**Our First Class – Sphere:**

```cpp
# ifndef SPHERE_H
# define SPHERE_H

class Sphere {
public:
    double getRadius();

private:
};

# endif
```

```cpp
#include "sphere.h"

double Sphere::getRadius() {
    return r_;
}
```

**Public vs. Private:**

<table>
<thead>
<tr>
<th>Situation</th>
<th>Protection Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helper function used internally in Sphere</td>
<td></td>
</tr>
<tr>
<td>Variable containing data about the Sphere</td>
<td></td>
</tr>
<tr>
<td>Sphere functionality provided to client code</td>
<td></td>
</tr>
</tbody>
</table>

**Hierarchy in C++:**

There `Sphere` class we’re building might not be the only `Sphere` class. Large libraries in C++ are organized into `namespace cs225 {`.

**Our first Program:**

```cpp
int main() {
    cs225::Sphere s;
    std::cout << "Radius: " << s.getRadius() << std::endl;
    return 0;
}
```

Several things about C++ are revealed by our first program:

1. `main.cpp:4`
2. `main.cpp:5, main.cpp:1`
3. `main.cpp:6, main.cpp:2`
4. However, our program is unreliable. **Why?**

**Default Constructor:**

Every class in C++ has a constructor – even if you didn’t define one!

- Automatic Default Constructor:

- Custom Default Constructor:

```cpp
namespace cs225 {
    class Sphere {
    public:
        Sphere();
    private:
    };
}
```
Custom, Non-Default Constructors:
We can provide also create constructors that require parameters when initializing the variable:

<table>
<thead>
<tr>
<th>sphere.h</th>
<th>sphere.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 class Sphere { 3 Sphere::Sphere(double r) { 4</td>
<td></td>
</tr>
<tr>
<td>5 public: 5 4</td>
<td></td>
</tr>
<tr>
<td>6 Sphere(double r); 5 5</td>
<td></td>
</tr>
<tr>
<td>... 6 */ ... */ 6</td>
<td></td>
</tr>
</tbody>
</table>

Puzzle #1: How do we fix our first program?

<table>
<thead>
<tr>
<th>main.cpp w/ above custom constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Sphere s; 8</td>
</tr>
<tr>
<td>9 cout &lt;&lt; &quot;Radius: &quot; &lt;&lt; s.getRadius() &lt;&lt; endl; 9</td>
</tr>
</tbody>
</table>

...run this yourself: run make puzzle and ./puzzle in the lecture source code.

Solution #1:

Solution #2:

The beauty of programming is both solutions work! There’s no one right answer, both have advantages and disadvantages!

Reference Variable
A reference variable is an alias to an existing variable. Modifying the reference variable modifies the variable being aliased. Internally, a reference variable maps to the same memory as the variable being aliased:

<table>
<thead>
<tr>
<th>main-ref.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 int main() { 3</td>
</tr>
<tr>
<td>4 int i = 7; 4</td>
</tr>
<tr>
<td>5 int &amp; j = i; // j is an alias of i 6</td>
</tr>
<tr>
<td>7 j = 4; // j and i are both 4. 8</td>
</tr>
<tr>
<td>8 std::cout &lt;&lt; i &lt;&lt; &quot; &quot; &lt;&lt; j &lt;&lt; std::endl; 9</td>
</tr>
<tr>
<td>9 i = 2; // j and i are both 2. 10</td>
</tr>
<tr>
<td>10 std::cout &lt;&lt; i &lt;&lt; &quot; &quot; &lt;&lt; j &lt;&lt; std::endl; 11</td>
</tr>
<tr>
<td>11 return 0; 12</td>
</tr>
<tr>
<td>12 } 13</td>
</tr>
</tbody>
</table>

...run this yourself: run make main-ref and ./main-ref in the lecture source code.

Three things to note about reference variables:

1. __________________________
2. __________________________
3. __________________________

Pointers and References – Introduction
A major component of C++ that will be used throughout all of CS 225 is the use of references and pointers. References and pointers both:
* Are extremely power, but extremely dangerous
* Are a level of indirection via memory to the data.

As a level of indirection via memory to the data:

1. __________________________
2. __________________________

CS 225 – Things To Be Doing:

1. Sign up for “Exam 0” (starts Tuesday, Jan. 23rd)
2. Complete lab_intro; due Sunday, Jan. 21st
3. MP1 released today; due Monday, Jan. 29th
4. Visit Piazza and the course website often!