面向对象方法与C++程序设计

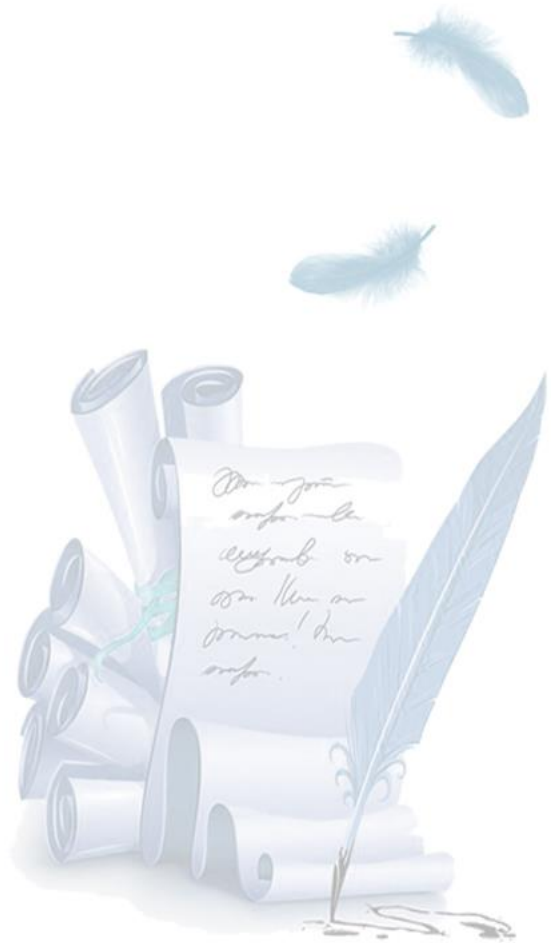
第2章　　类与对象

大连理工大学
主讲人-赵小薇

理解友元



类的私有成员只能在类的定义范围内使用，也就是说，类的私有成员只能通过本类的成员函数来访问，但有时候又需要在类的外部访问类的私有成员，甚至需要同时访问多个类的私有成员。为此引入友元。



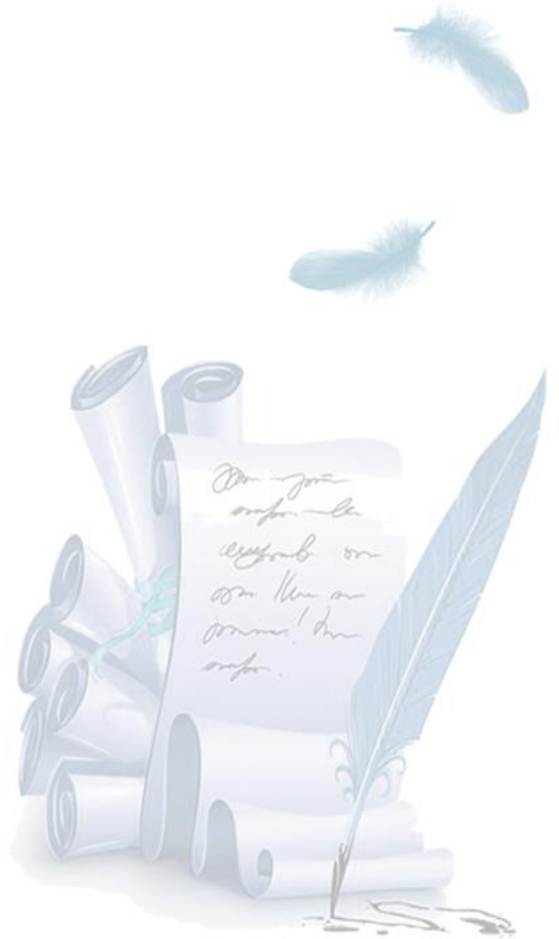
友元函数



- 友元函数是在类声明中用关键字friend说明的非成员函数。
- 友元函数不是当前类的成员函数，而是独立于当前类的外部函数，可以访问该类的所有对象的私有或公有成员；
- 友元函数声明可以放在私有部分，也可放在公有部分（因为它不是成员，不受访问控制权限的约束）。
- 友元函数不能通过this指针引用对象的成员，因为友元函数非对象成员。

