template<class Typew , class Typep>

class Knap {

friend Typep Knapsack(Typep\* , Typew\*, typew ,int );

private:  
  
      Typep Bound(int  i);  
      void Backtrack(k(int i);  
      Typew c;  
      int n;  
      Typew  \*w;  
      Typep \*p;  
      Typew cw;  
      Typep  cp;  
      Typep  bestp;  
};  
  
      template<class Typew, class Typep>  
  
      void Knap<Typew, Typep>::Backtrack(int i) {  
  
      if (i>n){  
      bestp = cp;  
      return;  
      }  
  
      if (cw+w[i] <=c) {  
      cw += w[i];  
      cp += p[i];  
  
      Backtrack(i+1);  
      cw -= w[i];  
  
      cp -= p[i];}

if (Bound(i+1) > bestp)  
Backtrack(i+1);  
}  
template<class Typew, class Typep>  
Typep Knap<Typew,  Typep>::Bound(int i) (  
Typew  cleft = c-cw;  
Typep  b=cp;  
while(i<=n8&w[i] <= cleft) {  
cleft -= w[i];  
b += p[i];  
it+;  
}  
  
if(i<=n)  
b += p[i]\*cleft/w[i];  
return b;  
}  
class object {  
friend int Knapsack(int\*, int\*, int, int);  
public:  
int operator<=(0bject a) const { return (d >= a.d); }  
private:  
int ID;  
float d;  
};  
  
template<class Typew, class Typep>  
Typep Knapsack(Typep p[ ], Typew w[ ], Typew c, int n) {  
Typew W = 0;  
Typep P = 0;  
Object \*Q= new Object [n];  
for (int i=1; i <=n; i++) {  
Q[i-1].ID= i;  
Q[i-1].d = 1.0\*p[i]/w[i];  
p+= p[i];  
w += w[i];  
}

if (W<=c)  
return P;  
Sort(Q, n);   
Knap<Typew, Typep> K;  
K.p = new Typep [n+1];  
K.W = new Typew [n+1];  
for (int i=1; i <= n; i++) {  
K.p[i] = p[Q[i-1].ID];  
K.w[i] = w[Q[i-1].ID];}  
K.cp=0;

K.cw .=e;

K.c = c;

K.n = n;

K.bestp = 0;

K. Backtrack(1);

delete[ ] Q;

delete[ ] K.w;

delete[ ] K.p;

return K.bestp;

}