TW-1

#include <stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

int numbers[50],tn=-1,to=-1;

char op[50];

void push\_num(int n){

numbers[++tn]=n;

}

void push\_op(char ch){

op[++to]=ch;

}

int pop\_num(){

return numbers[tn--];

}

char pop\_op(){

return op[to--];

}

int infix\_eval(int numbers[50],char op[50]){

int x,y;

char ope;

x=pop\_num();

y=pop\_num();

ope=pop\_op();

switch(ope){

case '+':

return x+y;

case '-':

return y-x;

case '\*':

return x\*y;

case '/':

if(x==0){

printf("\nCannot divide by 0");

exit(0);

}

else{

return y/x;

}

}

return 0;

}

int is\_operator(char ch){

return (ch=='+' || ch=='-'||ch=='\*'|| ch=='/'||ch=='^');

}

int precedence(char c){

switch(c){

case '+':

return 1;

case '-':

return 1;

case '\*':

return 2;

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int eval(char expr[20]){

int i,num,output,r;

char c;

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

c=expr[i];

if(isdigit(c)!=0){

num=0;

while(isdigit(c)){

num=num\*10+(c-'0');

i++;

if(i<strlen(expr)){

c=expr[i];

}

else

break;

}

i--;

push\_num(num);

}

else if(c=='('){

push\_op(c);

}

else if(c==')'){

while(op[to]!='('){

r=infix\_eval(numbers,op);

push\_num(r);

}

pop\_op();

}

else if(is\_operator(c)){

while(to!=-1 && precedence(c)<=precedence(op[to])){

output=infix\_eval(numbers,op);

push\_num(output);

}

push\_op(c);

}

}

while(to!=-1){

output=infix\_eval(numbers,op);

push\_num(output);

}

return pop\_num();

}

int main()

{

char expr[50];

printf("Enter the infix expression to evaluate: ");

gets(expr);

printf("%d",eval(expr));

return 0;

}

TW-2

//TW 2 Conversion of infix exp to postfix exp

#include <stdio.h>

//#include <stdlib.h>

#include <ctype.h>

#define MAX 20

char stack[MAX];

int top=-1;

void push(char x)

{

if(top==MAX-1)

{

printf("\nStack Overflow");

return;

}

stack[++top]=x;

}

char pop()

{

if(top==-1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x=='(')

return 0;

if(x=='+' || x== '-')

return 1;

if(x=='\*' || x== '/')

return 2;

}

int main()

{

char exp[20], \*e, x;

printf("\nEnter the exp: ");

scanf("%s", exp);

e=exp;

while(\*e!='\0')

{

if(isalnum(\*e))

printf("%c", \*e);

else

if(\*e=='(')

push(\*e);

else

if(\*e==')')

{

while((x=pop())!='(')

printf("%c", x);

}

else

{

while(priority(stack[top])>=priority(\*e))

printf("%c", pop());

push(\*e);

}

e++;

}

while(top!=-1)

printf("%c", pop());

return 0;

}

TW-3

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int data;

struct node \*next;

struct node \*prev;

}NODE;

NODE\* add(NODE\*, int);

void disp(NODE\*);

//int search(NODE\*, int);

int main()

{

//Create an empty warehouse

NODE \*head=NULL;

int opt, item;

while(1)

{

printf("\n1: Add item 2: Display ");

printf("3: exit");

printf("\nEnter your option: ");

scanf("%d", &opt);

switch(opt)

{

case 1: printf("\nEnter item to add to list: ");

scanf("%d", &item);

head=add(head, item);

break;

case 2: disp(head);

break;

case 3: exit(0);

}

}

return 0;

}

NODE\* add(NODE\* head, int item)

{

NODE \*start, \*curr;

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

if(newnode==NULL)

{

printf("\nMalloc failure");

exit(1);

}

newnode->data=item;

newnode->next=NULL;

newnode->prev=NULL;

// Case i - List is empty

if(head==NULL)

head=newnode;

else // Case ii - adding the smallest item

if(item < head->data)

{

newnode->next = head;

head = newnode;

}

else // Case iii

{

start=head;

curr=head->next;

while(curr && item > curr->data)

{

start=start->next;

curr=curr->next;

}//end of while

newnode->next=curr;

newnode->prev=start;

start->next=newnode

}//end of else

return head;

}

void disp(NODE \*head)

{

if(head==NULL)

{

printf("\nlist is empty");

return;

}

printf("\nThe list items are: ");

while(head)

{

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

head=head->next;

}

}

TW-4

#include<stdio.h>

#define N 5

int main()

{

int queue[N],ch=1,front=0,rear=0,i,j=1,x=N;

printf("Queue to implement FCFS algorithm:");

printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");

while(ch)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else

{

printf("\n Enter no %d:",j++);

scanf("%d",&queue[rear++]);

}

break;

case 2:

if(front==rear)

{

printf("\n Queue is empty");

}

else

{

printf("\n Deleted Element is %d",queue[front++]);

x++;

}

break;

case 3:

printf("\nQueue Elements are:\n ");

if(front==rear)

printf("\n Queue is Empty");

else

{

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

{

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

printf("\n");

}

break;

case 4:

exit(0);

default:

printf("Wrong Choice");

}

}

}

return 0;

}

TW-5

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left, \*right;

};

struct node \*head = NULL;

void insert(int a)

{

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

struct node \*curr = head;

struct node \*prev = NULL;

temp->data = a;

temp->left = NULL;

temp->right = NULL;

if (head == NULL)

{

head = temp;

return;

}

while (curr != NULL)

