

CSE 143

Classes

[Chapter 3, pp. 125-131]

10/6/00 E-1

ADTs: Great Idea, but...

- How do we actually get modularity, abstraction, ADTs, black boxes, etc. in our programs?
- How do we actually encapsulate?
- Key programming construct: the **class**
 - New and major difference between C++ and C
 - Based on C struct.

10/6/00 E-2

Classes vs. Structs

- A lot like a C struct in syntax:

```
class GradeTranscript {  
    // Class member declarations  
};
```
- Two big enhancements to support encapsulation
 - Members (= components) can be functions not just data
 - Can specify *private* vs. *public* members
Restrict access to (hide) implementation details that are not part of the interface

10/6/00 E-3

A Bank Account Class

```
// Representation of a bank account  
class BankAccount {  
public:  
    // set account owner to given name  
    void init(string name);  
    // add amount to account balance  
    void deposit(double amount);  
    // = current account balance  
    double amount();  
private:  
    string owner;    // account holder's name  
    double balance;  // current account balance  
};
```

- Inside the BankAccount declaration, you can see variables (*data members*) and function prototypes (*member functions* or *methods*)
- Some members are *public*, some are *private*

10/6/00 E-4

Public vs. Private

- By default, "private" is assumed for classes
- Members in the private interface are hidden from clients.
 - The compiler will *not* allow client code to access them.
 - There's a "wall" around them
- Public members may be used directly by clients
 - Windows or holes through the wall
- For the BankAccount class,
 - How many data members? private? public?
 - How many "methods"?
 - What can the client use directly?

10/6/00 E-5

How Clients Use a Class

- A class is treated like any programmer-defined type. For example, you can:
 - Declare variables of that type:

```
BankAccount anAccount;
```
 - Can have arguments (parameters) of that type:

```
void doSomething (BankAccount anotherAccount);
```
 - Use one type to build other types:

```
class Bank {  
public:  
    ...  
private:  
    BankAccount accounts[100];  
};
```

10/6/00 E-6

Member Functions

- A class can contain member that are functions
- To call a member function, specify an object (class instance), select the function member with a '.', and append a parameter list

```
BankAccount anAccount;  
  
anAccount.init("Fred Flinstone");
```

Object Member function Parameter(s)

10/6/00 E-7

A Class is a Type

```
BankAccount a1, a2;
```

- The code above creates two instances of the BankAccount class.
- Each instance has its own copy of the data members of the class:

owner: "Jack" balance: 200.17	owner: "Jill" balance: 940.15
a1	a2

10/6/00 E-8

Operations on instances

- Most built-in operations DO NOT apply to class instances
- You cannot (for example):
 - use the "+" to add two BankAccount instances
 - use the "==" to compare two accounts for equality
- To the client, the only valid operations on instances are
 - assignment ("=")
 - member selection (".")
 - plus, can use any operations defined in the public interface of the class.

10/6/00 E-9

Terminology

- Think of a class as a cookie cutter, used to stamp out concrete **objects** (instances)
- Another view: objects as simple creatures that we communicate with via "messages." (function calls)
 - From the "object-oriented" programming world - we will use the terminology lightly

```
BankAccount myAccount;  
  
myAccount.deposit(300.15);
```

instance argument
receiver selection message

10/6/00 E-10

Information Hiding

- The *private* access modifier supports and enforces information hiding

```
// A client program . . .  
  
BankAccount account;  
  
account.balance = 10000.0; // NO! why?  
cout << account.balance; // NO! why?  
  
account.init("Jill"); // ok?  
account.deposit(40.0); // ok?  
cout << account.amount(); // ok?  
cout << account.amount; // ????  
cout << account; // ????
```

10/6/00 E-11

Class Packaging

- C++ allows many legal ways to "package" classes. In CSE143 we generally follow this pattern:
- For each class named X, a pair of files: X.cpp and X.h
- X.h (specification file)
 - the declaration of only one class X
 - maybe some constants
- X.cpp (implementation file)
 - #include "X.h"
 - contains all the member function definitions and any other functions needed to implement them
- Client programs have #include "X.h"
- Sometimes very closely related classes are packaged together

10/6/00 E-12

Interface as Contract

The public parts of a class declaration define the **interface** that clients can use.

