Class

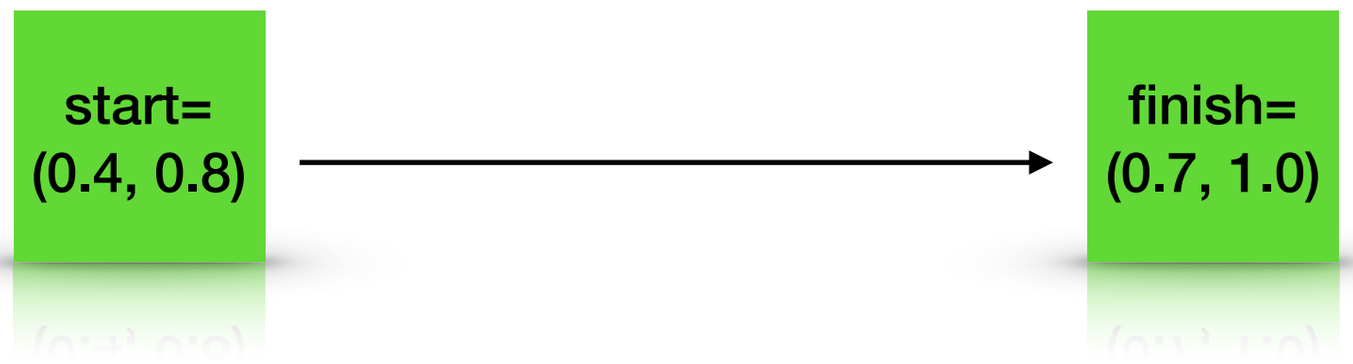
CSE2013-17F

slides are from CSE2013-16S at SKKU & 6.096 at MIT

Representing a Vector

- In the context of geometry, a vector consists of 2 points: a start and a finish

- Each point itself has an x and y coordinate



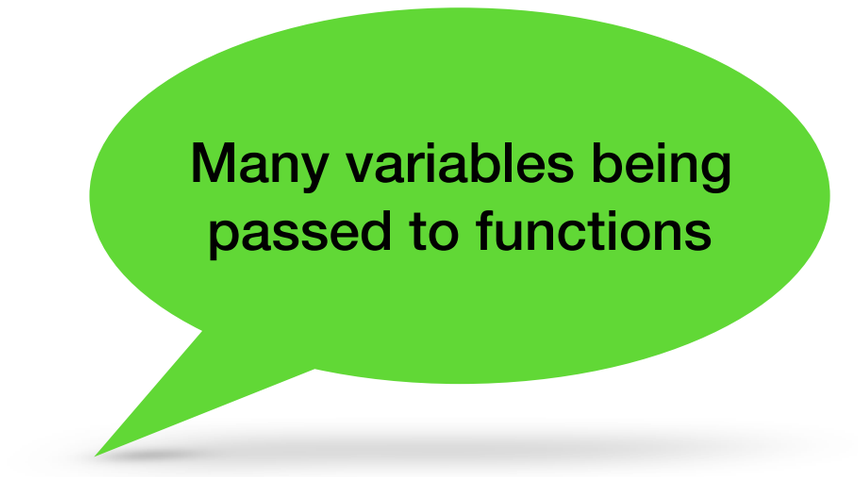
Representing a Vector

- Our representation so far? Use 4 doubles (startx, starty, endx, endy)

