C++ VIII

Header Files

What are they? Why have them?

- What are they?
  - Files that contain declarations, interfaces
    - An interface are the classes and functions without their implementations
  - They are included in cpp files with the # notation (see example next slide)
  - Their contents are copied into cpp or other source files that include them
- Why have them?
  - Each file is compiled separately: they don't know what is in other files, including headers allow other files to access the classes and functions defined in other files
  - They speed compile time: if all your code were in one file, you would need to compile your entire program every time you make a change
  - Keeps code more organized
  - Separation of interface from implementation

A Simple Example

- The compiler replaces the #include statements with the code in myclass.h
- main.cpp can now use MyClass
- Don’t include cpp files

```
1 // in myclass.h
2
3 class MyClass
4 {
5 public:
6   void foo();
7   int bar;
8 };
```

```
1 // in myclass.cpp
2 #include "myclass.h"
3
4 void MyClass::foo()
5 {
6 }
```

```
1  //in main.cpp
2  #include "myclass.h"
3
4  int main()
5  {
6    MyClass a;
7    return 0;
8  }
```

Using *include* in Header files

- What do you think will happen?
- Does this mean we should never use #include in our headers?

```
1 // x.h
2 class X { };
```

```
1 // a.h
2 #include "x.h"
3
4 class A { X x; };
```

```
1 // b.h
2 #include "x.h"
3
4 class B { X x; };
```

```
1 // main.cpp
2
3 #include "a.h"
4 #include "b.h"
```

Guards

- Use the #ifndef directive to check if the header has been defined (with the unique identifier you supply)
- If not, define it with the identifier
- Close the #if with an #endif
- According to this article: http://www.cplusplus.com/forum/articles/10627/, you should always do this

```
1 //x.h
2
3 #ifndef __X_H_INCLUDED__
4 #define __X_H_INCLUDED__
5
6 class X { };
7
8 #endif
```

The “right way” to include

When should you use an include, as opposed to a forward declaration for dependencies? That is, if class A has class B as a dependency:

- Forward declaration: declaration earlier in your cpp file
  - if A contains a B pointer or reference
  - one or more functions has a B object/pointer/reference as a parameter or return type
- Include:
  - if B is a parent class of A
  - A contains a B object

  Bottom line: do the least you can to avoid include-related hazards

from http://www.cplusplus.com/forum/articles/10627/

Header Structure

This header file illustrates the points made in the last slide

```
1  //=================================
2  // include guard
3  #ifndef __MYCLASS_H_INCLUDED__
4  #define __MYCLASS_H_INCLUDED__
5
6  //=================================
7  // forward declared dependencies
8  class Foo;
9  class Bar;
10
11 //=================================
12 // included dependencies
13 #include <vector>
14 #include "parent.h"
15
16 //=================================
17 // the actual class
18 class MyClass : public Parent
19 {
20 public:
21   std::vector<int> avector;
22   Foo* foo;
23   void Func(Bar& bar);
24
25   friend class MyFriend;
26
27 };
28
29 #endif // __MYCLASS_H_INCLUDED__
```

Why use `#include` inside header files?

Because objects in header files may themselves have dependencies: without using include in them, we would have to include their dependencies as well whenever they are used --- compare with on the right

Here is an example of why so-and-so's method is bad:

```
1 //example.cpp
2
3 //  I want to use MyClass
4 #include "myclass.h"
5    // ERROR 'Parent' undefined
```

so-and-so: "Hrm... okay...."

```
1 #include "parent.h"
2 #include "myclass.h"
3    // ERROR 'std::vector' undefined
```

```
1 #include "parent.h"
2 #include <vector>
3 #include "myclass.h"
4    // ERROR 'Support' undefined
```

so-and-so: "WTF? MyClass doesn't even *use* Support! But alright..."

```
1 #include "parent.h"
2 #include <vector>
3 #include "support.h"
4 #include "myclass.h"
5    // ERROR 'Support' undefined
```