temp==NULL)

{

printf("\nCan't insert"); return;

}

}

ptr->next=temp->next; temp->next=ptr;

printf("\n Node inserted");

}

}

specificdel(int loc)

{

NODE \*ptr,\*ptr1; int i;

ptr=start; for(i=0;i<loc;i++)

{

ptr1=ptr; ptr=ptr->next; if(ptr==NULL)

{

printf("\nThere are less than %d elements in the list",loc); return 0;

}

}

ptr1->next=ptr->next; free(ptr);

printf("\nNode Deleted"); getch();

return 0;

}

void search()

{

NODE \*ptr;

int item,i=0,flag; ptr=start; if(ptr==NULL)

{

printf("\nEmpty list");

}

else

{

printf("\nEnter item which you want to search?"); scanf("%d",&item);

while(ptr!=NULL)

{

if(ptr->data==item)

{

printf("\n Item found at location %d ",(i+1)); flag=0;

getch();

}

else

{

flag=1;

} i++;

ptr=ptr->next;

}

if(flag!=1)

{

printf("\nItem not found");

}

}

getch();

}

int main()

{

int choice,loc,item; do

{

printf("\n\t\tMENU");

printf("\n1. INSERT AT BEGINING"); printf("\n2. INSERT AT SPECIFIC LOCATION");

printf("\n3. INSERT AT END"); printf("\n4. DELETE FROM BEGINING");

printf("\n5. DELETE FROM SPECIFIC LOCATION");

printf("\n6. DELTE FROM END"); printf("\n7. SEARCH THE ELEMENT"); printf("\n8. DISPLAY THE LIST");

printf("\n9.Exit");

printf("\n Input your choice "); scanf("%d", &choice); switch(choice)

{

case 1:printf("\n Input the Node Data: "); scanf("%d",&item); insBegining(item);

break;

case 2:printf("\n Input the Node Data: "); scanf("%d",&item); specificins(item);

break;

case 3:printf("\n Input the Node Data: "); scanf("%d",&item); Lastinsert(item);

break; case 4:delbeg();

break; case 5:

printf("\nEnter the location:"); scanf("%d",&loc); specificdel(loc);

break; case 6:enddel();

break; case 7:search();

break;

case 8:displayList(); getch(); break;

case 9:exit(0);

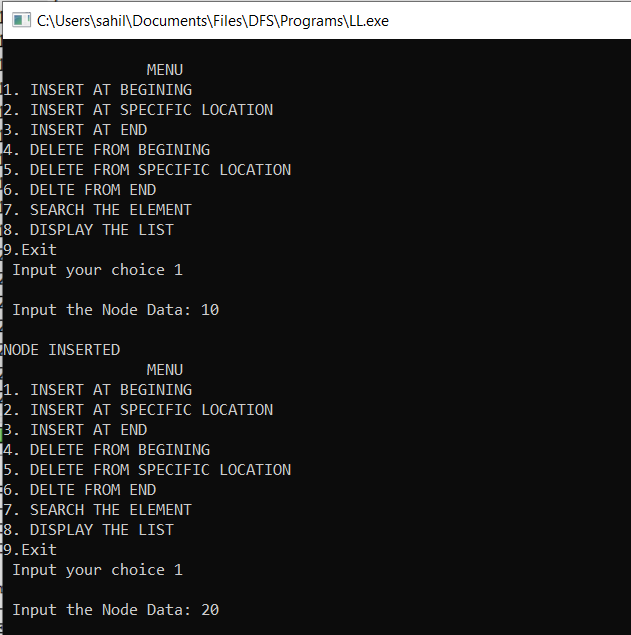
default : printf("\nPlease enter the correct choice!!!");

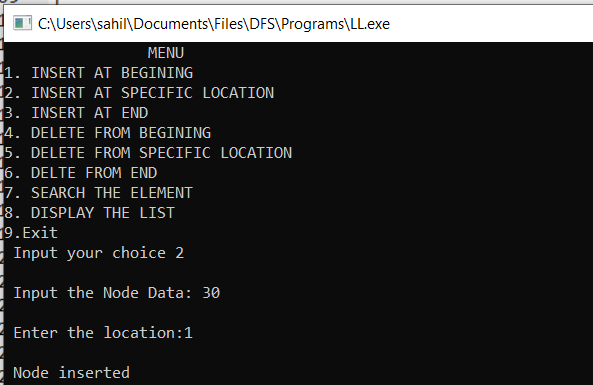
}

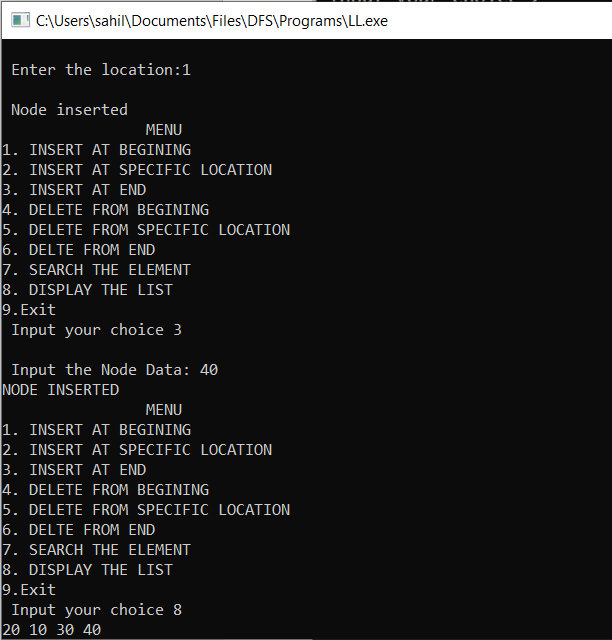
}while(choice!=10);

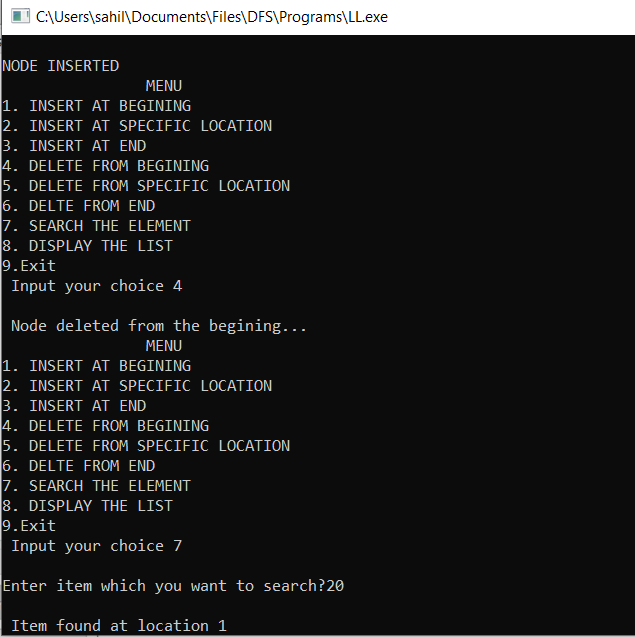
return 0;

}









# BP2

A polynomial is composed of different terms where each of them holds a coefficient and an exponent. Write a program to represent the following polynomials: 4x4 + 4x3 -2x2 + x and 11x3 + 7x2 - 4x with linear linked list, and then perform addition of the given polynomials.

#include<stdio.h> #include<malloc.h> #include<stdlib.h> struct node

{

int coe; int pow;

struct node \*next;

};

typedef struct node NODE; void createPoly(NODE \*\*); void printPoly(NODE \*);

void addPoly(NODE \*\*, NODE \*, NODE \*);

void main()

{

NODE \*p, \*q, \*r;

printf("\nCREATE FIRST POLYNOMIAL");

createPoly(&p);

printf("\nFIRST POLYNOMIAL: ");

printPoly(p);

printf("\nCREATE SECOND POLYNOMIAL");

createPoly(&q);

printf("\nSECOND POLYNOMIAL: ");

printPoly(q);

printf("\nADDITION OF TWO POLYNOMIALS: ");

addPoly(&r,p,q); printPoly(r);

getch();

}

void createPoly(NODE \*\*start)

{

int flag;

int coe, pow; NODE \*node;

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

\*start = node; do

{

printf("\n\nINPUT THE COEFFICIENT: ");

scanf("%d",&coe); printf("\nINPUT THE POWER: ");

scanf("%d",&pow); node->coe = coe; node->pow = pow; node->next = NULL;

printf("\n\nDO YOU WANT TO ADD MORE TERMS TO POLYNOMIAL: (0//1)");

scanf("%d",&flag); if(flag)

{

NODE \*newNode = (NODE \*)malloc(sizeof(NODE)); node->next = newNode;

node = newNode; node->next = NULL;

}

} while(flag);

}

void printPoly(NODE \*node)

{

printf("\nPOLYNOMIAL EXPRESSION: ");

while(node!=NULL)

{

printf("%d^%d", node->coe, node->pow); node = node->next;

if(node!=NULL)

{

printf(" + ");

