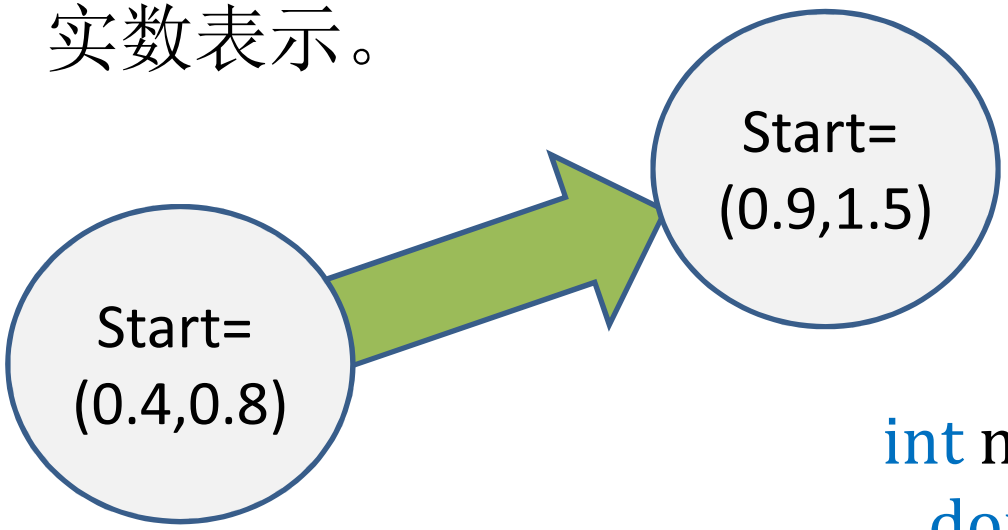
用户定义类型
User-defined Datatypes

classes and structs

几何向量(Geometry Vector)

- 二维平面上的向量由起点和终点构成。
- 每个点包含两个坐标(x, y),因此一个向量需要四个实数表示。




```
int main() {
   double xStart = 0.4;
   double xEnd = 0.8;
   double yStart = 0.9;
   double yEnd = 1.5;
}
```

几何向量(Geometry Vector)

- 传递一个向量给函数，需要传递4个double

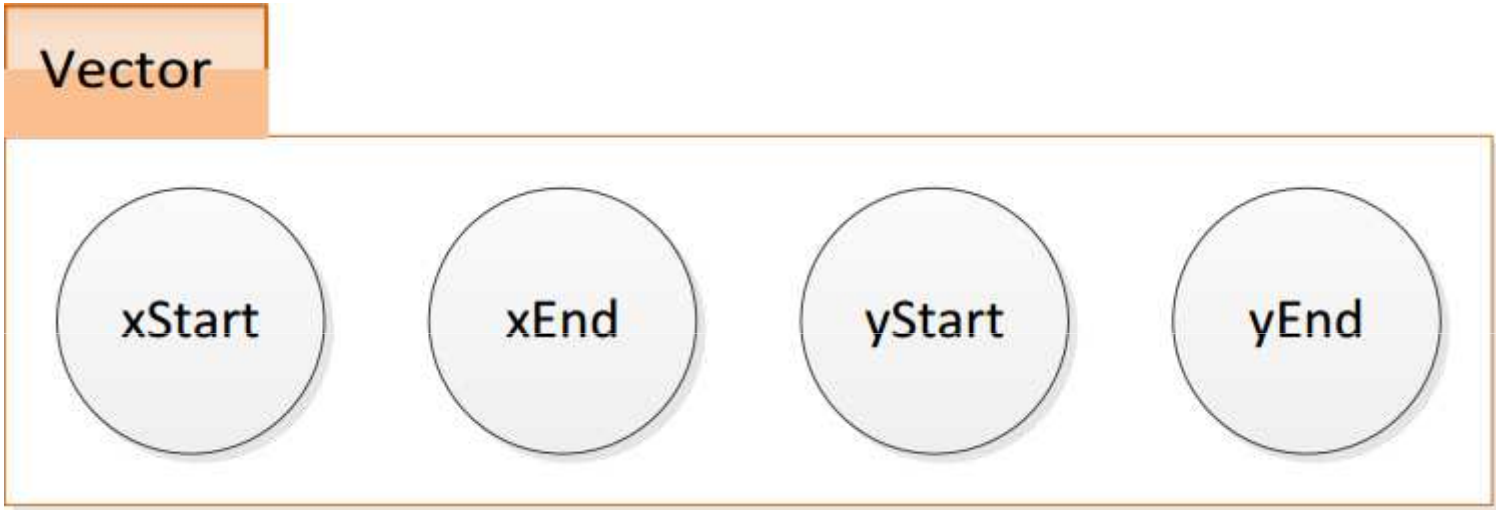
```
void printVector(double x0,double y0,double x1,double y1){
      cout << "(" << x0 << "," << y0 << ") -> ("
            << x1 << "," << y1 << ")" << endl;
}

int main( ) {
  double xStart = 0.4;
  double xEnd = 0.8;
  double yStart = 0.9;
  double yEnd = 1.5;
  printVector(xStart, yStart, xEnd, yEnd);
}
```

```
void offsetVector (double &x0, double &y0, double &x1,
                   double &y1,double offsetX, double offsetY) {
   x0 += offsetX;     y0 += offsetY;
   x1 += offsetX;      y1 += offsetY;
}
void printVector(double x0,double y0,double x1,double y1){
      cout << "(" << x0 << "," << y0 << ") -> ("
           << x1 << "," << y1 << ")" << endl;
}
int main( ) {
  double xStart = 0.4;
  double xEnd = 0.8;
  double yStart = 0.9;
  double yEnd = 1.5;
  offsetVector(xStart, yStart, xEnd, yEnd,1.2, 2.3);
  printVector(xStart, yStart, xEnd, yEnd);
}
```

类class

- 用户定义类型用以将相关的信息组装在一起。



类class的定义语法

类名

```
class Vector {
public:
   double xStart ;
   double xEnd ;
   double yStart ;
   double yEnd ;
};
```

C++11标准

```
class Vector {
public:
   double xStart = 0.4;
   double xEnd = 0.8;
   double yStart = 0.9;
   double yEnd = 1.5;
};
```

- 表示定义一个叫做Vector的类，即 Vector是用户定义的一种新类型。

类class的定义语法

```
class Vector {
public:
  double xStart;
  double xEnd;
  double yStart;
  double yEnd;
};
```