- We need to pass all 4 doubles to functions

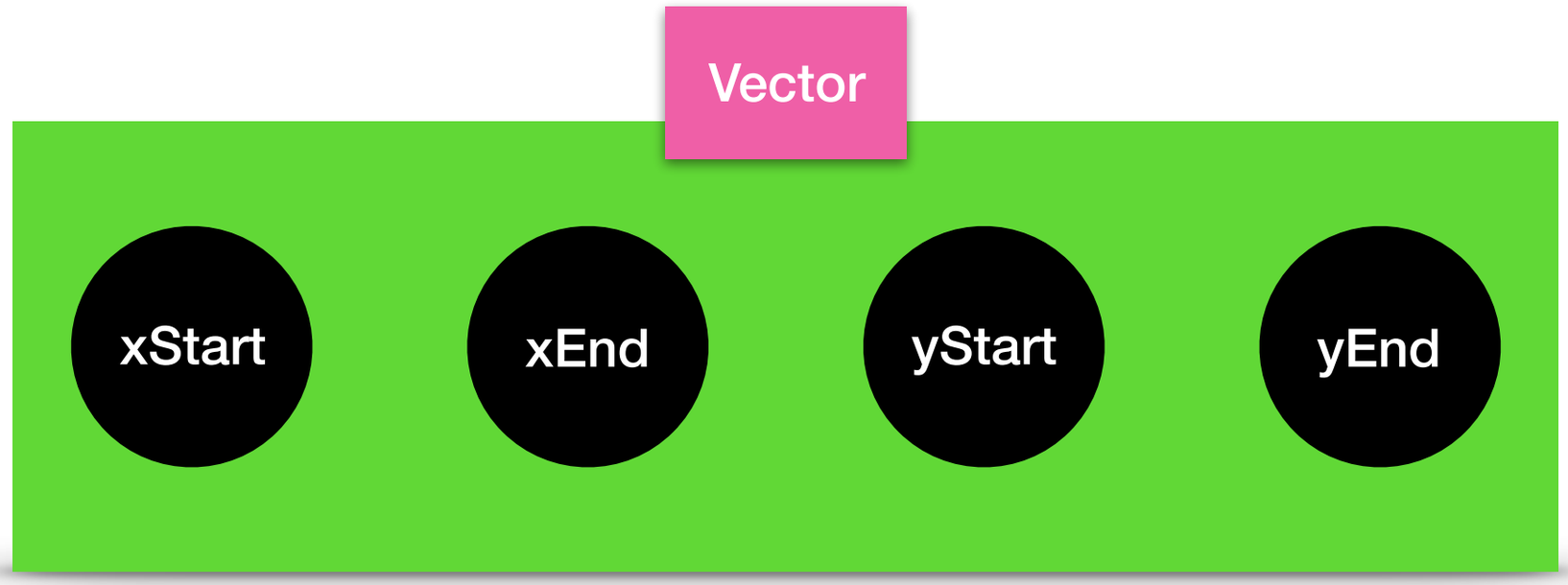
```
int main() {
    double xStart = 1.2;
    double xEnd = 2.0;
    double yStart = 0.4;
    double yEnd = 1.5;
}
```

```
void offsetVector (double &x0, double &x1, double &y0, double &y1,
double offsetX, double offsetY) {
    …
}

void printVector (double x0, double x1, double y0, double y1) {
    cout << … << endl;
}

int main() {
    double xStart = 1.2;
    double xEnd = 2.0;
    double yStart = 0.4;
    double yEnd = 1.5;

    offsetVector (xStart, xEnd, …);
    printVector (xStart, …);
}
```



