And Even More and More C++
Outline

- C++ Classes
- Special Members
Classes

- Classes are an expanded version of data structures (structs)
  - Like structs, they hold data members
  - They also hold functions as members
  - Can specify access permissions also
- Defined with the keyword “class”:

```cpp
class class_name {
    accessSpecifier_1: member1;
    accessSpecifier_2: member2;
    ...
} object_names;
```
Classes

- **Access Specifiers:**
  - private
    - Only accessible from within members of the same class or from “friends”
  - protected
    - Private access plus members of derived classes can have access
  - public
    - Accessible anywhere the object is visible
  - Default access is private
Classes

• An example:

```cpp
class Rectangle {
    int width, height;
    public:
    void set_values (int, int);
    int area (void);
} rect;
```

• To access data and function members:

```cpp
rect.set_values (3, 4);
myarea = rect.area();
```
• The full Rectangle example:

```cpp
// classes example
#include <iostream>
using namespace std;

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

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

int main () {
    Rectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```

Classes

- The Scope Operator:

```cpp
class Rectangle {
    int width, height;
    public:
    void set_values (int,x, int,y);
    int area() {return width*height;}
};

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

- Used to define a member function of a class outside of the class definition
  - `area()` is defined within the class
  - `set_values()` is defined outside the class
- Scope operator (::) specifies the class to which the function belongs
Classes define a data type

- As always, we can have many objects of the class type
- And each object will have its own member variables and functions that operate on those variables
Constructors

- Used to create a new object of the class data type
- Initializes any member variables that need to be initialized
- May do other work if needed

- Unlike Python, the constructor function name is the same as the class name
- They cannot be called like regular member functions
  - They are only executed once – when the object is instantiated
- They have no return values – not even void
Classes

• Constructor Example

```cpp
// example: class constructor
#include <iostream>
using namespace std;

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

Rectangle::Rectangle (int a, int b) {
    width = a;
    height = b;
}

int main () {
    Rectangle rect (3,4);
    Rectangle rectb (5,6);
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```
rect area: 12
rectb area: 30
Classes

- Overloading constructors
  - Classes can have more than one constructor
    - All named the same, since they are constructors
    - But with different parameter lists
      - Remember talking about a method signature?
        - The name of the method, and its parameters and their data types
  - The default constructor is called when an object is declared but a constructor is not specified
    - This is different than a constructor with no parameters
  - An example would be appropriate here...
```cpp
// overloading class constructors
#include <iostream>
using namespace std;

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

Rectangle::Rectangle () {
    width = 5;
    height = 5;
}

Rectangle::Rectangle (int a, int b) {
    width = a;
    height = b;
}

int main () {
    Rectangle rect (3,4);
    Rectangle rectb;
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}

Rectangle rectb;  // ok, default constructor called
Rectangle rectc(); // oops, default constructor NOT called
rect area: 12
rectb area: 25
```
Classes

- Member initialization
  - C++ offers a clean way of initializing member variables

```cpp
Rectangle::Rectangle (int x, int y) { width=x; height=y; }
Rectangle::Rectangle (int x, int y) : width(x), height(y) { }
```
A subtle constructor example

```cpp
// member initialization
#include <iostream>
using namespace std;

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;}
};

int main () {
    Cylinder foo (10,20);
    cout << "foo's volume: " << foo.volume() << '
';
    return 0;
}
```

foo's volume: 6283.19
```cpp
// pointer to classes example
#include <iostream>
using namespace std;

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

int main() {
   Rectangle obj(3, 4);
   Rectangle * foo, * bar, * baz;
   foo = &obj;
   bar = new Rectangle(5, 6);
   baz = new Rectangle[2]{{2,5}, {3,6}};
   cout << "obj's area: " << obj.area() << 'n';
   cout << "*foo's area: " << foo->area() << 'n';
   cout << "*bar's area: " << bar->area() << 'n';
   cout << "baz[0]'s area: " << baz[0].area() << 'n';
   cout << "baz[1]'s area: " << baz[1].area() << 'n';
   delete bar;
   delete[] baz;
   return 0;
}
```

<table>
<thead>
<tr>
<th>expression</th>
<th>can be read as</th>
</tr>
</thead>
<tbody>
<tr>
<td>*x</td>
<td>pointed to by x</td>
</tr>
<tr>
<td>&amp;x</td>
<td>address of x</td>
</tr>
<tr>
<td>x.y</td>
<td>member y of object x</td>
</tr>
<tr>
<td>x-&gt;y</td>
<td>member y of object pointed to by x</td>
</tr>
<tr>
<td>(*x).y</td>
<td>member y of object pointed to by x (equivalent to the previous one)</td>
</tr>
<tr>
<td>x[0]</td>
<td>first object pointed to by x</td>
</tr>
<tr>
<td>x[1]</td>
<td>second object pointed to by x</td>
</tr>
<tr>
<td>x[n]</td>
<td>(n+1)th object pointed to by x</td>
</tr>
</tbody>
</table>
Summary

- C++ Classes
- Special Members