声明为友元函数的一般形式为：

friend<数据类型><友元函数名>(参数表);

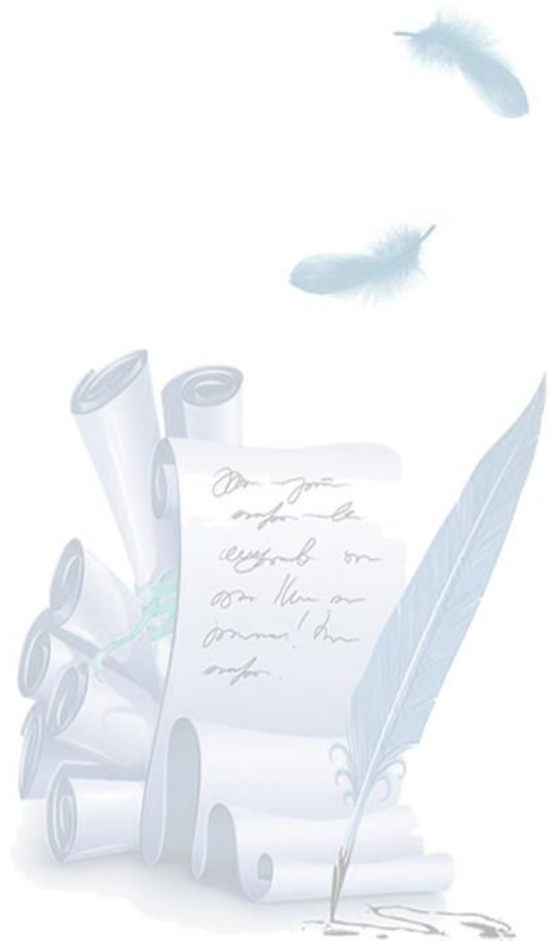


举例1

普通

```
int main(){
    Point p1(1.0,2.0), p2(3.0,4.0);
    p1.disp();          p2.disp();
   cout<<"距离为: "<<distance(p1,p2);
    return 0; }
```

```
class Point
{
private:
   double x, y;                  //x,y 坐标
public:
   Point(double x=0.0, double y=0.0);
   void disp( );
   friend double distance(Point p1,Point p2); //声明为Point的友元
};
Point::Point (double x, double y){  this->x = x;  this->y = y; }
void  Point::disp( ){  cout<<"点（"<<x<<","<<y<<"）";  }
double distance(Point p1,Point p2){
   return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
```