{

prev = curr;

if (a > curr->data)

curr = curr->right;

else

curr = curr->left;

}

if (a >= prev->data)

prev->right = temp;

else

prev->left = temp;

}

void inorder(struct node \*n)

{

if (n != NULL)

{

inorder(n->left);

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

inorder(n->right);

}

}

void preorder(struct node \*n)

{

if (n != NULL)

{

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

preorder(n->left);

preorder(n->right);

}

}

void postorder(struct node \*n)

{

if (n != NULL)

{

postorder(n->left);

postorder(n->right);

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

}

}

int main()

{

int a, ch;

while (1)

{

printf("Enter no.:");

scanf("%d", &a);

insert(a);

printf("Press 1 to enter more data:\n");

scanf("%d", &ch);

if (ch != 1)

{

break;

}

}

printf("Inorder\n");

inorder(head);

printf("\nPreorder\n");

preorder(head);

printf("\nPostorder\n");

postorder(head);

}

TW-6

#include<stdio.h>

#include<time.h>

int main()

{

int A[1000],n,k,f=0,i,j;

clock\_t s,e;

double t;

printf("\n enter n value");

scanf("%d",&n); // read n value

for(i=0;i<n;i++) // read data

A[i]=i;

printf("\n Array Elements are ...");

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

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

printf("\n Enter Key element");

scanf("%d", &k); // read key element

s=clock(); // Start Processor clock

for(i=0;i<10000;i++) // Delay loop

for(j=0;j<10000;j++)

f=Binary\_Search(A,k,0,n-1);

e=clock(); // End Processor clock

t=(double)(e-s)/CLK\_TCK; // Calculate total time taken

printf("Time taken = %lf", t);

if(f==1)

printf("Key element found");

else

printf("Key element not found");

}

int Binary\_Search(int A[],int k,int l,int h) // Binary Search code

{

int m;

while(l<=h)

{

m=(l+h)/2;

if(A[m]==k)

return 1;

else

if(k>A[m])

l=m+1;

else

h=m-1;

}

return 0;

}

TW-7

#include <stdio.h>

#include <stdlib.h>

#include<time.h>

#define MAX 20000

int A[MAX];

void Quicksort( int low , int high);

int Partition(int low , int high);

void swap(int \*p , int \*q);

int main()

{

int n,i,j;

int low , high;

clock\_t s , e ;

double cpu\_exe\_t;

printf("\nPlease enter the size of the array :");

scanf("%d",&n);

/\*printf("\nPlease enter the elements of the array :");\*/

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

{

A[i]=rand()%100; /\*scanf("%d",&A[i]);\*/

}

printf("\nThe array elements are :\n");

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

{

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

}

s=clock();

for(j=0;j<1000;j++) //Delay loops

for(i=0;i<1000;i++)

{

low=0;

high=n-1;

Quicksort(low,high);

}

e=clock();

cpu\_exe\_t=(double)(e-s)/CLK\_TCK;

printf("\nThe sorted array is :\n");

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

{

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

}

printf("\nCPU execution time is %lf",cpu\_exe\_t);

return 0;

}

int Partition( int low , int high)

{

int i,j;

int pivot=A[low];

i=low;

j=high+1;

while(i<j)

{

do

{

++i;

}while(A[i]<=pivot);

do

{

--j;

}while(A[j]>pivot);

if(i<j)

{

swap(&A[i],&A[j]);

/\*t=A[i];

A[i]=A[j];

A[j]=t;\*/

}

}

swap(&A[low],&A[j]);

/\*t=A[low];

A[low]=A[j];

A[j]=t;\*/

return j;

}

void Quicksort( int low , int high)

{

int j;

if(low<high)

{

j=Partition(low,high);

Quicksort(low , j-1);

Quicksort(j+1 , high);

}

}

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

{

int t;

t= \*a;

\*a=\*b;

\*b=t;

}

TW-8

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

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

scanf("%d",&n);

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

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

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

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

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

if(cost[i][j]< min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n Minimun cost=%d",mincost);

}

TW-9

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

void Floyd(int W[][10],int n)

{

int i,j,k;

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

{ printf("matrix D[%d]\n", k) ;

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

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

{

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

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

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

}

printf("\n");

}

}

}

int main()

{

int i,j,n;

int W[MAX][MAX];

printf("Floyds Algorithm \n");

printf("Enter number of nodes :");

scanf("%d",&n);

printf("\n Enter the edge weights :");

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

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

scanf(“%d”, &W[i][j]);

Floyd(W,n);

return 0;

}

TW-10

#include<stdio.h>

//#include<conio.h>

#include<math.h>

int a[30],count=0;

//row= index & column=value ie i=row and a[i]=column

int place(int pos)

{

int i;

for(i=1;i<pos;i++) //from 1st row to (current row-1)to check with all prev queens

{

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0; //if a queen exists in same column or diagonally

}

return 1;

}

void printsol(int n)

{

int i,j;

count++;

printf("\n\nSolution #%d \n\n",count);

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

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

{ if(a[i]==j) printf("Q\t");

else printf("\*\t");

}

printf("\n");

}

}

void queen(int n)

{

int k=1;

a[k]=0;

while(k!=0)

{

a[k]=a[k]+1;

while(a[k]<=n && !place(k)) //while columns are left and k cant be placed

a[k]++;

if(a[k]<=n)

{

if(k==n)

printsol(n);

else

{

k++;

a[k]=0;

}

}

else

k--;

}

}

void main()

{ int n;

printf("Enter the Number of queens\n");

scanf("%d",&n);

queen(n);

printf("\nTotal Number of Solutions=%d",count);

}