

Our First Class – Cube:

Cube.h		Cube.cpp	
1	#pragma once	1	#include "Cube.h"
2		2	
3	class Cube {	3	double Cube::getVolume() {
4	public:	4	
5	double getVolume();	5	
6		6	}
7		7	
8		8	
9		9	
10		10	
11	private:	11	
12		12	
13		13	
14		14	
15		15	
16	};	16	

Public vs. Private:

Situation	Protection Level
Cube functionality provided to client code	
Variable containing data about the Cube	
Helper function used in Cube	

Hierarchy in C++:

There **Cube** class we're building might not be the only **Cube** class. Large libraries in C++ are organized into _____.

Cube.h		Cube.cpp	
1	#pragma once	1	#include "Cube.h"
2		2	
3	namespace cs225 {	3	namespace cs225 {
4	class Cube {	4	double
5	public:	5	Cube::getVolume() {
6	double getVolume();	6	return length_ *
7		7	length_ * length_;
...		...	}

Our First Program:

main.cpp	
1	#include "Cube.h"
2	#include <iostream>
3	
4	int main() {
5	cs225::Cube c;
6	std::cout << "Volume: " << c.getVolume() << std::endl;
7	return 0;
8	}

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

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/Implicit Default Constructor:
- Custom Default Constructor:

Cube.h		Cube.cpp	
...		...	
4	class Cube {	3	Cube::Cube() {
5	public:	4	
6	Cube();	5	
...	/* ... */	6	}
...		...	

Custom, Non-Default Constructors:

We can provide also create constructors that require parameters when initializing the variable:

Cube.h		Cube.cpp	
...		...	
4	class Cube {	3	Cube::Cube(double length) {
5	public:	4	
6	Cube(double length);	5	
...	/* ... */	6	}
		...	

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

puzzle.cpp w/ above custom constructor	
...	
8	cs225::Cube c;
9	cout << "Volume: " << c.getVolume() << endl;
...	

...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!

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.
- Pointers are **level of indirection** via memory to our data.

As a level of indirection via memory to the data:

1. _____
2. _____

Often, we will have direct access to our object:

```
Cube c1; // A variable of type Cube
```

Occasionally, we have a reference or pointer to our data:

```
Cube & s1; // A reference variable of type Cube  
Cube * s1; // A pointer that points to a Cube
```

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:

main-ref.cpp	
3	int main() {
4	int i = 7;
5	int & j = i; // j is an <u>alias</u> of i
6	
7	j = 4; // j and i are both 4.
8	std::cout << i << " " << j << std::endl;
9	
10	i = 2; // j and i are both 2.
11	std::cout << i << " " << j << std::endl;
12	return 0;
13	}

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

Three things to note about reference variables:

1. _____
2. _____
3. _____

CS 225 – Things To Be Doing:

1. Sign up for “Exam o” (exam starts Thursday, Sept. 5th)
2. Attend lab and complete lab_intro; due Sunday, Sept. 1nd
3. MP1 released Friday; due Monday, Sept. 9th
4. Visit Piazza and the course website often!