//1. 学生档案管理（访问公有基类的成员）

//创建一个学生基本信息类BaseInfo，包括学号(num)、姓名(name)和性别(gender)

//基本信息类中含有能够显示基本信息的成员函数

//创建一个学生类Student，继承基本信息类，新增成员年龄(age)、地址(address)

//学生类Student中包含显示学生全部信息的成员函数

//编写测试主程序，从键盘输入两个学生的信息，然后在终端显示

#include <iostream>

#include <cstring>

using namespace std;

//基本信息类

class BaseInfo

{

private:

    int num;

    string name;

    bool gender;

public:

    BaseInfo();

    BaseInfo(int n, string nm, bool sex);

    BaseInfo(const BaseInfo& obj);

    ~BaseInfo();

    int getnum() const;

    string getname() const;

    bool getgender() const;

    void show();

};

BaseInfo::BaseInfo()

{

}

BaseInfo::BaseInfo(int n, string nm, bool sex)

:num(n), name(nm), gender(sex)

{

}

BaseInfo::BaseInfo(const BaseInfo& obj)

:num(obj.num), name(obj.name), gender(obj.gender)

{

}

BaseInfo::~BaseInfo()

{

}

int BaseInfo::getnum() const

{

    return num;

}

string BaseInfo::getname() const

{

    return name;

}

bool BaseInfo::getgender() const

{

    return gender;

}

void BaseInfo::show()

{

    string s1 = "Male";

    string s2 = "Female";

    cout << "num: " << num << endl;

    cout<< "name: " << name << endl;

    if(gender)

        cout << "gender: " << s1 << endl;

    else

        cout << "gender: " << s2 << endl;

}

//学生信息类

class Student : public BaseInfo

{

private:

    int age;

    string address;

public:

    Student();

    Student(int n, string nm, bool sex, int g, string addr);

    Student(const Student& obj);

    ~Student();

    void show();

};

Student::Student()

{

}

Student::Student(int n, string nm, bool sex, int g, string addr)

:BaseInfo(n, nm, sex), age(g), address(addr)

{

}

Student::Student(const Student& obj)

:BaseInfo(obj.getnum(), obj.getname(), obj.getgender()), age(obj.age), address(obj.address)

{

}

Student::~Student()

{

}

void Student::show()

{

    BaseInfo::show();

    cout << "age: " << age << endl;

    cout << "address: " << address << endl;

}

int main()

{

    Student st1(10010, "Wang-li", 'f', 19, "115 Beijing Road,Shanghai");

    Student st2(10011, "Zhang-fan", 'm', 21, "213 Shanghai Road,Beijing");

    st1.show();

    cout<<endl;

    st2.show();

    Student st3(st1);

    cout<<endl;

    st3.show();

    return 0;

}

//2. 创建一个矩阵类

//重载运算符+，使之能完成矩阵加法运算

//重载运算符-，使之能完成矩阵减法运算

//给出完成矩阵转置运算的函数transport()

//重载运算符\*，使之能完成矩阵乘法运算

//重载运算符=，使之能完成矩阵赋值运算

#include <iostream>

#include <cassert>

using namespace std;

class Matrix

{

private:

    int m;

    int n;

    int\*\* pMat;

public:

    Matrix();

    Matrix(int m1, int n1, int\*\* p);

    Matrix(const Matrix& obj);

    ~Matrix();

    Matrix operator+(const Matrix& obj);

    Matrix operator-(const Matrix& obj);

    Matrix transport();

    Matrix operator\*(const Matrix& obj);

    Matrix operator=(const Matrix& obj);

    void show();

};

Matrix::Matrix()

:m(0), n(0), pMat(nullptr)

{

}

Matrix::Matrix(int m1, int n1, int\*\* p)

:m(m1), n(n1)

{

    int i, j;

    pMat = new int\*[m];

    for(i = 0; i < m; i++) {

        pMat[i] = new int[n];

        for (j = 0; j < n; j++)

            pMat[i][j] = p[i][j];

    }

}

Matrix::Matrix(const Matrix& obj)

:m(obj.m), n(obj.n)

{

    int i, j;

    pMat = new int\* [m];

    for (i = 0; i < m; i++) {

        pMat[i] = new int[n];

        for (j = 0; j < n; j++)

            pMat[i][j] = obj.pMat[i][j];

    }

}

Matrix::~Matrix()

{

    int i;

    if (pMat) {

        for (i = 0; i < m; i++)

        {

            if (pMat[i])

                delete[] pMat[i];

        }

        delete[] pMat;

    }

}

Matrix Matrix::operator+(const Matrix& obj)

{

    assert(m == obj.m && n == obj.n);

    int i, j;

    int\*\* p = new int\*[m];

    for (i = 0; i < m; i++)

