**Assignment No. 1**

**Program: Factorial using recursive function**

#include<iostream>

using namespace std;

int factorial(int n);

int main()

{

int n;

cout<< "Enter a positive integer: ";

cin>> n;

cout<< "Factorial of " << n << " = " << factorial(n);

return 0;

}

int factorial(int n)

{

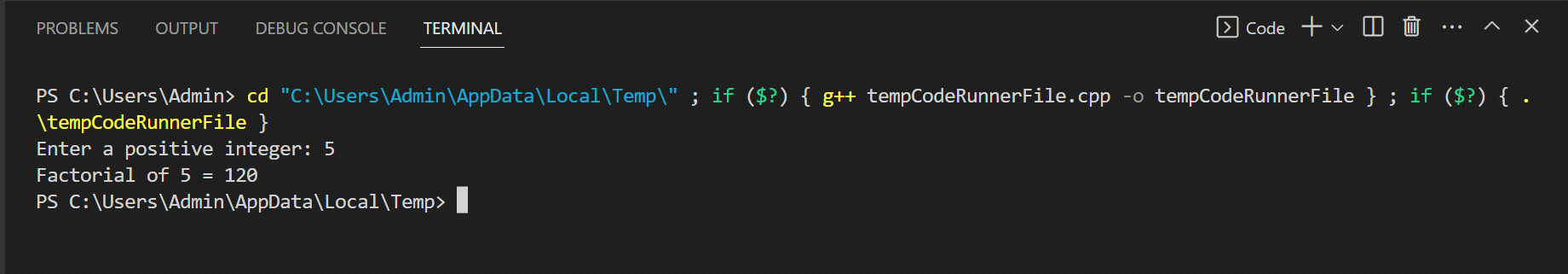
if(n > 1)

return n \* factorial(n - 1);

else

return 1;

}



**//Program: factorial using iterative function**

#include <iostream>

using namespace std;

int fact\_iter(int n){

int result = 1;

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

result \*= i;

}

return result;

}

int main(){

int n;

while (1){

cout<<"Enter interger (0 to exit): ";

cin>>n;

if (n == 0)

break;

cout<<fact\_iter(n)<<endl;

}

return 0;

}



**Assignment No. 2**

***Program For Binary Search***

#include<iostream>

using namespace std;

int main()

{

int i, arr[10], num, first, last, middle;

cout<<"Enter 10 Elements (in ascending order): ";

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

cin>>arr[i];

cout<<"\nEnter Element to be Search: ";

cin>>num;

first = 0;

last = 9;

middle = (first+last)/2;

while(first <= last)

{

if(arr[middle]<num)

first = middle+1;

else if(arr[middle]==num)

{

cout<<"\nThe number, "<<num<<" found at Position "<<middle+1;

break;

} else

last = middle-1;

middle = (first+last)/2;

