**Madesh**

**192011133**

**1.Write a program to return all the possible subsets for a given integer array. Return the solution in any order.**

**Input nums= [1,2,3]**

**Output : [ [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]]**

**Program :**

#include <stdio.h>

char string[50], n;

void subset(int, int, int);

int main()

{

int i, len;

printf("Enter the len of main set : ");

scanf("%d", &len);

printf("Enter the elements of main set : ");

scanf("%s", string);

n = len;

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

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

subset(0, 0, i);

}

void subset(int start, int index, int num\_sub)

{

int i, j;

if (index - start + 1 == num\_sub)

{

if (num\_sub == 1)

{

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

printf("%c\n", string[i]);

}

else

{

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

{

for (i = start;i < index;i++)

printf("%c", string[i]);

printf("%c\n", string[j]);

}

if (start != n - num\_sub)

subset(start + 1, start + 1, num\_sub);

}

}

else

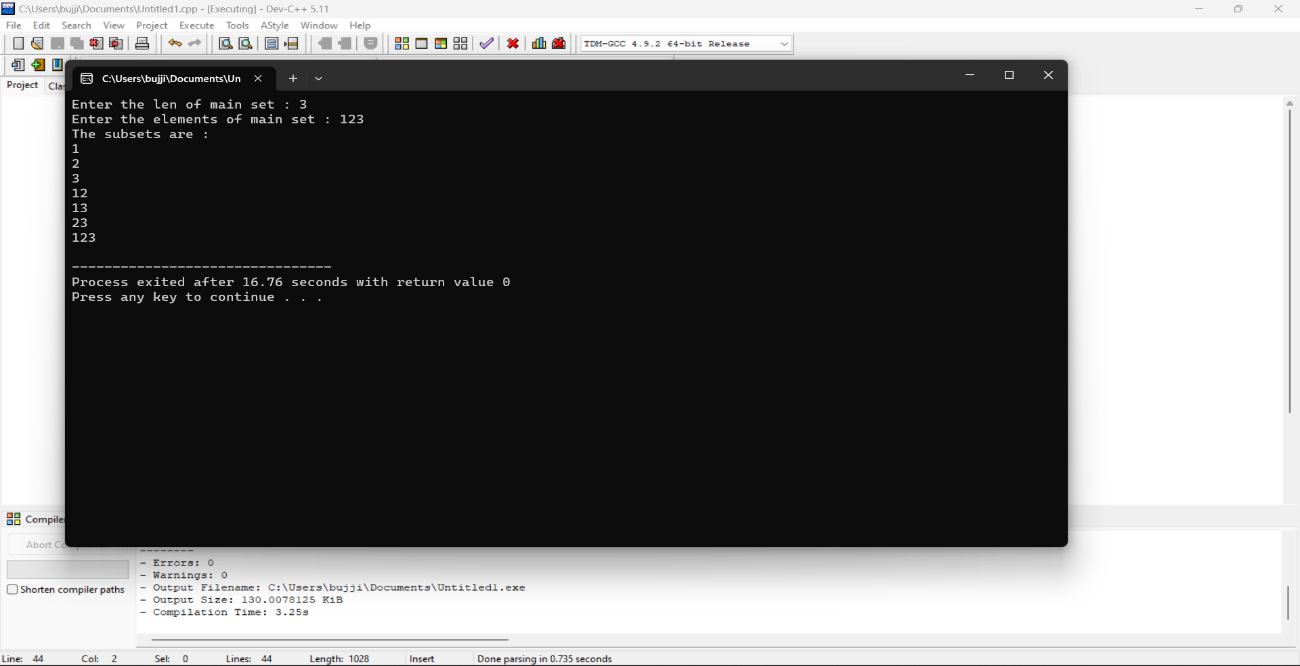
{

subset(start, index + 1, num\_sub);

}

}

**Output :**



**2.Write a program to perform sum of subsets problem using backtracking and estimate time complexity. Identify the test cases.**

**A. Set (s) = (6, 2,8,1,5) sum is 9 B. Set (s) = (6, -4, 7,-1,5, 2,8,1,) sum is 10**

**Program :**

#include <stdio.h>

#include <stdlib.h>

static int total\_nodes;

void printValues(int A[], int size){

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

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

}

printf("\n");