    {

        p[i] = new int[n];

        for (j = 0; j < n; j++)

            p[i][j] = pMat[i][j] + obj.pMat[i][j];

    }

    return Matrix(m, n, p);

}

Matrix Matrix::operator-(const Matrix& obj)

{

    assert(m == obj.m && n == obj.n);

    int i, j;

    int\*\* p = new int\*[m];

    for (i = 0; i < m; i++)

    {

        p[i] = new int[n];

        for (j = 0; j < n; j++)

            p[i][j] = pMat[i][j] - obj.pMat[i][j];

    }

    return Matrix(m, n, p);

}

Matrix Matrix::transport()

{

    int i, j;

    int m1 = n;

    int n1 = m;

    int \*\*p = new int\*[m1];

    for (i = 0; i < m1; i++)

    {

        p[i] = new int[n1];

        for (j = 0; j < n1; j++)

            p[i][j] = pMat[j][i];

    }

    return Matrix(m1, n1, p);

}

Matrix Matrix::operator\*(const Matrix& obj)

{

    assert(n == obj.m);

    int i, j, k;

    int m1 = m;

    int n1 = obj.n;

    int\*\* p = new int\*[m1];

    for (i = 0; i < m1; i++)

    {

        p[i] = new int[n1];

        for (j = 0; j < n1; j++)

        {

            p[i][j] = 0;

            for(k=0; k<n1; k++)

            {

                p[i][j] =+ pMat[i][k]\*obj.pMat[k][j];

            }

        }

    }

    return Matrix(m1, n1, p);

}

Matrix Matrix::operator=(const Matrix& obj)

{

    int i, j;

    if (pMat) {

        for (i = 0; i < m; i++)

        {

            if (pMat[i])

                delete[] pMat[i];

        }

        delete[] pMat;

    }

    m = obj.m;

    n = obj.n;

    pMat = new int\*[m];

    for(i = 0; i < m; i++)

    {

        pMat[i] = new int[n];

        for (j = 0; j < n; j++)

            pMat[i][j] = obj.pMat[i][j];

    }

    return \*this;

}

void Matrix::show()

{

    int i, j;

    for (i = 0; i < m; i++)

    {

        for (j = 0; j < n; j++)

        {

            cout << pMat[i][j];

            if (j < n - 1)

                cout << ",";

        }

        cout << endl;

    }

}

int main()

{

    int i, j;

    const int M1 = 2;

    const int N1 = 3;

    int a1[M1][N1] = {

        {2,3,4},

        {5,6,7}

    };

    int\*\* p1 = new int\*[M1];

    for(i=0; i<M1; i++)

    {

        p1[i] = new int[N1];

        for(j=0; j<N1; j++)

        {

            p1[i][j] = a1[i][j];

        }

    }

    Matrix mat1(M1, N1, p1);

    mat1.show();

    cout<<endl;

    const int M2 = 3;

    const int N2 = 2;

    int a2[M2][N2] = {

        {1, 3},

        {5, 6},

        {7, 8}

    };

    int\*\* p2 = new int\*[M2];

    for(i=0; i<M2; i++)

    {

        p2[i] = new int[N2];

        for(j=0; j<N2; j++)

        {

            p2[i][j] = a2[i][j];

        }

    }

    Matrix mat2(M2, N2, p2);

    mat2.show();

    cout<<endl;

    Matrix mat3 = mat1\*mat2;

    mat3.show();

    cout<<endl;

    Matrix mat4 = mat1;

    mat4.show();

    cout<<endl;

    return 0;

}

/\*

11. 设计一个直线类Line，直线的起点为p0(x0, y0)，终点p1(x1, y1)。

(1) 给出类的完整定义；(25分)

(2) 求直线的长度；(25分)

(3) 判断两条直线是否相交？(25分)

(4) 求两条直线的交点坐标。(25分)

\*/

//Point1.h

#include<iostream>

using namespace std;

class Point1

{

public:

   Point1(int x, int y);

   Point1(const Point1& obj);

   void setX(int x);

   void setY(int y);

   int getX(void) const;

   int getY(void) const;

   void getShow() const;

private:

   int x;

   int y;

};

Point1::Point1(int x, int y)

:x(x), y(y)

{

}

Point1::Point1(const Point1& obj)

:x(obj.x), y(obj.y)

{

}

void Point1::setX(int x)

{

   this->x = x;

}

void Point1::setY(int y)

{

   this->y = y;

}

int Point1::getX(void) const

{

   return x;

}

int Point1::getY(void) const

{

   return y;

}

void Point1::getShow() const

{

   cout<<"("<<x<<", "<<y<<")"<<endl;