- 表示定义一个叫做Vector的类，即 Vector是用户定义的一种新类型。

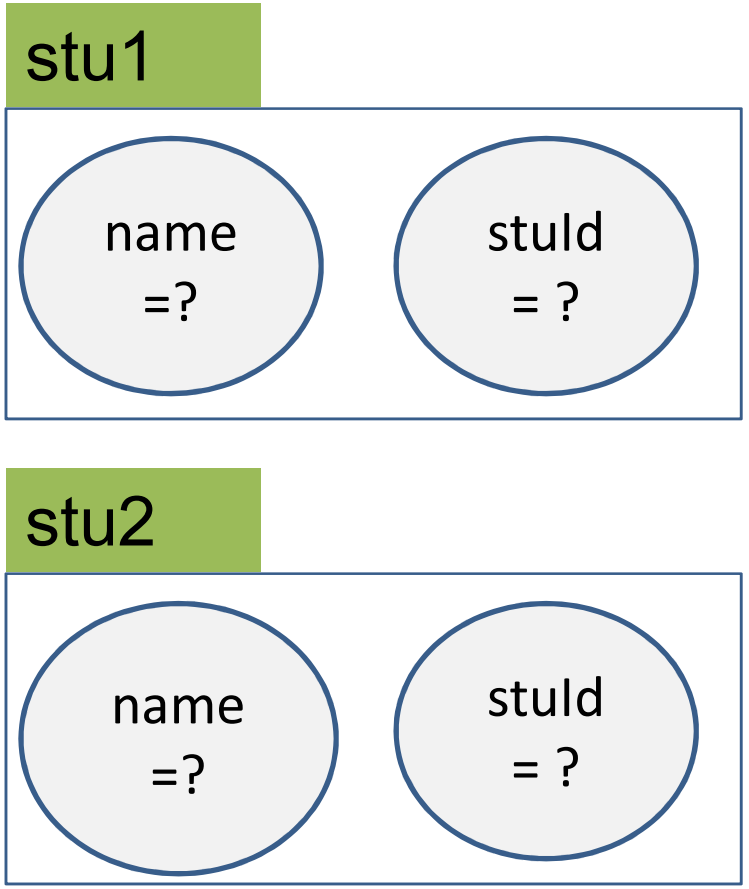
Student

```
class Student {
public:
  char *name ;
  int stuId;
};
```

- 类类型Student具有2个不同类型的数据成员。

类的实例-具体的类变量

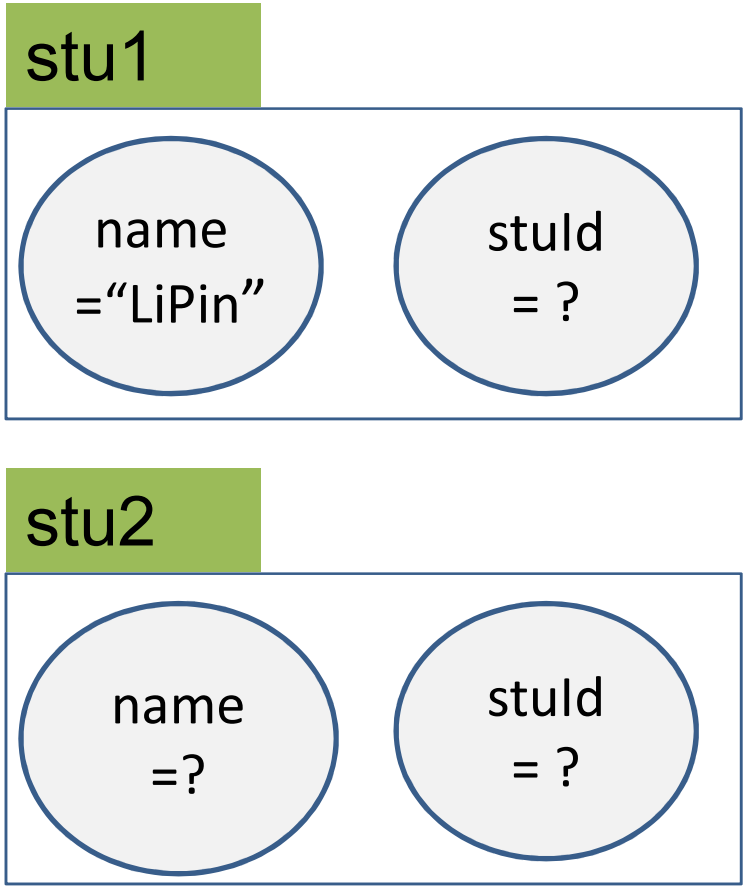
```
class Student {
public:
  char *name ;
  int stuId;
};
void main( ){
  Student stu1, stu2;
}
```



- 某种类类型的一个具体变量叫做这个类的一个实例。

类的实例-具体的类变量

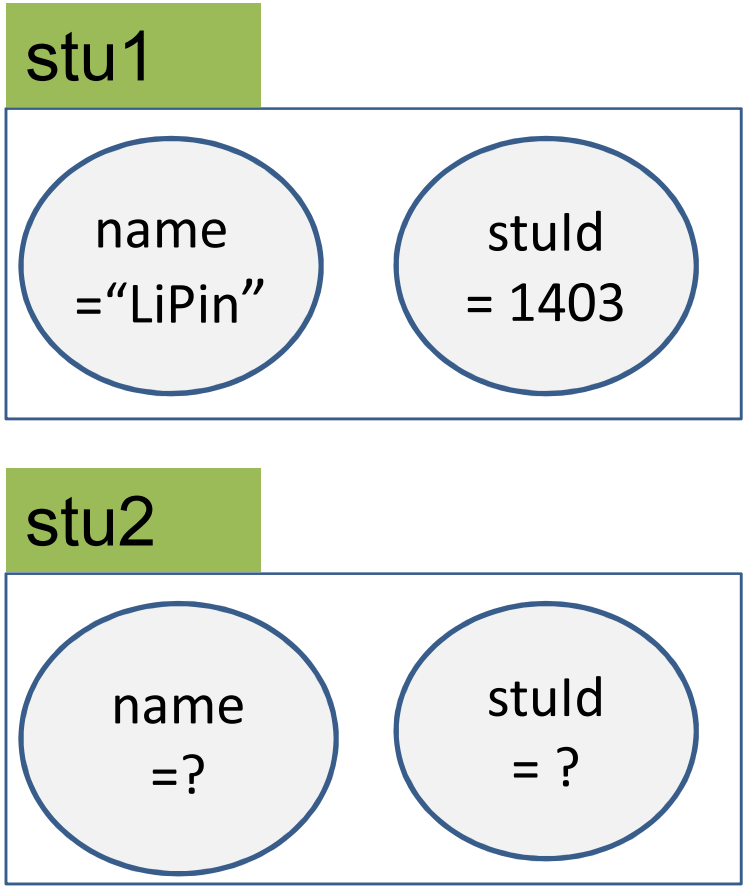
```
class Student {
public:
  char *name ;
  int stuId;
};
void main( ){
  Student stu1, stu2;
  stu1.name = “LiPin”;
}
```



