Inheritance
In nearly all object-oriented languages (including C++), classes can be extended to build other classes. We call the class being extended the base class and the class inheriting the functionality the derived class.

<table>
<thead>
<tr>
<th>Shape.h</th>
<th>Square.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Shape {</td>
<td>#include &quot;Shape.h&quot;</td>
</tr>
<tr>
<td>public:</td>
<td>class Square : public Shape</td>
</tr>
<tr>
<td>Shape();</td>
<td>{</td>
</tr>
<tr>
<td>double getLength() const;</td>
<td></td>
</tr>
<tr>
<td>private:</td>
<td>public:</td>
</tr>
<tr>
<td>double length_;</td>
<td>double getArea() const;</td>
</tr>
<tr>
<td></td>
<td>private:</td>
</tr>
<tr>
<td></td>
<td>double length_;</td>
</tr>
</tbody>
</table>
|               |     // Nothing!         |}

In the code, Square is derived from the base class Shape:
- All public functionality of Shape is part of Square:

```
main.cpp
int main() {
    Square sq;
    sq.getLength(); // Returns 1, the len init'd
    // by Shape's default ctor
    ...
    ...
}
```

- [Private Members of Shape]:

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

Example:

<table>
<thead>
<tr>
<th>Cube.cpp</th>
<th>RubikCube.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube::print_1() {</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Cube&quot; &lt;&lt; endl;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>// No print_1()</td>
</tr>
<tr>
<td>Cube::print_2() {</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Cube&quot; &lt;&lt; endl;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>// No print_3()</td>
</tr>
<tr>
<td>virtual Cube::print_3() {</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Cube&quot; &lt;&lt; endl;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>// No print_3()</td>
</tr>
<tr>
<td>Cube::print_4() {</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; &quot;Cube&quot; &lt;&lt; endl;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>// No print_5()</td>
</tr>
<tr>
<td>Cube::print_5() = 0;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cube c;</th>
<th>RubikCube c;</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.print_1();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_2();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_3();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_4();</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_5();</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Polymorphism
Object-Orientated Programming (OOP) concept that a single object may take on the type of any of its base types.
- A RubikCube may polymorph itself to a Cube
- A Cube cannot polymorph to be a RubikCube (base types only)
**Pure Virtual Methods**
In `Cube`, `print_5()` is a **pure virtual** method:

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

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

---

**C++ Templates:**

1. 

2. 

3. 

**Templated Functions:**

```
functionTemplate1.cpp
1
2
3 T maximum(T a, T b) {
4    T result;
5    result = (a > b) ? a : b;
6    return result;
7 }
```

Where to put templated code?

---

**Templated Classes:**

```
List.h
1 #pragma once
2
3 class List {
4    public:
5
6    private:
7
8 }
9
10
11
12
13
14
15
```

```
List.hpp
1
2
3
4
5
```

---

**CS 225 – Things To Be Doing:**

1. mp_stickers due next Monday
2. lab_intro extended deadline Sunday
3. new lab released today also due Sunday
4. Daily POTDs