}

void subset\_sum(int s[], int t[], int s\_size, int t\_size, int sum, int ite, int const target\_sum){

total\_nodes++;

if (target\_sum == sum) {

printValues(t, t\_size);

subset\_sum(s, t, s\_size, t\_size - 1, sum - s[ite], ite + 1, target\_sum);

return;

}

else {

for (int i = ite; i < s\_size; i++) {

t[t\_size] = s[i];

subset\_sum(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum);

}

}

}

void generateSubsets(int s[], int size, int target\_sum){

int\* tuplet\_vector = (int\*)malloc(size \* sizeof(int));

subset\_sum(s, tuplet\_vector, size, 0, 0, 0, target\_sum);

free(tuplet\_vector);

}

int main(){

int set[] = { 5, 6, 12 , 54, 2 , 20 , 15 };

int size = sizeof(set) / sizeof(set[0]);

printf("The set is ");

printValues(set , size);

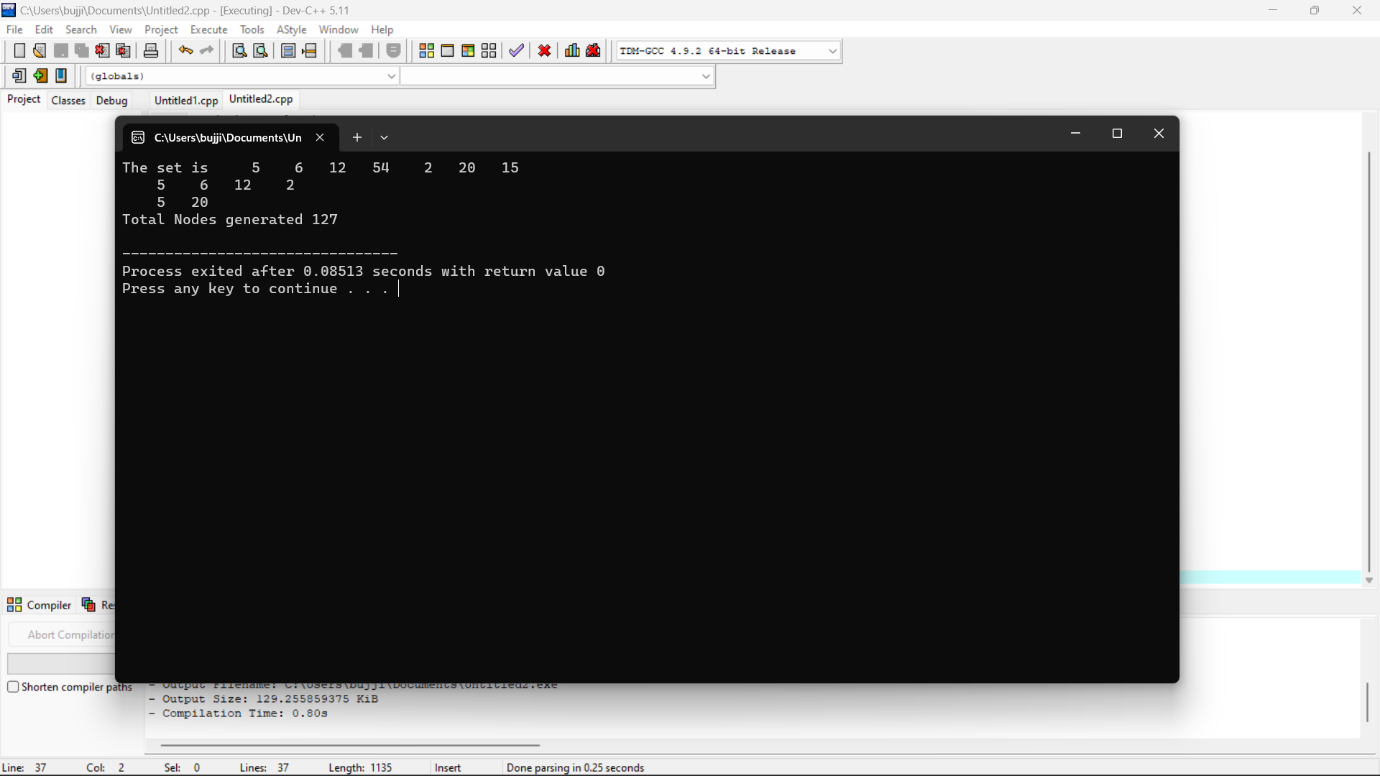
generateSubsets(set, size, 25);

printf("Total Nodes generated %d\n", total\_nodes);

return 0;