}

}

}

void addPoly(NODE \*\*start, NODE \*p, NODE \*q)

{

NODE \*node = (NODE \*)malloc(sizeof(NODE)); node->next = NULL;

\*start = node;

while(p && q) //LOOP WHILE BOTH LISTS HAVE VALUES

{

if(p->pow > q->pow)

{

node->pow = p->pow; node->coe = p->coe; p = p->next;

}

else if(p->pow < q->pow)

{

node->pow = q->pow; node->coe = q->coe; q = q->next;

}

else

{

node->pow = p->pow;

node->coe = p->coe + q->coe; p = p->next;

q = q->next;

}

if(p && q) //GROW THE LINKED LIST ON CONDITION

{

NODE \*newNode = (NODE \*)malloc(sizeof(NODE)); node->next = newNode;

node = newNode; node->next = NULL;

}

}

while(p || q)

{

NODE \*newNode = (NODE \*)malloc(sizeof(NODE)); node->next = newNode;

node = newNode; node->next = NULL; if(p)

{

node->pow = p->pow; node->coe = p->coe; p = p->next;

}

if(q)

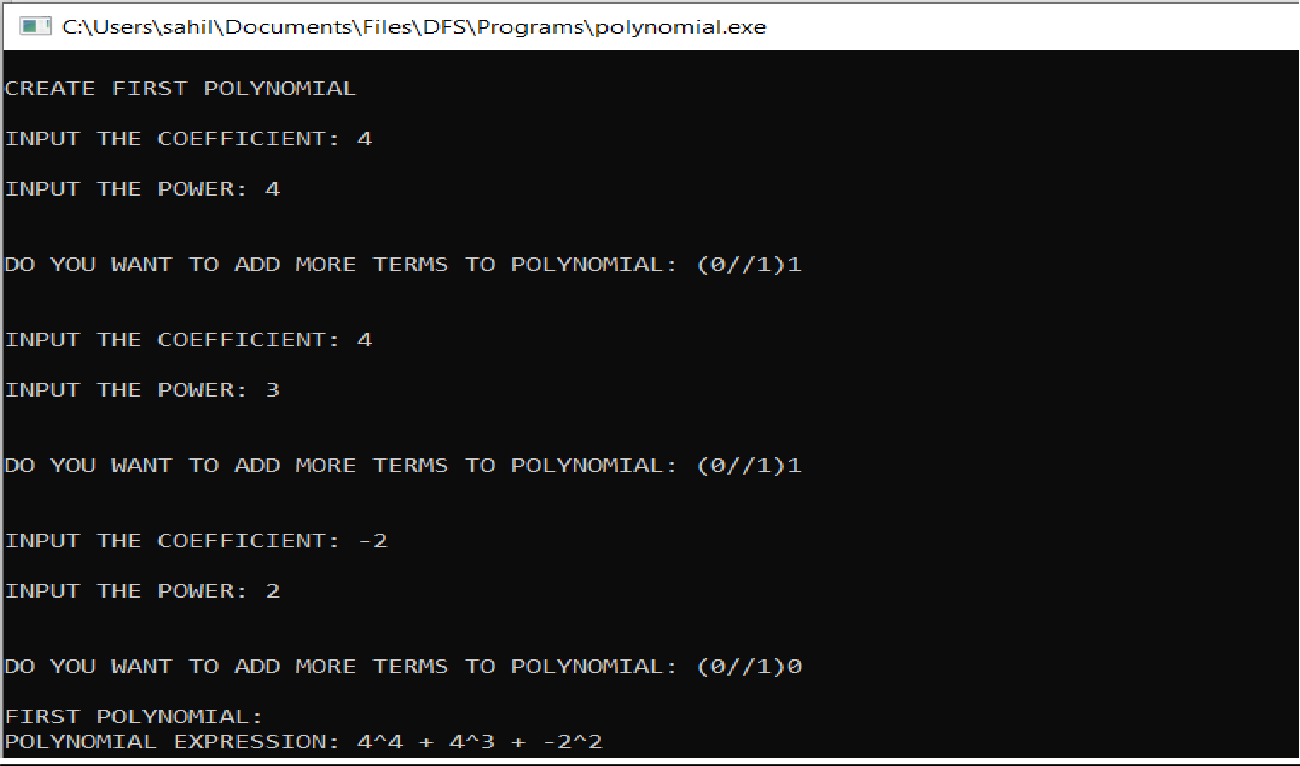
{

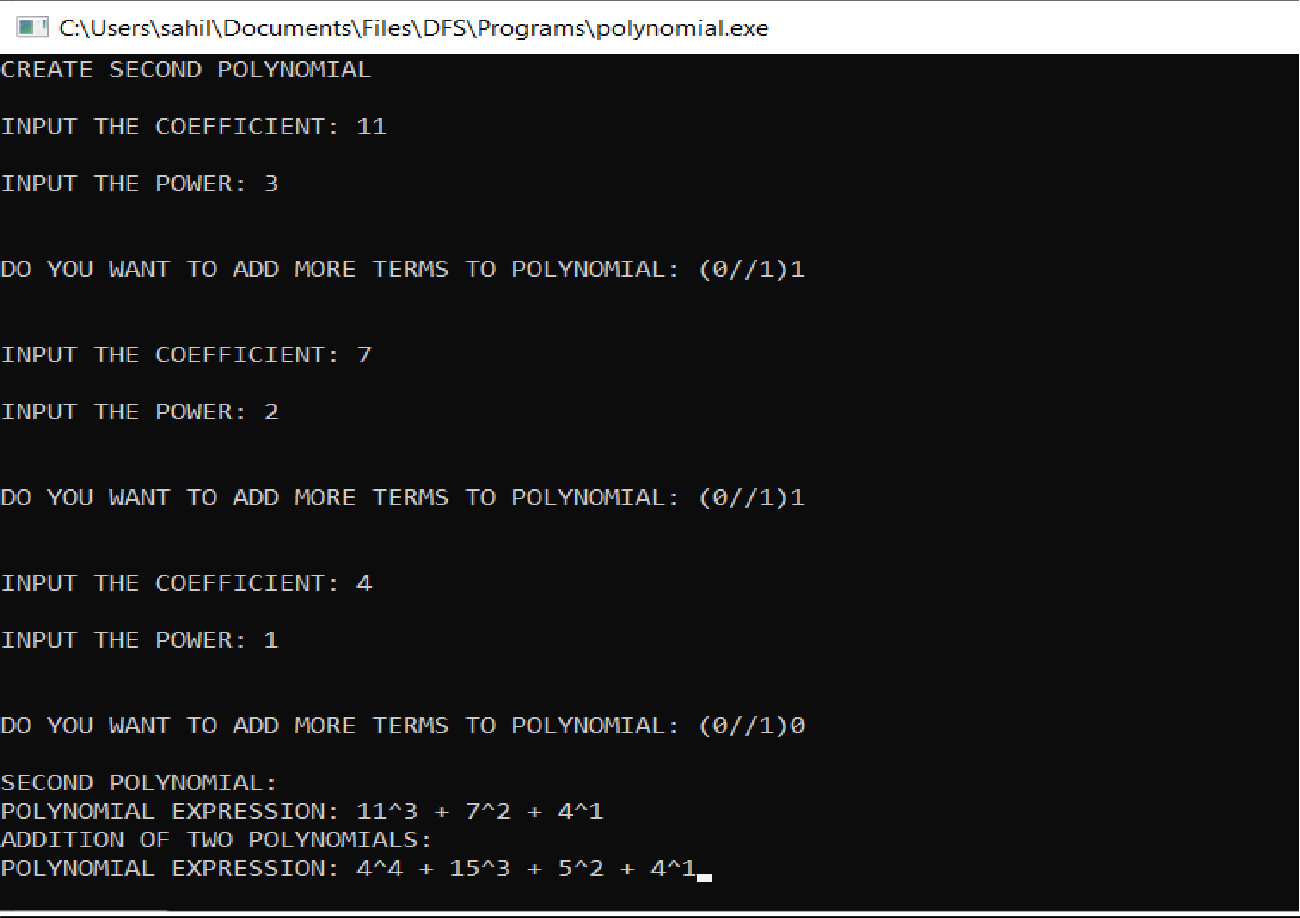
node->pow = q->pow; node->coe = q->coe; q = q->next;

}

}

}





# CP1

Write a menu-driven program which implements a stack (using one dimensional array) with following operations:

1. Push (insert an element)
2. Pop (delete an element)
3. Display (print all the elements of stack)

#include <stdio.h> #define MAX 5 void push(int);

int pop();

int isempty(); int isfull(); void display();

int stack[MAX],top=-1; void main()

{

int choice ,item; do{

printf("\t\tMENU"); printf("\n1.PUSH AN ELEMENT");

printf("\n2 POP AN ELEMENT");

printf("\n3 DISPLAY ELEMENTS IN THE STACK");

printf("\n4.EXIT"); printf("\nInput your choice : "); scanf("%d",&choice); switch(choice){

case 1:

if(isfull()){

printf("\nstack overflow");

