Class Design

March 23, 2015

Introduction

> Programs must be written for people to read, and only incidentally for machines to execute[1]
>
> It's hard to overstate the value of simple design and clear code[2]

Programming Ideals[3]

- Correctness
- Reliability
- Affordable
- Maintainable

[1] H. Abelson & G.J. Sussman
[2] Sutter & Alexandrescu 2005, §6
[3] Stroustrup 2009, §1.6

Class design

- Member function design
  - Function design
    - Return type choice
    - Argument type choice
    - Name choice
    - Error handling policy
  - Choise of modifiers: const, static
- Member variable type choice
- Choice of member variable modifiers: const, mutable, static, volatile
- Interface design

- The Big Four
- Class hierarchy
- Design Patterns

Example

```
class Complex {
public:

 Complex( double real, double imaginary = 0 )
    : _real(real), _imaginary(imaginary) {};

  void operator+ ( Complex other ) {
    _real = _real + other._real;
    _imaginary = _imaginary + other._imaginary;
  }

  void operator<<( ostream os ) {
    os << "(" << _real << "," << _imaginary << ")";
  }
```

```
  Complex operator++() {
    ++_real;
    return *this;
  }

  Complex operator++( int ) {
    Complex temp = *this;
    ++_real;
    return temp;
  }

private:
  double _real, _imaginary;
};
```

Conclusions

- x is either really an x or a short-living simple-use variable or coder is unaware of the literature
- x can both be positive or negative
- there is no value for x to initialize it with yet and coder is unaware of the literature
- x will have its value changed at least once

Example 2

What can be concluded from the following code?

```
unsigned int x;
```

Example 2

What can be concluded from the following code?

```
unsigned int n_countries = 27;
```

Conclusions

- n_countries is probably a number of countries
- n_countries is always positive
- coder is unaware of the literature or some complex code is coming
- n_countries will have its value changed at least once