- 某种类类型的一个具体变量叫做这个类的一个实例。

类的实例-具体的类变量

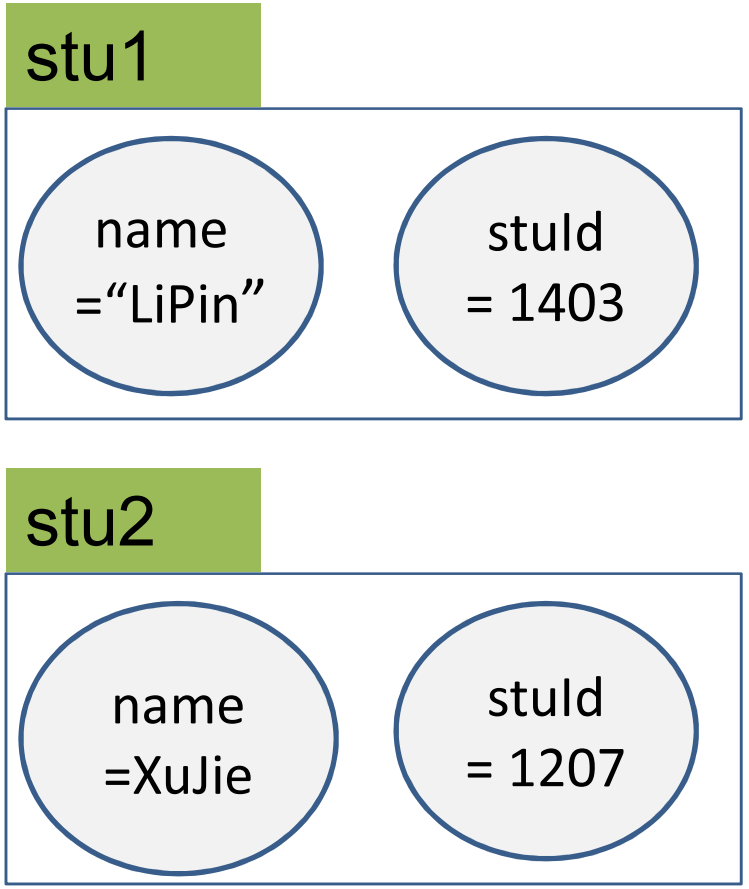
```
class Student {
public:
  char *name ;
  int stuId;
};
void main( ){
  Student stu1, stu2;
  stu1.name = “LiPin”;
  stu1.stuId = 1403;
}
```



- 某种类类型的一个具体变量叫做这个类的一个实例。

类的实例-具体的类变量

```
class Student {
public:
   char *name ;
   int stuId;
};
void main( ){
   Student stu1, stu2;
   stu1.name = “LiPin”;
   stu1.stuId = 1403;
   stu2.name = “XuJie”;
   stu2.stuId = 1207;
}
```



- 某种类类型的一个具体变量叫做这个类的一个实例。

操作成员/属性(Accessing Fields)

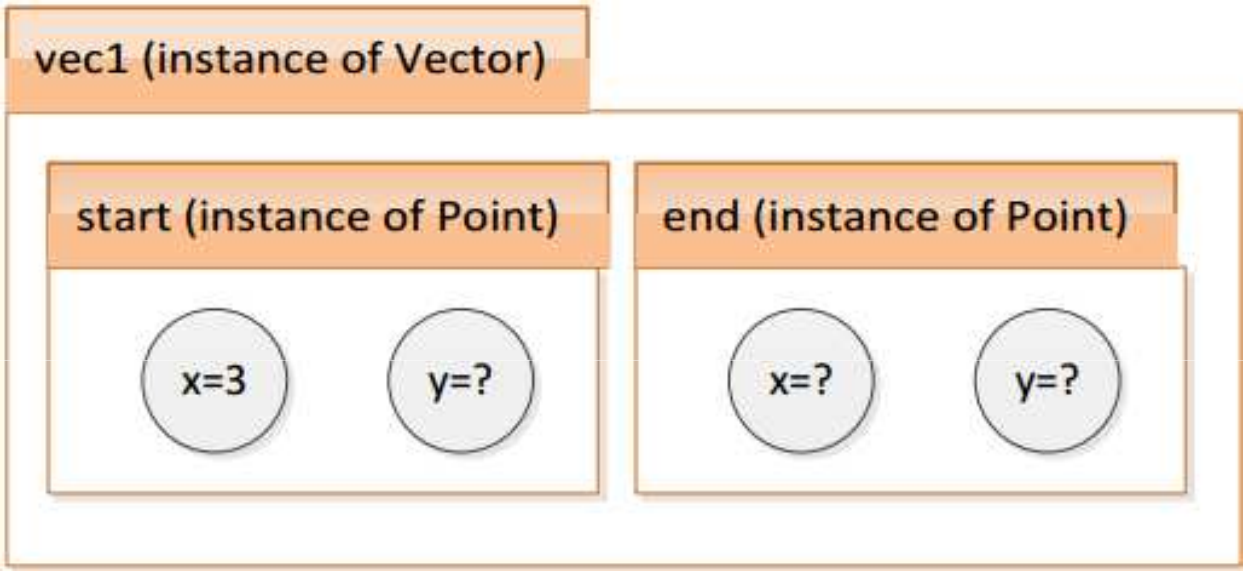
- 通过 “**变量名.成员名**”来访问类实例的成员(属性)

```
clss Student {
public:
  char *name ;
  int stuId;
};
void main( ){
  Student stu1, stu2;
  stu1.name = “LiPin”;
  stu1.stuId = 1403;
  stu2.name = “XuJie”;
  stu2.stuId = 1207;
  cout<<“the name of stu1 is:” << stu1.name<<‘\n’;
}
```

