COMSC-200 Lecture Topic 5
Friend Functions And Classes

Reference
Deitel Ch.10.3-10.4

Related Classes
when objects are used as data members...
textbook example
  `Employee` class has
    `Date` class members
options: member object, or...
  pointer/reference to an object

Forward Declaration in H Files
Example: `class X;`
minimum alternative to `#include`
  works for *pointers* and *refs* to objects
  not sufficient for *actual objects*
  or references to *class members*
place in `.h` above any reference to the class
avoids infinite reflection problem
or use in CPP in the case of single-file apps

Required Class Definition
Example: `#include "X.h";`
when there are member objects
Example: `vector`s of objects
not required for pointers or refs
required for references to *class members*

Initializer Lists
a constructor syntax
use to initialize member objects
if initialized in constructor body,
  garbage values assigned first
  results in double assignment
Example: `MyClass():x(0){}` instead of...
  `MyClass(){x = 0;}`
if member object's class contains `const`
  member variables, initialized in constructor body
  when used with the STL:
    requires `operator=` with `const_cast`
...so use the initializer list!

initializer lists for non-inline contructors:

```
class Time
{
  const int h, m, s;
  ...
  Time();
  ...
};

Time::Time():h(0), m(0), s(0)
```

Friend Classes
not exactly Java's `package`
allows access to `private` members (unlike Java)
declare friends in host class (unlike Java)
Example: `friend class X;`
  ...allows X to access class' private members

Friend Functions
allows stand-alone functions to access `private` members
  ...of a parameter list object
advantage over use of getters/setters:
  efficiency: less code
  performance: faster
declare friends in host class
Example: `friend void fun(X&);`
  ...allows `fun` to access class' private members

Friend Declarations
"member access notions are not relevant"
that is, not affected by public or private keywords
forward declaration or H include, still needed

`char` pointers vs. `char` arrays
`char a[] = "hello";` // mutable
`const char a[] = "hello";` // immutable
`char* a = "hello";` // be careful!
`const char* a = "hello";` // a reassignable
`const char* const a = "hello";` // immutable
`char* const a = "hello";` // be careful!

Solving The Shortest-Route Problem
the model: a network of interconnected nodes
nodes represent (e.g.) cities, project milestones,...
connections represent (e.g.) milage, cost, duration,...
  connections are one-way
objective: find shortest route from point A to point B
approach:
  model each connection
  build and compare routes
using recursion

The `Leg` Class
a journey of one leg
member variables:
  starting & ending cities; distance
using `const` : constructor with initializer list

The `Route` Class
a journey of one or more legs
encapsulates an array of `Leg`s
needs a output function to show journey

```
{
  ...
}
```

☐ **The `ShortestRoute` Class**
models a network with legs
includes recursive function for solution

Working With Projects
programs with multiple CPP files (IDE and command line)
Microsoft Visual C++
UNIX/Linux g++ (via WinScp)
Mac, MinGW g++, and Cygnus g++

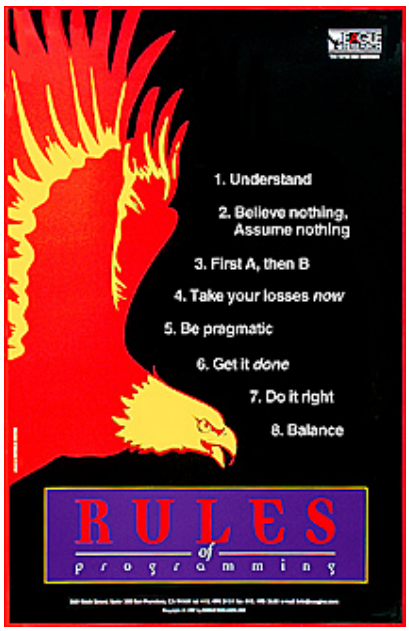
```
cl test1.cpp Date.cpp -EHs
...or... cl test2.cpp Employee.cpp Date.cpp -EHs
...or... cl Date.cpp -c -EHs

g++ test1.cpp Date.cpp
...or... g++ test2.cpp Employee.cpp Date.cpp
...or... g++ -c Date.cpp
```

Guidelines for Applying Forward Declaration vs. #include in Multi-File Projects:

1. A class' CPP file should always #include it's own H file. For example, Date.cpp should have a #include "Date.h" in it.
2. A class' H file (like Date.h) that refers to another class by name (like `Time`) should have either a forward declaration for that other class (like `class Time;` or `struct Time;`), or a `#include` for the other class' H file (like `#include "Time.h"`). For H files that contain only have references to the other class, either pointers (e.g., `Time*`) or reference variables (e.g. `Time&`), and refer to none of that class' members, use a forward declaration. But for H files that have another class' name without star or ampersand (e.g., `Time` or `vector<Time>`) and/or refers to members of the other class (like `Time::setHour`), the `#include` of the other class' H is necessary.
3. A class' CPP file that refers to another class by name should have either a forward declaration for that other class, or a `#include` for the other class' H file. Unlike a class' H file, you have *the option* of using a forward declaration if the CPP contains only pointers and references to the class, and no references to any of its members. There is no problem just using the `#include` the class' H file in any case, because there is not an infinite reflection issue -- CPPs are never #included.
4. Do NOT rely on the forward declaration or include used in the class' H file to be sufficient for the class' CPP. Every H or CPP that refers to a class by name should have its own forward declaration or #include, fully independent of any other H or CPP in the project.

In short, if your H contains only `Time*` and `Time&`, and does not call any Time member functions, use `class Time;` (or struct). Otherwise use `#include "Time.h"`. DO NOT put a #include in any of your H files if the requirements for using a forward declaration are satisfied. DO NOT reply on trial-and-error to determine if you made the right choice -- follow the rules!



The Rules Of Programming

- Understand
- Believe nothing; assume nothing
- First A, then B
- Take your losses now
- Be pragmatic
- Get it done
- Do it right
- Balance

Test driver example with
(1) ifndef testing,
(2) const object copy testing with assignment upon declaration, and

(3) object copy testing with assignment after declaration:

```
// TimeDriver.cpp, by Joe Student (1234567)
#include <iostream>
#include <string>
using namespace std;

#include "Time.h"
#include "Time.h" // testing ifndef (1)

int main()
{
  cout << "Lab 5, by Joe Student (1234567)\n";

  ...
  Time time(12, 0, 0);
  ...test all getters and setters

  // const object copy testing, with assignment UPON declaration (2)
  {
    const Time copy = time;
    ...use getters to confirm that copy's contents match time's
  }

  // object copy testing, with assignment AFTER declaration (3)
  {
    Time copy(0, 0, 0); copy = time; // using any available constructor
    ...use getters to confirm that copy's contents match time's
  }
}
```