}else{

printf("\n Input the item : "); scanf("%d",&item); push(item);

}

break; case 2:

if(isempty()){

printf("\nstack underflow");

}else{

printf("\nItem deleted=%d",pop());

}

break;

case 3:

if(isempty()){

printf("\nstack underflow");

}else{

printf("\nElements in stack"); display();

}

break;

}

}while(choice!=4);

}

void push(int item){ top=top+1; stack[top]=item;

}

int pop(){

int temp=stack[top]; top=top-1;

return temp;

}

int isempty(){ if(top==-1){ return 1;

}else{

return 0;

}

}

int isfull(){ if(top==MAX-1){

return 1;

}else{

return 0;

}

}

void display()

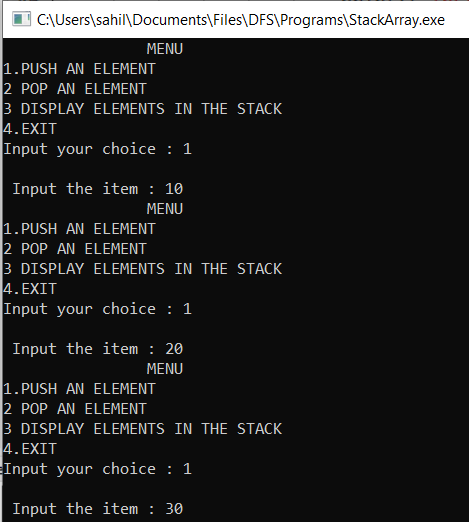
{

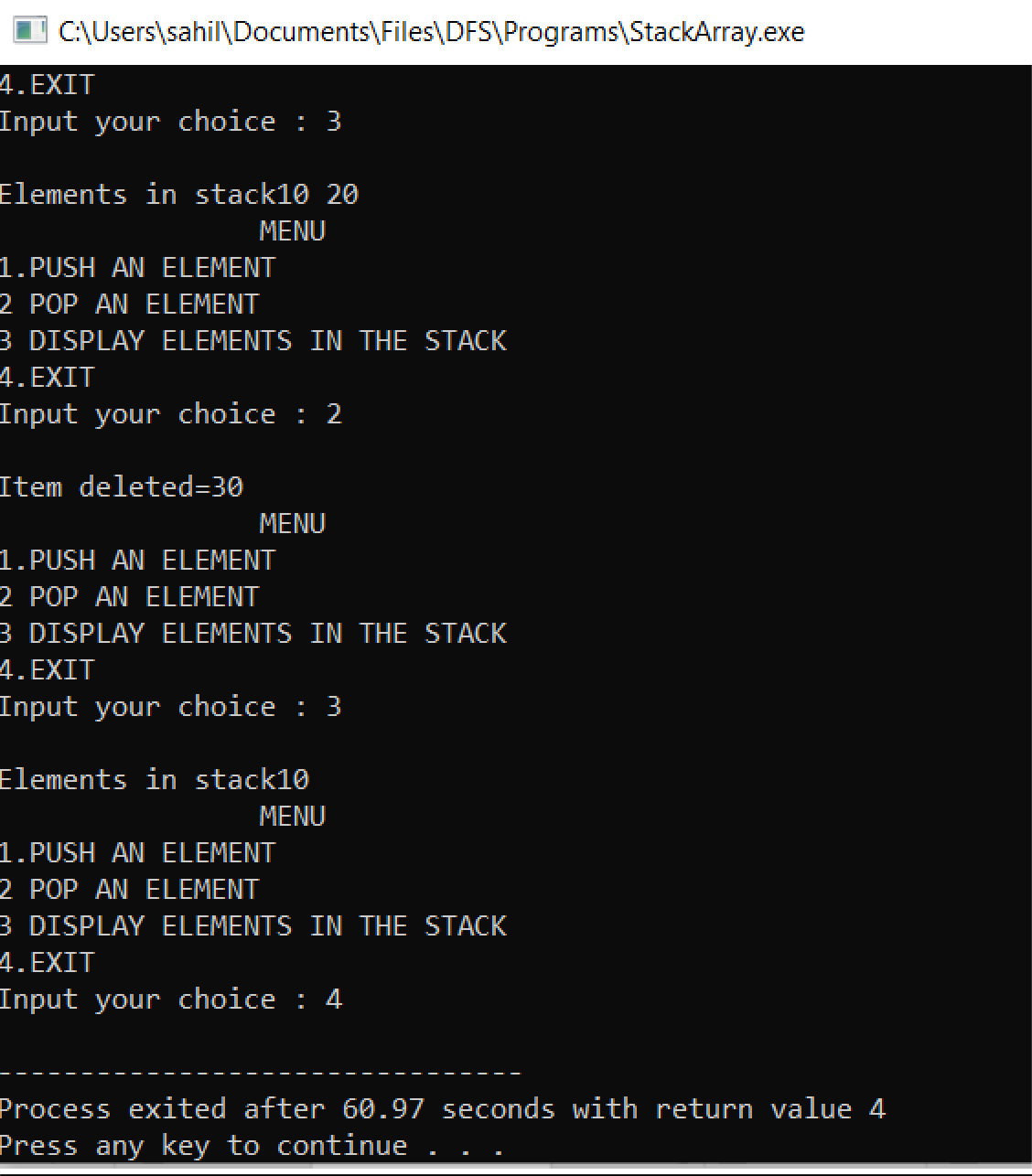
int i; for(i=0;i<top;i++){

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

}

}





# CP2

Write a menu-driven program which implements a linear queue (using one-dimensional array) with following operations:

1. Enqueue (insert an element)
2. Dequeue (delete an element)
3. Display (print all the elements of queue)

#include<stdio.h> #include<conio.h> #define MAX 5

void enqueue(int); int dequeue(); void display();

int queue[MAX],front,rear; void main()

{

int choice,item; do

{

printf("\n\t\tMENU"); printf("\n1.Enqueue"); printf("\n2.Dequeue"); printf("\n3.Display"); printf("\n4.Exit"); printf("\nInput your choice: "); scanf("%d",&choice); switch(choice)

{

case 1:printf("\nEntered item:"); scanf("%d",&item); enqueue(item);

break;

case 2:printf("\nItem Deleted :%d ",queue[front],dequeue()); break;

case 3:printf("\nElelment of Queue:"); display();

break;

}

}while(choice!=4);

}

void enqueue(int item)

{

if(front == 0 && rear == MAX-1)

{

printf("\n Queue is full");

}

else if(front == -1)

{

front=0; rear=0;

}

else

{

rear=rear+1; queue[rear]=item;

}

}

int dequeue()

{

int temp; if(front==-1)

{

printf("\nQueue is empty");

}

else if(rear==front)

{

temp=queue[front]; rear=-1;

front=-1;

}

else

{

temp=queue[front]; front=front+1;

}

return temp;

}

void display()

{

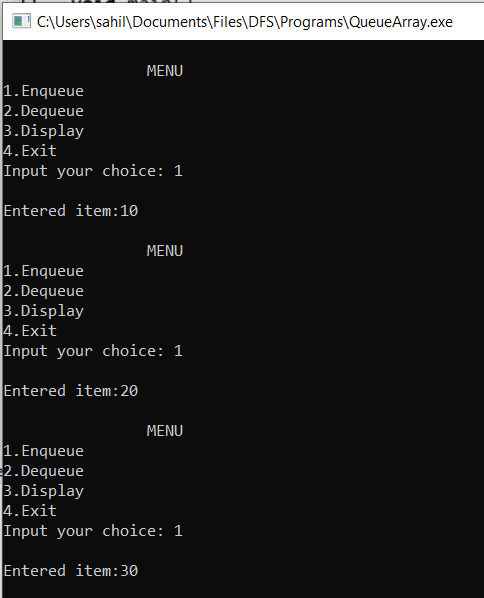
int i; for(i=front+1;i<=rear;i++)

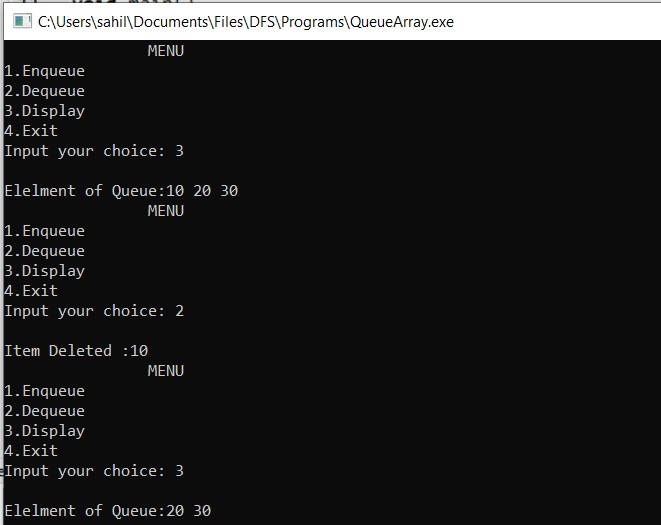
{

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

}

}





# CP3

Write a menu-driven program which implements a circular queue (using one-dimensional array) with following operations:

