**算法设计：**使用c语言程序设计语言编写源程序。首先声明头文件和常量，定义物品总量n和背包容量c（挑选每一行的属性数值不能超过规定值，这个规定值即可看作是背包的总容量）定义数组m[i][j]并使用非递归算法来求解m[i][j],然后编写一个函数用来寻找问题的最优解，再编写一个函数用于输入问题的最优解，最后在主函数中定义各个物品的价值和重量，最终计算最优解并输出。

**算法分析：**使用这个算法来计算多维背包问题的算法复杂度极高，因为它要一行一行属性进行运行求解，不能一次性直接求解，问题中说明有10个属性，则该算法要运行10次才能求出这个数据集中问题的最优解，操作过程复杂且相对比较麻烦。

**伪代码：**Begin(算法开始)

定义NUM n,CONTENT,v[NUM],w[NUM], c, m[NUM ][CONTENT], I ,j jMax

IF ((w[n])-1) 则jMax = w[n]-1;

否则jMax = c;

/\* 初始化m[n][j] \*/

for(j = 0; j) 则m[n][j] = 0;

for(j = jMax +1; j) 则m[n][j] = v[n];

/\*使用非递归的算法来求解m[i][j] \*/

if(m[i+1][j] >= (m[i+1][j-w[i]]+v[i])) 则m[i][j] = m[i+1][j] ;

否则m[i][j] = m[i+1][j-w[i]]+v[i];

/\*寻找最优解\*/

if(m[i][c] == m[i+1][c]) 则flag[i] = 0;

否则flag[i] = 1; c-=w[i];

if(flag[i] == 1)

printf(" %d %d %d\n",i,w[i],v[i]);

/\*计算最优值\*/

knapsack(value,weight,c,maxvalue);

/\*构造最优解\*/

traceback(flag,weight,maxvalue);

/\*打印程序的结果\*/

printResult(flag,weight,value,maxvalue);

End (算法结束)