class

- A user-defined datatype which groups together related pieces of information



class definition syntax

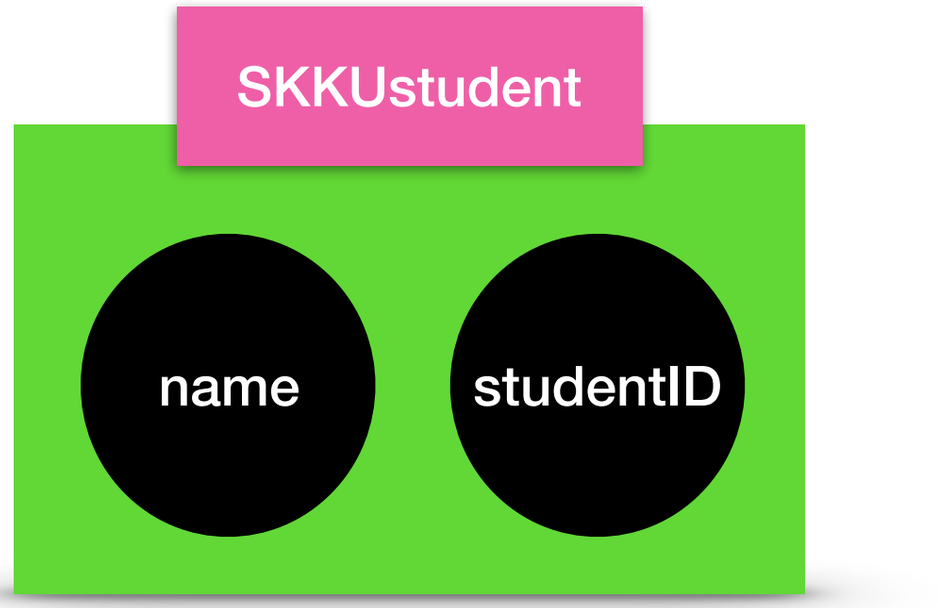
name

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

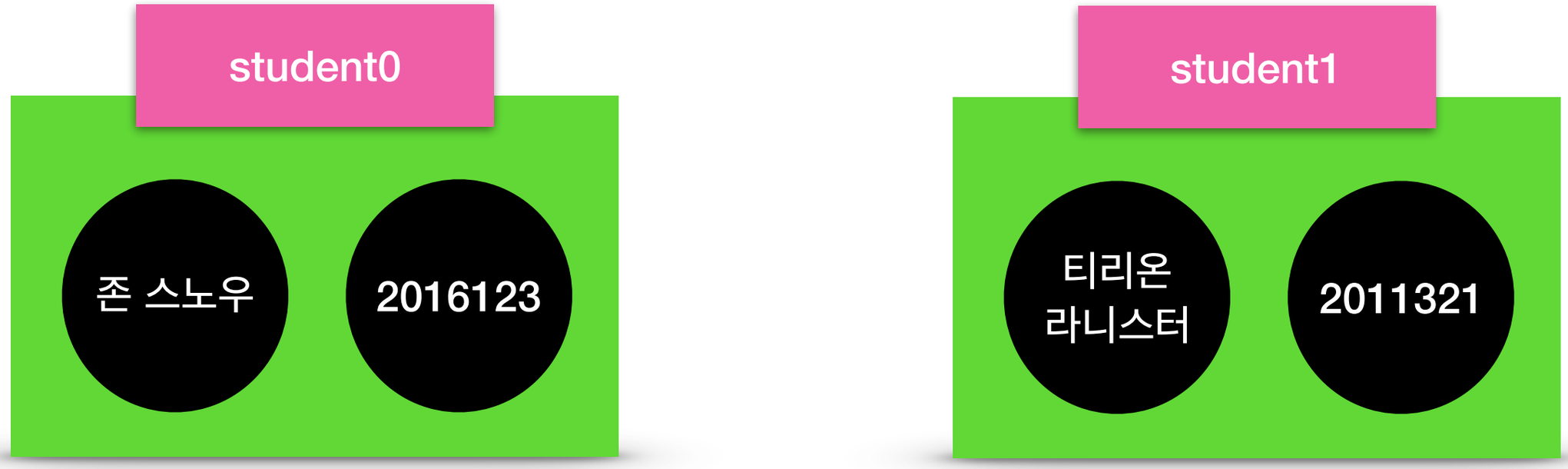
fields or member

Fields can have different types

```
class SKKUstudent {
public:
     char *name;
     int studentID;
};
```

