```
#ifndef QVECTOR_H_INCLUDED
#define QVECTOR_H_INCLUDED
#include<iostream>
#include<string.h>
using namespace std;
class Qvector{
    public:
        struct Qpair{//Qvector 内部类
        private:
            char sym;//物品的标识（如：'a','b','c','d','e','f'）
            double w;//物品的重量
            double v;//物品的价值
            double avg;//avg=w/v
            bool sel;//表示物品是否装入背包,0 表示没有装入，1 表示装入
         public:
            Qpair(char ch='$',double tw=1,double tv=0):sym(ch),w(tw),v(tv),sel(false){avg=v/w;}//构造函数
            Qpair(const Qpair &rhs)//拷贝构造函数
            {
              operator=(rhs);
            }
             Qpair& operator=(const Qpair &rhs)//重载运算符=（参数必须为引用，解释略）
            {
                if(this!=&rhs)//防止对自己赋值
                {
                  sym=rhs.sym;
                  w=rhs.w;
                  v=rhs.v;
                  avg=rhs.avg;
                  sel=rhs.sel;
                }
               return *this;
            }
             double getW()const//获取物品的质量
            {
              return w;
            }
            double getV()const//获取物品的价值
            {
               return v;
            }
            char getSym()//获取物品的标识符
            {
                  return sym;
```

```
    }
    double getAvg()const//获取物品的单位价值
    {
     return avg;
    }
    double getSel()const//获取物品是否装入背包
    {
     return sel;
    }
          void setSym(char ch)//设置物品的标识符
    {
          sym=ch;
    }
    void setAvg(bool b)//设置物品的单位价值
    {
       sel=b;
    }
    void setW(double w)//设置物品的重量
    {
    this->w=w;
    }
    void setV(double v)//设置物品的价值
    {
     this->v=v;
    }
   void setSel(bool b)//设置物品是否装入背包
   {
     sel=b;
   }
    bool lessThan(const Qpair& rhs,int i)//i 用来表示用那个数据成员进行比较
    {
        if(i==3)
            return avg<rhs.avg;//贪心策略：选取单位重量价值最大的物品
        if(i==2)
             return v<rhs.v;//贪心策略：选取价值最大者。
        if(i==1)
             return w>rhs.w;//贪心策略：：选取重量最小。
    }
  friend ostream& operator<<(ostream& out,const Qpair &rhs)//重运算符载<<
   {

out<<rhs.sym<<"\t\t"<<rhs.w<<"\t\t"<<rhs.v<<"\t\t"<<rhs.avg<<"\t\t"<<rhs.sel<<endl;
      return out;
   }
```

```
 };
private:
int theSize;
int theCapacity;
Qpair *objects;
public:
Qvector(int i=0):theSize(0),theCapacity(theSize+15)//构造函数
{
    objects=new Qpair[theCapacity];
}
~Qvector()//析构函数
{
    delete[] objects;
}
const Qvector& operator=(const Qvector &rhs)//重运算符载=（参数必须为引用，解释略）
{

    if(this!=&rhs)//判断是否对自己赋值
    {
        delete[] objects;
        theSize=rhs.theSize;
        theCapacity=rhs.theCapacity;
        objects=new Qpair[theCapacity];//因为这句话，犯了很多错误！注意注意！
        for(int k=0;k<rhs.size();++k)
        {
            objects[k]=rhs.objects[k];
        }
    }
    return *this;
}
Qvector( Qvector &rhs):objects(NULL)//拷贝构造函数
{
    operator=(rhs);
}
void reserve(int newCapacity)//重新设置 theCapacity 的大小
{
    if(newCapacity<theCapacity)
        return ;
    Qpair *old=objects;
    theCapacity=newCapacity;
    objects=new Qpair[theCapacity];
    for(int k=0;k<theSize;++k)
    {
        objects[k]=old[k];
```

```
        }
        delete[] old;
  }
  void resize(int newSize)
  {
        if(newSize>theCapacity)
              reserve(newSize*2+1);
        theSize=newSize;
  }
  Qpair & operator[](int index)//重载运算符[]
  {
        return objects[index];
  }
  const Qpair& operator[](int index)const//运算符重载[]
  {
        return objects[index];
  }
int size()const
{
      return theSize;
}
int capacity()const
{
      return theCapacity;
}
void push_back(struct Qpair x)
{
      if(theSize==theCapacity)
      {
         reserve(theCapacity*2+1);
      }
      objects[theSize++]=x;
}
  void sort(int s)//根据所选取的贪心策略，进行排序，并输出结果
  {
        if(0==s) //使用前三种所有贪心策略
        {
              for(int k=1;k<=3;++k)
              {
                    select(k);//排序
                    outPut(k);//贪心选择，并输出结果
                    initSel();//恢复至初始值
              }
              select(3);//第四种贪心策略，进行启发式贪婪算法
```

```
            Qvector temp=(*this);
            if(this->size()>2)
                    for(int k1=0;k1<=2;++k1)//启发式贪婪算法中，k 的预定义值为 2.
                    {
                       Rec(temp,1,k1);
                    }
            else//如果 temp.size()==2 则只能执行此过程。
                    Rec(temp,1,0);
            outPut(4);
       }
       else
       {
            if((s>0)&&(s<4))
            {
                 select(s);
                 outPut(s);
            }
            else//启发式贪婪算法
            {
                 select(3);//按价值密度（即价值重量比）进行排序。
                 Qvector temp=(*this);
                 if(this->size()>2)
                     for(int k1=0;k1<=2;++k1)//启发式贪婪算法中，k 的预定义值为 2.