}

**Output :**



**3.Determine an optimal tour in a weighted, directed graph. The weights are nonnegative numbers. The inputs are weighted, directed graph, and n, the number of vertices in the graph. The graph is represented by a two-dimensional array W, which has both its rows and columns indexed from 1 to n, where W [i] [j] is the weight on the edge from the ith vertex to the jth vertex. Write a program for travelling salesman problem using dynamic programming for the below given graph.**

**Program :**

#include <stdio.h>

#include <stdbool.h>

#define MAX 20

#define INF 99999

int n, d[MAX][MAX], x[MAX];

int best\_tour\_length = INF, tour\_length[MAX];

void backtrack(int curr\_pos) {

int i;

if (curr\_pos == n) {

tour\_length[curr\_pos] = d[x[n - 1]][x[0]];

int tour = 0;

for (i = 0; i < n; i++) tour += tour\_length[i];

if (tour < best\_tour\_length) best\_tour\_length = tour;

return;

}

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

if (x[i] == -1) {

x[i] = curr\_pos;

tour\_length[curr\_pos] = d[x[curr\_pos - 1]][i];

backtrack(curr\_pos + 1);

x[i] = -1;

}

}

}

int main() {

int i, j;

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

scanf("%d", &n);

printf("Enter the distance matrix:\n");

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

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

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

x[i] = -1;

}

x[0] = 0;

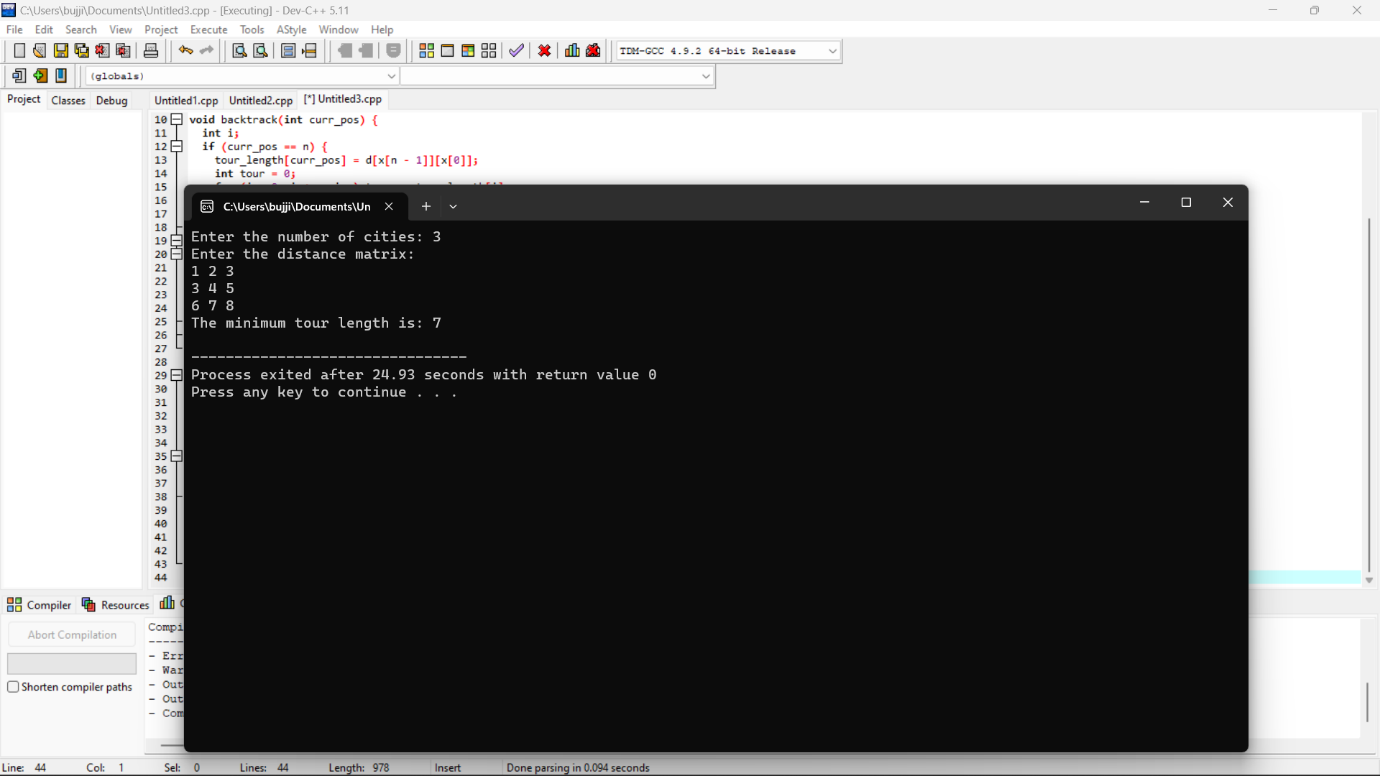
backtrack(1);

