Polymorphism
Object-Orientated Programming (OOP) concept that a single object may take on the type of any of its base types.
- A Planet may polymorph itself to a Sphere
- A Sphere cannot polymorph to be a Planet (base types only)

Virtual
- The `virtual` keyword allows us to override the behavior of a class by its derived type.

<table>
<thead>
<tr>
<th>Sphere.cpp</th>
<th>Planet.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere::print_1() {</td>
<td>// No print_1() defined</td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Sphere&quot; &lt;&lt; endl;</td>
<td>// in Planet</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>Sphere::print_2() {</td>
<td>Planet::print_2() {</td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Sphere&quot; &lt;&lt; endl;</td>
<td>cout &lt;&lt; &quot;Earth&quot; &lt;&lt; endl;</td>
</tr>
<tr>
<td>}</td>
<td>// No print_3() defined</td>
</tr>
<tr>
<td>virtual Sphere::print_3() {</td>
<td>// in Planet</td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Sphere&quot; &lt;&lt; endl;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>virtual Sphere::print_4() {</td>
<td>Planet::print_4() {</td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Sphere&quot; &lt;&lt; endl;</td>
<td>cout &lt;&lt; &quot;Earth&quot; &lt;&lt; endl;</td>
</tr>
<tr>
<td>}</td>
<td>// No print_5() defined</td>
</tr>
<tr>
<td>// .h:</td>
<td>// in Planet</td>
</tr>
<tr>
<td>virtual Sphere::print_5() = 0;</td>
<td></td>
</tr>
</tbody>
</table>

Why?
Suppose you’re managing an animal shelter that adopts cats and dogs.

Option 1 – No Inheritance

```
animalShelter.cpp
1 Cat & AnimalShelter::adopt() { ... }
2 Dog & AnimalShelter::adopt() { ... }
3 ...
```

Option 2 – Inheritance

```
animalShelter.cpp
1 Animal & AnimalShelter::adopt() { ... }
```

Pure Virtual Methods
In Sphere, `print_5()` is a **pure virtual** method:

```
Sphere.h
1 virtual Sphere::print_5() = 0;
```

A pure virtual method does not have a definition and makes the class and **abstract class**.

Abstract Class:
1. [Requirement]:
2. [Syntax]:
3. [As a result]:

Abstract Class Animal
In our animal shelter, **Animal** is an abstract class:
Assignment Operator
I didn’t cover a few details of the assignment operator — let’s do that:

1. [Default Assignment Operator]
C++ generate a default assignment operator for simple classes:
   - No non-static const variables
   - No reference variables

2. [Self-Assignment]
   - Programmers are never perfect and are never optimal. Consider the following:
   
   ```cpp
   #include "Sphere.h"
   int main() {
      cs225::Sphere s(10);
      s = s;
      return 0;
   }
   ```
   - Ensure your assignment operator doesn’t self-destroy:
   
   ```cpp
   #include "Sphere.h"
   Sphere& Sphere::operator=(const Sphere &other) {
      if (&other != this) {
         _destroy();
         _copy(other);
      }
      return *this;
   }
   ```

Abstract Data Types (ADT):

<table>
<thead>
<tr>
<th>List ADT - Purpose</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

List Implementation
What types of List do we want?

Templated Functions:

```cpp
T maximum(T a, T b) {
   T result;
   result = (a > b) ? a : b;
   return result;
}
```

Templated Classes:

```cpp
#ifndef LIST_H
#define LIST_H

class List {
   public:
   
   private:
   
};
#endif
```

CS 225 – Things To Be Doing:

1. Theory Exam #1 – Today’s the final day of the exam.
2. MP2 due Jan. 12 (10 days), EC deadline in 3 days!
3. Lab Extra Credit → Attendance in your registered lab section!
4. Daily POTDs