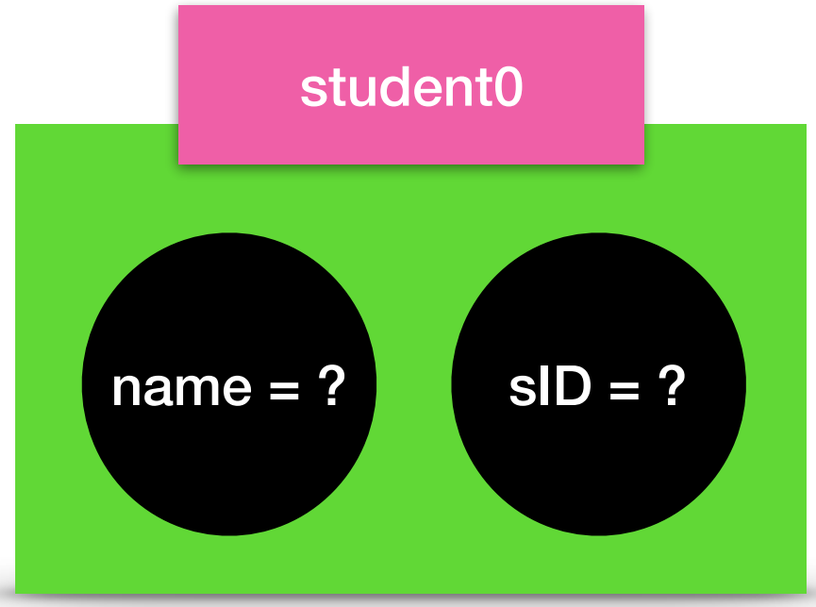


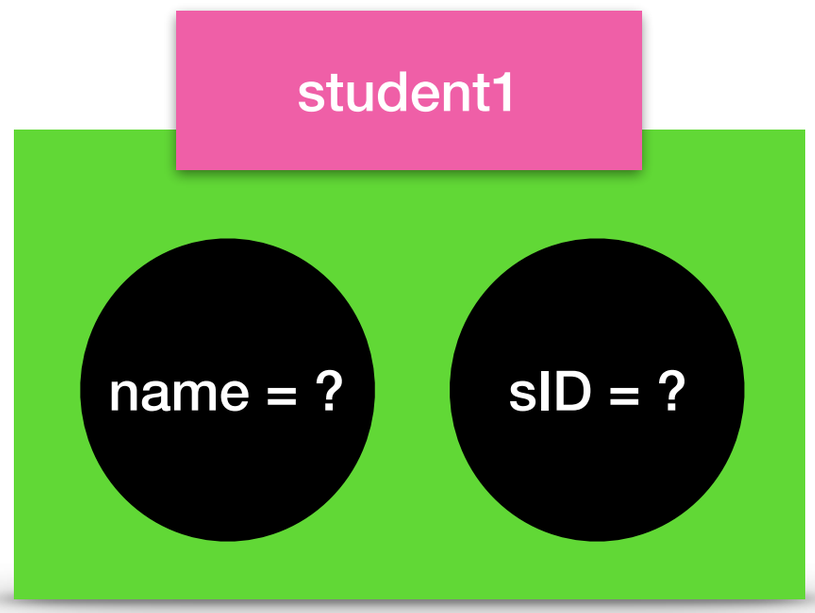
Instances



Declaring an Instance

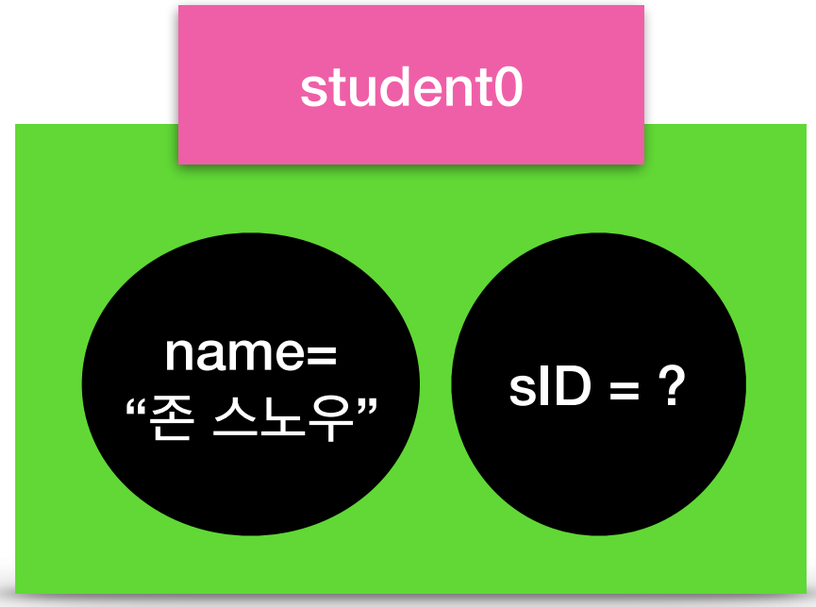
```
class SKKUstudent {
public:
    char *name;
    int studentID;
};

int main() {
    SKKUstudent student0;
    SKKUstudent student1;
}
```