printf("The minimum tour length is: %d\n", best\_tour\_length);

return 0;

}

**Output :**



**4.The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other. Given an integer n, return all distinct solutions to the n-queens puzzle. You may return the answer in any order. Write a program for the same.**

**Program :**

#include <stdio.h>

#include <stdbool.h>

#define N 8

int col[N];

bool check(int row) {

int i;

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

if (col[i] == col[row] ||

row - i == col[row] - col[i] ||

row - i == col[i] - col[row])

return false;

return true;

}

void backtrack(int row) {

int i;

if (row == N) {

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

printf("\n");

return;

}

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

col[row] = i;

if (check(row)) backtrack(row + 1);

}

}

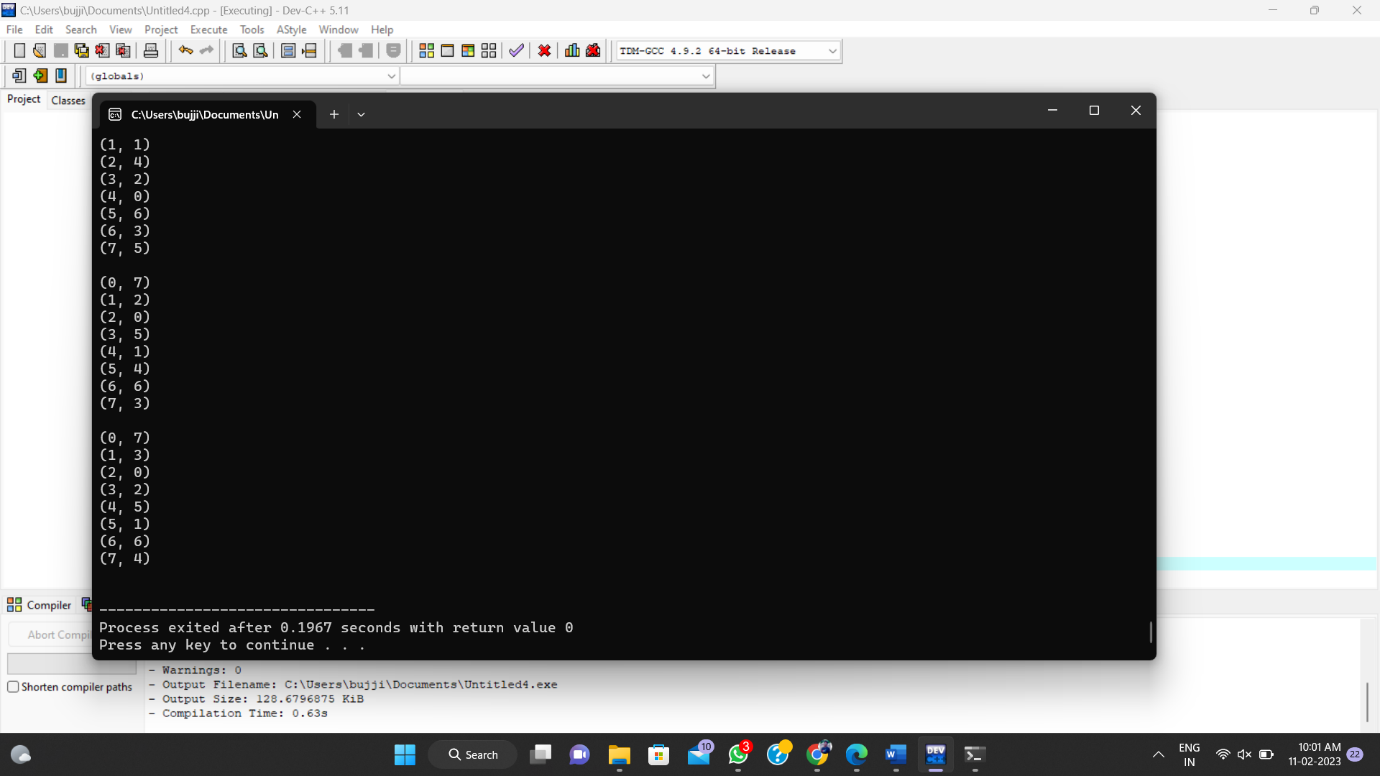
int main() {

backtrack(0);

return 0;

}

**Output :**



**5.Write a program to perform Minimum spanning tree using greedy techniques and estimate time complexity for the given set of values.**

**Program :**

#include <stdio.h>

#include <limits.h>

#define V 5

int minKey(int key[], int mstSet[]) {

int min = INT\_MAX, min\_index;

int v;

for (v = 0; v < V; v++)

if (mstSet[v] == 0 && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

int printMST(int parent[], int n, int graph[V][V]) {

int i;

printf("Edge Weight\n");

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

printf("%d - %d %d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V], i, v, count;

int mstSet[V];

for (v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, V, graph);