Point → Vector

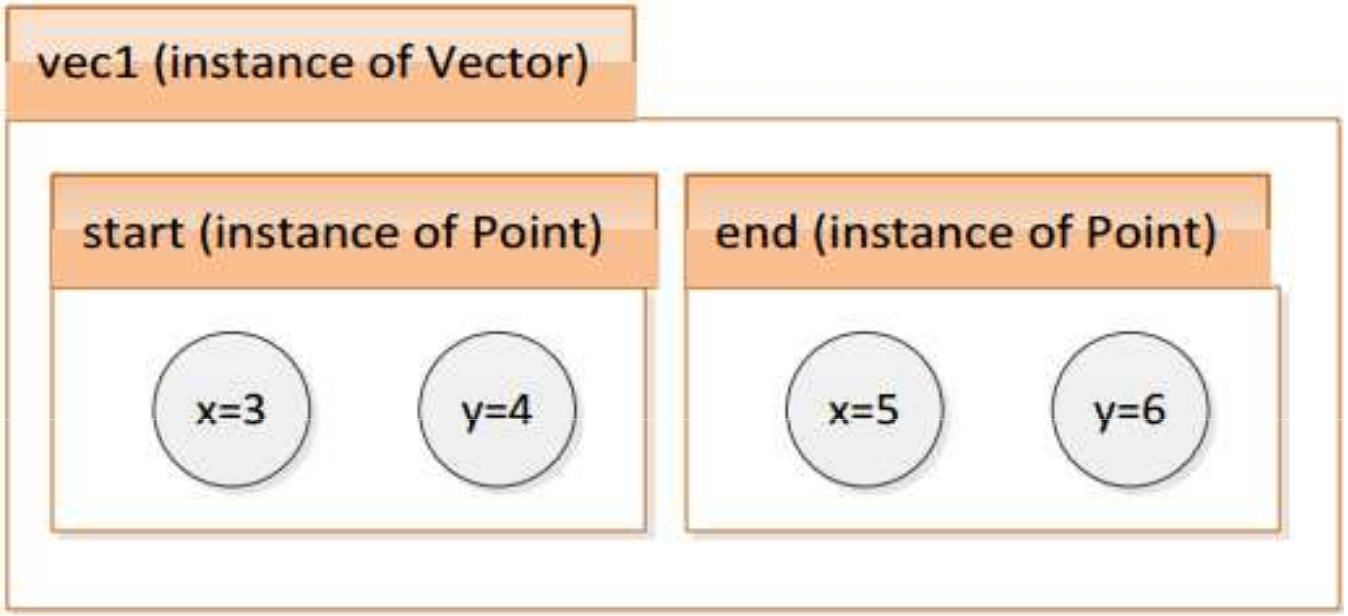
- 一个向量由2点(起点和终点)构成.

```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
void main() {
  Vector vec1;
  vec1.start.x = 3.0;
}
```