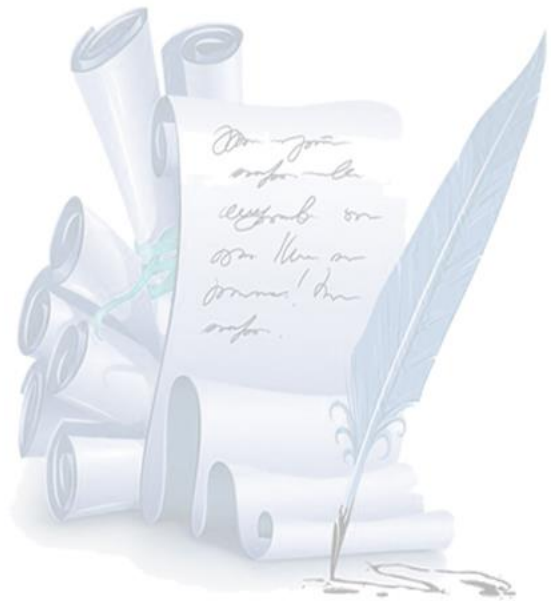


举例2

普通函数作为多个类的友元函数



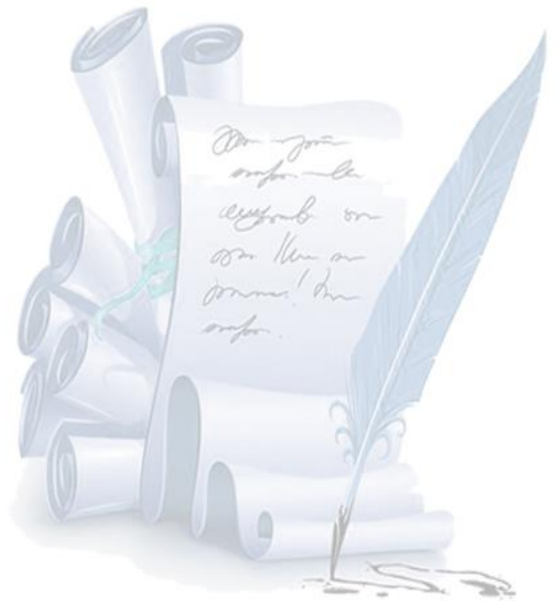
```
class Point
{
 private:
   double x, y;                    //x,y 坐标
 public:
   Point(double x=0.0, double y=0.0);
   void disp( );
   friend double distance(Point p,Line l); //声明为Point的友元
};
Point::Point (double x, double y){
          this->x = x;  this->y = y;
}
void  Point::disp( ){  cout<<"点（"<<x<<","<<y<<"）";  }
```



普通函数作为多个类的友元函数



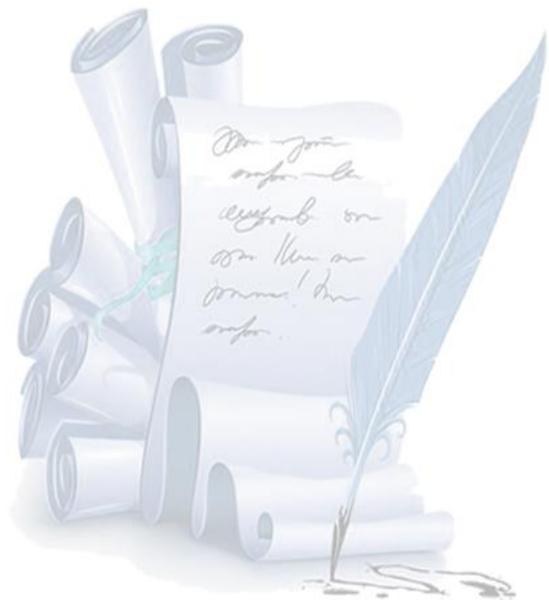
```
class Line
{
 private:
   double a, b, c;              //直线为ax+by+c=0的系数
 public:
   Line(double a=0.0, double b=0.0, double c=0.0);
   void disp( );
   friend double distance(Point p,Line l); //声明为Line的友元
};
Line::Line(double a, double b, double c){
          this->a = a;  this->b = b;  this->c = c;
}
void  Line::disp( ){
  cout<<"线（"<<a<<"x+"<<b<<"y+"<<c<<"=0）";}
```



普通函数作为多个类的友元函数

```
double distance(Point p,Line l){
   return fabs(l.a*p.x+l.b*p.y+l.c)/sqrt(l.a*l.a+l.b*l.b);
}
int main(){
    Point p1(1.0,2.0);
    Line l1(3.0,4.0,5.0);
    p1.disp();
    l1.disp();
    cout<<"距离为："<<distance(p1,l1)<<endl;
    return 0;
}
```

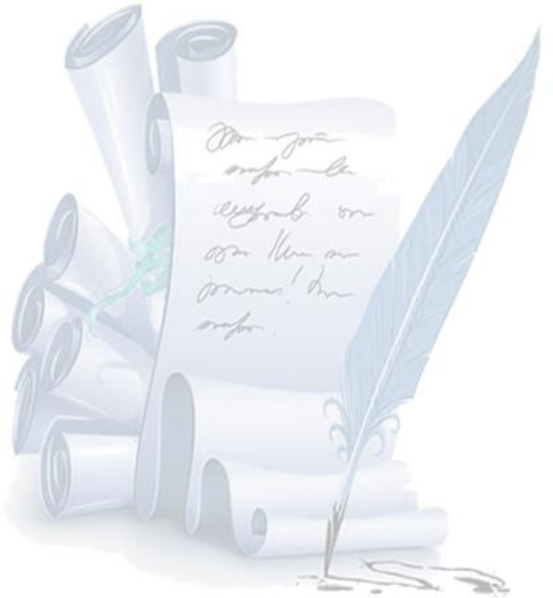
两个类的互相不可见的私有数据在distance函数都可以访问



友元函数



- 如果一个类的成员函数是另一个类的友元函数，则称这个成员函数为友元成员函数；
- 通过友元成员函数，不仅可以访问自己所在类对象中的私有和公有成员，还可访问由关键字friend声明语句所在的类对象中的私有和公有成员。