}

//Line1.h

#include "Point1.h"

#include <cmath>

class Line1 : public Point1

{

public:

   Line1(int x0, int y0, int x1, int y1);

   void setPts(int x1, int y1);//修改直线的起点坐标

   void setPte(int x1, int y1);//修改直线的终点坐标

   Point1 getPts(void) const;//返回直线的起点坐标

   Point1 getPte(void) const;//返回直线的终点坐标

   double getLength(void) const;//返回直线的长度

   bool isIntersect(const Line1& obj);//判断两直线是否存在交点

   Point1 getIntersect(const Line1& obj);//返回两直线的交点坐标

private:

   //继承的(x, y)作为线段的起点

   Point1 pte;//新增成员作为线段的终点

};

Line1::Line1(int x0, int y0, int x1, int y1)

:Point1(x0, y0), pte(x1, y1)

{

   if(y0>y1)//考虑到直线的正负划分性，将y坐标较小的端点作为直线的起点

   {

      setPts(x1, y1);

      setPte(x0, y0);

   }

}

//修改直线的起点坐标

void Line1::setPts(int x1, int y1)

{

   setX(x1);

   setY(y1);

}

//修改直线的终点坐标

void Line1::setPte(int x1, int y1)

{

   pte.setX(x1);

   pte.setY(y1);

}

//返回直线的起点坐标

Point1 Line1::getPts(void) const

{

   return Point1(getX(), getY());

}

//返回直线的终点坐标

Point1 Line1::getPte(void) const

{

   return pte;

}

//返回直线的长度

double Line1::getLength(void) const

{

   int x0, y0, x1, y1;

   x0 = getX();

   y0 = getY();

   x1 = pte.getX();

   y1 = pte.getY();

   return std::sqrt((x1-x0)\*(x1-x0) + (y1-y0)\*(y1-y0));

}

//判断两直线是否存在交点

bool Line1::isIntersect(const Line1& obj)

{

   //线段l1

   Point1 p0 = getPts();

   Point1 p1 = getPte();

   //线段l2

   Point1 p2 = obj.getPts();

   Point1 p3 = obj.getPte();

   int l1x0 = p0.getX();

   int l1y0 = p0.getY();

   int l1x1 = p1.getX();

   int l1y1 = p1.getY();

   int l2x0 = p2.getX();

   int l2y0 = p2.getY();

   int l2x1 = p3.getX();

   int l2y1 = p3.getY();

   int sx1 = l1x0>l1x1 ? l1x0: l1x1;//返回线段1的最大x坐标

   int sx2 = l2x0<l2x1 ? l2x0: l2x1;//返回线段2的最小x坐标

   int sy1 = l1y0>l1y1 ? l1y0: l1y1;//返回线段1的最大y坐标

   int sy2 = l2y0<l2y1 ? l2y0: l2y1;//返回线段2的最小y坐标

   int sx3 = l2x0>l2x1 ? l2x0: l2x1;//返回线段2的最大x坐标

   int sx4 = l1x0<l1x1 ? l1x0: l1x1;//返回线段1的最小x坐标

   int sy3 = l2y0>l2y1 ? l2y0: l2y1;//返回线段2的最大y坐标

   int sy4 = l1y0<l1y1 ? l1y0: l1y1;//返回线段1的最小y坐标

   //线段1的最大x坐标小于线段2的最小x坐标，两线段必分离，余同理

   if( sx1 < sx2 || sy1 < sy2 || sx3 < sx4 || sy3 < sy4)

      return false;

   int xi1 = ((l1x0-l2x0)\*(l2y1-l2y0) - (l1y0-l2y0)\*(l2x1-l2x0));

   int xi2 = ((l1x1-l2x0)\*(l2y1-l2y0) - (l1y1-l2y0)\*(l2x1-l2x0));

   int xi3 = ((l2x0-l1x0)\*(l1y1-l1y0) - (l2y0-l2y0)\*(l1x1-l1x0));

   int xi4 = ((l2x1-l1x0)\*(l1y1-l1y0) - (l2y1-l1y0)\*(l1x1-l1x0));

   if( (xi1 \* xi2)>0 || (xi3 \* xi4)>0 )

      return false;

   return true;

}

Point1 Line1::getIntersect(const Line1& obj)

{

   double k1, k2;

   double b1, b2;

   int x, y;

   int x0, y0, x1, y1;

   int x2, y2, x3, y3;

   //线段l1

   Point1 p0 = getPts();

   Point1 p1 = getPte();

   x0 = p0.getX();

   y0 = p0.getY();

   x1 = p1.getX();

   y1 = p1.getY();

   //线段l2

   Point1 p2 = obj.getPts();