}

int main() {

2 3

(0)--(1)--(2)

| / \ |

6| 8/ \5 |7

| / \ |

(3)-------(4)

9 \*/

int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },

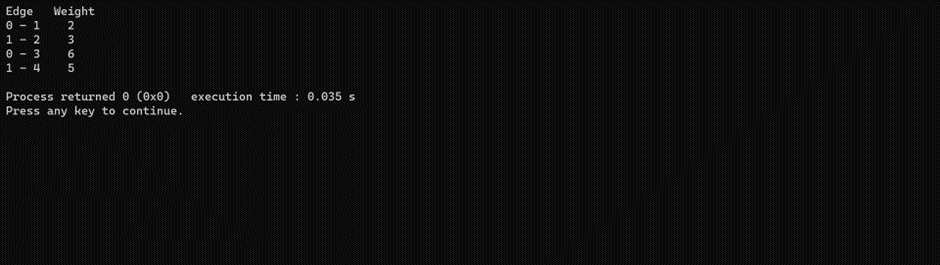
{ 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };

primMST(graph);

return 0;

}

**Output :**



**6.Writa a C program for binary seach tree and find the time complexity**

**Program :**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*left;

struct node\*right;

}\*root=NULL,\*newnode;

struct node\*create(struct node\*root,int ele)

{

if(root==NULL)

{

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

newnode->data=ele;

newnode->left=NULL;

newnode->right=NULL;

return(newnode);

}

else if(ele>root->data)

root->right=create(root->right,ele);

else if(ele<root->data)

root->left=create(root->left,ele);

return(root);

}

void inorder(struct node \*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d\t",root->data);

inorder(root->right);

}

}

void preorder(struct node \*root)

{

if(root!=NULL)

{

printf("%d\t",root->data);

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d\t",root->data);

}

}

int main()

{

int choice;

while(1)

{

printf("\nMAIN MEANU\n");

printf("\n1.CREATE\n");

printf("\n2.INORDER\n");

printf("\n3.PREORDER\n");

printf("\n4.POSTORDER\n");

printf("\n5.EXIT\n");

printf("\nENTER THE CHOICE:\t");

scanf("%d",&choice);

switch(choice)

{

case 1:

int ele;

printf("ENTER THE ELEMENT:");

scanf("%d",&ele);

root=create(root,ele);

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:

exit(0);

break;

default:

printf("\nWRONG CHOICE\n");