1. Enqueue (insert an element)
2. Dequeue (delete an element)
3. Display (print all the elements of queue)

#include<conio.h> #include<stdio.h> #include<stdlib.h> int cqueue[6];

int front = -1, rear = -1, n=6; void enqueue(int val){

if ((front == 0 && rear == n-1) || (front == rear+1)) { printf("Queue Overflow \n");

return;

}

if (front == -1) { front = 0;

rear = 0;

}

else {

if (rear == n - 1) rear = 0;

else

}

rear = rear + 1;

cqueue[rear] = val ;

}

void dequeue(){ if (front == -1) {

printf("Queue Underflow\n"); return ;

}

printf("Element deleted from queue is : %d ", cqueue[front]); if (front == rear) {

front = -1;

rear = -1;

}

else {

if (front == n - 1) front = 0;

else

}

}

front = front + 1;

void display(){

int f = front, r = rear; if (front == -1) {

printf("Queue is empty"); return;

}

printf("Queue elements are :\n"); if (f <= r) {

while (f <= r){

printf("%d ", cqueue[f]); f++;

}

}

else {

while (f <= n - 1) { printf("%d",cqueue[f]); f++;

}

f = 0;

while (f <= r) {

printf("%d",cqueue[f]); f++;

}

}

}

void main()

{

int value,choice; do

{

printf("\n\t\tMENU");

printf("\n 1.Add value to the list"); printf("\n 2. Delete value to the list"); printf("\n 3. Travesre/View List"); printf("\n 4. exit");

printf("\n Please enter your choice: "); scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Input for insertion: "); scanf("%d",&value); enqueue(value);

break;

case 2:

dequeue(); break;

case 3:

display(); break;

case 4:

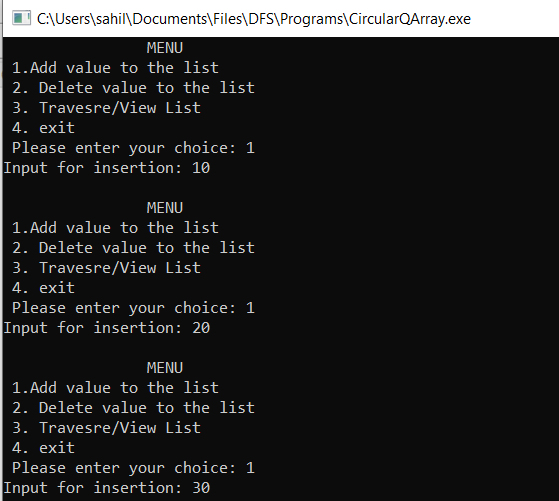
exit(0); default:

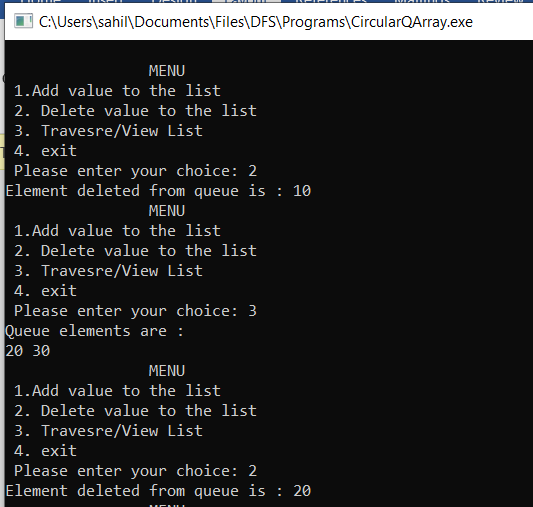
printf("invalid choice");

}

}while(choice!=4);

}





# DP1

Write a menu-driven program which implements a binary tree (using linked list) with following operations:

1. Insertion of a node
2. Deletion of a node
3. Preorder traversal
4. In-order traversal
5. Post-order traversal
6. Determine total number of leaf nodes

#include<stdio.h> #include<stdlib.h> #include<malloc.h>

//creating structure of a tree node struct node

{

int data;

struct Node \*left; struct Node \*right;

};

typedef struct node Node;

Node \*root = NULL;

Node \*newNode(int); Node \*insert(Node \*,int); Node \*minNode(Node \*);

Node \*deleteNode(Node \*,int); void preorder(Node \*);

void inorder(Node \*); void postorder(Node \*); int leafNode(Node \*);

Node \*newNode(int item)

{

Node \*node = (Node \*)malloc(sizeof(Node)); node->data = item;

node->left = NULL; node->right = NULL;

return node;

}

Node \*insert(Node \*root,int item)

{

if(root==NULL)

{

return newNode(item);

}

if(item < root->data)

{

}

else

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

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

return root;

}

Node \*minNode(Node \*node)

{

Node \*curr = node;

while(curr && curr->left!=NULL) curr = curr->left;

return curr;

}

Node \*deleteNode(Node \*root,int item)

{

if(root == NULL)

return root;

if(item < root->data)

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

else if(item > root->data)

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

else

{

if(root->left == NULL)

{

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

return temp;

}

else if(root->right == NULL)

{

Node \*temp = root->left; free(root);

return temp;

}

Node \*temp = minNode(root->right); root->data = temp->data;

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

}

return root;

}

void preorder(Node \*root)

{

if(root!=NULL)

{

printf("%d ",root->data); preorder(root->left); preorder(root->right);

}

}

void inorder(Node \*root)

{

if(root!=NULL)

{

inorder(root->left); printf("%d ",root->data); inorder(root->right);

}

}

void postorder(Node \*root)

{

if(root!=NULL)

{

postorder(root->left); postorder(root->right); printf("%d ",root->data);

}

}

int leafNode(Node \*root)

{

if(root==NULL);

return 0;

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

{

return 1;

}

else

{

}

}

return leafNode(root->left) + leafNode(root->right);

//main function int main()

{

int ch,item; do{

printf("\n\n\t\tMENU"); printf("\n1.INSERT AN ITEM IN THE TREE");

printf("\n2.DELETE AN ITEM FROM THE TREE"); printf("\n3.PREORDER TRAVERSAL"); printf("\n4.INORDER TRAVERSAL"); printf("\n5.POSTORDER TRAVERSAL");

printf("\n6.DETERMINE TOTAL NUMBER OF LEAF NODE IN THE TREE");

printf("\n7.EXIT...");

printf("\n Input your choice : "); scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the item to insert : ");

scanf("%d",&item); root = insert(root,item);

printf("Item inserted successfully"); break;

case 2: printf("\nEnter the item to delete : ");

scanf("%d",&item);

root = deleteNode(root,item); printf("Item deleted successfully"); break;

case 3: printf("\nPREORDER TRAVERSAL : ");

preorder(root); break;

case 4: printf("\nINORDER TRAVERSAL : ");

inorder(root); break;

case 5: printf("\nPOSTORDER TRAVERSAL : ");

postorder(root); break;

case 6: printf("Leaf count of the tree is %d", leafNode(root)); break;

case 7: exit(0);

break;

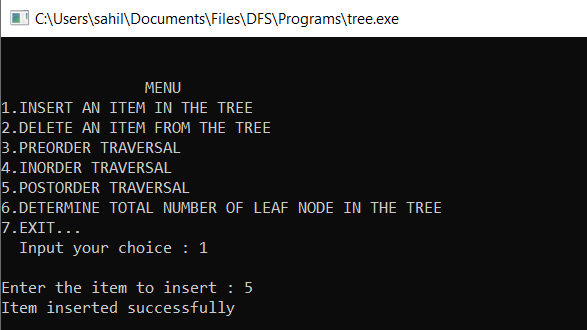
default : printf("INVALID CHOICE!!!");