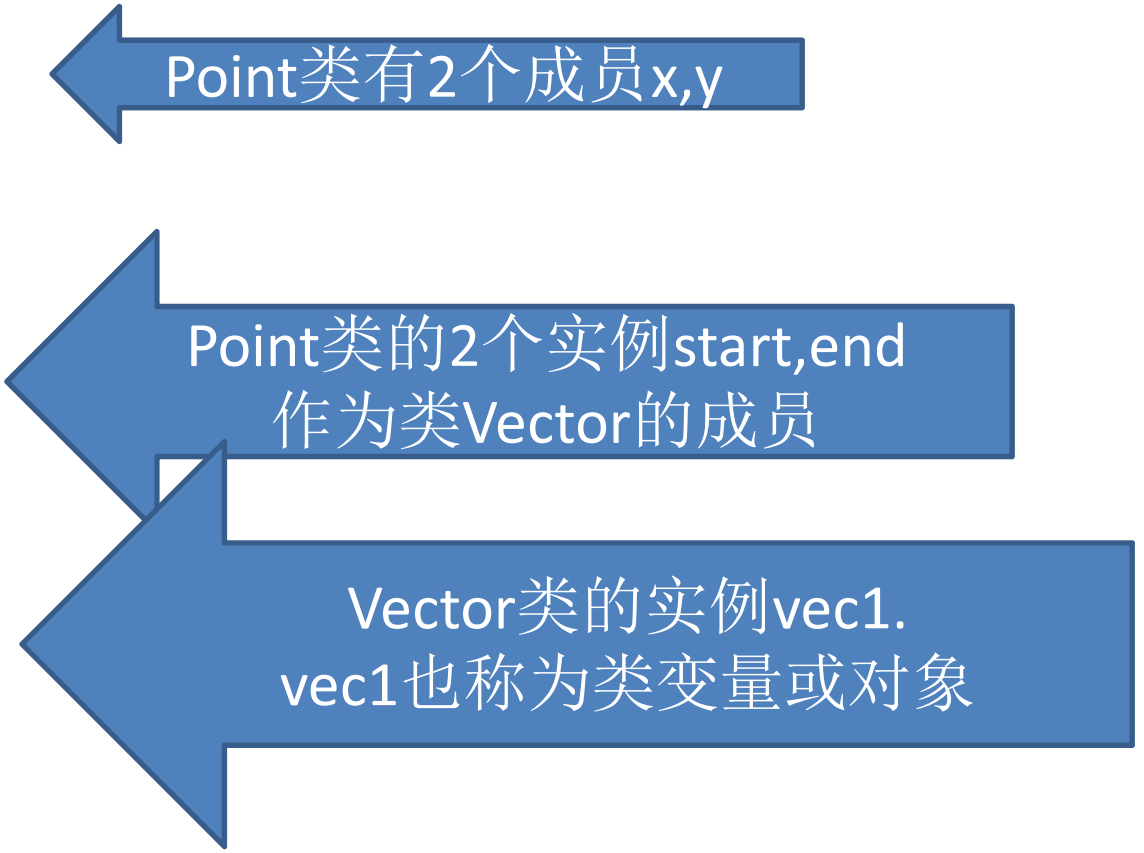


Point → Vector

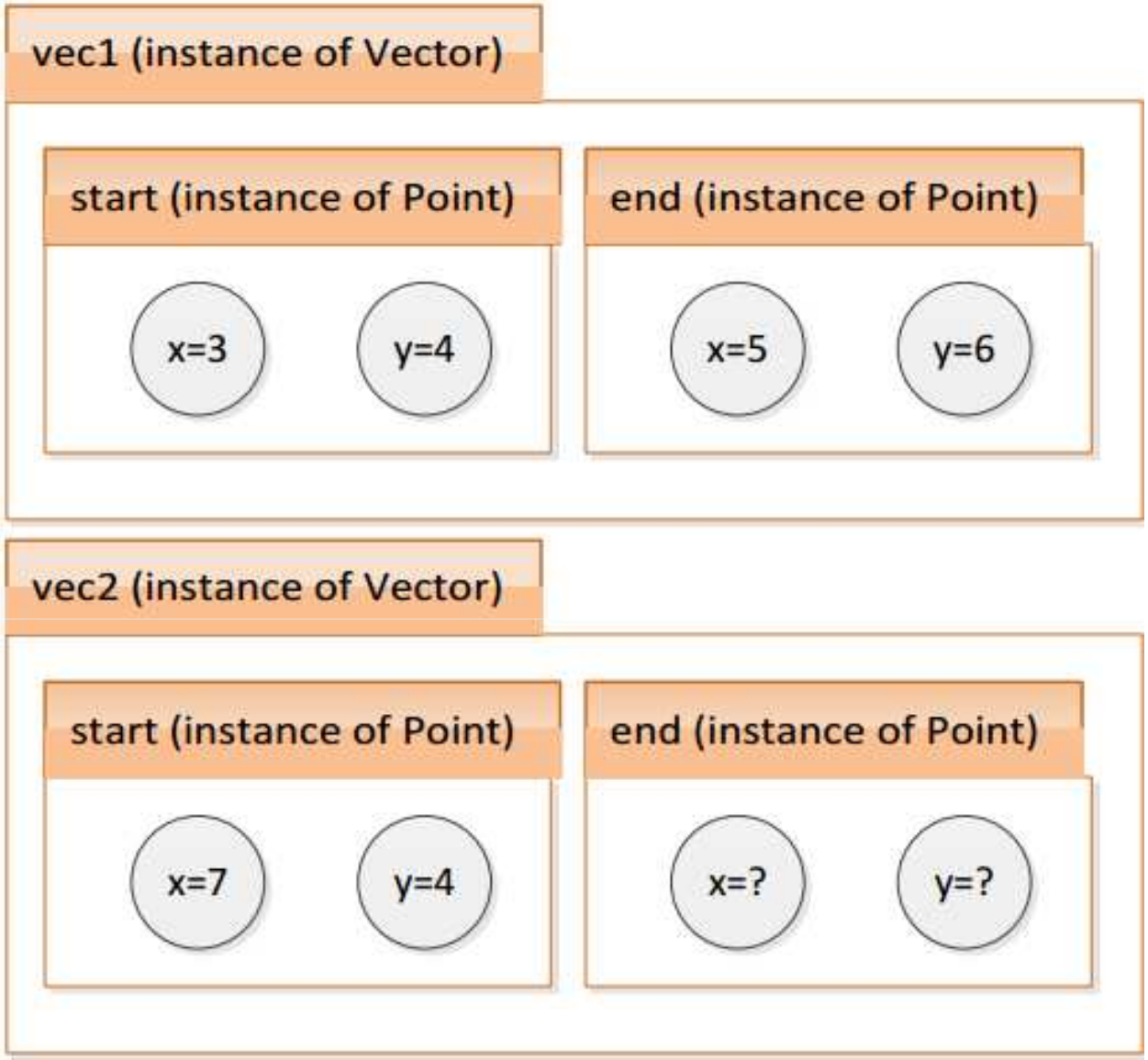
```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
void main() {
  Vector vec1;
  vec1.start.x = 3.0;
  vec1.start.y = 4.0;
  vec1.end.x = 5.0;
  vec1.end.y = 6.0;
}
```



```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
void main() {
  Vector vec1;
  vec1.start.x = 3.0;
  vec1.start.y = 4.0;
  vec1.end.x = 5.0;
  vec1.end.y = 6.0;
 Vector vec2;
  vec2.start = vec1.start;
  vec2.start.x = 7.0;
}
```



```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
void main() {
  Vector vec1;
  vec1.start.x = 3.0;
  vec1.start.y = 4.0;
  vec1.end.x = 5.0;
  vec1.end.y = 6.0;
 Vector vec2;
  vec2.start = vec1.start;
  vec2.start.x = 7.0;
}
```



```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
```

```
int main() {
  Vector vec;
  vec.start.x = 1.2; vec.end.x = 2.0; vec.start.y = 0.4; vec.end.y = 1.6;
}
```

```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};


void printVector(Vector v) {
  cout << "(" << v.start.x << "," << v.start.y << ") -> (" << v.end.x <<
        "," << v.end.y << ")" << endl;
}
int main() {
  Vector vec;
  vec.start.x = 1.2; vec.end.x = 2.0; vec.start.y = 0.4; vec.end.y = 1.6;
  printVector(vec);  }
```

```
class Point {
public:
  double x,y;
};
class Vector {
public:
  Point  start,  end;
};
```

如果不需要修改传进来的v，可以传值

```
void printVector(Vector v) {
  cout << "(" << v.start.x << "," << v.start.y << ") -> (" << v.end.x <<
        "," << v.end.y << ")" << endl;
}
int main() {
  Vector vec;
  vec.start.x = 1.2; vec.end.x = 2.0; vec.start.y = 0.4; vec.end.y = 1.6;
  printVector(vec);  }
```

```
class Point {
public:  double x,y;
};
class Vector {
public:  Point  start,  end;
};

void offsetVector(Vector &v, double offsetX, double offsetY) {
    v.start.x += offsetX;   v.end.x += offsetX;
     v.start.y += offsetY; v.end.y += offsetY;
}
void printVector(Vector v) {
  cout << "(" << v.start.x << "," << v.start.y << ") -> (" << v.end.x <<
          "," << v.end.y << ")" << endl;
}
int main() {
  Vector vec;
  vec.start.x = 1.2; vec.end.x = 2.0; vec.start.y = 0.4; vec.end.y = 1.6;
  offsetVector(vec, 1.0, 1.5);   printVector(vec);   }
```

```
class Point {
public:  double x,y;
};
class Vector {
public:  Point  start,  end;
};

void offsetVector(Vector &v, double offsetX, double offsetY) {
    v.start.x += offsetX;   v.end.x += offsetX;
     v.start.y += offsetY; v.end.y += offsetY;
}
void printVector(Vector v) {
  cout << "(" << v.start.x << "," << v.start.y << ") -> (" << v.end.x <<
          "," << v.end.y << ")" << endl;
}
int main() {
  Vector vec;
  vec.start.x = 1.2; vec.end.x = 2.0; vec.start.y = 0.4; vec.end.y = 1.6;
   offsetVector(vec, 1.0, 1.5);   printVector(vec);   }
```

