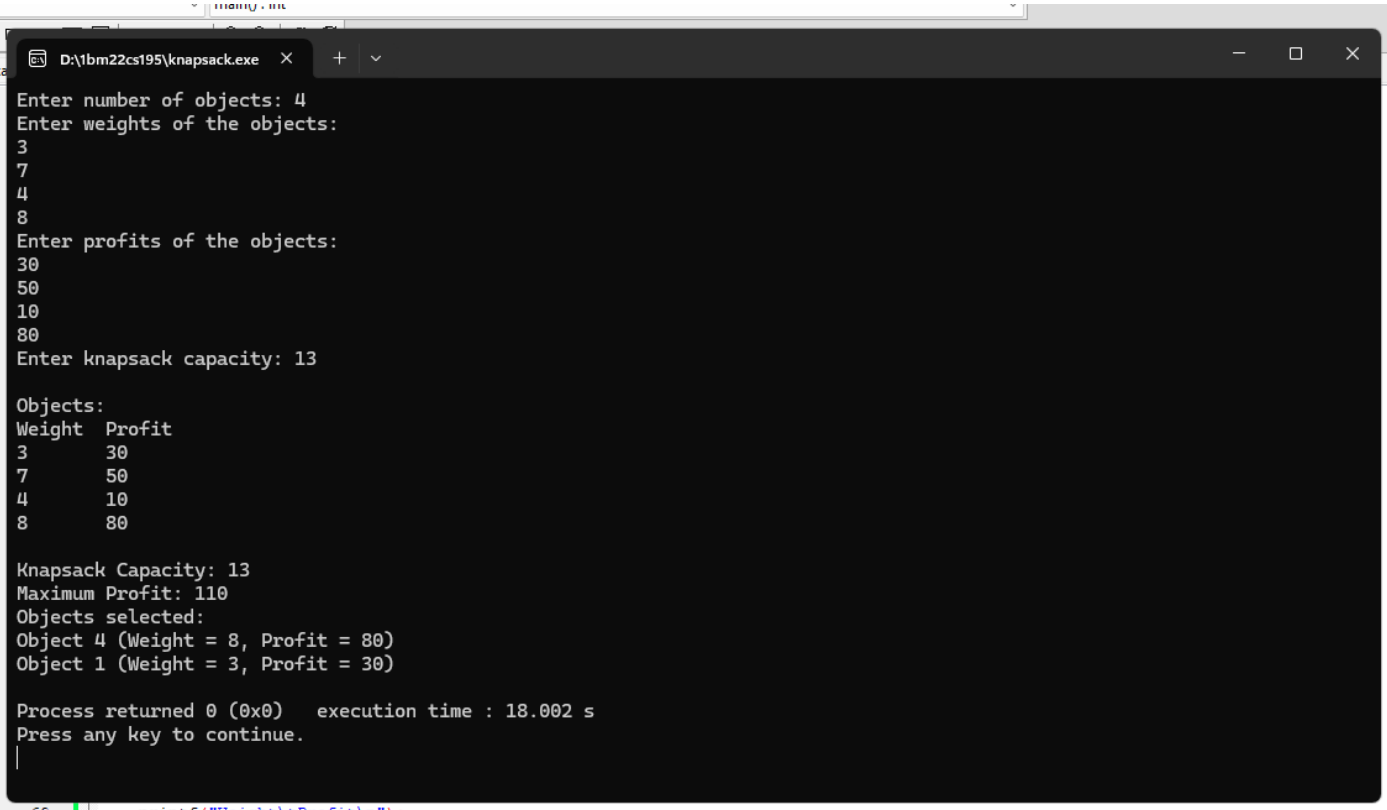
```c
#include <stdio.h>

// Function to find maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve the Knapsack problem using Dynamic Programming
void knapsack(int n, int capacity, int weights[], int profits[]) {
    int dp[n+1][capacity+1]; // DP table
    int i, w;

    // Initialize DP table
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i-1] <= w)
                dp[i][w] = max(profits[i-1] + dp[i-1][w - weights[i-1]], dp[i-1][w]);
            else
                dp[i][w] = dp[i-1][w];
        }
    }

    // Maximum profit will be in dp[n][capacity]
    int max_profit = dp[n][capacity];
    printf("Maximum Profit: %d\n", max_profit);

    // Find objects included in the knapsack
    printf("Objects selected:\n");
    w = capacity;
    for (i = n; i > 0 && max_profit > 0; i--) {
        if (max_profit == dp[i-1][w])
            continue; // Item not included
        else {
            // Item included
            printf("Object %d (Weight = %d, Profit = %d)\n", i, weights[i-1], profits[i-1]);
            max_profit -= profits[i-1];
            w -= weights[i-1];
        }
    }
}

int main() {
    int n; // Number of objects
    printf("Enter number of objects: ");
    scanf("%d", &n);
```

```c
        else {
            // Item included
            printf("Object %d (Weight = %d, Profit = %d)\n", i, weights[i-1], profits[i-1]);
            max_profit -= profits[i-1];
            w -= weights[i-1];
        }
    }
}

int main() {
    int n; // Number of objects
    printf("Enter number of objects: ");
    scanf("%d", &n);

    int weights[n]; // Weights of the objects
    int profits[n]; // Profits of the objects

    // Input weights and profits
    printf("Enter weights of the objects:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }

    printf("Enter profits of the objects:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &profits[i]);
    }

    int capacity; // Capacity of the knapsack
    printf("Enter knapsack capacity: ");
    scanf("%d", &capacity);

    // Display the weights and profits
    printf("\nObjects:\n");
    printf("Weight\tProfit\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\n", weights[i], profits[i]);
    }

    printf("\nKnapsack Capacity: %d\n", capacity);

    // Solve knapsack problem
    knapsack(n, capacity, weights, profits);

    return 0;
}
```




```
69      printf("Weight\tProfit\n");
70      for (int i = 0; i < n; i++) {
71          printf("%d\t%d\n", weights[i], profits[i]);
72      }
73
74      printf("\nKnapsack Capacity: %d\n", capacity);
```

```c
#include <stdio.h>
#define INF 9999

void prim(int n, int cost[n][n]) {
    int s[n]; // To track which vertices are in the MST