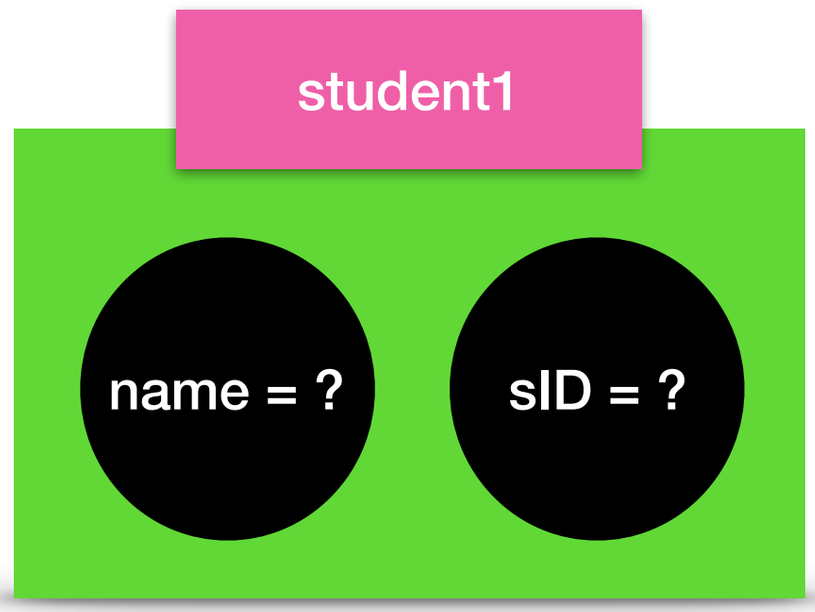




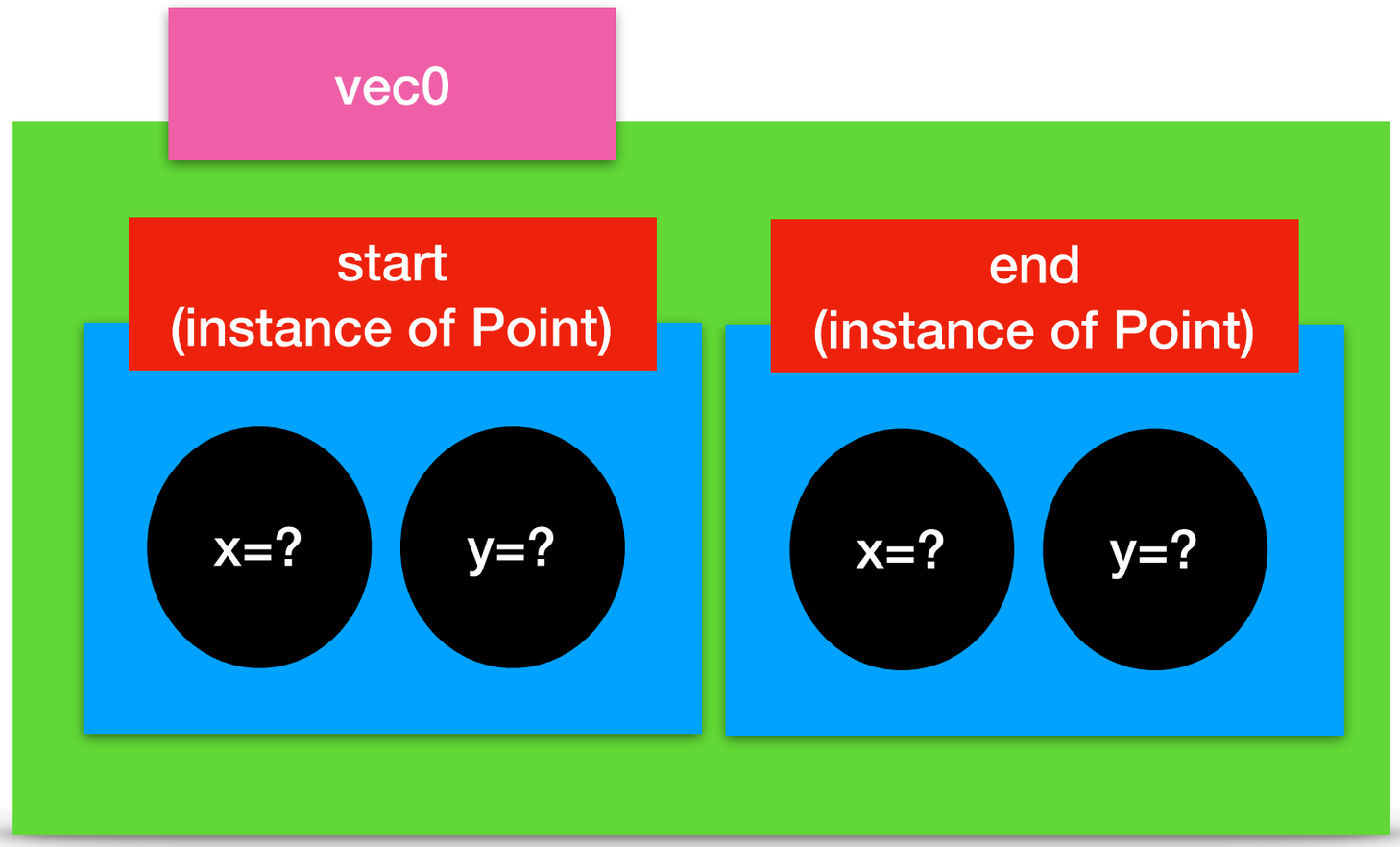
Accessing Fields

```
class SKKUstudent {
public:
     char *name;
     int studentID;
};

int main() {
     SKKUstudent student0;
     SKKUstudent student1;
     student0.name = "존 스노우";
    cout << "student0 name is"
         << student0.name << endl;
}
```

