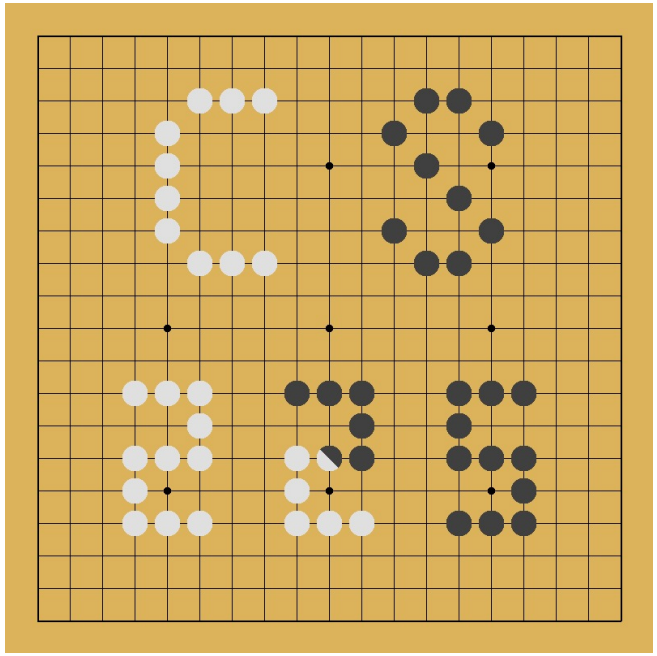
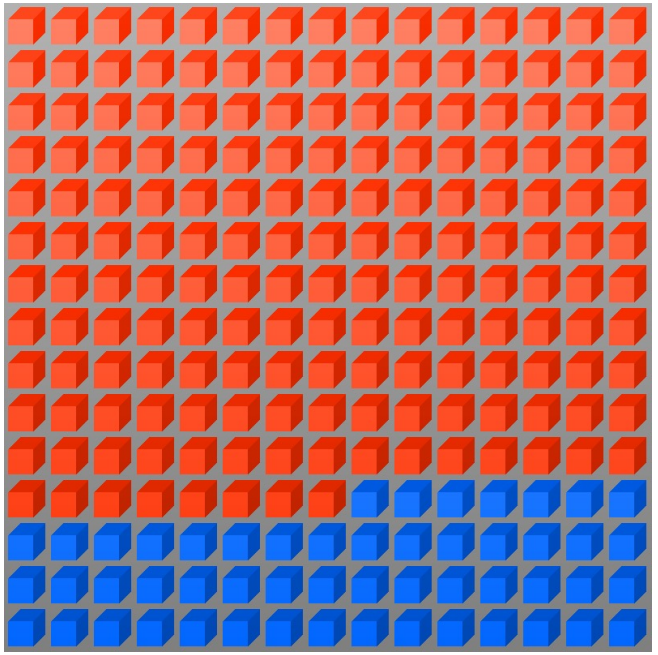




Data Structures

September 10 – Assignment and Inheritance

G Carl Evans

One Very Special Operator

Definition Syntax (.h):

```
Cube & operator=(const Cube& s)
```

Implementation Syntax (.cpp):

```
Cube & Cube::operator=(const Cube& s)
```

Assignment Operator

Similar to Copy Constructor:

Different from Copy Constructor:

Example:

assignmentOpSelf.cpp

```
1  #include "Cube.h"
2
3  int main() {
4    cs225::Cube c(10);
5    c = c;
6    return 0;
7  }
```

Example:

assignmentOpSelf.cpp

```
 1 #include "Cube.h"
 …
40 Cube& Cube::operator=(const Cube &other) {
41
42   _destroy();
43   _copy(other);
44
45   return *this;
46 }
```

Assignment Operator

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

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”

Corollary to Rule of Five

Classes that **declare** custom destructors, copy/move constructors or copy/move assignment operators should deal exclusively with ownership. Other classes should not **declare** custom destructors, copy/move constructors or copy/move assignment operators

–Scott Meyers

Inheritance

Square.h

```
 1  #pragma once
 2
 3  #include "Shape.h"
 4
 5  class Square : public Shape {
 6    public:
 7      double getArea() const;
 8
 9    private:
10      // Nothing!
11  };
```

Shape.h

```
 4  class Shape {
 5    public:
 6      Shape();
 7      Shape(double length);
 8      double getLength() const;
 9
10    private:
11      double length_;
12  };
```

Square.cpp

```
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
…
```

Derived Classes

[Public Members of the Base Class]:

main.cpp

```
5  int main() {
6    Square sq;
7    sq.getLength(); // Returns 1, the length init'd
8                    // by Shape's default ctor
…    ...
…  }
```

[Private Members of the Base Class]:

Polymorphism

The idea that a single interface my take multiple types or that a single symbol may be different types.

In Object-Orientated Programming (OOP) a key example is that a single object may take on the type of any of its base types.