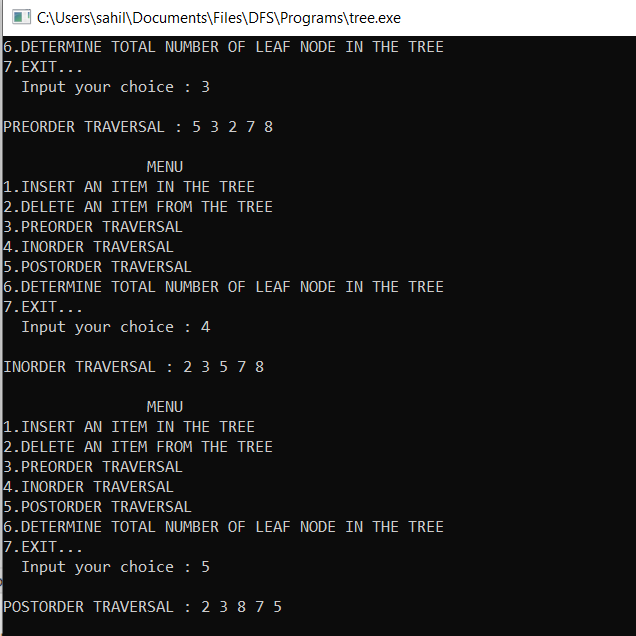
}

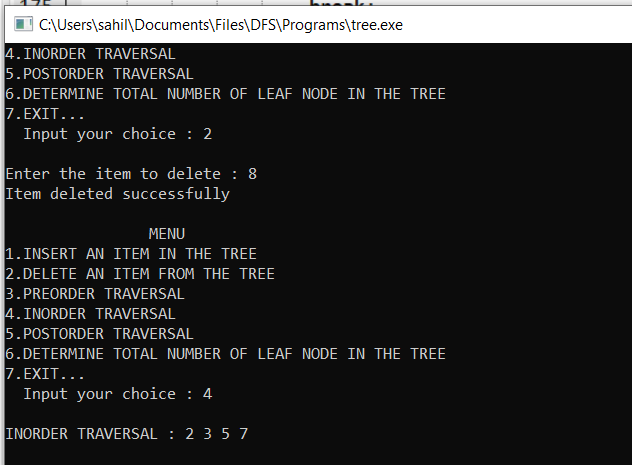
}while(ch!=7);

return 0;

}







# DP2

Write a menu-driven program which implements a heap (using one dimensional array) with following operations:

1. Insertion of a node
2. Deletion of a node
3. Display (print all the elements of heap)

#include <stdio.h> #include <stdlib.h> int size = 0;

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

{

int temp = \*b;

\*b = \*a;

\*a = temp;

}

void heapify(int array[], int size, int i)

{

if (size == 1)

{

printf("Single element in the heap");

}

else

{

int largest = i; int l = 2 \* i + 1; int r = 2 \* i + 2;

if (l < size && array[l] > array[largest]) largest = l;

if (r < size && array[r] > array[largest]) largest = r;

if (largest != i)

{

swap(&array[i], &array[largest]); heapify(array, size, largest);

}

}

}

void insert(int array[], int newNum)

{

if (size == 0)

{

array[0] = newNum; size += 1;

}

else

{

int i;

array[size] = newNum; size += 1;

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

{

heapify(array, size, i);

}

}

}

void deleteRoot(int array[], int num)

{

int i;

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

{

if (num == array[i]) break;

}

swap(&array[i], &array[size - 1]); size -= 1;

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

{

heapify(array, size, i);

}

}

void printArray(int array[], int size)

{

int i;

for (i = 0; i < size; ++i) printf("%d ", array[i]); printf("\n");

}

int main()

{

int array[10],i,d,ch; do

{

printf("MENU");

printf("\n1.Insert a node\n2.Delete a node\n3.Display\n4.Exit\nEnter your choice: "); scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter the node to be inserted:"); scanf("%d",&i);

insert(array,i); break;

case 2:

printf("\nEnter node to be deleted:"); scanf("%d",&d);

deleteRoot(array,d); break;

case 3:

printArray(array, size);

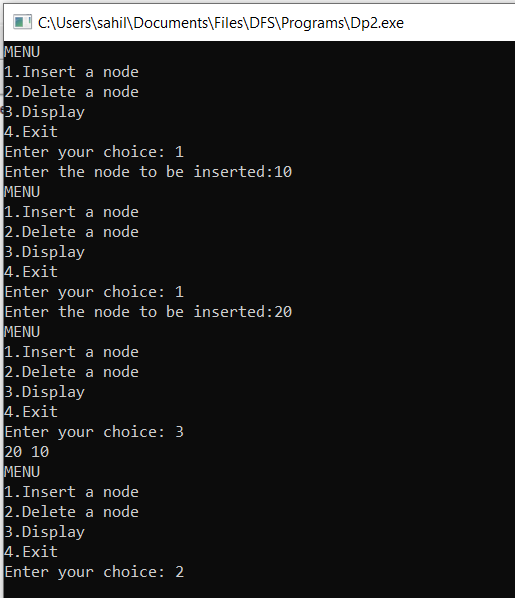
break; case 4:

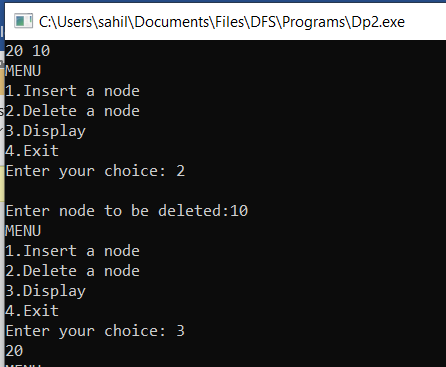
exit(0); break;

}

}while(ch!=4); return 0;

}





# DP3

Write a menu-driven program which implements a binary search tree (using linked list) with following operations:

1. Insertion of a node
2. Deletion of a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal

#include <stdio.h> #include <stdlib.h>

struct BST

{

int data;

struct BST \*left; struct BST \*right;

};

typedef struct BST NODE; NODE \*node;

NODE\* createtree(NODE \*node, int data)

{

if (node == NULL)

{

NODE \*temp;

temp= (NODE\*)malloc(sizeof(NODE)); temp->data = data;

temp->left = temp->right = NULL; return temp;

}

if (data < (node->data))

{

node->left = createtree(node->left, data);

}

else if (data > node->data)

{

node -> right = createtree(node->right, data);

}

return node;

}

void inorder(NODE \*node)

{

if(node != NULL)

{

inorder(node->left); printf("%d\t", node->data); inorder(node->right);

}

}

void preorder(NODE \*node)

{

if(node != NULL)

{

printf("%d\t", node->data); preorder(node->left); preorder(node->right);

}

}

void postorder(NODE \*node)

{

if(node != NULL)

{

postorder(node->left); postorder(node->right); printf("%d\t", node->data);

}

}

NODE\* findMin(NODE \*node)

{

if(node==NULL)

{

return NULL;

}

if(node->left)

return findMin(node->left);

else

return node;

}

NODE\* del(NODE \*node, int data)

{

NODE \*temp;

if(node == NULL)

{

printf("\nElement not found");

}

else if(data < node->data)

{

node->left = del(node->left, data);

}

else if(data > node->data)

{

}

else

{

{

}

else

{

node->right = del(node->right, data);

if(node->right && node->left)

temp = findMin(node->right); node -> data = temp->data;

node -> right = del(node->right,temp->data);

temp = node;

if(node->left == NULL) node = node->right;

else if(node->right == NULL) node = node->left; free(temp);

}

}

return node;

}

void main()

{

int data, ch, i, n; NODE \*root=NULL;

while (1)

{

printf("\n1.Insertion in Binary Search Tree"); printf("\n2.Search Element in Binary Search Tree"); printf("\n3.Delete Element in Binary Search Tree"); printf("\n4.Inorder");

printf("\n5.Preorder"); printf("\n6.Postorder");

printf("\n7.Exit"); printf("\nEnter your choice: "); scanf("%d", &ch);

switch (ch)

{

case 1: printf("\nEnter N value: " );

scanf("%d", &n);

printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n"); for(i=0; i<n; i++)

{

scanf("%d", &data); root=createtree(root, data);

}

break;

case 3: printf("\nEnter the element to delete: "); scanf("%d", &data);

root=del(root, data); break;

case 4: printf("\nInorder Traversal: \n");

inorder(root); break;

case 5: printf("\nPreorder Traversal: \n");

preorder(root); break;

case 6: printf("\nPostorder Traversal: \n");

postorder(root); break;

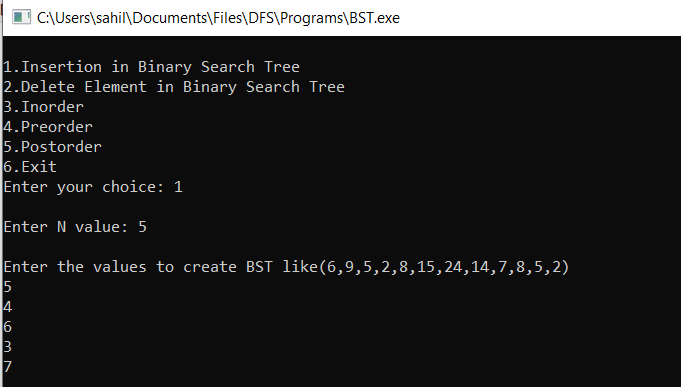
case 7: exit(0); default:printf("\nWrong option");

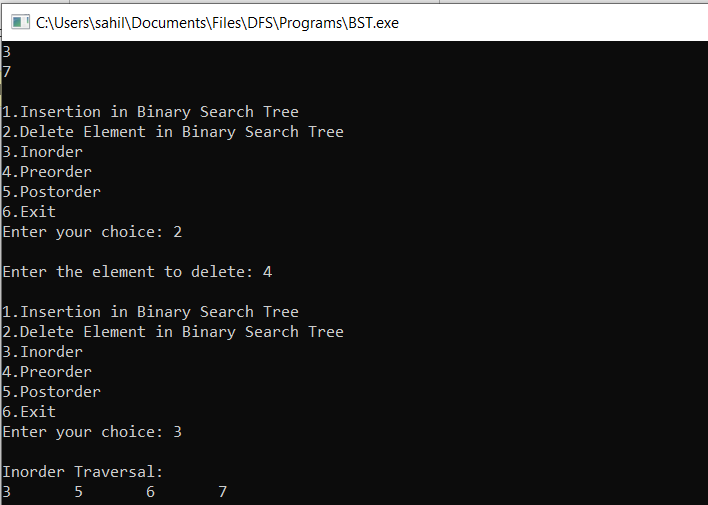
break;