如果要修改传进来的v，则传引用

类的函数成员(方法)

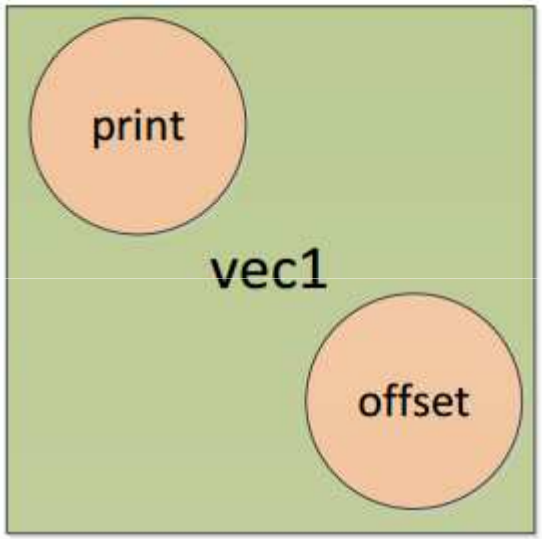
- 类的方法(Method)是一种属于类的函数，用于访问类的属性(数据成员)。

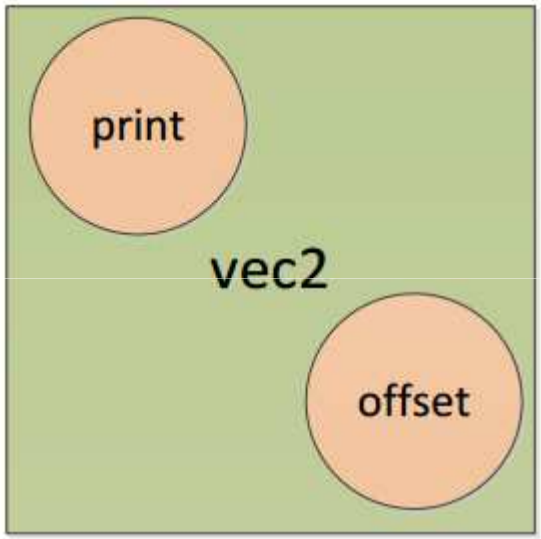
```
Vector vec;
vec.start.x = 1.2; vec.end.x = 2.0;
vec.start.y = 0.4; vec.end.y = 1.6;
vec.print();
vec.offset( 1.0, 1.5);
```

- 类的方法(Method)如同**盒子**(类实例)上的**按钮**，当按钮被按下(调用这个方法)就盒子产生某种动作。

类的函数成员(方法)

- 类的方法(Method)如同**盒子**(类实例)上的**按钮**，当按钮被按下(调用这个方法)就盒子产生某种动作

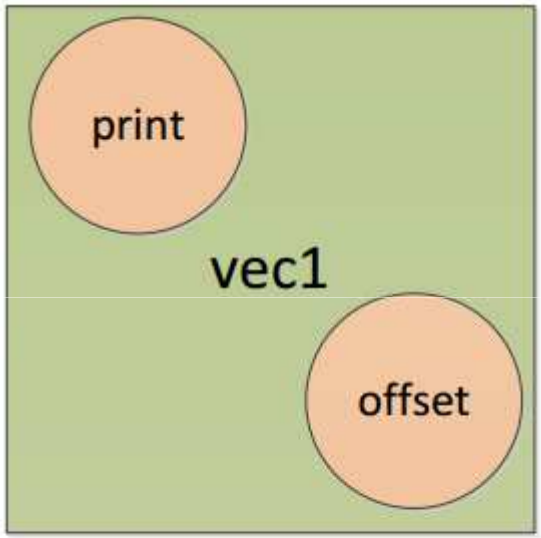


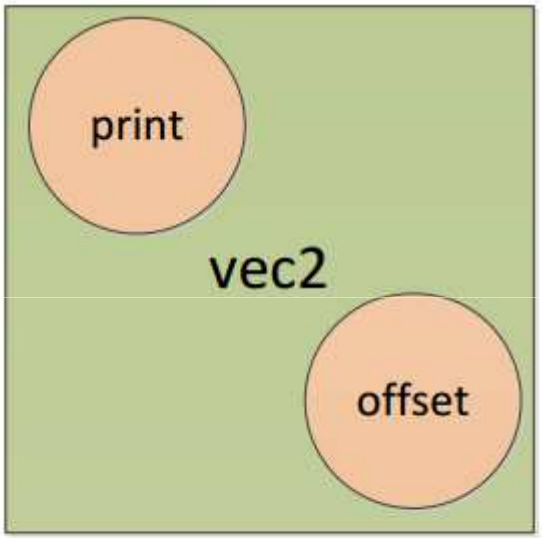



```
Vector vec1;
Vector vec2;
// initialize vec1 and vec2
vec1.print();
```

类的函数成员(方法)

