Class recap

- Class definition
  - Creates a new data type
  - Does not allocate memory!
  - Tell the compiler what the parts of the class are
  - Each part has a type and a name (looks just like a variable)
  - The parts of a class are called *members*

```
class employee
{
 public:
    string name;
    string position;
    int review_score;
};
```

Class recap

- Object variable declaration:
  - Variable whose type is a class
    - As opposed to a primitive (int, char, double, bool, etc.)
  - Variable declaration is always:

```
type name;
```

  - So in this example:

```
employee emp1;
```

  - Allocates memory space for an *instance* of the class
    - 1 string, 1 int, 1 double
  - Gives the whole thing a name
  - A class instance is also called an *object*

Class recap

- Using an object variable
  - The *member access operator* (.) indicates part of an object
  - The parts are also variables:

```
emp.name = "peter";
cin >> emp.position = "manager";
emp.review_score = emp.review_score + 1;
```

Classes vs. Structs

- In C, structured data was stored in *structs*
- In C++, classes were added
  - They're nearly identical under the hood
  - You can use them more or less interchangeably

```
class point
{
 public:
    int x, y;
};
```

```
struct point
{
   int x, y;
};
```

Classes vs. Structs

- Classes are typically used when *object-oriented* features are needed
  - We'll be discussing those later

- Most C++ programmers use structs when those features aren't necessary
  - Even though structs and classes both technically support those features

- For simplicity in this course, we'll just use classes instead of structs across the board

Whole vs. part

- Arrays and classes have parts
  - Accessed by the subscript `[]` and member `.` operators
  - The whole is a piece of memory you can refer to
  - The parts are also pieces of memory you can refer to
  - The wholes and parts all have data types
    - e.g . int, string, array of doubles, employee object
- Functions only work with particular data types
  - Specified in the formal parameter list:

```
void myfn( int x, employee y, double a[], string s );
```

  - Function calls must provide matching argument types
    - Could be whole variables, or parts

Whole vs. part

- Operators also only work with particular data types
  - The addition operator (+)
    - Works with `double`, `int`, `char` (addition)
    - Works with `string` (concatenation)
    - Does not work with any arrays!
    - Does not work with our `employee` class objects!
  - The insertion operator (<<)
    - Works with `double`, `int`, `char`, `string`
    - Does not work with any arrays!
    - Does not work with our `employee` class objects!
- Therefore, to print an array/object
  - Must print them out one part (element or member) at a time

Copying objects

- How to copy from one object to another?
  - Assignment operator (=) works with primitive data types
    - int, char, double, bool, etc.
  - Assignment operator (=) works with strings
  - Assignment operator does not work with arrays
    - Have to copy element-by-element
  - What about class objects?

Copying objects

- How to copy from one object to another?
  - Assignment operator (=) works with primitive data types
    - int, char, double, bool, etc.
  - Assignment operator (=) works with strings
  - Assignment operator does not work with arrays
    - Have to copy element-by-element
  - What about class objects?
    - Can copy member-by-member
    - Can also assign an entire object (provided it contains no arrays!)

Copying objects

- How does C++ support object assignment?
  - Behind the scenes, it's member-by-member copy
- Why does C++ support object assignment?
  - Because it allows objects to be passed-by-value to functions
  - Also allows objects to be returned by functions
- Notice that arrays don't support assignment
  - And arrays are implicitly passed by reference

Other operations

- And the other built-in operations?
  - +, −, ==, etc.
  - No built-in support for aggregate operations
  - All must be done member-by-member
- This includes:
  - Stream input (>>) to an object
  - Printing (<<) an object
  - Comparing two objects

Class Composition

- Once defined, a class is used like any other data type
  - E.g `int`, `double`, `char`
  - `string` is actually a class
- So a class can hold:
  - All primitive types (e.g. 2 ints and a double)
  - A mix of primitives and classes (e.g. an int and a string)
    - Including other class types you've defined

Example: Time Class

- What are the pieces of data that make up a time?

Example: Time Class

- What are the pieces of data that make up a time?
  - Hours (int)
  - Minutes (int)
  - Seconds (int)
  - AM/PM (bool, char, int, string…)
  - Time Zone (int, string)

Exercise: Composing Classes

- Define a class `Date` to hold a calendar date

- Define a class `CalEntry` to hold a calendar entry, including:
  - Event description
  - Date
  - Time

Exercise: Using a Composed Object

- Declare a `CalEntry` object
- Write statements to set the values:
  - Event name: “Quiz 5”
  - Date: 4/7/2010
  - Time: 10:45am

- Declare an array of `CalEntry` objects
- Write statements to:
  - Set the name of the fourth event to “Lunch at Bob’s”
  - Set the time in the first event to the time of the third event
  - Write a loop to print all the event names

Exercise: Email Inbox

- Consider your inbox
  - Multiple emails
    - Each email has a sender, recipient(s), subject and body (text only)
    - Each email is received at a particular date and time

1. Write a class definition for an email
   - Use the `Date` and `Time` classes we already defined

2. Declare a variable to hold all the emails in your inbox
   - And another to keep count!

Exercise: Email Inbox

3. Given the email class and your inbox variables:
   - Write a function to print your inbox
     - Should have date/time received, sender and subject on each line
     - What parameters need to be passed in?
   - Write a function to print a particular email
     - What are the 2 different sets of parameters you could pass in?
     - Should look like:

From: bob@bob
To: etomai@cs.panam.edu
Received: 10:32pm 11/18/2010
Subject: about those labs

Blah, blah, blah. Blah, blah, blah. Blah, blah, blah. Blah, blah, blah.
Blah, blah, blah. Blah, blah, blah. Blah, blah, blah. Blah, blah, blah.
Blah, blah, blah. Blah, blah, blah. Blah, blah, blah. Blah, blah, blah.