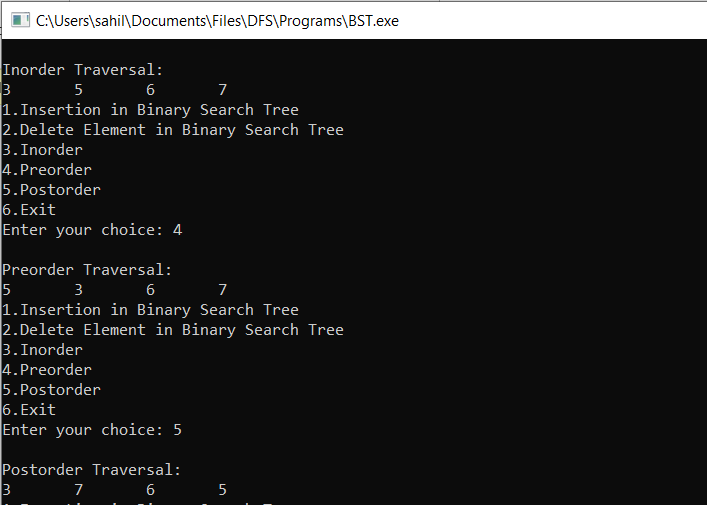
}

}

}







**DP4**

Write a program which takes an array of n integers and sorts the integers in ascending order using heap sort technique.

#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) { int i;

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

{

heapify(arr, n, i);

}

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

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

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

int i;

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

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

printf("\n");

}

int main()

{

int n,i;

printf("Enter the size : "); scanf("%d",&n);

int arr[n];

printf("\nEnter the elements in the array : "); for(i=0;i<n;i++)

{

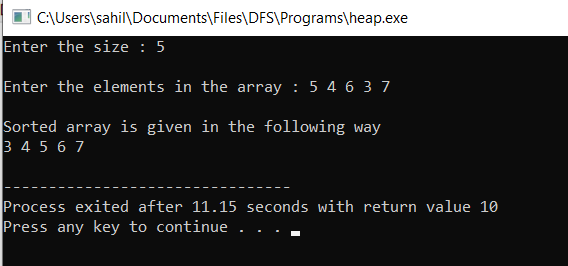
scanf("%d",&arr[i]);

}

heapSort(arr, n);

printf("\nSorted array is given in the following way \n"); printArray(arr, n);

}



**EP1**

1 Write a menu-driven program which implements a graph (using adjacency matrix) with following operations:

1. Insertion of a vertex
2. Insertion of an edge
3. Deletion of a vertex
4. Deletion of an edge
5. Calculation of degree of each vertex
6. Calculation of number of self-loops in the graph

#include<stdio.h>

int adjMat[9][9],vCount=0;

int queue[9],front=-1,rear=-1,visited[9], stack[10], stackSize=9, top = 1;

void push(int v){

if( top == stackSize ) printf("\nStack Overflow!"); else {

stack[top] = v; top++;

}

}

int peek() {

return stack[top-1];

}

void resetVisited() { int i;

for(i=1; i<= vCount; i++) { visited[i] = 0;

}

}

int stackisEmpty(){ if( top == 1) return 1; else

return 0;

}

int pop(){

if( stackisEmpty() )

{ printf("\nStack is Empty!"); return 0;

}

else { top--;

int n = stack[top]; stack[top] = 0; return n;

}

}

void insVertex(int v)

{

int i;

vCount = vCount+1; adjMat[vCount][0]=v; adjMat[0][vCount]=v; for(i=1; i<=vCount; i++)

{

adjMat[vCount][i] = 0;

adjMat[i][vCount] = 0;

}

}

void delVertex(int v)

{

int i,j,k; j=0; for(i=1;i<=vCount;i++)

{

if(adjMat[i][j]==v)

{

adjMat[i][j] = 0; for(k=1; k<=vCount; k++)

{

adjMat[i][k] = 0;

}

}

} i=0;

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

{

if(adjMat[i][j]==v)

{

adjMat[i][j] = 0; for(k=0;k<=vCount;k++)

{

adjMat[k][j] = 0;

}

}

}

}

void insEdge(int sE, int dE)

{

adjMat[sE][dE] = 1;

adjMat[dE][sE] = 1;

}

void delEdge(int sE, int dE)

{

adjMat[sE][dE] = 0;

adjMat[dE][sE] = 0;

}

void disMatrix()

{

int i,j;

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

{

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

{

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

}

printf("\n");

}

}

void enqueue(int x)

{

if(front==-1)

{

front=rear=0;

}

else

{

rear=rear+1;

}

queue[rear] = x;

}

int dequeue()

{

int temp;

temp = queue[front]; if(rear==front)

{

front=rear=-1;

}

else

{

front = front+1;

}

return temp;

}

int isEmpty()

{

if(front==-1) return 1; else

return 0;

}

void main()

{

int v,choice = 0,sE,dE,source; do

{

resetVisited();

printf("\n\*\*\*\*\* MENU \*\*\*\*\*"); printf("\n1. INSERT A VERTEX"); printf("\n2. DELETE A VERTEX"); printf("\n3. INSERT AN EDGE"); printf("\n4. DELETE AN EDGE");

printf("\n5. CALCULATION OF DEGREE OF EACH VERTEX"); printf("\n6. CALCULATION OF SELF-LOOPS");

printf("\n7. EXIT"); printf("\n\nINPUT YOUR CHOICE: "); scanf("%d",&choice);

switch(choice)

{

case 1: printf("\nINPUT THE VERTEX TO INSERT: ");

scanf("%d",&v); insVertex(v); break;

case 2: printf("\nINPUT THE VERTEX TO DELETE: ");

scanf("%d",&v); delVertex(v); break;

case 3: printf("\nINPUT THE SOURCE VERTEX: "); scanf("%d",&sE); printf("\nINPUT THE DESTINATION VERTEX: ");

scanf("%d",&dE); insEdge(sE,dE); break;

case 4: printf("\nINPUT THE SOURCE VERTEX: "); scanf("%d",&sE); printf("\nINPUT THE DESTINATION VERTEX: ");

scanf("%d",&dE); delEdge(sE,dE); break;

case 5: printf("\nCALCULATE DEGREE OF VERTEX: ");

scanf("%d", &source); break;

case 6: printf("\nCALCULATE SELF LOOPS ");

scanf("%d", &source); break;

case 7: printf("\n\nExiting...");

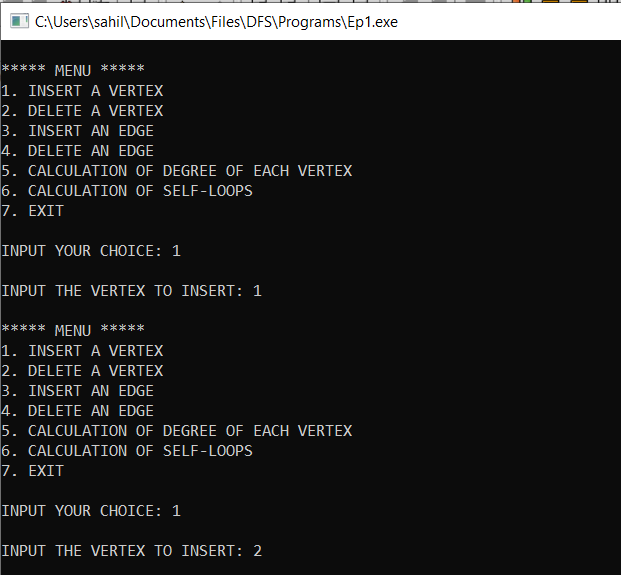
choice = 0; break;

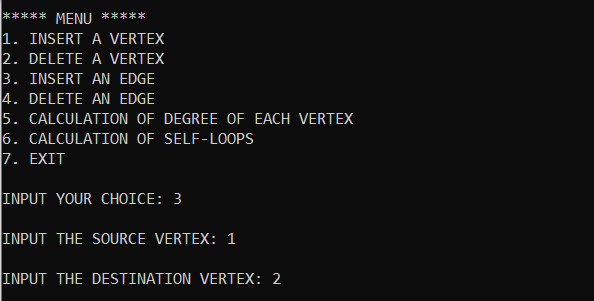
default: printf("\nInvalid Choice!\n");