                     {
                        Rec(temp,1,k1);
                     }
                 else
                     Rec(temp,1,0);
                 outPut(s);
            }
       }
}
void initSel()//对 objects 的各个 Qpair 对象进行恢复至初始值。
   {
     objects[0].setV(0);
     for(int i=1;i<size();++i)
     {
          if(1==objects[i].getSel())
          {
               objects[0].setW(objects[0].getW()+objects[i].getW());
               objects[i].setSel(false);
          }
     }
   }
```

```
private :
  void select(int s)//对输入的每个物品进行按某种策略进行排序（冒泡排序）
  {
      for(int i=1;i<size()-1;++i)
          {
              int flag=0;
              for(int j=1;j<size()-i;++j)
              {
                 if(objects[j].lessThan(objects[j+1],s))
                 {
                 Qpair temp=objects[j];
                 objects[j]=objects[j+1];
                 objects[j+1]=temp;
                 }
                 flag=1;
              }
            if(0==flag)
                 return;
          }
  }
 void Greedy()//进行贪心选择
 {
     int q=1;
     while((q<size()))
     {
          if((objects[q].getW()<=objects[0].getW())&&objects[q].getV()!=0)
         {
          objects[q].setSel(true);
          objects[0].setV(objects[0].getV()+objects[q].getV());
          objects[0].setW(objects[0].getW()-objects[q].getW());
         }
          ++q;
     }
 }
void   outPut(int s)//对结果进行输出。
 {
      if(4!=s)
          Greedy(); //进行贪心选择，启发式贪心策略不使用本 Greedy()函数。
      if(4==s)
           cout<<"//贪心策略 4：启发式贪婪算法";
      if(3==s)
           cout<<"//贪心策略 3：选取单位重量价值最大的物品";
      if(2==s)
           cout<<"//贪心策略 2：选取价值最大者";
```

```
        if(1==s)
            cout<<"//贪心策略 1：选取重量最小";
        cout<<"(isSelected 表示是否将该物品装入背包):"<<endl;
        cout<<"Symbol"<<"\t\t"<<"Weigh";
        if(1==s)
            cout<<"*";
        cout<<"\t\tValue";
        if(2==s)
            cout<<"*";
        cout<<"\t\tValue/Weight";
        if(3==s)
            cout<<"*";
        cout<<"\tisSelected"<<endl;
        cout<<*this<<endl; //运算符重载<<
        cout<<"装入背包物品的最大总价值为: "<<(*this)[0].getV()<<endl<<endl;
    }
    void Deal(Qvector& temp)//启发贪婪算法的处理函数
    {
        double tw=0,tv=0;
        int initW=0,initV=0;
        initW=temp[0].getW();//用来保存 tempp[0]的初始值。
        initV=temp[0].getV();
        for(int i=1;i<temp.size();++i)
        {
            if(temp[i].getSel())
            {
                tw+=temp[i].getW();
                tv+=temp[i].getV();
            }
        }//当 tw>temp[0].getW()时，丢弃。
        if(tw<=temp[0].getW())//如果 S 子集物品总重量小于背包容量
        {
            temp[0].setW(temp[0].getW()-tw);
            temp[0].setV(temp[0].getV()+tv);
            if(temp[0].getW()>0)
            {
                for(int j=1;j<temp.size();++j)//进行贪心选择
                {
                    if(temp[j].getSel())
                        continue;
                    if((temp[j].getW()<=temp[0].getW())&&(temp[j].getV()>0))//当物品 j 的
重量小于当前背包的容量并且价值大于 0 时。
                    {
                     temp[0].setW(temp[0].getW()-temp[j].getW());
```

```
                        temp[0].setV(temp[0].getV()+temp[j].getV());
                        temp[j].setSel(true);
                    }
                }
            }
            if((*this)[0].getV()<temp[0].getV())
            {
                (*this)=temp;
            }
            temp[0].setW(initW);//对 temp[0]进行恢复
            temp[0].setV(initV);
        }
       /*for(int ss=1;ss<temp.size();++ss)
            cout<<temp[ss].getSel()<<' ';
        cout<<endl;*/
    }
    void Rec(Qvector& temp,int po,int rt)//启发式贪婪算法的从 po 到 temp.size()-1 中选取 rt
个 Qpair 对象
    {
        if(0==rt)//递归结束条件
        {
            Deal(temp);
            return;
        }
        if((temp.size()-po)==rt)//递归结束条件
        {
            for(int i=po;i<temp.size();++i)
            {
                temp[i].setSel(true);
            }
            Deal(temp);
        return;
        }
      temp[po].setSel(true);
      Rec(temp,po+1,rt-1);
      for(int k2=po+1;k2<temp.size();++k2)//注意
          temp[k2].setSel(false);

      temp[po].setSel(false);
      Rec(temp,po+1,rt);
      for(int k3=po+1;k3<temp.size();++k3)//注意
          temp[k3].setSel(false);
    }
     friend ostream& operator<<(ostream &out,Qvector &rhs)//重载运算符<<
```

```
    {
        int n=0;
        for(int i=1;i<rhs.size();++i)
        {
            out<<rhs.objects[i];
            if(rhs.objects[i].getSel()!=0)
                ++n;
        }
       cout<<endl<<"背包的剩余容量为: "<<rhs.objects[0].getW()<<endl
                 <<"装入背包的物品数量为："<<n;
        return out;
    }
};
#endif // QVECTOR_H_INCLUDED
```