ALGORITHM:

```
Input:
 Values (stored in array v or profit)
 Weights (stored in array w or weight)
 Number of distinct items (n)
 Knapsack capacity (W)
for j from 0 to W do
  m[0, j] = 0
end for
for i from 1 to n do
  for j from 0 to W do
    if w[i] <= j then
      m[i, j] = max(m[i-1, j], m[i-1, j-w[i]] + v[i])
    else
      m[i, j] = m[i-1, j]
    end if
  end for
end for
```

CODE:

```
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 10;
inline int max(int a, int b);
void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX]
[MAX]);
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int
l[MAX]);<\pre>

/
**********************************************************************
**********
*Function   : main
*Input parameters: no parameters
*RETURNS    :   0 on success
**********************************************************************
**********/<\pre>
    int main(void)
    {
        int i, j, totalProfit;
        int weight[MAX];
        int profit[MAX];
        int capacity;
        int num;
        int loaded[MAX];
        int table[MAX][MAX];
        cout<<"Enter the maxium number of objects : ";
     cin >> num;

        cout << "Enter the weights : \n";
```

```
        for (i=1; i<=num; i++)
    {
        cout << "\nWeight " << i << ": ";
        cin >> weight[i];
    }
    cout << "\nEnter the profits : \n";
    for (i=1; i<=num; i++)
    {
        cout << "\nProfit " << i << ": ";
        cin >> profit[i];
    }
    cout << "\nEnter the maximum capacity : ";
    cin >> capacity;


    totalProfit = 0;

    for( i=1; i<=num; i++)
        loaded[i] = 0;

    fnProfitTable(weight,profit,num,capacity,table);
    fnSelectItems(num,capacity,table,weight,loaded);
    cout << "Profit Matrix\n";
    for (i=0; i<=num; i++)
    {
        for(j=0; j<=capacity; j++)
        {
        cout <<"\t"<<table[i][j];
        }
        cout << endl;
    }

    cout << "\nItem numbers which are loaded : \n{ ";
    for (i=1; i<=num; i++)
    {
        if (loaded[i])
        {
            cout <<i << " ";
            totalProfit += profit[i];
        }
    }
    cout << "}" << endl;

    cout << "\nTotal Profit : " << totalProfit << endl;
    return 0;
}

inline int max(int a, int b)
{
    return a>b ? a : b;
}

/
**********************************************************************
```

```
**********
*Function   : fnProfitTable
*Description    : Function to construct the profit table
*Input parameters:
*   int w[MAX]  -   weight vector
*   int p[MAX]  -   profit vector
*   int n   - no of items
*   int c   - knapsack capacity
*   int t[MAX][MAX] - profit table
*RETURNS    : no value
**********************************************************************
**********/

void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX]
[MAX])
{
    int i,j;

    for (j=0; j<=c; j++)
        t[0][j] = 0;

    for (i=0; i<=n; i++)
        t[i][0] = 0;

    for (i=1; i<=n; i++)
    {
        for (j=1; j<=c; j++)
        {
            if (j-w[i] < 0)
                t[i][j] = t[i-1][j];
            else
                t[i][j] = max( t[i-1][j], p[i] + t[i-1][j-w[i]]);
        }
    }
}

/
**********************************************************************
**********
*Function   : fnSelectItems
*Description: Function to determine optimal subset that fits into
the knapsack
*Input parameters:
*   int n   - no of items
*   int c   - knapsack capacity
*   int t[MAX][MAX] - profit table
*   int w[MAX]  -   weight vector
*   int l[MAX]  -   bit vector representing the optimal subset
*RETURNS    : no value
**********************************************************************
**********/

void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int
l[MAX])
```

```
{
    int i,j;

    i = n;
    j = c;

    while (i >= 1 && j >= 1)
    {
            if (t[i][j] != t[i-1][j])
            {
                l[i] = 1;
                j = j - w[i];
                i--;
            }
            else
                i--;
    }
}
```

OUTPUT:

Enter the maxium number of objects : 4 Enter the weights :

Weight 1: 2

Weight 2: 1

Weight 3: 3

Weight 4: 2

Enter the profits :

Profit 1: 12

Profit 2: 10

Profit 3: 20

Profit 4: 15

Enter the maximum capacity : 5 Profit Matrix 0 0 0 0 0 0 0 0 12 12
12 12 0 10 12 22 22 22 0 10 12 22 30 32 0 10 15 25 30 37

Item numbers which are loaded : { 1 2 4 }

Total Profit : 37