CS 225 Spring 2019 :: TA Lecture Notes
2/1 Templates

By Wenjie

- **Assignment Operator**
  - Check if both cubes are the same one

```
1  Cube & Cube::operator=(const Cube & other){
2    if (this != &other) {      //If I'm not copying
3  myself
       _destroy();
       _copy(other);
     }
     return *this;
  };
```

- **Virtual**
  - Allow us to override the function in derived classes
  - Evaluate rules:
    - Check the type of the variable
    - Check the matching function in that type. If virtual, go to drive type (the actual instance type)
    - If function not found, check the base type, repeat

Cube.cpp

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

RubikCube.cpp

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

CS 225 Spring 2019 :: TA Lecture Notes
2/1 Templates

By Wenjie

16	`// pure virtual function`	16	}

main.cpp	**Cube c;**	**RubikCube c;**	**RubikCube rc; Cube *c = rc; (polymorphed into the base type Cube)**
c.print_1();	Cube	Cube	Cube
c.print_2();	Cube	Rubik	Cube
c.print_3();	Cube	Cube	Cube (go to drive type, didn't find 3, go back)
c.print_4();	Cube	Rubik	Rubik (drive type function evaluated because of **virtual**)
c.print_5();	**Error! Can't create an instance of an abstract class, does not compile**	Rubik	Rubik

- Pure virtual function
  - No implementation
  - Makes Cube an **abstract class**
  - Act as a placeholder function that every derived class must implement
- In a chain of inheritance, every class that was used as a base class need to have its functions **virtual**
  - Shape -> Cube -> RubikCube -> MyCube, then functions in Shape/Cube/RibukCube are going to be virtual, so they can be overridden

CS 225 Spring 2019 :: TA Lecture Notes
2/1 Templates

By Wenjie

- **Abstract Class**
  - Requirement: one or more pure virtual functions
  - Syntax: nothing - no abstract keyword in cpp
  - As a result: cannot create an instance of an abstract class

- **Virtual Destructor**
  - All destructors in base classes must be virtual
  - Destructors will call the base classes destructors

virtual-dtor.cpp

```
1   class Cube {
2     public:
3       ~Cube() { std::cout << "~Cube() invoked." <<
4   std::endl; }
5   };
6   class RubikCube : public Cube {
...   public:
28      ~RubikCube() { std::cout << "~RubikCube()
29  invoked." << std::endl; }
3   };
0   class CubeV {
31    public:
32      virtual ~CubeV() { std::cout << "~CubeV()
33  invoked." << std::endl; }
34  };
35  class RubikCubeV : public CubeV {
36    public:
        ~RubikCubeV() { std::cout << "~RubikCubeV()
    invoked." << std::endl; }
    };
    int main() {
        std::cout << "Non-virtual dtor:" <<
    std::endl;
        Cube *ptr = new RubikCube();
        delete ptr;
        std::cout << "Virtual dtor:" << std::endl;
        CubeV *ptrV = new RubikCubeV();
        delete ptrV;
```

CS 225 Spring 2019 :: TA Lecture Notes
2/1 Templates

By Wenjie

```
    return 0;
}
```

- In this case we have rubikcube dtor invoked first then cube dtor is invoked
- Abstract Data Type (ADT)
    - English definition / the basic operations of a data structure
    - ADT describes functionality but not implementation details

List ADT	Definition of Functionality
Create the empty list	Creates and empty list.
Add data to the list	Store data.
Get data from the list	Access data.
Remove data from the list	Remove data.
Check if a list is empty/size	How much data is in the list.

- **Templates: a dynamic data type**
    - Using “Template <typename T>” so that we do not need to write same function for various types
    - Template type are checked at compile time
        - maximum(3, 5): T = int
        - maximum(“world”, “hello”): T = string
        - maximum(cube(7), cube(42)) - but this may not complied since no > op defined
    - We can use other replace for T but using T is universally standard way

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