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:

```cpp
#include "Cube.h"

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

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

<table>
<thead>
<tr>
<th></th>
<th>Copies an object</th>
<th>Destroys an object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy constructor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy Assignment operator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
```cpp
#pragma once

#include "Shape.h"

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

    private:
        // Nothing!
};

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

    private:
        double length_;  
};
```
Derived Classes

[Public Members of the Base Class]:

```cpp
5 | int main() { 
6 | Square sq; 
7 | sq.getLength(); // Returns 1, the length init'd
8 | // by Shape’s default ctor
9 | ...
10 | ...
11 | }
```

[Private Members of the Base Class]:

Polymorphism

The idea that a single interface may 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

<table>
<thead>
<tr>
<th>virtual-main.cpp</th>
<th>Cube c;</th>
<th>RubikCube c;</th>
<th>RubikCube rc; Cube &amp;c = rc;</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.print_1();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_2();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_3();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_4();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_5();</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Polymorphism?
```cpp
class Animal {
    public:
    void speak() {
    }
};

class Dog : public Animal {
    public:
    void speak() {
    }
};

class Cat : public Animal {
    public:
    void speak() {
    }
};
```
Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:

class Cube { 
public: 
    ~Cube(); 
}; 

class RubikCube : public Cube { 
public: 
    ~RubikCube(); 
};
class PNG {
public:
    PNG();
    PNG(unsigned int width, unsigned int height);
    PNG(PNG const & other);
    ~PNG();

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

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

private:
    unsigned int width_; 
    unsigned int height_; 
    HSLAPixel * imageData_; 
    void _copy(PNG const & other); 
};
Abstract Data Type
List ADT
What types of “stuff” do we want in our list?
template1.cpp

```cpp
T maximum(T a, T b) {
    T result;
    result = (a > b) ? a : b;
    return result;
}
```
#pragma once

class List {
    public:
    
    private:
    
};