Information Hiding

• Strategy for implementing abstractions so that programmer using abstraction does not have to know details
• What is necessary:
  – Record structure (struct)
  – Type definition
  – Functions on type
• Can be done with procedures

Encapsulation

• Prevent access to internal information of abstraction
• Requires language mechanism for enforcement
• Can simulate with separate compilation (*.h, *.c)
• Provided by classes in C++ (private)

Example: Dates

• Why bother abstracting dates?
• Scenarios of using dates:
  – Determine when to do self-test
  – Interest computation
  – Forced password update
• What kinds of operations on dates?

“Procedural” Dates

• Create Date abstraction using only
  – A struct
  – A typedef
  – Functions

Date “Objects”

• Create Date abstraction using a class

A C Struct

```c
struct EmpLabel {
  char name[NAME_SIZE];
  char id[ID_SIZE];
  short yearsExperience;
  int gender;
  unsigned char dependents;
  unsigned char exemptions;
} f;

typedef struct EmpLabel Employee;
```
A C++ Class

```cpp
class Employee {
    public:
        char name[NAME_SIZE];
        char id[ID_SIZE];
        short yearsExperience;
        int gender;
        unsigned char dependents;
        unsigned char exemptions;
    }
};
```

Good Abstractions

- Good abstractions are good abstractions whether implemented using structs/functions or classes
- Similar design problems in terms of choosing how to organize data and functions

Why use classes?

- More supportive mechanisms
  - Class interface
  - Constructors/destructors automatically called
  - Localized declaration
  - Encapsulation

How “Small” A Class?

- A class can hold any data
- Possible to define an Integer class
  - Objects hold ints – unnecessary abstraction
  - Methods often static – class becomes a module, or container of useful functions on ints
- General principle: don’t define class for unneeded abstraction