   Point1 p3 = obj.getPte();

   x2 = p2.getX();

   y2 = p2.getY();

   x3 = p3.getX();

   y3 = p3.getY();

   if(x0!=x1 && x2!=x3)

   {

      k1 = static\_cast<double>((y1-y0)/(x1-x0));

      k2 = static\_cast<double>((y3-y2)/(x3-x2));

      b1 = static\_cast<double>(y0-k1\*x0);

      b2 = static\_cast<double>(y2-k2\*x2);

      x = static\_cast<int>((b1-b2)/(k2-k1));

      y = static\_cast<int>(k1\*x + b1);

      return Point1(x,y);

   }

   else if(x1 == x0)

   {

      k2 = static\_cast<double>(y3-y2)/(x3-x2);

      b2 = static\_cast<double>(y2-k2\*x2);

      x = x1;

      y = static\_cast<int>(k2\*x + b2);

      return Point1(x, y);

   }

   else{

      k1 = static\_cast<double>(y1-y0)/(x1-x0);

      b1 = static\_cast<double>(y0-k1\*x0);

      x = x2;

      y = static\_cast<int>(k1\*x + b1);

      return Point1(x,y);

   }

}

//main.cpp

#include"Line1.h"

int main()

{

    Line1 l1(0, 0, 6, 6);

    Line1 l2(0, 6, 6, 0);

    if(l1.isIntersect(l2))

    {

        Point1 p = l1.getIntersect(l2);

        cout<<"交点为";

        p.getShow();

    }

    else

        cout<<"No intersect!"<<endl;

    return 0;

}

/\*

12.创建一个矩形类；

从键盘输入两个矩形参数，创建两个矩形对象;

判断两个矩形是否相交？如果相交，返回Ture，否则返回False;

如果相交，求相交部分的面积。

\*/

//Rect.h

#include"Point1.h"

class Rect: public Point1

{

private:

    int w;

    int h;

public:

    Rect(int x, int y, int w, int h);

    int getW() const;

    int getH() const;

    void setW(int w);

    void setH(int h);

    bool isIntersect(const Rect& obj);

    int area() const;

};

Rect::Rect(int x, int y, int w, int h)

:Point1(x, y), w(w), h(h)

{

}

int Rect::getW() const

{

    return w;

}

int Rect::getH() const

{

    return h;

}

void Rect::setW(int w)

{

    this->w = w;

}

void Rect::setH(int h)

{

    this->h = h;

}

bool Rect::isIntersect(const Rect& obj)

{

    int x0, y0, x1, y1;

    x0 = getX();

    y0 = getY();

    x1 = obj.getX();

    y1 = obj.getY();

    if((x0+w)<x1 || (x1+obj.w)<x0 || (y0+h)<y1 || (y1+obj.h)<y0)

        return false;

    else

        return true;

}

int Rect::area() const

{

    return w\*h;

}

//main.cpp

#include"Rect.h"

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int main()

{

    Rect rect1(1, 1, 4, 4);

    Rect rect2(2, 2, 6, 6);

    if(rect1.isIntersect(rect2))

    {

        vector<int> a(4), b(4);

        a[0] = rect1.getX();

        a[1] = a[0] + rect1.getW();

        a[2] = rect2.getX();

        a[3] = a[2] + rect2.getW();

        b[0] = rect1.getY();

        b[1] = b[0] + rect1.getH();

        b[2] = rect2.getY();

        b[3] = b[2] + rect2.getH();

        sort(a.begin(), a.end());

        sort(b.begin(), b.end());

        Rect rect3(a[1], b[1], a[2]-a[1], b[2]-b[1]);

        cout<<"The area is "<<rect3.area()<<endl;

    }

    else

        cout<<"Rectangles are not intersect, the area is 0."<<endl;

    return 0;

}

/\*

13. 多项式代数表达式形如：f(x)=a0+a1\*x+a2\*x^2+a3\*x^3+....+an\*x^n

(1) 设计一个多项式类Polynomial (10分)

(2) 计算多项式的值 (10分)

(3) 显示多项式的系数向量 (10分)

(4) 重载加法运算符“+”, 可以实现两多项式的加法运算 (10分)

(5) 重载赋值运算符“=”, 可以实现多项式的赋值运算 (10分)

(6) 重载乘法运算符“\*”, 可以实现两多项式的乘法运算 (10分)

(7) 编写一个测试主程序:

       给定多项式列表f1, f2,输出其系数向量 (10分)

       测试加法运算f3=f1+f2, 输出其系数向量 (10分)

       测试乘法运算f4=f1\*f2, 输出其系数向量 (10分)

       给定x的值, 计算f1、f2、f3、f4的值 (10分)

