



## Data Structures

*February 7 – Inheritance and polymorphism*

*Jing Jiang*

# 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.

# 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 }
```

# Learning objectives

- Understand why inheritance is used
- Compare different ways of using polymorphic functions
- Distinguish the impact of “virtual” on destructor

# Learning objectives

- Understand why inheritance is used
- Compare different ways of using polymorphic functions
- Distinguish the impact of “virtual” on destructor



# 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]:

# Learning objectives

- Understand why inheritance is used
- Compare different ways of using polymorphic functions
- Distinguish the impact of “virtual” on destructor

# Polymorphism

*Object-Orientated Programming (OOP) concept 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

<u>virtual-main.cpp</u>	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]:**

# Learning objectives

- Understand why inheritance is used
- Compare different ways of using polymorphic functions
- Distinguish the impact of “virtual” on destructor

## virtual-dtor.cpp

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

```
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$ make
clang++ virtual-dtor.cpp -lm -o virtual-dtor
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$ ./virtual-dtor
Non-virtual dtor:
~Cube() invoked.
Virtual dtor:
~RubikCubeV() invoked.
~CubeV() invoked.
waf@siebl-2215-02:/mnt/c/Users/waf/Desktop/cs225/_lecture/09-linkedMemory$ □
```