}

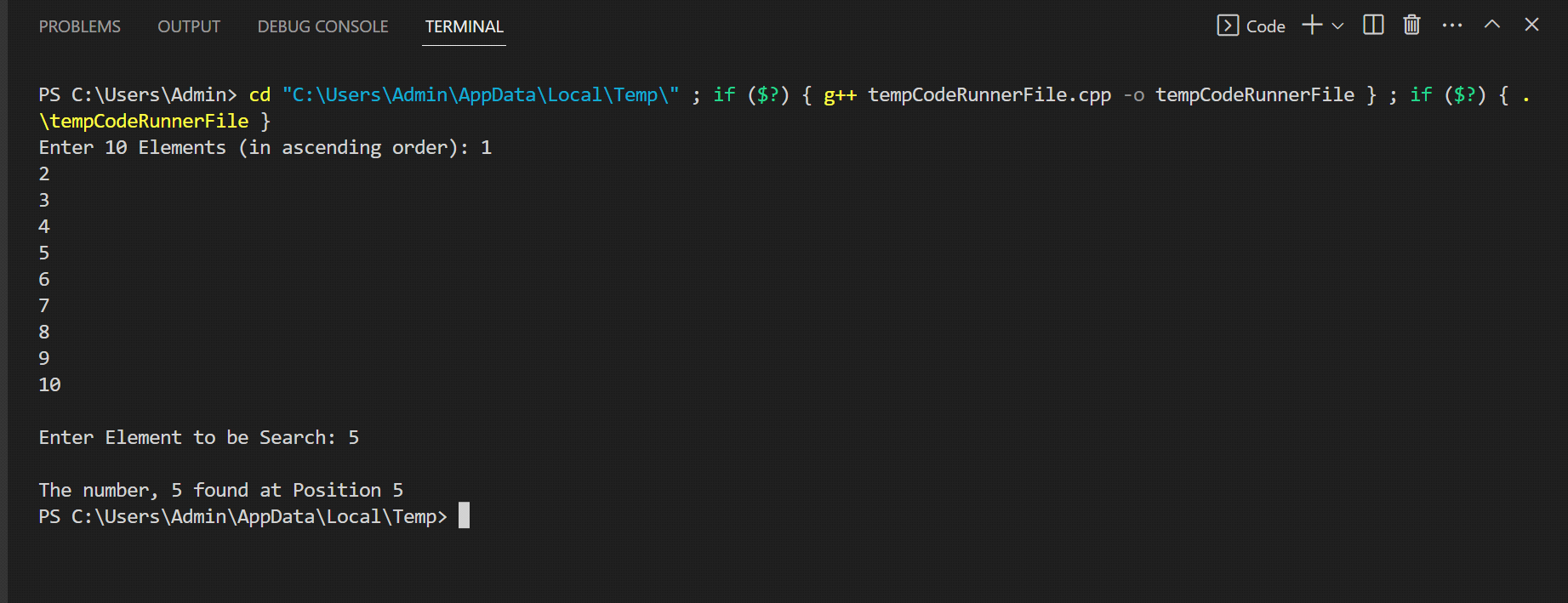
if(first>last)

cout<<"\nThe number, "<<num<<" is not found in given Array";

cout<<endl;

return 0;

}



**Assignment No.3**

***//Program: Write a program to implement quick sort in C++ language.***

#include <iostream>

using namespace std;

int partition (int a[], int start, int end)

{int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

// If current element is smaller than the pivot

if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}int t = a[i+1];

a[i+1] = a[end];

a[end] = t;

return (i + 1);}

/\* function to implement quick sort \*/

void quick(int a[], int start, int end) /\* a[] = array to be sorted, start = Starting index, end = Ending index \*/

{

if (start < end){

int p = partition(a, start, end); //p is the partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}}

/\* function to print an array \*/

void printArr(int a[], int n){

int i;

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

cout<<a[i]<< " ";}

int main(){

int a[] = { 23, 8, 28, 13, 18, 26 };

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

cout<<"Before sorting array elements are - \n";

printArr(a, n);

quick(a, 0, n - 1);

cout<<"\nAfter sorting array elements are - \n";

printArr(a, n);