测试用例:

    f1(x) = 1-3x+5x^2-7x^3

    f2(x) = 1-2x+4x^2

    x = 2.0

\*/

#include<iostream>

#include<cmath>

#include<vector>

#include<iterator>

using namespace std;

template<class T>

class Polynomial

{

private:

    vector<T> coef;

public:

    Polynomial();

    Polynomial(const vector<T>& coef);

    Polynomial(const Polynomial& obj);

    ~Polynomial();

    T value(T x) const;

    void show() const;

    Polynomial operator+(const Polynomial& obj);

    Polynomial operator-(const Polynomial& obj);

    Polynomial operator=(const Polynomial& obj);

    Polynomial operator\*(const Polynomial& obj);

};

template<class T>

Polynomial<T>::Polynomial()

{

}

template<class T>

Polynomial<T>::Polynomial(const vector<T>& coef)

:coef(coef)

{

}

template<class T>

Polynomial<T>::Polynomial(const Polynomial& obj)

:coef(obj.coef)

{

}

template<class T>

Polynomial<T>::~Polynomial()

{

}

template<class T>

T Polynomial<T>::value(T x) const

{

    T val = 0;

    int i;

    for(i=0; i<=coef.size(); i++)

    {

        val += coef[i] \* pow(x, i);

    }

    return val;

}

template<class T>

void Polynomial<T>::show() const

{

    cout<<"[";

    copy(coef.begin(), coef.end()-1, ostream\_iterator<T>(cout, ", "));

    cout<<\*(coef.end()-1);

    cout<<"]";

    cout<<endl;

}

template<class T>

Polynomial<T> Polynomial<T>::operator+(const Polynomial& obj)

{

    int len1 = coef.size();

    int len2 = obj.coef.size();

    int i;

    int m = len1>len2?len1:len2;

    vector<T> v;

    T val;

    for(i=0; i<m; i++)

    {

        val = 0;

        if(i<len1)

            val += coef[i];

        if(i<len2)

            val += obj.coef[i];

        v.push\_back(val);

    }

    return Polynomial(v);

}

template<class T>

Polynomial<T> Polynomial<T>::operator-(const Polynomial& obj)

{

    int len1 = coef.size();

    int len2 = obj.coef.size();

    int i;

    int m = len1>len2?len1:len2;

    vector<T> v;

    T val;

    for(i=0; i<m; i++)

    {

        val = 0;

        if(i<len1)

            val += coef[i];

        if(i<len2)

            val -= obj.coef[i];

        v.push\_back(val);

    }

    return Polynomial(v);

}

template<class T>

Polynomial<T> Polynomial<T>::operator=(const Polynomial& obj)

{

    coef = obj.coef;

    return \*this;

}

template<class T>

Polynomial<T> Polynomial<T>::operator\*(const Polynomial& obj)

{

    if(!coef.size() || !obj.coef.size())

    {

        coef.clear();

        return \*this;

    }

    vector<T> v;

    Polynomial f;

    int i, j, k;

    for(i=0; i<coef.size(); i++)

    {

        for(j=0; j<obj.coef.size(); j++)

        {

            v.clear();

            for(k=0; k<i+j+1; k++)

            {

                if(k<i+j)

                    v.push\_back(0);

                else

                    v.push\_back(coef[i]\*obj.coef[j]);

            }

            f = f + Polynomial(v);

        }

    }

    return f;

}

int main()

{

    vector<double> v1;

    v1.push\_back(1);

    v1.push\_back(-3);

    v1.push\_back(5);

    v1.push\_back(-7);

    vector<double> v2;

    v2.push\_back(1);

    v2.push\_back(-2);

    v2.push\_back(4);

    Polynomial<double> f1(v1), f2(v2);

    f1.show();

    f2.show();

    Polynomial<double> f3=f1\*f2;

    f3.show();

    cout<<f3.value(1)<<endl;

    return 0;

}

/\*

4. 建立一个学生类Student，学生信息包括学号(id)、姓名(name)、性别(gender)、分数(score)。

(1) 建立带参构造函数及复制构造函数，用于初始化学生对象；

(2) 建立一个返回学生所有信息的函数；

(3) 从键盘输入5个学生的信息，按照分数进行排序，输出每个学生的信息；

(4) 显示所有性别为男性的学生信息；

(5) 统计全班同学的平均成绩。

\*/

#include<iostream>

#include<cstring>

#include<vector>

#include<algorithm>

#include<iterator>

using namespace std;

class Student

{

private:

    int id;

    string name;

    bool gender;

    float score;

public:

    Student();

    Student(int d, string nm, bool g, float s);

    Student(const Student& obj);