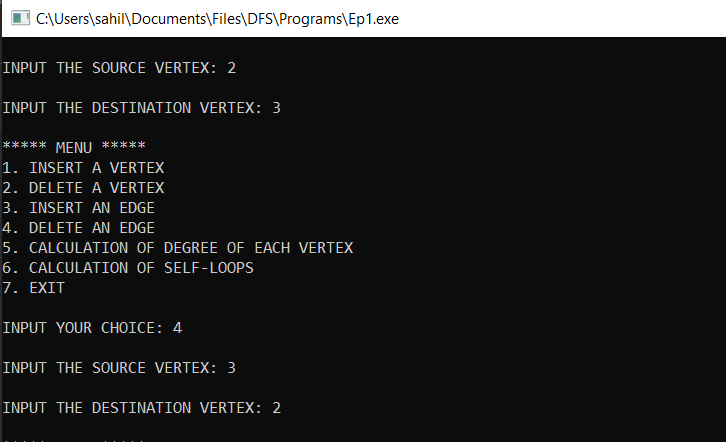
}

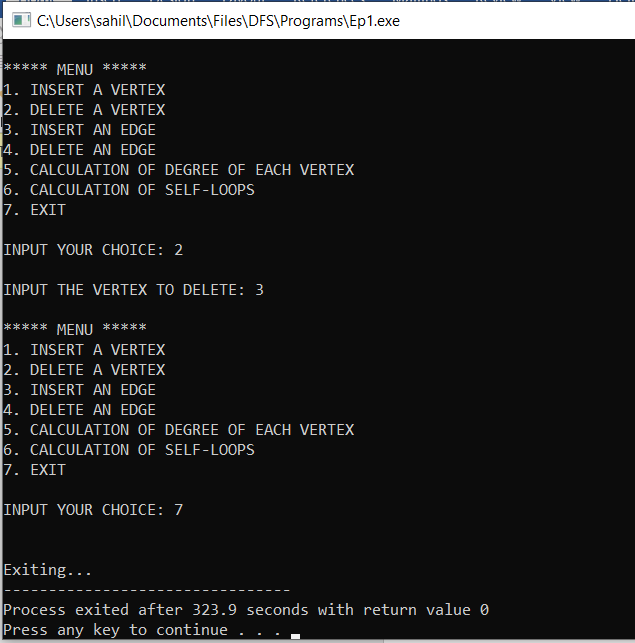
}while(choice);

}



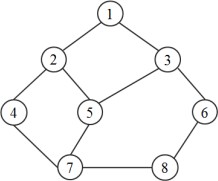






**EP2**

Write a program to traverse the following graph using breadth first search and depth first search techniques.



#include<stdio.h> #include<stdlib.h>

// its still pending only BFS is Done Dfs is left struct queue

{

int size; int f; int r;

int\* arr;

};

int isEmpty(struct queue \*q){ if(q->r==q->f){

return 1;

}

return 0;

}

int isFull(struct queue \*q){ if(q->r==q->size-1){

return 1;

}

return 0;

}

void enqueue(struct queue \*q, int val){ if(isFull(q)){

printf("This Queue is full\n");

}

else{

q->r++;

q->arr[q->r] = val;

// printf("Enqued element: %d\n", val);

}

}

int dequeue(struct queue \*q){ int a = -1;

if(isEmpty(q)){

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

}

else{

q->f++;

a = q->arr[q->f];

}

return a;

}

int main(){

// Initializing Queue (Array Implementation) struct queue q;

q.size = 400;

q.f = q.r = 0;

q.arr = (int\*) malloc(q.size\*sizeof(int));

// BFS Implementation int node,j;

int i = 1;

int visited[8] = {0,0,0,0,0,0,0,0}; int a [8][8] = {

{0,1,1,0,0,0,0,0},

{1,0,0,1,1,0,0,0},

{1,0,0,0,1,1,0,0},

{0,2,0,0,0,0,1,0},

{0,1,1,1,0,0,1,0},

{0,0,1,0,0,0,0,1},

{0,0,0,1,1,0,0,1},

{0,0,0,0,0,1,1,0}

};

printf("\nBFS Traversal : "); printf("%d", i);

visited[i] = 1;

enqueue(&q, i); // Enqueue i for exploration while (!isEmpty(&q))

{

int node = dequeue(&q); for (j = 0; j < 8; j++)

{

if(a[node][j] ==1 && visited[j] == 0){ printf("%d", j);

visited[j] = 1; enqueue(&q, j);

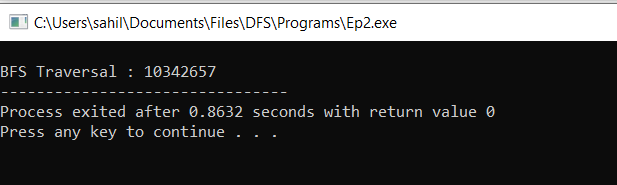
}

}

}

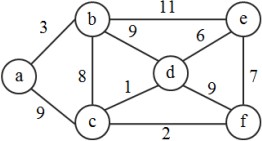
return 0;

}



**EP3**

Write a program to determine shortest path from a to f (using Dijkstra‟s algorithm) in the following graph.



#include<stdio.h> #include<conio.h> #define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode); int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. 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]); printf("\nEnter the starting node:"); scanf("%d",&u);

dijkstra(G,n,u); getch();

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX]; int visited[MAX],count,mindistance,nextnode,i,j; for(i=0;i<n;i++)

for(j=0;j<n;j++) if(G[i][j]==0) cost[i][j]=INFINITY; else cost[i][j]=G[i][j];

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

{

distance[i]=cost[startnode][i]; pred[i]=startnode; visited[i]=0;

}

distance[startnode]=0; visited[startnode]=1; count=1; while(count<n-1)

{

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

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i]; nextnode=i;

}

visited[nextnode]=1; for(i=0;i<n;i++) if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i]; pred[i]=nextnode;

}

count++;

}

for(i=0;i<n;i++) if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]); printf("\nPath=%d",i);

j=i; do

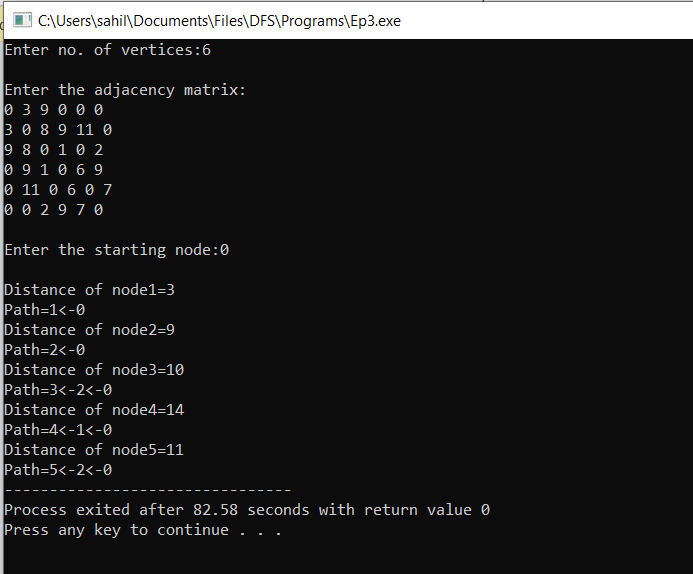
{

j=pred[j]; printf("<-%d",j);

}while(j!=startnode);

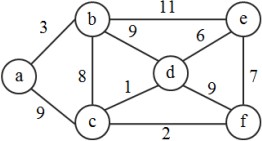
}

}



EP4

Write a program to determine shortest paths between every pair of vertices (using Floyd Warshell‟s algorithm) in the following graph.



#include <stdio.h> #include <conio.h> #define nV 6

#define INF 999

void printMatrix(int matrix[][nV]); void floydWarshall(int graph[][nV]) { int matrix[nV][nV], i, j, k;

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

matrix[i][j] = graph[i][j]; for (k = 0; k < nV; k++) { for (i = 0; i < nV; i++) {

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

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

}

printMatrix(matrix);

}

void printMatrix(int matrix[][nV])

{

int i,j;

printf("The shortest path between all paris of vertices are as follows : \n"); for (i = 0; i < nV; i++) {

for (j = 0; j < nV; j++) { if (matrix[i][j] == INF)

printf("%4s", "INF");

else

}

printf("%4d", matrix[i][j]);

printf("\n");

}

getch();

}

int main() {

int graph[nV][nV] =

{{0, 3, 9, INF,INF,INF},

{3, 0, 8, 9,11,INF},

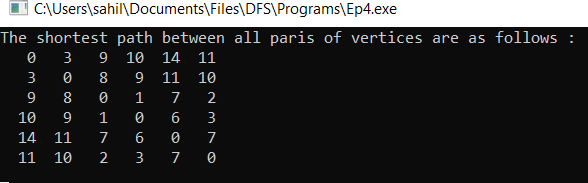
{9, 8, 0, 1,INF,2},

{INF, 9, 1, 0,6,9},

{INF,11,INF,6,0,7},

{INF,INF,2,9,7,0}};

floydWarshall(graph);

}

**FP1**

Write a program that generates n random integers and stores them in a text file, named as “All.txt”. Then, retrieve the stored integers from this file and copy to “Odd.txt” and „Even.txt‟ based upon the type of number, i.e. if the retrieved integer is odd number then store in “Odd.txt” file or if the retrieved integer is even then store in “Even.txt” file. Finally, display the contents of all three files.

