

#8: Templates

2 5 September 14, 2018 · Wade Fagen-Ulmschneider

Assignment Operator – Self Destruction

• Programmers are sometimes not perfect Consider the following:

assignmentOpSelf.cpp		
1	<pre>#include "Cube.h"</pre>	
2		
3	<pre>int main() {</pre>	
4	cs225::Cube c(10);	
5	c = c;	
6	return 0;	
7	}	

• Ensure your assignment operator doesn't self-destroy:

	Cube.cpp		
1	<pre>#include "Cube.h"</pre>		
40	Cube& Cube::operator=(const Cube &other) {		
41	if (&other != this) {		
42	destroy();		
43	copy(other);		
44	}		
45	return *this;		
46	}		

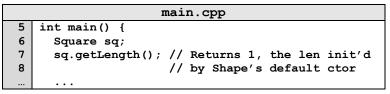
Inheritance

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

Shape.h		Square.h	
	class Shape {		<pre>#include "Shape.h"</pre>
	public:		
	Shape () ;		class Square : public Shape
	Shape(double length);		{
	<pre>double getLength() const;</pre>		public:
			<pre>double getArea() const;</pre>
	private:		_
	double length ;		private:
	};		<pre>// Nothing!</pre>
			};

In the above code, **square** is derived from the base class **shape**:

• All **<u>public</u>** functionality of **shape** is part of **square**:



• [Private Members of **Shape**]:

Virtual

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

Example:

Cube.cpp	RubikCul	be.cpp
<pre>Cube::print_1() { cout << "Cube" << endl; }</pre>	<pre>// No print_1()</pre>	
<pre>Cube::print_2() { cout << "Cube" << endl; }</pre>	RubikCube::prin cout << "Rubi }	
<pre>virtual Cube::print_3() { cout << "Cube" << endl; }</pre>	<pre>// No print_3()</pre>	
<pre>virtual Cube::print_4() { cout << "Cube" << endl; }</pre>	RubikCube::prin cout << "Rubi }	
<pre>// In .h file: virtual Cube::print_5() = 0;</pre>	RubikCube::prin cout << "Rubi }	

	Cube c;	RubikCube c;	RubikCube rc; Cube &c = rc;
c.print_1();			
c.print_2();			
c.print_3();			
c.print_4();			
c.print_5();			

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 can<u>not</u> polymorph to be a **RubikCube** (*base types only*)

Why Polymorphism? Suppose you're managing an animal shelter that adopts cats and dogs:

Option 1 – No Inheritance

	animalShelter.cpp		
ſ		Cat & AnimalShelter::adopt() { }	
	2	<pre>Dog & AnimalShelter::adopt() { }</pre>	
	3	•••	

Option 2 – Inheritance

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

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**.

Abstract Class:

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

Abstract Class Animal

In our animal shelter, Animal is an abstract class:

Abstract Data Types (ADT):

List ADT - Purpose	Function Definition

List Implementation

What types of List do we want?

Templates in C++

Two key ideas when using templates in C++:

1.

2.

1 2

3

4

5

6

Templated Functions:

functionTemplate1.cpp T maximum(T a, T b) { T result; result = (a > b) ? a : b; return result; }

CS 225 – Things To Be Doing:

- 1. Theory Exam #1 is ongoing; ensure you take it!
- 2. MP2 due Sept. 24 (10 days), EC deadline in 3 days!
- **3.** Lab Extra Credit \rightarrow Attendance in your registered lab section!
- **4.** Daily POTDs