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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define max(a,b) (a>b?a:b)
void knapsack( );
void optimal( );
int i, j, x[10], n, m, v[10][10], w[10], p[10], item=0;
void main( )
{
printf("\n**** KNAPSACK PROBLEM ****\n\n");
printf("Enter the total number of items: ");
scanf("%d", &n);
printf("\n\nEnter the weight of each item: ");
for(i=1;i<=n;i++)
scanf("%d", &w[i]);
printf("\n\nEnter the profit of each item: ");
for(i=1;i<=n;i++)
scanf("%d", &p[i]);
printf("\n\nEnter the knapsack capacity: ");
scanf("%d", &m);
knapsack( );
printf("\n\nThe contents of the knapsack table are\n");
for(i=0; i<=n; i++)
{
for(j=0; j<=m; j++)
{
printf("%d\t", v[i][j]);
}
printf("\n");
}
optimal( ); //call optimal function
}
void knapsack( ) /*function to prepare the knapsack table*/
{
for(i=0; i<=n; i++) // every individual item i
{
for(j=0; j<=m; j++) // for the available knapsack capacity j
{
// if there exists no item or sack is full or has 0 capacity
if(i==0 | |j==0) v[i][j]=0;
// if the available capacity is less than the item weight
else if(j < w[i]) v[i][j]=v[i-1][j];
// if the available capacity is sufficient for the item weight
else v[i][j]=max(v[i-1][j], v[i-1][j-w[i]]+p[i]);
}
}
}
void optimal( ) /*function to find the optimal solution*/
{
int i = n, j = m;
while( i != 0 && j != 0)
```

```
{
if(v[i][j] != v[i-1][j])
{
x[i] = 1; j = j-w[i];
}
i = i-1;
}
printf("\n\nOptimal solution is %d\n\n", v[n][m]);
printf("Selected items are: ");
for(i=1; i<= n;i++)
if(x[i] == 1)
{
printf("%d, ", i);
item=1;   // flag item is set if there are any items that can be placed in Knapsack
}
printf("\b\b ");
if(item == 0)
printf("NIL\n\t Sorry ! No item can be placed in Knapsack\n");
printf(“\n********* ********************************************”);
}
```