#include <stdio.h> #include <stdlib.h>

int isEven(const int NUM);

void printRandoms(int lower, int upper,int count)

{

int i;

for (i = 0; i < count; i++) { int num = (rand() %

(upper - lower + 1)) + lower; printf("%d ", num);

}

}

int main()

{

FILE \* fPtrIn, \* fPtrEven, \* fPtrOdd, \* fPtrPrime; int lower=1000,upper=2000,count=30;

int num, success;

fPtrIn = fopen("all.txt", "w"); if(fPtrIn!=NULL){

int i;

for (i = 0; i < count; i++) { int num = (rand() %

(upper - lower + 1)) + lower; fprintf(fPtrIn, "%d\n", num);

}

}

else{

}

printf("\nunable to open file ");

fclose(fPtrIn);

fPtrIn = fopen("all.txt", "r"); fPtrEven = fopen("even.txt" , "w"); fPtrOdd = fopen("odd.txt" , "w");

if(fPtrIn == NULL || fPtrEven == NULL || fPtrOdd == NULL)

{

printf("\nUnable to open file.\n"); exit(EXIT\_FAILURE);

}

printf("\nFile opened successfully. Reading integers from file. \n\n");

while (fscanf(fPtrIn, "%d", &num) != -1)

{

if (isEven(num))

fprintf(fPtrEven, "%d\n", num); else

fprintf(fPtrOdd, "%d\n", num);

}

fclose(fPtrIn); fclose(fPtrEven); fclose(fPtrOdd);

printf("Data written to files successfully.");

return 0;

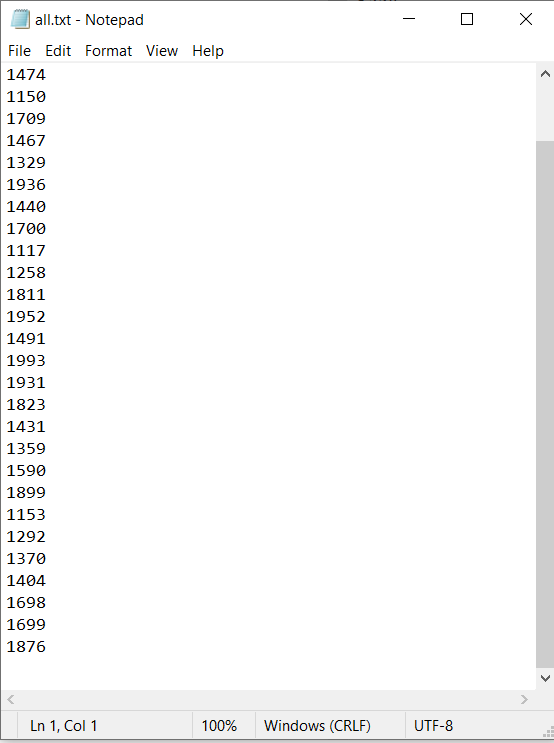
}

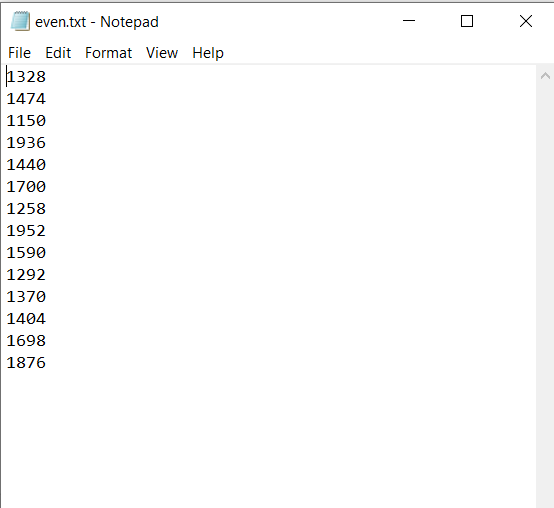
int isEven(const int NUM)

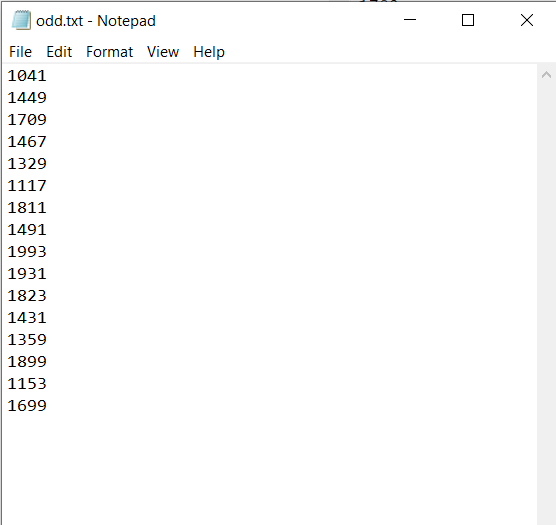
{

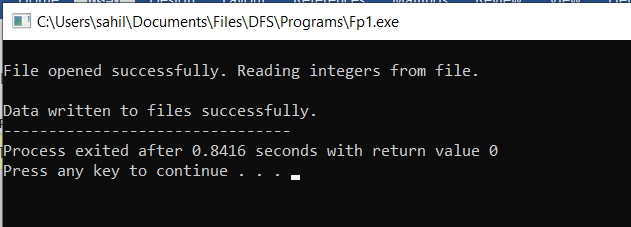
return !(NUM & 1);

}









**FP2**

A text file contains student‟s grade, followed by student‟s name. Sample data is following:

8.3 Gautam

9.4 Jasleen

6.7 Gaurav

9.4 Naman

5.7 Ishika

7.5 Rakesh

Write a program to find the the highest grade, and list all the students who have highest grade. Also, list the details of students‟ having 3rd highest grade.

#include <stdio.h> #include <conio.h>

struct student{ double marks; char name[30];

}s[10];

void main()

{

FILE \*f1;

int num, size=0,i;

double max=0.0,second=0.0,third=0.0; f1 = fopen("students.txt", "r");

while (!feof(f1))

{

fscanf(f1,"%lf",&s[size].marks);

fscanf(f1,"%s",&s[size].name); size++;

}

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

{

if(max<s[i].marks) max=s[i].marks;

}

printf("\nThe highest marks: %.2f",max);

printf("\n\nThe list of students having highest grade: "); for(i=0;i<size;i++)

{

if(s[i].marks==max) printf("\n%s",s[i].name);

}

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

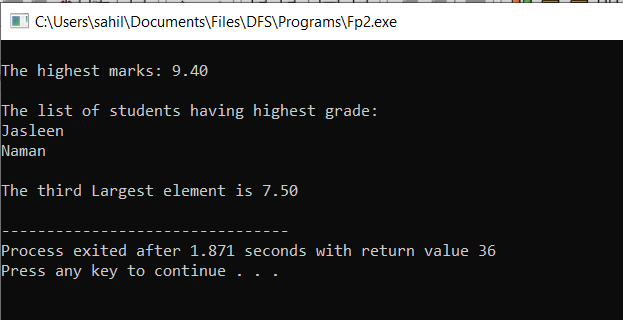
if (s[i].marks > second && s[i].marks < max) second = s[i].marks;

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

if (s[i].marks > third && s[i].marks < second) third = s[i].marks;

printf("\n\nThe third Largest element is %.2f\n", third);

}



FP3

Write a program to implement 20 integers (with some duplicate integers) in a hash table (with separate chaining for collision resolution).

#include<stdio.h> #include<stdlib.h>

#define size 7 struct node

{

int data;

struct node \*next;

};

struct node \*chain[size];

void init()

{

int i;

for(i = 0; i < size; i++) chain[i] = NULL;

}

void insert(int value)

{

//create a newnode with value

struct node \*newNode = malloc(sizeof(struct node)); newNode->data = value;

newNode->next = NULL;

//calculate hash key int key = value % size;

//check if chain[key] is empty if(chain[key] == NULL)

chain[key] = newNode;

//collision else

{

//add the node at the end of chain[key]. struct node \*temp = chain[key]; while(temp->next)

{

temp = temp->next;

}

temp->next = newNode;

}

}

void print()

{

int i;

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

{

struct node \*temp = chain[i]; printf("chain[%d]-->",i); while(temp)

{

printf("%d -->",temp->data); temp = temp->next;

}

printf("NULL\n");

}

}

int main()

{

//init array of list to NULL init();

insert(7); insert(0); insert(3); insert(10); insert(4); insert(5); insert(76); insert(26); insert(37); insert(59); insert(21); insert(65); insert(88); insert(5); insert(10); insert(59); insert(15); insert(26); insert(55); insert(33);

print();

return 0;

}

