```
/*
 * Problem Statement :- Write a program to solve a 0-1 Knapsack problem using dynamic
 *                                   programming or branch and bound strategy.
 *
 * Time complexity  : O(n*c)      (n=>number of elements  c=>capacity of knapsack)
 * Space Complexity : O(n*c)
 */

#include<bits/stdc++.h>
using namespace std;

int knapsack(vector<int> val, vector<int> wt, int c)
{
      int n = wt.size();

      vector<vector<int>> dp(n+1, vector<int>(c+1, 0));
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=c; j++)
        {
            dp[i][j] = dp[i-1][j];
            if(wt[i-1] <= j) dp[i][j] = max(dp[i][j], val[i-1]+dp[i-1][j-wt[i-1]]);
        }
    }
    return dp[n][c];
}

int main()
{
      int n, c;

      cout<<"\n\t Enter number of elements : ";
      cin>>n;
    cout<<"\n\t Enter the capacity of knapsack : ";
    cin>>c;

    vector<int> wt(n, 0), val(n, 0);

    cout<<"\n\t Enter the value/cost of all elements : ";
    for(int i=0; i<n; i++) cin>>val[i];

    cout<<"\n\t Enter the weight of all elements : ";
    for(int i=0; i<n; i++) cin>>wt[i];

    int max_val = knapsack(val, wt, c);
    cout<<"\n\t Maximum total value in the knapsack : "<<max_val<<endl;
}
```