- 类的方法(Method)如同**盒子**(类实例)上的**按钮**，当按钮被按下(调用这个方法)就盒子产生某种动作



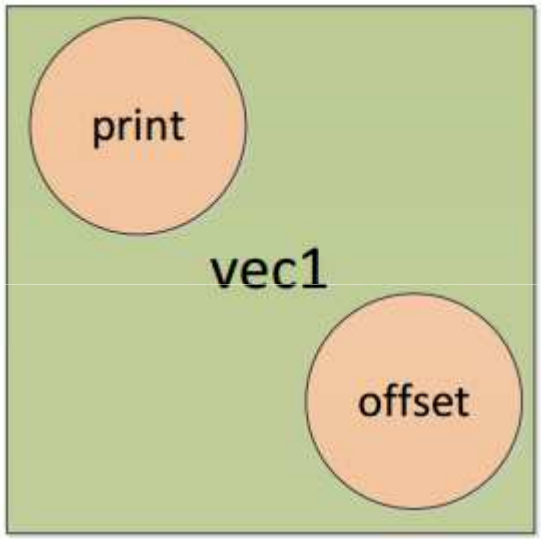


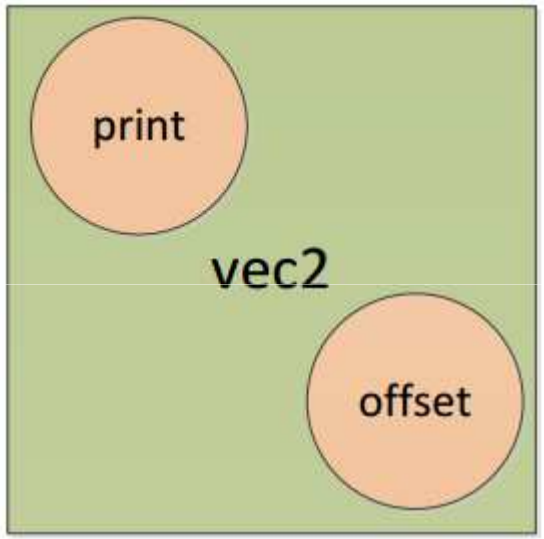

```
Vector vec1;
Vector vec2;
// initialize vec1 and vec2
vec1.print();
```

哪一个盒子？

类的函数成员(方法)

- 类的方法(Method)如同**盒子**(类实例)上的**按钮**，当按钮被按下(调用这个方法)就盒子产生某种动作



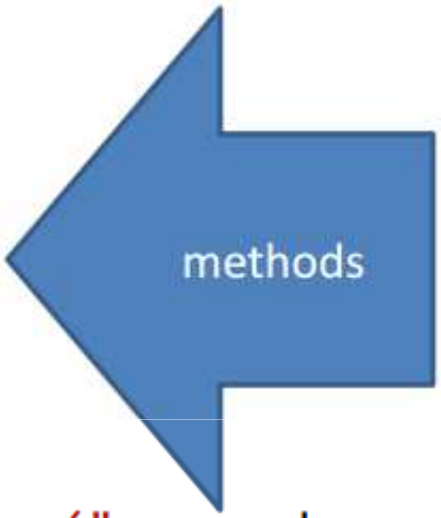



```
Vector vec1;
Vector vec2;
// initialize vec1 and vec2
vec1.print();
```

哪一个按钮？

```
class Vector {
public:
  Point start;
  Point end;

  void offset(double offsetX, double offsetY) {
    start.x += offsetX;
    end.x += offsetX;
    start.y += offsetY;
    end.y += offsetY;
  }
  void print() {
    cout << "(" << start.x << "," << start.y << ") -> (" << end.x <<
"," << end.y << ")" << endl;
  }
};
```



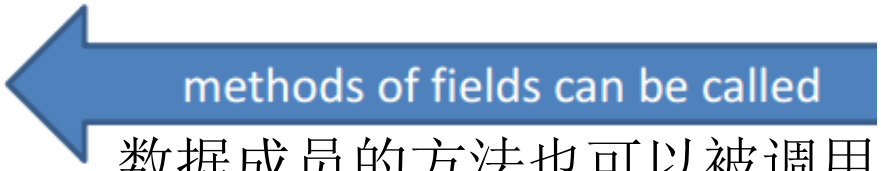
```
class Vector {
public:
  Point start;
  Point end;

  void offset(double offsetX, double offsetY) {
    start.x += offsetX;
    end.x += offsetX;
    start.y += offsetY;
    end.y += offsetY;
  }
  void print() {
    cout << "(" << start.x << "," << start.y << ") -> (" << end.x <<
"," << end.y << ")" << endl;
  }
};
```

数据成员可以被方法访问

```
class Vector {
public:
  Point start, end;

  void offset(double offsetX, double offsetY) {
    start.offset(offsetX, offsetY);
    end.offset(offsetX, offsetY);
  }
  void print() {
    start.print();
    cout << " -> ";
    end.print();
    cout << endl;
  }
};
```



数据成员的方法也可以被调用

```
class Point {
public:
  double x, y;
  void offset(double offsetX, double offsetY) {
    x += offsetX; y += offsetY;
  }
  void print() {
    cout << "(" << x << "," << y << ")";
  }
};
```

类方法的原型和实现的分离

- 如何函数的原型和实现可以分开一样，类的函数成员(方法)与其原型也可以分离。
- .h头文件是函数原型说明，.cpp文件是方法的实现

```
// vector.h - header file
class Point {
public:
  double x, y;
  void offset(double offsetX, double offsetY);
  void print();
};

class Vector {
public:
  Point start, end;
  void offset(double offsetX, double offsetY);
  void print();
};
```

```
#include "vector.h"
// vector.cpp - method implementation
void Point::offset(double offsetX, double offsetY) {
  x += offsetX; y += offsetY;
}
void Point::print() {
  cout << "(" << x << "," << y << ")";
}
void Vector::offset(double offsetX, double offsetY) {
  start.offset(offsetX, offsetY);
  end.offset(offsetX, offsetY);
}
void Vector::print() {
  start.print();
  cout << " -> ";
  end.print();
  cout << endl;
}
```

:: indicates which class' method is being implemented

- 许多初始化是乏味的，能否在定义类的实例时就自动进行初始化？

```
Vector vec;
vec.start.x = 0.0;
vec.start.y = 0.0;
vec.end.x = 0.0;
vec.end.y = 0.0;
```

```
Point p;
p.x = 0.0;
p.y = 0.0;
```

构造函数(Constructors)

- 在类实例创建时自动调用的方法。

```
class Point {
public:
  double x, y;
  Point() {
    x = 0.0; y = 0.0; cout << "Point instance created" << endl;
  }
};

int main() {
  Point p; // Point instance created
  // p.x is 0.0, p.y is 0.0
}
```

构造函数(Constructors)

- 可以接受参数

```
class Point {
public:
  double x, y;
  Point(double nx, double ny) {
    x = nx; y = ny; cout << "2-parameter constructor" << endl;
  }
};

int main() {
  Point p(2.0, 3.0); // 2-parameter constructor
  // p.x is 2.0, p.y is 3.0
}
```

构造函数(Constructors)

- 可以有多个构造函数

```
class Point {
public:
  double x, y;
  Point() {
    x = 0.0; y = 0.0; cout << "default constructor" << endl;
  }
  Point(double nx, double ny) {
    x = nx; y = ny; cout << "2-parameter constructor" << endl;
  }
};

int main() {
  Point p; // default constructor
  // p.x is 0.0, p.y is 0.0)
  Point q(2.0, 3.0); // 2-parameter constructor
  // q.x is 2.0, q.y is 3.0)
}
```