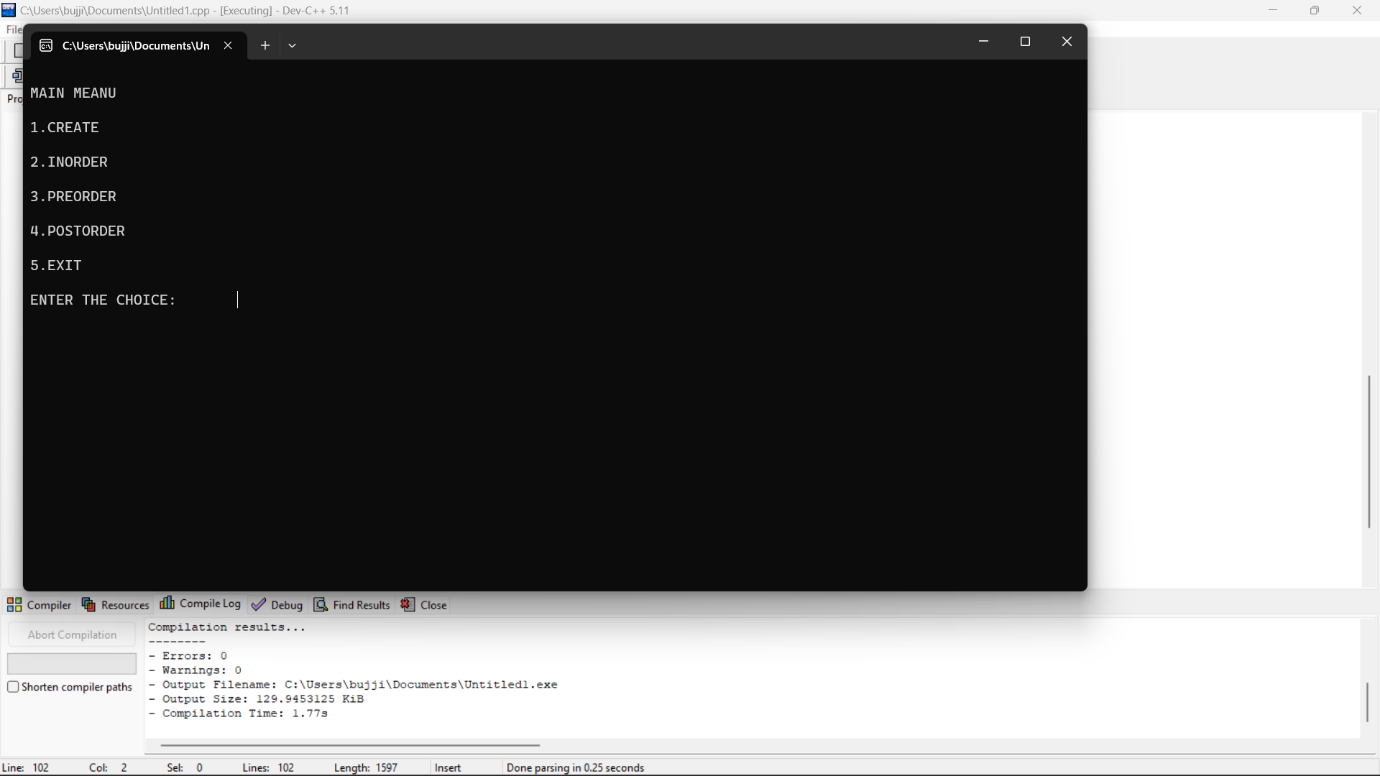
break;

}

}

}

**Output :**



**7.Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized. Write a program to solve a assignment problem for the given data sets using branch and bound.**

**Job 1 Job 2 Job 3 Job 4**

**Person A 12 8 9 10**

**Person B 11 10 10 9**

**Person C 9 11 8 12**

**Person D 11 9 23 7**

**Program :**

**­­­**#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct Job {

char id;

int dead;

int profit;

} Job;

int compare(const void\* a, const void\* b)

{

Job\* temp1 = (Job\*)a;

Job\* temp2 = (Job\*)b;

return (temp2->profit - temp1->profit);

}

int min(int num1, int num2)

{

return (num1 > num2) ? num2 : num1;

}

void printJobScheduling(Job arr[], int n)

{

qsort(arr, n, sizeof(Job), compare);

int result[n];

bool slot[n];

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

slot[i] = false;

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

for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {

if (slot[j] == false) {

result[j] = i;

slot[j] = true;

break;

}

}

}

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

if (slot[i])

printf("%c ", arr[result[i]].id);

}

int main()

{

Job arr[] = { { 'a', 12, 8, 9, 10 },

{ 'b', 11, 10, 10, 9 },

{ 'c', 9, 11, 8, 12 },

{ 'd', 11, 9, 23, 7 } };

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

printf(

"Following is maximum profit sequence of jobs \n");

printJobScheduling(arr, n);

return 0;

}