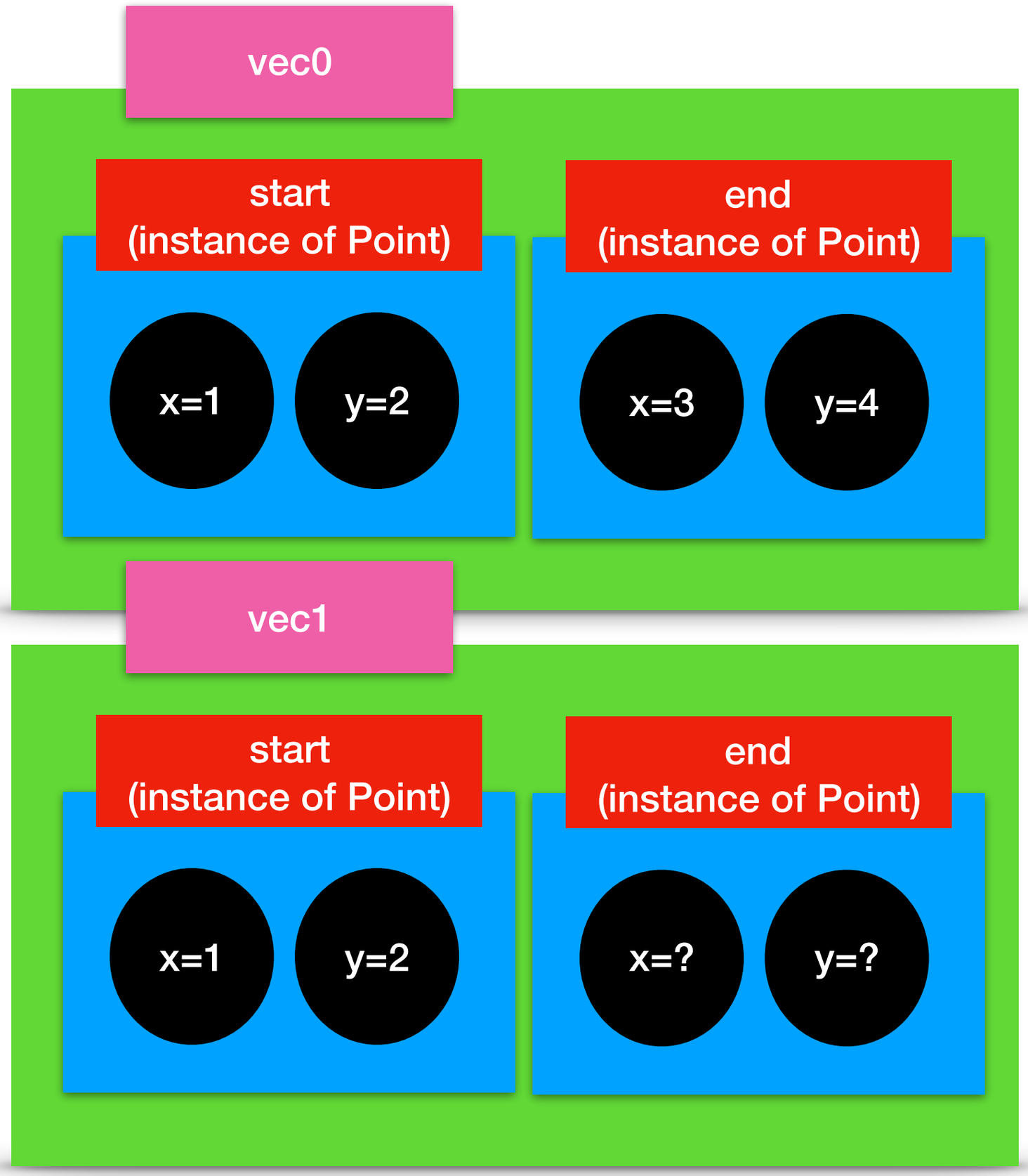




```
class Point {
public:
    double x;
    double y;
};

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

int main() {
    Vector vec0;
}
```



