4) Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

#include <stdio.h>

#include <stdlib.h>

// Function to find maximum of two integers

int max(int a, int b) {

return (a > b) ? a : b;

}

// Function to solve 0/1 Knapsack problem using Dynamic Programming

int knapsack(int W, int wt[], int val[], int n) {

int i, w;

int K[n + 1][W + 1];

// Build table K[][] in bottom up manner

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

for (w = 0; w <= W; w++) {

if (i == 0 || w == 0)

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);

else

K[i][w] = K[i - 1][w];

}

}

return K[n][W];

}

int main() {

int n, W;

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

scanf("%d", &n);

int val[n], wt[n];

printf("Enter the values and weights of items:\n");

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

printf("Value[%d]: ", i + 1);

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

printf("Weight[%d]: ", i + 1);

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

}

printf("Enter the capacity of knapsack: ");

scanf("%d", &W);

printf("Maximum value that can be obtained: %d\n", knapsack(W, wt, val, n));

return 0;

}

OUTPUT:

Enter the number of items: 3

Enter the values and weights of items:

Value[1]: 60

Weight[1]: 10

Value[2]: 100

Weight[2]: 20

Value[3]: 120

Weight[3]: 30

Enter the capacity of knapsack: 50

Maximum value that can be obtained: 220