Virtual

Cube.cpp

```
1  Cube::print_1() {
2    cout << "Cube" << endl;
3  }
4
5  Cube::print_2() {
6    cout << "Cube" << endl;
7  }
8
9  virtual Cube::print_3() {
10   cout << "Cube" << endl;
11 }
12
13 virtual Cube::print_4() {
14   cout << "Cube" << endl;
15 }
16
17 // In .h file:
18 virtual print_5() = 0;
19
20
21
22
```

RubikCube.cpp

```
1  // No print_1() in RubikCube.cpp
2
3
4
5  RubikCube::print_2() {
6    cout << "Rubik" << endl;
7  }
8
9  // No print_3() in RubikCube.cpp
10
11
12
13 RubikCube::print_4() {
14   cout << "Rubik" << endl;
15 }
16
17 RubikCube::print_5() {
18   cout << "Rubik" << endl;
19 }
20
21
22
```

Runtime of Virtual Functions

virtual-main.cpp	`Cube c;`	`RubikCube c;`	`RubikCube rc;` `Cube &c = rc;`
`c.print_1();`			
`c.print_2();`			
`c.print_3();`			
`c.print_4();`			
`c.print_5();`			

Why Polymorphism?

animalShelter.cpp

```
 1 class Animal {
 2   public:
 3     void speak() {                                    }
 4 };
 5
 6 class Dog : public Animal {
 7   public:
 8      void speak() {                                     }
 9 };
10
11 class Cat : public Animal {
12   public:
13      void speak() {                                     }
14 };
```

Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:

virtual-dtor.cpp

```
15 class Cube {
16   public:
17     ~Cube();
18 };
19
20 class RubikCube : public Cube {
21   public:
22     ~RubikCube();
23 };
```

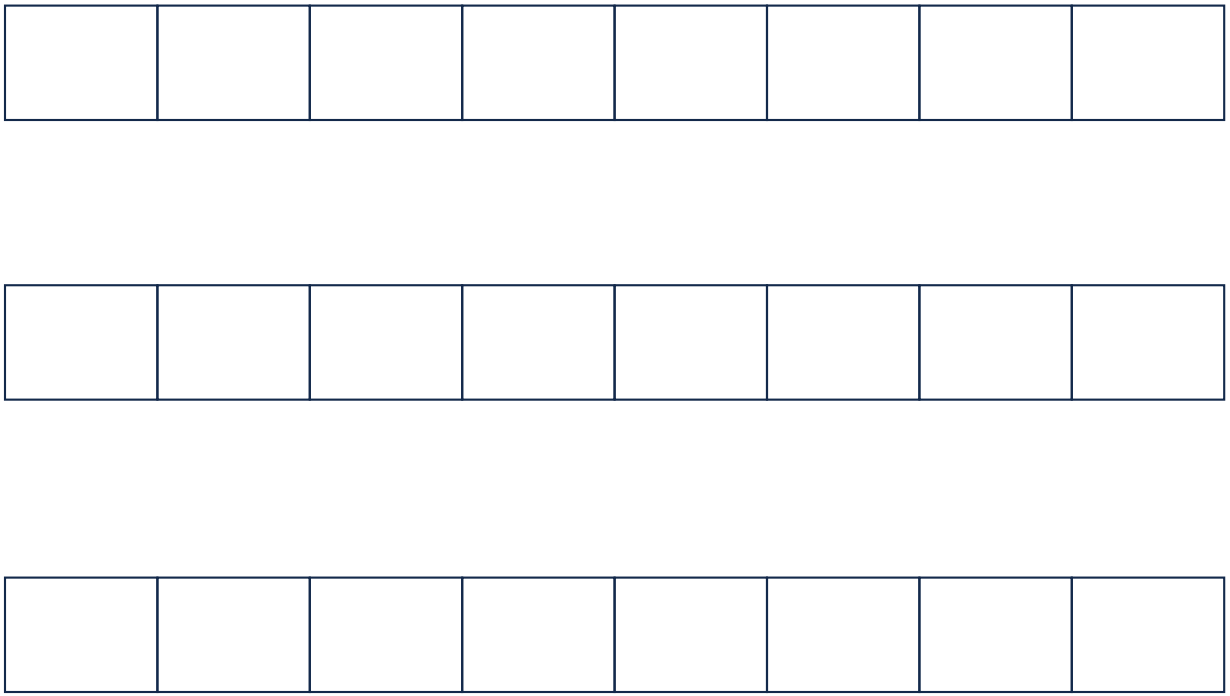
MP2: cs225/PNG.h

```
 18 class PNG {
 19   public:
 23     PNG();
 30     PNG(unsigned int width, unsigned int height);
 37     PNG(PNG const & other);
 43     ~PNG();

 50     PNG & operator= (PNG const & other);
 57     bool operator== (PNG const & other) const;

 73     bool readFromFile(string const & fileName);
 80     bool writeToFile(string const & fileName);
 90     HSLAPixel & getPixel(unsigned int x, unsigned int y) const;
 96     unsigned int width() const;
        // ...

118   private:
119     unsigned int width_;
120     unsigned int height_;
121     HSLAPixel *imageData_;
127     void _copy(PNG const & other);
132 };
```

Abstract Data Type

List ADT

What types of “stuff” do we want in our list?



Templates

template1.cpp

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

List.h

```
1  #pragma once
2
3
4
5  class List {
6    public:
7
8
9
10
11
12
13
14
15   private:
16
17
18
19 };
20
21 #endif
22
```

List.cpp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```