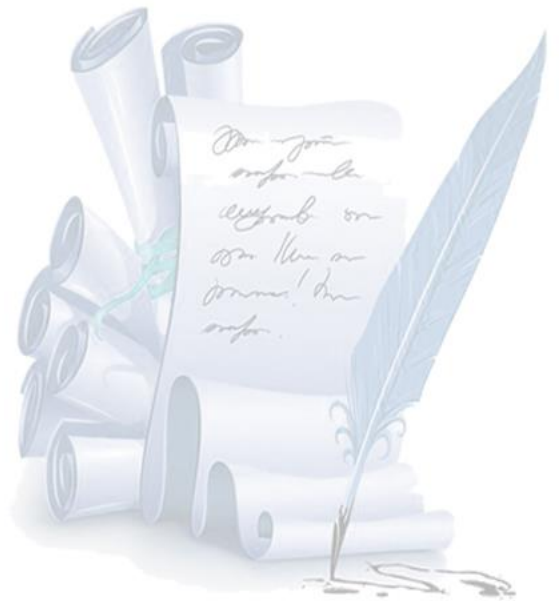


举例3

成员函数作为类的友元函数



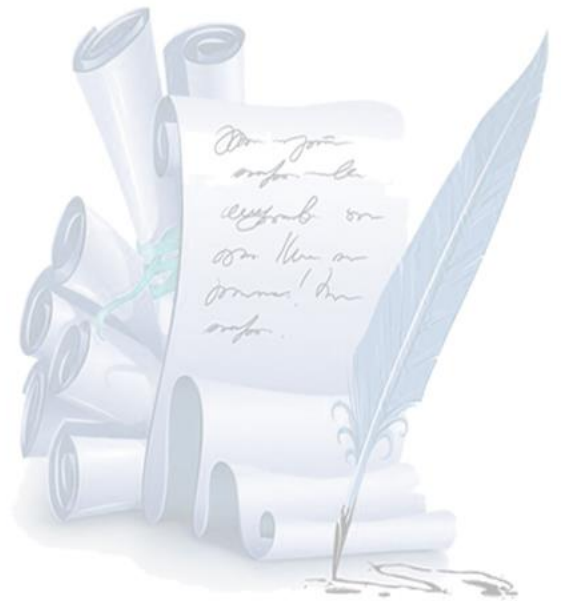
```
class Point
{
 private:
   double x, y;                    //x,y 坐标
 public:
   Point(double x=0.0, double y=0.0);
   void disp( );
   double distance(Line l);  //声明为Point的成员
};
Point::Point (double x, double y){
          this->x = x;  this->y = y;
}
void  Point::disp( ){  cout<<"点（"<<x<<","<<y<<"）"; }
```



成员函数作为类的友元函数



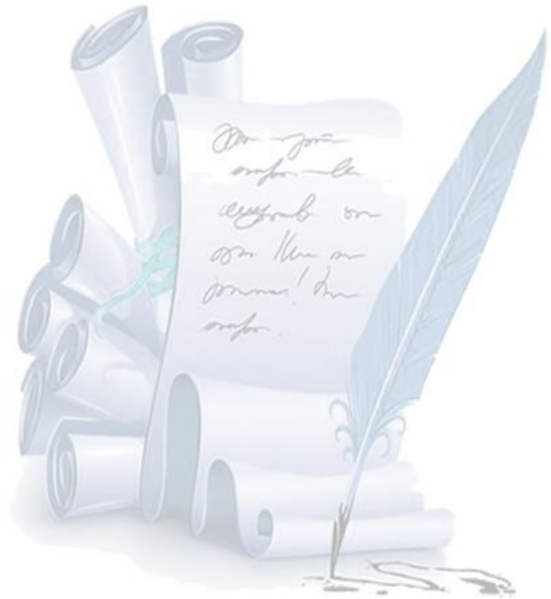
```
class Line
{
 private:
   double a, b, c;              //直线为ax+by+c=0的系数
 public:
   Line(double a=0.0, double b=0.0, double c=0.0);
   void disp( );
   friend double Point::distance(Line l); //声明为Line的友元
};
  Line::Line(double a, double b, double c){
          this->a = a;  this->b = b;  this->c = c;    }
void  Line::disp( ){
    cout<<“线（"<<a<<"x+"<<b<<"y+"<<c<<"=0）";}
```