Module interface acts as a contract between client and implementer

- Client depends on interface not changing
- Doesn't need to know any details of how module works, just what it does
- Implementer can change anything not in the interface, (e.g. to improve performance)
- Implementation is a "black box" (**encapsulation**), providing **information hiding**

10/6/00 E-13

Class Declaration: Interface

```
#ifndef BANKACCOUNT_H
#define BANKACCOUNT_H

// Representation of a bank account
class BankAccount {
public:
    // set account owner to given name
    void init(string name);
    // add amount to account balance
    void deposit(double amount);
    // = current account balance
    double amount();
private:
    string owner;    // account holder's name
    double balance; // current account balance
};

#endif
```

BankAccount.h

10/6/00 E-14

Building the Class: Implementation (Code)

```
#include "BankAccount.h"
// set account owner to given name
void BankAccount::init(string name) {
    balance = 0.0;
    owner = name;
}
// = current account balance
double BankAccount::amount() {
    return balance;
}
// add amount to account balance
void BankAccount::deposit(double amount) {
    balance = balance + amount;
}
```

BankAccount.cpp

10/6/00 E-15

Implementing Member Functions

- Implementations of member functions use **classname::** prefix
 - indicate which class the member belongs to
 - "**::**" is called the **scope resolution operator**
- Within member function body:
 - Refer to members directly
 - Can access any member, whether public or private!
 - Don't reuse class member names for formal parameters and local variables (bad style)

10/6/00 E-16

Declaration vs Definition

- In C++ (and C) there is a careful distinction between declaring and defining an item.
- **Declaration**: A specification that gives the information needed to use an item
 - function prototype
 - class declaration (specification in header file)
- **Definition**: The C++ construct that actually creates the item.
 - full function w/body

10/6/00 E-17

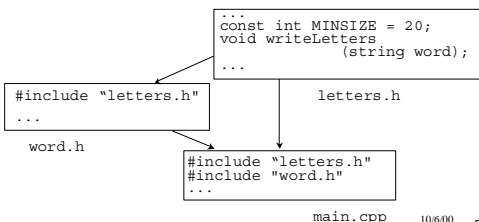
One-Definition Rule (ODR)

- An item (class, function, etc.) may be **declared** as many times as needed in a program (i.e., the same declaration may be #included in many files), but...
- An item must be **defined** (actually created or implemented) **exactly once** in a program.

10/6/00 E-18

Multiple Inclusion

Although an item may be declared in many different compilation units, it is a compile-time error if identifiers (function names, constants, etc.) are declared multiple times in one compilation unit:



10/6/00 E-19

Multiple Inclusion Hack

- To avoid this problem, use preprocessor directives:

```

// letters.h
#ifndef LETTERS_H
#define LETTERS_H
...
const int MINSIZE = 20;
void writeLetters (string word);
...
#endif
    
```

Preprocessor directive

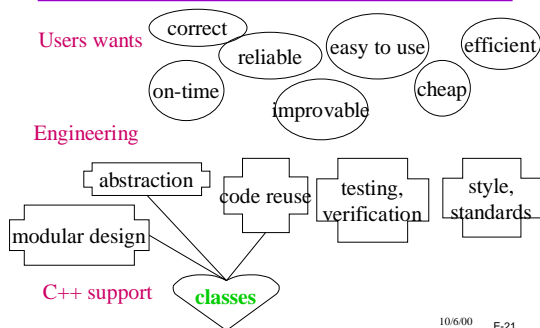
- Read the above as:

"If the symbol `LETTERS_H` has not been defined, compile the code through `#endif` (and define `LETTERS_H`), otherwise skip that code"

Effect: the header is only processed the first time it encountered (`#included`) when compiling a particular source file

10/6/00 E-20

Classes in the Big Picture



10/6/00 E-21

Summary

- class construct for Abstract Data Types
 - Function members (operations)
 - Data members (representation)
- public vs. private members
- Specification vs Implementation
 - Related concept: Declaration vs Definition
 - Implementation signaled by `classname::`
 - Implementations can access all members, public or private
- Clients generally have multiple instances of a few classes

10/6/00 E-22