CPP Tracing 7 (Header Files 1)

The compiler has **no** knowledge of what names are declared in other compilation units. That means that if you define a class or function or global variable, you must provide a declaration of that thing in each additional .cpp file that uses it. Each declaration of that thing must be exactly identical in all files. A slight inconsistency will cause errors, or unintended behavior, when the linker attempts to merge all the compilation units into a single program.

To minimize the potential for errors, **C++** has adopted the convention of using **header files** to contain declarations. You make the declarations in a header file, then use the #include directive in every .cpp file or other header file that requires that declaration. The **#include** directive inserts a copy of the **header file** directly into the .cpp file prior to compilation.

The following example shows a common way to declare a class and then use it in a different source file. We'll start with the header file, **my_class.h**. It contains a class definition, but note that the definition is incomplete; the member function do_something is not defined:

```
// my_class.h

namespace N

{

    class my_class
    {
    public:
        void do_something();
    };

}
```

Next, create an **implementation** file (typically with a .cpp or similar extension). We'll call the file my_class.cpp and provide a definition for the member declaration. We add an **#include** directive for "**my_class.h**" file in order to have the my_class declaration inserted at this point in the .cpp file :

```
// my_class.cpp
#include "my_class.h" // header in local directory
#include <iostream> // header in standard library

using namespace N;
using namespace std;

void my_class::do_something()
{
    cout << "Doing something!" << endl;
}
```

Now we can use my_class in another .cpp file:

```
// my_program.cpp
#include "my_class.h"
```

```
using namespace N;

int main()
{
    my_class mc;
    mc.do_something();
    return 0;
}
```