成员函数作为类的友元函数

```
double Point::distance(Line l){
   return fabs(l.a*x+l.b*y+l.c)/sqrt(l.a*l.a+l.b*l.b);
}
int main(){
    Point p1(1.0,2.0);
    Line l1(3.0,4.0,5.0);
    p1.disp();
    l1.disp();
    cout<<"距离为："<<p1.distance(l1)<<endl;
    return 0;
}
```



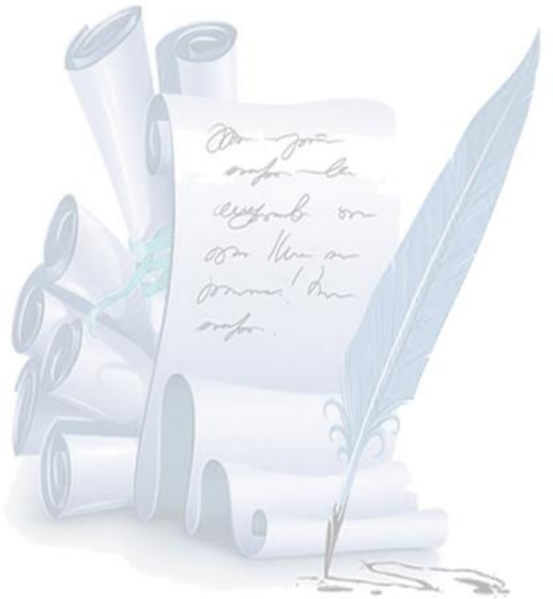
友元类



- 当一个类作为另一个类的友元时，称这个类为友元类。
- 当一个类成为另一个类的友元类时，这个类的所有成员函数都成为另一个类的友元函数
- 友元类的声明可以放在类声明中的任何位置。
- 友元关系是不能传递的。类B是类A的友元，类C是类B的友元，并不表示类C是类A的友元。
- 友元关系是单向的。类A是类B的友元，类A的成员函数可以访问类B的私有成员和保护成员，反之，类B不是类A的友元，类B的成员函数却不可以访问类A的私有成员和保护成员。

友元类声明的形式如下：

firiend class <友元类名>;