return 0;}



--**Assignment No.4 (Part-A)**

-**#include <iostream>**

**#include <algorithm>**

**using namespace std;**

**const int MAX = 1e4 + 5;**

**int id[MAX], nodes, edges;**

**pair <long long, pair<int, int> > p[MAX];**

**void init(){**

**for(int i = 0;i < MAX;++i)**

**id[i] = i;**

**}int root(int x){**

**while(id[x] != x){**

**id[x] = id[id[x]];**

**x = id[x];**

**} return x;}**

**void union1(int x, int y){**

**int p = root(x);**

**int q = root(y);**

**id[p] = id[q];**

**}long long kruskal(pair<long long, pair<int, int> > p[]){**

**int x, y;**

**long long cost, minimumCost = 0;**

**for(int i = 0;i < edges;++i){**

**x = p[i].second.first;**

**y = p[i].second.second;**

**cost = p[i].first;**

**if(root(x) != root(y)){**

**minimumCost += cost;**

**union1(x, y);}**

**} return minimumCost;**

**}int main(){**

**int x, y;**

**long long weight, cost, minimumCost;**

**init();**

**cout <<"Enter Nodes and edges";**

**cin >> nodes >> edges;**

**for(int i = 0;i < edges;++i){**

**cout<<"Enter the value of X, Y and edges";**

**cin >> x >> y >> weight;**

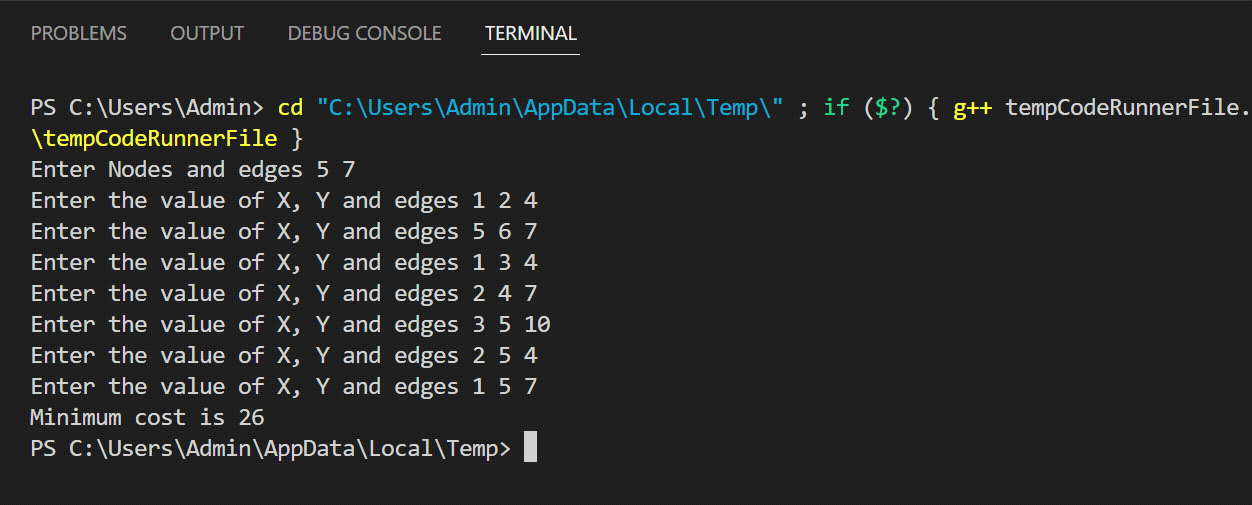
**p[i] = make\_pair(weight, make\_pair(x, y));**

**}sort(p, p + edges);**

**minimumCost = kruskal(p);**

**cout <<"Minimum cost is "<< minimumCost << endl;**

**return 0;}**



-------------------------------------------------xxxxxxxxxxxxx-------------------------------------------------------------

// Kirti Lulle

// SY B\_29

//Pratical 5

#include<iostream>

#include<stdio.h>

using namespace std;

int shortest(int ,int);

int cost[10][10],dist[20],i,j,n,k,m,S[20],v,totcost,path[20],p;

main()

{

int c;

cout <<"enter no of vertices: ";

cin >> n;

cout <<"enter no of edges: ";

cin >>m;

cout <<"\nenter EDGE Cost:";

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

{

cin >> i >> j >>c;

cost[i][j]=c;

}

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

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

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

cost[i][j]=31999;

cout <<"enter initial vertex: ";

cin >>v;

cout << v<<"\n";

shortest(v,n);

}

int shortest(int v,int n)

{

int min;

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

{

S[i]=0;

dist[i]=cost[v][i];

}

path[++p]=v;

S[v]=1;

dist[v]=0;

for(i=2;i<=n-1;i++)

{

k=-1;

min=31999;

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

{

if(dist[j]<min && S[j]!=1)

{

min=dist[j];

k=j;

}

}

if(cost[v][k]<=dist[k])

p=1;

path[++p]=k;

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

cout<<path[j];

cout <<"\n";

//cout <<k;

S[k]=1;

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

if(cost[k][j]!=31999 && dist[j]>=dist[k]+cost[k][j] && S[j]!=1)

dist[j]=dist[k]+cost[k][j];

}

}

OUTPUT:

/tmp/cTVNemssYL.o

enter no of vertices: 5

enter no of edges: 3

enter EDGE Cost:4 5

5 6

7 8

9 10

11 12

enter initial vertex: 12

121

122

123

-------------------------------------xxxxxxxxxxxxxxxxxx----------------------------------------------------------------

// Kirti Lulle

// SY B\_29

//Pratical 6

#include <iostream>

using namespace std;

int knaps(int n,int m,int w[],int p[])