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

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

int main() {
     Vector vec0;
     vec0.start.x=1.0;
     vec0.start.y=2.0;
     vec0.end.x=3.0;
     vec0.end.y=4.0;
     Vector vec1;
     vec1.start = vec0.start;
}
```



Passing classes to functions

- Passing by value passes a copy of the class instance to the function; changes aren’t preserved

```
class Point {
public: double x; double y;
};
class offsetPoint(Point P, double x, double y) {
    p.x += x;
    p.y += y;
}
int main() {
    Point p; p.x = 3; p.y = 4;
    offsetPoint(p, 1, 2); // does nothing
}
```

Passing classes to functions

- Pass by reference

```
class Point {
public: double x; double y;
};
class offsetPoint(Point &P, double x, double y) {
    p.x += x;
    p.y += y;
}
int main() {
    Point p; p.x = 3; p.y = 4;
    offsetPoint(p, 1, 2); // works properly
}
```

Methods

- Functions which are part of a class

```
class Point {
public: double x, y;
    void print() {
        cout << ‘(‘ << x << ‘, ‘ << y << ‘)’ << endl;
};
int main() {
    Point p; p.x = 3; p.y = 4;
    p.print(); // print (3, 4)
}
```

```
// header.h
class Point {
public: double x, y;
    void print();
};
void Point::print() {
    cout << '(' << x << ', ' << y << ')' << endl;
}
```

:: indicates which class' method is being implemented

Constructors

- Method that is called when an instance is created

```
class Point {
public:
    double x, y;
    Point() {
        x = 0.0; y = 0.0;
    }
    Point(double a, double b) {
        x = a; y = b;
    }
};
int main() {
    Point p0; // p0.x = 0.0, p0.y = 0.0
    Point p1(1, 2); // p1.x = 1, p1.y = 2
    Point p2 = p1; // p2.x = 1, p2.y = 2
}
```

Destructor

- Cleanup

```
class Point {
public:
    double x, y;
    Point() {
        x = 0.0; y = 0.0;
    }
    ~Point() {
        x = 0.0; y = 0.0;
        // mostly used to deallocate data structures
        // e.g. delete []array;
    }
};
int main() {
    Point p0; // p0.x = 0.0, p0.y = 0.0
}
```

Access Methods

```
class Point {
private:
    double x1, y1;
public:
    double x, y;
    Point(double a, double b) {
        x = a; y = b;
    }
};
int main() {
    Point p1(1, 2); // p1.x = 1, p1.y = 2
    p1.x1 = 3; // not allowed
}
```

Access Methods

```
class Point {
private:
    double x1, y1;
public:
    double x, y;
    Point(double a, double b) {
        x = a; y = b;
    }
    setX1Y1(double a, double b) {
        x1 = a; y1 = b;
    }
};
int main() {
    Point p1(1, 2); // p1.x = 1, p1.y = 2
    p1.x1 = 3; // not allowed
    p1.setX1Y1(3, 4); // allowed
}
```

Inheritance

```
class Person {
  int age;
  char name[20];
public:
  int getAge() {
    …
  }
  char *getName() {
    …
  }
}
class Student: public Person {
public:
  …
  void printInfo() {
    cout << “Name: “ << getName() << endl;
    cout << “Age: “ << getAge() << endl;
  }
  …
}
```

Polymorphism

- Run-time interpretation of object type - “Late Binding”
- implemented using virtual functions

```
class Figure
{
public:
      virtual string draw () = 0;
};
```

```
class Triangle : public Figure
{
public:
      virtual string draw (){ return “T” };
}
```

```
class Square : public Figure
{
public:
      virtual string draw (){ return “S” };
}
```

```
class Circle : public Figure
{
public:
      virtual string draw (){ return “C” };
}
```

```
int main ()
{
    Figure *F1 = new Triangle;
    Figure *F2 = new Square;
    Figure *F3 = new Circle;

    cout << F1->draw () << endl;
    cout << F1->draw () << endl;
    cout << F1->draw () << endl;

    delete F1; delete F2; delete F3;
}
```

stdout:
T
S
C