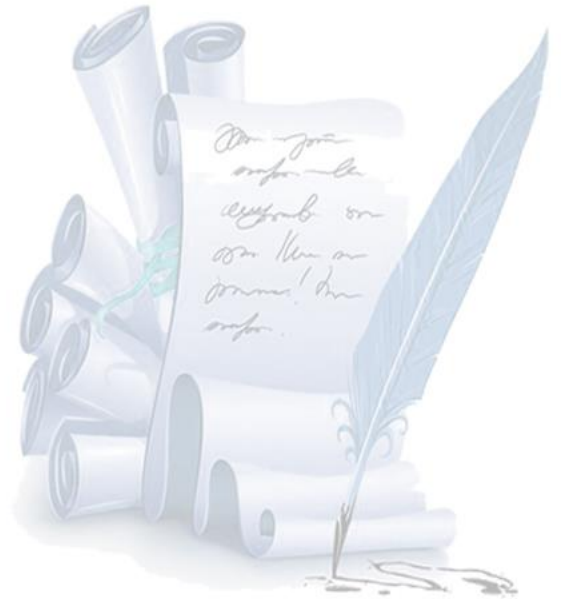


举例4

友元类



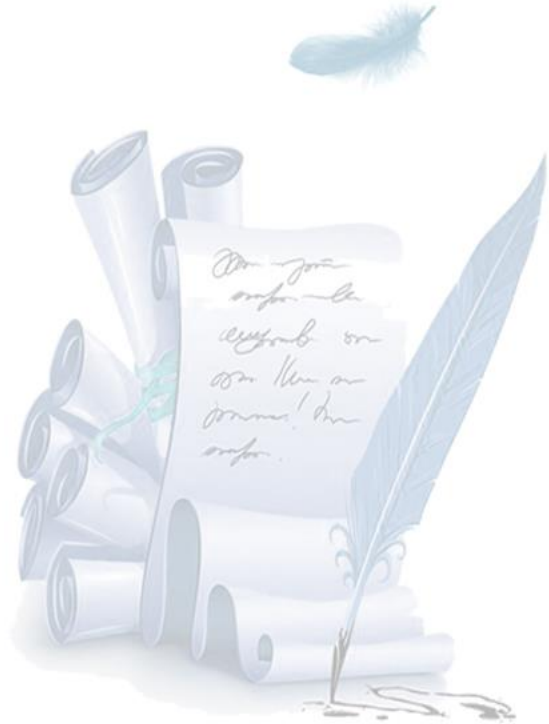
```
class Point
{
 private:
   double x, y;                    //x,y 坐标
 public:
   Point(double x=0.0, double y=0.0);
   void disp( );
   friend class ComputeTools; //声明为Point的友元类
};
Point::Point (double x, double y){
          this->x = x;  this->y = y;
}
void  Point::disp( ){  cout<<"点（"<<x<<","<<y<<"）"; }
```



友元类

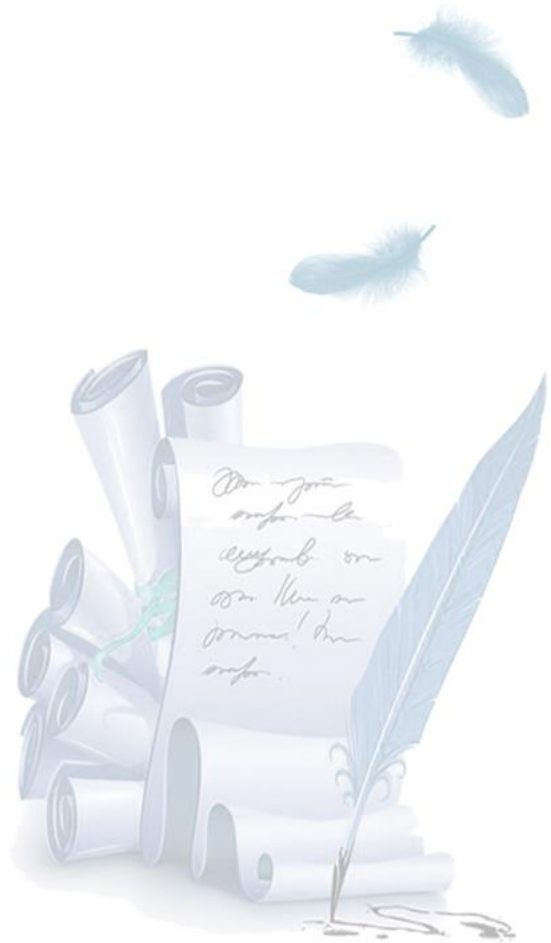


```
class Line
{
 private:
   double a, b, c;              //直线为ax+by+c=0的系数
 public:
   Line(double a=0.0, double b=0.0, double c=0.0);
   void disp( );                    // 输出私有变量的成员函数
   friend class ComputeTools; //声明为Line的友元类
};
Line::Line(double a, double b, double c){
          this->a = a;  this->b = b;  this->c = c;  }
void  Line::disp( ){
cout<<"线（"<<a<<"x+"<<b<<"y+"<<c<<"=0）";}
```



友元类

```
class ComputeTools{
public:
    static double distance(Point p1,Point p2); //重载点与点距离
    static double distance(Point p,Line l); //重载点与直线距离
};
double ComputeTools::distance(Point p1,Point p2){
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}
double ComputeTools::distance(Point p,Line l){
   return fabs(l.a*p.x+l.b*p.y+l.c)/sqrt(l.a*l.a+l.b*l.b);
 }
```



友元类

```
int main(){
    Point p1(1.0,2.0),p2(3.0,4.0);
    Line l1(3.0,4.0,5.0);
    p1.disp();  p2.disp();
    cout<<"距离为："<<ComputeTools::distance(p1,p2)<<endl;
    p1.disp();  l1.disp();
    cout<<"距离为："<<ComputeTools::distance(p1,l1)<<endl;
    return 0;
}
```