{

int i,j;

int knapsack[n+1][m+1];

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

knapsack[0][j]=0;

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

knapsack[i][0]=0;

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

{

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

{

if(w[i-1]<=j)

knapsack[i][j]=max(knapsack[i-1][j],p[i-1]+knapsack[i-1][j-w[i-1]]);

else

knapsack[i][j]=knapsack[i-1][j];

}

}

return knapsack[n][m];

}

int main()

{

int i,j,n,m;

cout<<"\nEnter number of item:\t";

cin>>n;

int w[n];

int p[n];

cout<<"\nEnetr weight & price of items:\t";

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

{

cin>>w[i]>>p[i];

}

cout<<"\nEnter capacity of knapsack:\t";

cin>>m;

int result=knaps(n,m,w,p);

cout<<"\nMaximum value that can be stored is:\t"<<result;

return 0;

}

OUTPUT

/tmp/cTVNemssYL.o

Enter number of item: 5

Enetr weight & price of items: 1 2

2 3

4 5

6 7

8 9

Enter capacity of knapsack: 5

Maximum value that can be stored is: 7

---------------------------------------------------------xxxxxxxxxxxxxxxx------------------------------------------------------

// Kirti Lulle

// SY B\_29

//Pratical 7

#include<iostream>

using namespace std;

#define INT\_MAX 999999

int n=4;

int dist[10][10] = {

{0,20,42,25},

{20,0,30,34},

{42,30,0,10},

{25,34,10,0}

};

int VISITED\_ALL = (1<<n) -1;

int dp[16][4];

int tsp(int mask,int pos){

if(mask==VISITED\_ALL){

return dist[pos][0];

}

if(dp[mask][pos]!=-1){

return dp[mask][pos];

}

//Now from current node, we will try to go to every other node and take the min ans

int ans = INT\_MAX;

//Visit all the unvisited cities and take the best route

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

if((mask&(1<<city))==0){

int newAns = dist[pos][city] + tsp( mask|(1<<city), city);

ans = min(ans, newAns);

}

}

return dp[mask][pos] = ans;

}

int main(){

/\* init the dp array \*/

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

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

dp[i][j] = -1;

}

}

cout<<"Travelling Salesman Distance is "<<tsp(1,0);

return 0;

}

OUTPUT:

/tmp/cTVNemssYL.o

Travelling Salesman Distance is 85

// Kirti Lulle

// SY B\_29

//Pratical 8

------------------------------------------xxxxxxxxxxxxxx------------------------------------------------------------------

#include <iostream>

#include <cstdio>

#include <cstdlib>

#define N 8

using namespace std;

/\* print solution \*/

void printSolution(int board[N][N])

{

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

{

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

cout<<board[i][j]<<" ";

cout<<endl;

}

}

/\* check if a queen can be placed on board[row][col]\*/

bool isSafe(int board[N][N], int row, int col)

{

int i, j;

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

{

if (board[row][i])

return false;

}

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

{

if (board[i][j])

return false;

}

for (i = row, j = col; j >= 0 && i < N; i++, j--)

{

if (board[i][j])

return false;

}

return true;

}

/\*solve N Queen problem \*/

bool solveNQUtil(int board[N][N], int col)

{

if (col >= N)

return true;

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

{

if ( isSafe(board, i, col) )

{

board[i][col] = 1;

if (solveNQUtil(board, col + 1) == true)

return true;

board[i][col] = 0;

}

}

return false;

}

/\* solves the N Queen problem using Backtracking.\*/

bool solveNQ()

{

int board[N][N] = {0};

if (solveNQUtil(board, 0) == false)

{

cout<<"Solution does not exist"<<endl;

return false;

}

printSolution(board);

return true;

}// Main

int main()

{

solveNQ();

return 0;

}

OUTPUT:

/tmp/cTVNemssYL.o

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 1 0 0

1. 0 1 0 0 0 0 0

---------------------------------------------------------xxxxxxxxxxxxx----------------------------------------------------------

// Kirti Lulle

// SY B\_29

//Pratical 9

#include<iostream>

using namespace std;

int main()

{

int i,j,k,n,min,g[20][20],c[20][20],s,s1[20][1],s2,lb;

cout << ("\n TRAVELLING SALESMAN PROBLEM");

cout << ("\n Input number of cities:");

cin >> n;

