CS 225

#8: Templates

September 10, 2021 · *G Carl Evans*

One Very Powerful Operator: Assignment Operator

Cube.h

```
Cube & operator=(const Cube & other);
```

Cube.cpp

```
Cube & Cube::operator=(const Cube & other) { ... }
```

Functionality Table:

	Copies an object	Destroys an object
Copy constructor		
Copy Assignment operator		
Destructor		

Assignment Operator – Self Destruction

- Programmers are sometimes not perfect Consider the following:

assignmentOpSelf.cpp

```
#include "Cube.h"

int main() {
  cs225::Cube c(10);
  c = c;
  return 0;
}
```

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

Cube.cpp

```
#include "Cube.h"
…
Cube& Cube::operator=(const Cube &other) {
  if (&other != this) {
    _destroy();
    _copy(other);
  }
  return *this;
}
```

The Rule of Three

If it is necessary to define any one of these three functions in a class, it will be necessary to define all three of these functions:

1.

2.

3.

The Rule of Zero

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**.

Shape.h

```
class Shape {
  public:
    Shape();
    Shape(double length);
    double getLength() const;

  private:
    double length_;
};
```

Square.h

```
#include "Shape.h"

class Square : public Shape
{
  public:
    double getArea() const;

  private:
    // Nothing!
};
```

In the above code, **`Square`** is derived from the base class **`Shape`**:

- All **<u>public</u>** 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:

Cube.cpp
```
Cube::print_1() {
  cout << "Cube" << endl;
}

Cube::print_2() {
  cout << "Cube" << endl;
}

virtual Cube::print_3() {
  cout << "Cube" << endl;
}

virtual Cube::print_4() {
  cout << "Cube" << endl;
}

// In .h file:
virtual print_5() = 0;
```

RubikCube.cpp
```
// No print_1()


RubikCube::print_2() {
  cout << "Rubik" << endl;
}

// No print_3()



RubikCube::print_4() {
  cout << "Rubik" << endl;
}

RubikCube::print_5() {
  cout << "Rubik" << endl;
}
```

	`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
```
Cat & AnimalShelter::adopt() { ... }
Dog & AnimalShelter::adopt() { ... }
...
```

Option 2 – Inheritance

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

Pure Virtual Methods

In **`Cube`**, **`print_5()`** is a **pure virtual** method:

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

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

CS 225 – Things To Be Doing:

1. mp_stickers EC deadline Sep. 13 (12 days).
2. Lab Extra Credit → Lab attendance is automatic this week.
3. Daily POTDs