构造函数(Constructors)

- 默认构造函数：无参数或参数有默认值的构造函数。

```
class Point{
    double x,y;
    Point(double nx = 0, double ny = 0) { x= nx; y= ny;}
};

int main(){
   Point P, Q(1.2) , R(2.6, 3);
}
```

构造函数(Constructors)

- 一个类实例给另一个复制时，将复制其每个成员(调用默认**拷贝构造函数**(default **copy constructor**))

```
class Point {
public:
  double x, y;
  Point() {
    x = 0.0; y = 0.0; cout << "default constructor" << endl;
  }
  Point(double nx, double ny) {
    x = nx; y = ny; cout << "2-parameter constructor" << endl;
  }
};

int main() {
  Point q(1.0, 2.0); // 2-parameter constructor
  Point r = q;
  // r.x is 1.0, r.y is 2.0)
}
```

Invoking the copy constructor

构造函数(Constructors)

- 你可以定义自己的拷贝构造函数

```
class Point {
public:
  double x, y;
  Point(double nx, double ny) {
    x = nx; y = ny; cout << "2-parameter constructor" << endl;
  }
  Point(Point &o) {
    x = o.x; y = o.y; cout << "custom copy constructor" << endl;
  }
};

int main() {
  Point q(1.0, 2.0); // 2-parameter constructor
  Point r = q; // custom copy constructor
  // r.x is 1, r.y is 2
}
```

构造函数(Constructors)

- 为何定义自己的拷贝构造函数？ 默认的拷贝构造函数硬拷贝每个成员可能不是我们需要的！

```
class Student {
public:
  char *name ;
  int stuId;
  Student(){
    name = “”;
    stuId = 0;
  }
};
```

```
void main( ){
  Student stu1;
  stu1.stuID = 98;
  char n[] = "foo";
  stu1.name = n;
  Student stu2 = stu1;
  stu2.name[0] = 'b';
  cout << stu1.name;  //boo
}
```

对stu2的修改引起了stu1的改变！

构造函数(Constructors)

- 为何定义自己的拷贝构造函数？ 默认的拷贝构造函数硬拷贝每个成员可能不是我们需要的！

```
class Student {
public:
  char *name ;
  int stuId;
  Student(){
    name = “”;
    stuId = 0;
  }
  Student(Student &o){
    name = strdup(o.name);
    stuId = o. stuId ;
  }
};
```

```
void main( ){
  Student stu1;
  stu1.stuID = 98;
  char n[] = "foo";
  stu1.name = n;
  Student stu2 = stu1;
  stu2.name[0] = 'b';
  cout << stu1.name;  //foo
}
```

对stu2的修改不会改变stu1！

构造函数(Constructors)

- 初始化成员列表

```
class Rectangle {
    int width,height;
  public:
    Rectangle(int,int);
    int area() {return width*height;}
};
```

```
Rectangle::Rectangle (int x, int y) { width=x; height=y; }
```

构造函数(Constructors)

- 初始化成员列表

```
class Rectangle {
    int width,height;
  public:
    Rectangle(int,int);
    int area() {return width*height;}
};
```

```
Rectangle::Rectangle (int x, int y) : width(x) { height=y; }
```

构造函数(Constructors)

- 初始化成员列表

```
class Rectangle {
    int width,height;
  public:
    Rectangle(int,int);
    int area() {return width*height;}
};
```

```
Rectangle::Rectangle (int x, int y) : width(x), height(y) { }
```

构造函数(Constructors)

- 对于没有默认构造函数的类成员，该成员必须在初始化成员列表中初始化！

```
class Circle {
    double radius;
  public:
    Circle(double r) : radius(r) { }
    double area() {return radius*radius*3.14159265;}
};
class Cylinder {
    Circle base;
    double height;
  public:
    Cylinder(double r, double h) : base (r), height(h) { }
    double volume() { return base.area() * height;}
};
```

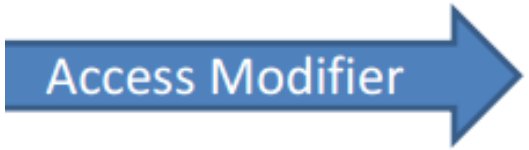
```
#include <iostream>

int main () {
  Cylinder foo (10,20);

  std:: cout << "foo's volume: " << foo.volume() << '\n';
  return 0;
}
```

访问修饰符(Access Modifiers)

- 定义类的属性能被？访问。




```
class Point {
public:
  double x, y;

  Point(double nx, double ny) {
    x = nx; y = ny;
  }
};
```

访问修饰符(Access Modifiers)

- public : 能从任何地方访问

```
class Point {
public:
  double x, y;

  Point(double nx, double ny) {
    x = nx; y = ny;
  }
};

int main() {
  Point p(2.0,3.0);
  p.x = 5.0; // allowed
}
```

访问修饰符(Access Modifiers)

- private : 只能从类内部的方法访问

```
class Point {
private:
  double x, y;

public:
  Point(double nx, double ny) {
    x = nx; y = ny;
  }
};

int main() {
  Point p(2.0,3.0);
  p.x = 5.0; // not allowed
}
```

访问修饰符(Access Modifiers)

- getters: 提供对私有数据成员的查询方法

```
class Point {
private:
  double x, y;

public:
  Point(double nx, double ny) {
    x = nx; y = ny;
  }
  double getX() { return x; }
  double getY() { return y; }
};

int main() {
  Point p(2.0,3.0);
  cout << p.getX() << endl; // allowed
}
```

默认的访问修饰符(Default Access Modifiers)

- class: 默认是private

```
class Point {
  double x, y;
};
```


```
class Point {
private:
  double x, y;
};
```

默认的访问修饰符(Default Access Modifiers)

- struct 默认是public
- struct 是对C中的struct的扩展(增加了方法)
- struct和class都是定义类类型，除了其默认访问修饰符不同！

```
class Point {
public:
  double x;
  double y;
};
```

```
struct Point {

  double x;
  double y;
};
```

默认的访问修饰符(Default Access Modifiers)

- struct: 默认是public
- class:默认是private

```
struct Point {
  double x, y;
};
```


```
struct Point {
public:
  double x, y;
};
```

```
class Point {
  double x, y;
};
```


```
class Point {
private:
  double x, y;
};
```

作业

- 定义一个表示三维向量类Vector3，能够完成常见的数学向量运算，如加减、点积、差积、求长等