    int d[n]; // Shortest distance from source to each vertex
    int p[n]; // Parent array to store the MST

    int i, j, min, source, sum = 0, k = 0;

    // Step 1: Choose a source vertex with minimum edge going out of it
    min = INF;
    source = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (cost[i][j] != 0 && cost[i][j] < min) {
                min = cost[i][j];
                source = i;
            }
        }
    }

    // Step 2: Initialization
    for (i = 0; i < n; i++) {
        s[i] = 0; // Not yet included in MST
        d[i] = cost[source][i]; // Initialize distances from source
        p[i] = source; // Parent of each vertex in MST
    }

    // Step 3: Add source to the MST
    s[source] = 1;

    // Step 4: Find MST
    for (i = 1; i < n; i++) {
        // Find vertex u such that d[u] is minimum and u not in s
        min = INF;
        int u = -1;
        for (j = 0; j < n; j++) {
            if (s[j] == 0 && d[j] < min) {
                min = d[j];
                u = j;
            }
        }

        // Add the edge (u, p[u]) to the MST
        printf("(%d, %d) ", u, p[u]);
        sum += cost[u][p[u]];
```

```
            if (s[j] == 0 && d[j] < min) {
                min = d[j];
                u = j;
            }
        }

        // Add the edge (u, p[u]) to the MST
        printf("(%d, %d) ", u, p[u]);
        sum += cost[u][p[u]];
        s[u] = 1; // Add u to the MST

        // Update d[] and p[] for adjacent vertices of u
        for (j = 0; j < n; j++) {
            if (s[j] == 0 && cost[u][j] < d[j]) {
                d[j] = cost[u][j];
                p[j] = u;
            }
        }
    }

    // Step 5: Check if MST exists
    if (sum >= INF) {
        printf("\nSpanning tree does not exist\n");
    } else {
        printf("\nThe cost of the Minimum Spanning Tree is %d\n", sum);
    }


nt main() {
    int n;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    int cost[n][n];
    printf("Enter the cost adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    printf("\nMinimum Spanning Tree edges:\n");
    prim(n, cost);

    return 0;
}
```

```
int s[n]; // To track which vertices are in the MST
```

D:\1bm22cs195\prims.exe

```
Enter number of vertices: 4
Enter the cost adjacency matrix:
0
9999
3
4
9999
0
8
7
3
8
0
6
4
7
6
0

Minimum Spanning Tree edges:
(2, 0) (3, 0) (1, 3)
The cost of the Minimum Spanning Tree is 14

Process returned 0 (0x0)   execution time : 26.706 s
Press any key to continue.
```

```
}

// Add the edge (u, p[u]) to the MST
printf("(%d, %d) ", u, p[u]);
sum += cost[u][p[u]];
s[u] = 1; // Add u to the MST

// Update d[] and p[] for adjacent vertices of u
```