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

{

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

{

c[i][j]=0;

}}

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

{

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

{

if(i==j)

continue;

else{

cout<<"input"<<" " <<i<<" to"<< " "<<j<<" cost:";

cin>>c[i][j];

}

}

}

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

{

g[i][0]=c[i][1];

}

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

{

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

{

if(i!=j)

g[i][j]=c[i][j]+g[j][0];

}

}

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

{

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

{

if(i!=j)

break;

}

}

for(k=2;k<=n;k++){

if(i!=k && j!=k){

if((c[i][j]+g[i][k])<(c[i][k]+g[k][j]))

{

g[i][j]=c[i][j]+g[j][k];

s1[i][j]=j;

}

else

{

g[i][1]=c[i][k]+g[k][j];

s1[i][1]=k;

}

}

}

min=c[1][2]+g[2][1];

s=2;

for(i=3;i<n;i++)

{

if((c[i][i]+g[i][i])<min)

{

min=c[1][i]+g[i][1];

s=i;

}

}

int y=g[i][1]+g[i][j]+g[i][i];

lb=(y/2);

cout<<"Edge Matrix";

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

{

cout<<"\n";

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

{

cout<<"\t"<<c[i][j];

}

}

cout<<"\n min"<<" "<<min;

cout<<"\n\b"<<lb;

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

{

if(s!=i && s1[s][1]!=i)

{

s2=i;

}

}

cout<<"\n"<<1<<"-->"<<s<<"-->"<<s1[s][1]<<"-->"<<s2<<"-->"<<1<<"\n";

return (0);

}

Output

/tmp/cTVNemssYL.o

TRAVELLING SALESMAN PROBLEM

Input number of cities:3

input 1 to 2 cost:56

input 1 to 3 cost:45

input 2 to 1 cost:63

input 2 to 3 cost:24

input 3 to 1 cost:84

input 3 to 2 cost:22

Edge Matrix

0 56 45

63 0 24

84 22 0

min 56

42

1-->2-->0-->3-->1

------------------------------------------------------------------xxxxxxxxxxxxxx------------------------------------------------

// Kirti Lulle

// SY B\_29

//Pratical 10

#include <iostream>

#include <cstdio>

#include <cstdlib>

#define V 5

using namespace std;

void printSolution(int path[]);

/\* check if the vertex v can be added at index 'pos' in the Hamiltonian Cycle \*/

bool isSafe(int v, bool graph[V][V], int path[], int pos)

{

if (graph [path[pos-1]][v] == 0)

return false;

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

if (path[i] == v)

return false;

return true;

}

/\* solve hamiltonian cycle problem \*/

bool hamCycleUtil(bool graph[V][V], int path[], int pos)

{

if (pos == V)

{

if (graph[ path[pos-1] ][ path[0] ] == 1)

return true;

else

return false;

}

for (int v = 1; v < V; v++)

{

if (isSafe(v, graph, path, pos))

{

path[pos] = v;

if (hamCycleUtil (graph, path, pos+1) == true)

return true;

path[pos] = -1;

}

}

return false;

}

/\* solves the Hamiltonian Cycle problem using Backtracking.\*/

bool hamCycle(bool graph[V][V])

{

int \*path = new int[V];

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

path[i] = -1;

path[0] = 0;

if (hamCycleUtil(graph, path, 1) == false)

{

cout<<"\nSolution does not exist"<<endl;

return false;

}

printSolution(path);

return true;

}

/\* Main \*/

void printSolution(int path[])

{

cout<<"Solution Exists:";

cout<<" Following is one Hamiltonian Cycle \n"<<endl;

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

cout<<path[i]<<" ";

cout<< path[0]<<endl;

}

int main()

{

bool graph1[V][V] = {{0, 1, 0, 1, 0},

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

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

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

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

};

hamCycle(graph1);

bool graph2[V][V] = {{0, 1, 0, 1, 0},

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

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

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

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

};

hamCycle(graph2);

return 0;

}

OUTPUT

/tmp/cTVNemssYL.o

Solution Exists: Following is one Hamiltonian Cycle

0 1 2 4 3 0

Solution does not exist