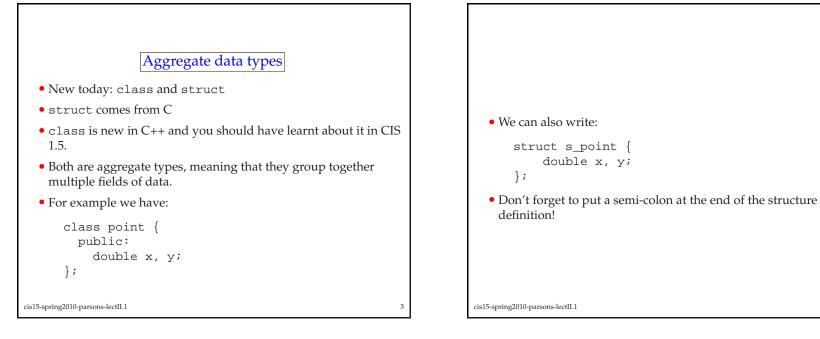


# Today

- We will start to talk about *object-oriented programming*
- In particular we will talk about struct and class.
- We will show how to use these features of C++ to define aggregate data types.
- We will show how to define *methods* that operate on these data types.
- This work is based on Pohl, Chapter 4.
- Much of the work we will do for the next couple of weeks will be concerned not only with what we can do in C++, but also the *style* in which we do it.

```
cis15-spring2010-parsons-lectII.1
```



### Aside: why is point useful?

- The idea behind point is that it represents information about the location of something.
- Think of it as a pair of (Cartesian) coordinates.
- We group the coordinates together because they make no sense separately if we have the x coordinate of a thing, then it has a y coordinate also.
- We will use point when we write a simulation of small eco-system and of a robot operating in a simulated world. We will do this in some of the homeworks.

```
cis15-spring2010-parsons-lectII.1
```

- The fields or elements of an aggregate data type are called *members*.
- Members are referred to using "dot notation", e.g.:

$$p.x = 7.0; p.y = 10.3;$$

• You can also use a *pointer* to access members of an aggregate data type, e.g.:

 $p \rightarrow x = 12.3;$ 

but we will discuss pointers in the next unit, so don't worry about this now...

#### Back to aggregate data types

- In C, the tag (point) is optional and does not constitute a data type (you need to use typedef as well).
- In C++, the tag is considered a data type, hence the above example is a data type definition.
- This means that you can use point as a data type, e.g.:

point p;

- In other words, you can declare a variable p which is of type point.
- p is called an *object* or an *instance* of class point.

cis15-spring2010-parsons-lectII.1

• Just as you can define an object of type point:

point p;

you can define an array of these objects

```
point myPoints[3];
```

and even

point myPoints[3] = {{1, 2}, {3, 4}, {5, 6}};

which defines the array myPoints to hold three elements each of which is a class of type point which holds two doubles, and sets the values of these.

• We can then access the individual members as before:

```
cout << myPoints[1].x;</pre>
```

```
will, for example, print out 3.
```

cis15-spring2010-parsons-lectII.1

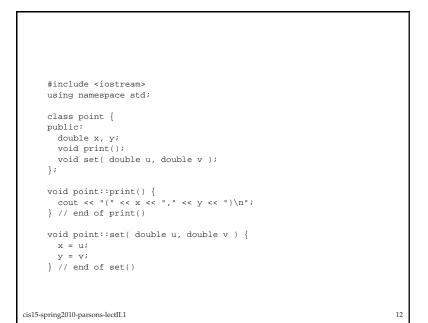
cis15-spring2010-parsons-lectII.1

#### Member functions

- In C++, members of aggregate data types can be functions
- (C only allows data members)
- In object-oriented programming (OOP) lingo, the word "method" is often used instead of "function"
- The reason to define functions inside an aggregate data type is to follow the OOP principle of *encapsulation*—operations should be packaged with data
- This is a *style* thing.
- For example:

cis15-spring2010-parsons-lectII.1

#### #include <iostream> using namespace std; class point { public: double x, y; void print() { cout << "(" << x << "," << y << ")\n"; } void set( double u, double v ) { x = u; y = v;}; // end of class--don't forget semi-colon! int main() { point w; w.set( 1.2, 3.4 ); cout << "point = "; w.print(); } cis15-spring2010-parsons-lectII.1



10

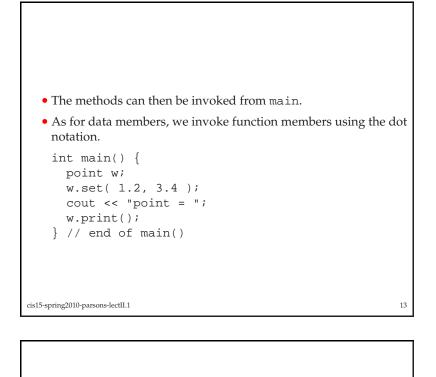
• Notes:

- Notice that the set method changes the values of the data members—this is considered good OOP practise
- Defining the methods inside the class definition is called "in-line declaration"; this is generally only okay for short, concise methods
- The *class scope* operator can be used when in-line declarations are inappropriate.

11

• For example:

cis15-spring2010-parsons-lectII.1



```
• Here is a (maybe confusing) example from the book:
 int count = 0; // declare global variable
 void how_many( double w[], double x, int& count ) {
   for ( int i=0; i<N; ++i ) {</pre>
      count += ( w[i] == x ); // local count
   ++::count; // global count
 } // end of how_many()
• We need to use the unary scope operator her since count is
 declared twice
• If you didn't have the ::count, then the second time, the use of
 count would also refer to the local variable
• It is better practise not to use global variables; or at least if you
```

do, give them unique names to avoid confusion :-)

cis15-spring2010-parsons-lectII.1

```
Class scope
```

• The class scope operator is two colons (::), as in our example:

```
void point::print() const {
  cout << "(" << x << "," << y << ")\n";
```

- The :: operator has the highest precedence in the language, so it always gets evaluated first
- There are two versions of the operator: binary and unary
- The binary version is the one we used before: point::print(), which is used to refer to a variable's "class scope" (also called "local scope").
- The unary version is like this: ::count and is used to refer to a variable's "external scope" (e.g., for a global variable).

```
cis15-spring2010-parsons-lectII.1
```

## "this" pointer

- The keyword this is used to refer to an instance of a class from within itself.
- It is a *pointer* something we will discuss at length in the next unit
- Here is a possible use to give you the idea.
- The data members are available anywhere inside any function members:

```
point::foo(double a) {
  if(x == a)
    cout << y;
```

```
cis15-spring2010-parsons-lectII.1
```

15

16

14