    ~Student();

    int getId() const;

    string getName() const;

    bool getGender() const;

    float getScore() const;

    friend ostream& operator<<(ostream& os, const Student& obj);

};

Student::Student()

{

}

Student::Student(int d, string nm, bool g, float s)

:id(d), name(nm), gender(g), score(s)

{

}

Student::Student(const Student& obj)

:id(obj.id), name(obj.name), gender(obj.gender), score(obj.score)

{

}

Student::~Student()

{

}

int Student::getId() const

{

    return id;

}

string Student::getName() const

{

    return name;

}

bool Student::getGender() const

{

    return gender;

}

float Student::getScore() const

{

    return score;

}

ostream& operator<<(ostream& os, const Student& obj)

{

    os<<"Num: "<<obj.id<<", ";

    os<<"Name: "<<obj.name<<", ";

    if(obj.gender)

        os<<"Gender: male, ";

    else

        os<<"Gender: female, ";

    os<<"Score: "<<obj.score<<endl;

    return os;

}

bool compareScore(const Student& obj1, const Student& obj2)

{

    if(obj1.getScore()<obj2.getScore())

        return true;

    else

        return false;

}

int main()

{

    int n;

    int i;

    int d;

    string nm;

    char ch;

    bool flag;

    float sc;

    vector<Student> vs;

    cout<<"Please enter num of student: ";

    cin>>n;

    for(i=0; i<n; i++)

    {

        cout<<"Please enter the "<<i+1<<"th student id: ";

        cin>>d;

        cout<<"Please enter the "<<i+1<<"th student name: ";

        cin>>nm;

        cout<<"Please enter the "<<i+1<<"th student gender(m/f): ";

        cin>>ch;

        cout<<"Please enter the "<<i+1<<"th student score: ";

        cin>>sc;

        if(ch=='m')

            flag = true;

        else

            flag = false;

        vs.push\_back(Student(d, nm, flag, sc));

    }

    sort(vs.begin(), vs.end(), compareScore);

    copy(vs.begin(), vs.end(), ostream\_iterator<Student>(cout, " "));

    return 0;

}

/\*

14. 建立一个好友的通信录应用程序。

创建类AddressBook，包括编号num，姓名name，电话telecode，地址address，出生日期birthday

成员函数包括缺省构造函数、带参构造函数、复制构造函数、释放函数，

也包括可以返回各个属性信息的成员函数和修改各属性信息的函数，

还包括显示好友基本信息的成员函数。

在主程序中创建一个通讯录对象数组，好友人数由键盘输入，

每个好友的信息由键盘输入，不得少于5人。

将所有好友按年龄由大到小排序，然后输出。

\*/

#include<iostream>

#include<iterator>

#include<cstring>

#include<algorithm>

#include<vector>

using namespace std;

class Date

{

private:

    int year;

    int month;

    int day;

public:

    Date();

    Date(int y, int m, int d);

    Date(const Date& obj);

    ~Date();

    int getYear() const;

    int getMonth() const;

    int getDay() const;

    bool operator<(const Date& obj);

};

Date::Date()

{

}

Date::Date(int y, int m, int d)

:year(y), month(m), day(d)

{

}

Date::Date(const Date& obj)

:year(obj.year), month(obj.month), day(obj.day)

{

}

Date::~Date()

{

}

int Date::getYear() const

{

      return year;

}

int Date::getMonth() const

{

      return month;

}

int Date::getDay() const

{

      return day;

}

bool Date::operator<(const Date& obj)

{

      if(year<obj.year)

            return true;

      else if(year>obj.year)

            return false;

      else if(month<obj.month)

            return true;

      else if(month>obj.month)

            return false;

      else if(day<obj.day)

            return true;

      else

            return false;

}

class AddressBook: public Date

{

private:

      int id;

      string name;

      string telcode;

      string address;

public:

      AddressBook();

      AddressBook(int id, const string& name, int y, int m, int d, const string& tel, const string& addr);

      AddressBook(const AddressBook& obj);

      ~AddressBook();

      int getId() const;

      string getName() const;

      string getTelcode() const;

      string getAddress() const;

      friend ostream& operator<<(ostream& os, const AddressBook& obj);

};

AddressBook::AddressBook()

{

}

AddressBook::AddressBook(int id, const string& name, int y, int m, int d, const string& tel, const string& addr)

:Date(y,m,d), id(id), name(name), telcode(tel), address(addr)

{

}

AddressBook::AddressBook(const AddressBook& obj)

:Date(obj.getYear(), obj.getMonth(), obj.getDay()),

id(obj.id), name(obj.name), telcode(obj.telcode), address(obj.address)

{

}

AddressBook::~AddressBook()

