#include <iostream>

#include <iomanip>

#include <math.h>

using namespace std;

//抽象基类Shape

class Shape {

public:

virtual void show() = 0;

virtual string getType() = 0;

virtual double distanceTo(Shape\*) = 0;

};

//用于点及相关功能的Point派生类

class Point :public Shape {

protected:

double x;

double y;

public:

friend class Line;//友元类

Point(double x, double y):x(x),y(y){}//点的构造函数

void show() {

cout << fixed << setprecision(3);//精确到小数点后三位

cout << "(" << x << "," << y << ")";

}

string getType() {

return "Point";//返回图形的类型

}

double getX() {

return x;//返回横坐标

}

double getY() {

return y;//返回纵坐标

}

double distanceTo(Shape\* another);//由于Line派生类尚未定义，在此暂且只做声明，不去具体给出实现

};

//用于线及相关功能的Line派生类

class Line :public Shape {

protected:

double a;

double b;

double c;

public:

friend class Point;//友元类

Line(double a0,double b0,double c0){//第一种构造函数

if(a0==0&&b0==0){

cout<<"Invalid Line. Reset: a=1, b=1"<<endl;

a=1; b=1; c=c0;//若a、b同时为0，则报错，并将他们都重设为1

}

else{a=a0; b=b0; c=c0;}

}

Line(Point p, double k) {//第二种构造函数

a = k;

b = -1;

c = p.getY() - k \* p.getX();

}

Line(Point p1, Point p2) {//第三种构造函数

if(p1.getX()==p2.getX()&&p1.getY()==p2.getY()){

cout<<"Invalid Line. Reset: P2("<<p2.getX()-1<<","<<p2.getY()-1<<")"<<endl;

a=1; b=-1;

c=p1.getY()-p1.getX();//若两点重合，则报错，将直线重设为经过P1，斜率为1的直线

}

else{

a = (p1.getY() - p2.getY()) / (p1.getX() - p2.getX());

b = -1;

c = p1.getY() - (p1.getY() - p2.getY()) / (p1.getX() - p2.getX()) \* p1.getX();

}

}

void show() {

cout << fixed << setprecision(3);//精确到小数点后三位

cout << a << "x" ;

if(b>=0){cout<<"+";}//避免+和-一起出现

cout<<b<<"y";

if(c>=0){cout<<"+";}

cout<<c<<"=0";

}

string getType() {

return "Line";//返回图形的类型

}

double getA(){

return a;//返回参数a

}

double getB(){

return b;//返回参数b

}

double getC(){

return c;//返回参数c

}

double distanceTo(Shape\* another){

if(another->getType()=="Point"){//若another的getType函数返回Point

Point\* p=dynamic\_cast<Point\*>(another);//将another转化为Point类

return fabs(a\*p->getX()+b\*p->getY()+c)/sqrt(a\*a+b\*b);

}

if(another->getType()=="Line"){//若another的getType函数返回Line

Line\* l2=dynamic\_cast<Line\*>(another);//将another转化为Line类

if(a\*l2->b==b\*l2->a){//如果平行，则按公式算出距离

return fabs(c-l2->c\*a/l2->a)/sqrt(a\*a+b\*b);

}

else return 0;//若相交，则返回0

}

else return -1;

}

Point getCross(Line l2) {

if(a\*l2.b==b\*l2.a){//若平行，则报错，设置交点cross为坐标原点

cout<<"There is no cross. Reset Cross:(0,0)"<<endl;

return Point(0,0);

}

else{//若相交，则按公式算出交点

double x = (b \* l2.c - c \* l2.b) / (a \* l2.b - b \* l2.a);

double y = (a \* l2.c - c \* l2.a) / (b \* l2.a - a \* l2.b);

return Point(x, y);

}

}

};

class GeometrySystem{

private:

Shape\*\* arr;//数组arr用于储存指向Shape类的指针

int n;//最大容量（为了简便，直接设定了最大容量）

int current;//目前一共装了多少个指针

public:

GeometrySystem(int n){

arr=new Shape\*[n];

for(int i=0;i<n;i++){

arr[i]=nullptr;

}

current=0;

}//构造函数设置最大容量，将所有指针初始化为空指针，

void add(Shape\* s){

if(current<n){//若还没有装满

arr[current]=s;

current++;//则可以添加，并将current值加一

}

else if(current>=n){//若已经装满，则报错

cout<<"The array is full."<<endl;

}

}

void showN(int i){//展示第i个数据

arr[i]->show();

}

void deleteN(int n){

if(arr[n]==nullptr){//如果要删除的是空指针，则报错

cout<<"Invalid action."<<endl;

}

else{//如果要删除的不是空指针

arr[n]=nullptr;//将要删除的指向空指针

for(int i=n;i<current;i++){

arr[i]=arr[i+1];//后面的每一个都向前面移动一位

}

current--;//目前数据个数减一

}

}

void showArr(){

cout<<"The whole array is :"<<endl;

for(int i=0;i<current;i++){

if(arr[i]!=nullptr){

cout<<"Number "<<i+1<<": ";

arr[i]->show();

cout<<endl;//展示arr内的每一个数据

}

}

}

};

//已经定义过Point和Line派生类了，可以在这里给出具体实现了，原理与Line的diatanceTo完全一致

double Point::distanceTo(Shape\* another){

if(another->getType()=="Point"){

Point\* p2=dynamic\_cast<Point\*>(another);

return sqrt((p2->x-x)\*(p2->x-x)+(p2->y-y)\*(p2->y-y));

}

if(another->getType()=="Line"){

Line\* l=dynamic\_cast<Line\*>(another);

return fabs(x\*l->getA()+y\*l->getB()+l->getC())/sqrt(l->getA()\*l->getA()+l->getB()\*l->getB());

}

else return -1;

}

int main(){

Point p1(3,5),p2(2,0),p3(-1,-5);//定义三个点

Line l1(2,-1,4),l2(p1,-1),l3(p2,p3);//定义三条线，分别用三种不同的构造函数

GeometrySystem geo(10);//设置容量为10的数组geo

geo.add(&p1); geo.add(&p2); geo.add(&p3);

geo.add(&l1); geo.add(&l2); geo.add(&l3);//将三个点和三条线添加到geo中

cout<<"Error test:"<<endl;//进行报错测试

Line lineError1(0,0,3);//a和b均为0，会重设a和b都为1

cout<<"Line error1: "; lineError1.show(); cout<<endl;

Line lineError3(p2,p2);//两个点相同，重设为经过这个点斜率为1的直线

cout<<"Line error3: "; lineError3.show(); cout<<endl;

Line lineParallel(2,-1,5);//定义一条l1的平行线

Point cross=l1.getCross(lineParallel);//由于两直线没有交点，此处会将交点设置为（0，0）

cout<<"The cross is "; cross.show(); cout<<endl;

cout<<"The distance between Line1 and Line3 is "<<l1.distanceTo(&l3)<<"."<<endl;//相交直线距离为0

geo.deleteN(8);//删除空指针会报错

cout<<endl<<"Function test:"<<endl;//进行功能测试

cout<<p1.getType(); p1.show(); cout<<endl;

cout<<l1.getType(); l1.show(); cout<<endl;//测试点和线的getType和show函数

cout<<"The distance between Point1 and Point2 is "<<p1.distanceTo(&p2)<<endl;//测试点点距

cout<<"The distance between Point1 and Line1 is "<<p1.distanceTo(&l1)<<endl;//测试点线距

cout<<"The distance between Line1 and lineParallel is "<<l1.distanceTo(&lineParallel)<<endl;//测试两平行直线距离

//两相交直线距离为0，已经在报错测试中测试过

geo.add(&lineParallel);//测试GeometrySystem的add函数

cout<<"Number 2: "; geo.showN(2); cout<<endl;

geo.deleteN(2);//测试删除函数

geo.showArr();//测试输出整个数组的函数

return 0;

}