Stroustrup
ch. 9
pp 299–336

9. Classes

```
class X {
    data members (fields)
    funct members (methods)
};
```

9.3 interface

- fields should be private or protected
- interface fns public
- helper fns priv/prot
- avoid bodies in interfaces
- used in header files

class - private by default
enum struct, union — public by default
use struct for convenience classes (nodes)
class for ADTs

9.4 implementation

members — ctors
— dtor (only one)
— member fns

this = current object
i.e. LHS argument in
p → f(x,y)
p.f(x,y)

Stroustrup ch.9

9.5 enumerations

```
enum color {RED, GREEN, BLUE};
```

(2)

defaults 0 1 2

```
enum code { FOO=8, BAR=19, BAZ=22 };
```

- don't use #defines if const or enum works.

9.6 operator overloading

```
color operator++ (color &c) { //++c
    c = c == BLUE ? RED : color(c+1);
    return c;
}
color operator++(color &c, int) { //c++
    color cr = c;
    ++c;
    return cr;
}
```

do not overload

```
&&
||
?:
,
```

Stroustrup
Ch. 9

9.7 Interfaces

principles: - complete
- as small as poss., but no smaller
- minimal
- ctors (2 basic + others)
- conversions ?
- copy ctor & op= (or prohibit)
- const member fns
- dtor frees resources
  - ∀ pointers

9.7.2 Copying

same

copy ctor T x(y) or T x=y;

op= x=y; ← not same

maybe finesse copy ctor:

```
foo::foo(const foo &that) {
    *this = that;
}
```

copy ctor used: pass value
return result
init var

9.7.3 ctors

default (no arg:) T x;
T *p = new T();
T *p = new T[n];

can't init arrays except w/ dot ctor
— use vector

NB: T x(); = fn decl not
x default ctor

Stroustrup
ch. 9

9.7.4 const mbr fns.

```
int foo() const;
```
↑ can't modify fields

"const correctness"

9.7.5 helper fns

- implement fn outside class if possible
- minimize class & corruption of invariants