{

}

int AddressBook::getId() const

{

      return id;

}

string AddressBook::getName() const

{

      return name;

}

string AddressBook::getTelcode() const

{

      return telcode;

}

string AddressBook::getAddress() const

{

      return address;

}

ostream& operator<<(ostream& os, const AddressBook& obj)

{

      os<<"Id: "<<obj.getId()<<", ";

      os<<"Name: "<<obj.getName()<<", ";

      os<<"Birthday: ";

      os<<obj.getYear()<<"."<<obj.getMonth()<<"."<<obj.getDay()<<", ";

      os<<"Telcode: "<<obj.getTelcode()<<", ";

      os<<"Address: "<<obj.getAddress()<<", "<<endl;

      return os;

}

int main()

{

      int i;

      int n;

      int id;

      string name;

      string tel;

      string addr;

      int y;

      int m;

      int d;

      cout<<"Please enter num of the friends: ";

      cin>>n;

      vector<AddressBook> abk;

      for(i=0; i<n; i++)

      {

            cout<<"Please enter the "<<i+1<<"th friend id: ";

            cin>>id;

            cout<<"Please enter the "<<i+1<<"th friend name: ";

            cin>>name;

            cout<<"Please enter the "<<i+1<<"th friend birthday(year month day):";

            cin>>y>>m>>d;

            cout<<"Please enter the "<<i+1<<"th friend telcode: ";

            cin>>tel;

            cout<<"Please enter the "<<i+1<<"th friend address: ";

            cin>>addr;

            abk.push\_back(AddressBook(id, name, y, m, d, tel, addr));

      }

      sort(abk.begin(), abk.end());

      copy(abk.begin(), abk.end(), ostream\_iterator<AddressBook>(cout, "\n"));

      return 0;

}

/\*

15. 创建一个设备管理应用程序。

(1). 创建一日期类Date

属性有年(year)、月(month)、日(day)

成员包括缺省构造函数、带参构造函数、复制构造函数、释放函数

成员包括可以返回年、月、日等的函数

成员包括可以比较两个时间先后顺序的函数，例如，如果时间t1早于时间t2，函数返回值为真

(2). 创建一个设备类Device

属性包括：编号(num)、名称(name)、单价(price)、数量(amount)、金额(money)

购买日期继承自日期类Date

成员包括缺省构造函数、带参构造函数、复制构造函数、释放函数，

成员包括获取设备各属性信息的函数

成员包括重载输出运算符，用以显示设备信息的函数。

(3). 编写测试主程序

输入5种设备信息，计算所有设备的总价

统计超过10万元以上的大型设备有多少件

将设备按照购买日期早晚的顺序排序后显示

测试用例:

-----------------------------------------------------------------------------

|   num     |   name        |   data    |   price   |   amount  |   money   |

-----------------------------------------------------------------------------

|   1       |   computer    |2015.10.15 |   4500    |   50      |   225000  |

-----------------------------------------------------------------------------

|   2       |   printer     |2018.02.05 |   1200    |   2       |   2400    |

-----------------------------------------------------------------------------

|   3       |   sever       |2021.09.10 |   120000  |   2       |   240000  |

-----------------------------------------------------------------------------

|   4       |   switch      |2022.05.01 |   20000   |   2       |   40000   |

-----------------------------------------------------------------------------

|   5       |   workstation |2019.07.05 |   15000   |   20      |   300000  |

-----------------------------------------------------------------------------

(4). 创建一个实验室管理应用程序

创建一个实验室类Laboratory

属性包含实验室名称(labname)、房间号(roomid)、实验员(name)以及若干设备资产

自行设计实验室类的成员函数

将上表中的计算机、服务器和工作站归类到计算实验室(cmptLab)中

将打印机和交换机归类到应用实验室(appLab)中

分别统计出两个实验室的资产总值。

\*/

#include<iostream>

#include<iterator>

#include<cstring>

#include<algorithm>

#include<vector>

using namespace std;

class Date

{

private:

    int year;

    int month;

    int day;

public:

    Date();

    Date(int y, int m, int d);

    Date(const Date& obj);

    ~Date();

    int getYear() const;

    int getMonth() const;

    int getDay() const;

    bool operator<(const Date& obj);

};

Date::Date()

{

}

Date::Date(int y, int m, int d)

:year(y), month(m), day(d)

{

}

Date::Date(const Date& obj)

:year(obj.year), month(obj.month), day(obj.day)

{

}

Date::~Date()

{

}

int Date::getYear() const

{

    return year;

}

int Date::getMonth() const

{

    return month;

}

int Date::getDay() const

{

    return day;

}

bool Date::operator<(const Date& obj)

{

    if(year<obj.year)

        return true;

    else if(year>obj.year)

        return false;

    else if(month<obj.month)

        return true;

    else if(month>obj.month)

        return false;

    else if(day<obj.day)

        return true;

    else

        return false;

}

class Device: public Date

{

private:

    int num;

    string name;

    float price;

    float amount;

    float money;

public:

    Device();

    Device(int num, const string& name, int y, int m, int d, float price, float amount, float money);

    Device(const Device& obj);

    ~Device();

    int getNum() const;

    string getName() const;

    float getPrice() const;

    float getAmount() const;

    float getMoney() const;

    friend ostream& operator<<(ostream& os, const Device& dev);

};

Device::Device()

{

}

Device::Device(int num, const string& name, int y, int m, int d, float price, float amount, float money)

:Date(y,m,d), num(num), name(name), price(price), amount(amount), money(money)

{

}

Device::Device(const Device& obj)

:Date(obj.getYear(), obj.getMonth(), obj.getDay()),

num(obj.num), name(obj.name), price(obj.price), amount(obj.amount), money(obj.money)

{

}

Device::~Device()

{

}

int Device::getNum() const

{

    return num;

}

string Device::getName() const

{

    return name;

}

float Device::getPrice() const

{

    return price;

}

float Device::getAmount() const

{

    return amount;

}

float Device::getMoney() const

{

    return money;

}

ostream& operator<<(ostream& os, const Device& dev)

{

    os<<"Num: "<<dev.num<<", ";

    os<<"Name: "<<dev.name<<", ";

    os<<"Purchase date: "<<dev.getYear()<<"."<<dev.getMonth()<<"."<<dev.getDay()<<", ";

    os<<"Price: "<<dev.price<<", ";

    os<<"Amount: "<<dev.amount<<", ";

    os<<"Money: "<<dev.money;

    return os;

}

class Laboratory

{

private:

    string labname;

    int roomid;

    string name;

    vector<Device> devs;

public:

    Laboratory();

    Laboratory(const string& lnm, int rid, const string& nm, vector<Device> devs);

    Laboratory(const Laboratory& obj);

    ~Laboratory();

    string getLname() const;

    int getRoomid() const;

    string getName() const;

    vector<Device> getDevs() const;

};

Laboratory::Laboratory()

{

}

Laboratory::Laboratory(const string& lnm, int rid, const string& nm, vector<Device> devs)

:labname(lnm), roomid(rid), name(nm), devs(devs)

{

}

Laboratory::Laboratory(const Laboratory& obj)

:labname(obj.labname), roomid(obj.roomid), name(obj.name), devs(obj.devs)

{

}

Laboratory::~Laboratory()

{

    devs.clear();

}

string Laboratory::getLname() const

{

    return labname;

}

int Laboratory::getRoomid() const

{

    return roomid;

}

string Laboratory::getName() const

{

    return name;

}

vector<Device> Laboratory::getDevs() const

{

    return devs;

}

int main()

{

    vector<Device> vd;

    vd.push\_back(Device(1, "computer",    2015, 10, 15, 4500  , 50, 225000));

    vd.push\_back(Device(2, "printer",     2018, 2,  5,  1200  , 2,  2400  ));

    vd.push\_back(Device(3, "sever",       2021, 9,  10, 120000, 2,  240000));

    vd.push\_back(Device(4, "switch",      2022, 5,  1,  20000 , 2,  40000 ));

    vd.push\_back(Device(5, "workstation", 2019, 7,  5,  15000 , 20, 300000));

    vector<Device> cmptLab, appLab;

    cmptLab.push\_back(vd[0]);

    cmptLab.push\_back(vd[2]);

    cmptLab.push\_back(vd[4]);

    appLab.push\_back(vd[1]);

    appLab.push\_back(vd[3]);

    int i;

    float sum;

    sum = 0;

    for(i=0; i<vd.size(); i++)

        sum += vd[i].getMoney();

    cout<<"Total value is "<<sum<<endl;

    int k = 0;

    for(i=0; i<vd.size(); i++)

    {

        if(vd[i].getPrice()>100000)

            k++;

    }

    cout<<"There are "<<k<<" device worth more than 100000 yuan."<<endl;

    sort(vd.begin(), vd.end());

    copy(vd.begin(), vd.end(), ostream\_iterator<Device>(cout, "\n"));

    sum = 0;

    for(i=0; i<cmptLab.size(); i++)

        sum += cmptLab[i].getMoney();

    cout<<"Total value of computer laboratory is "<<sum<<endl;

    sum = 0;

    for(i=0; i<appLab.size(); i++)

        sum += appLab[i].getMoney();

    cout<<"Total value of application laboratory is "<<sum<<endl;

    return 0;

}