Header Files in C++

Introduction

- What is a header file?
  - It is a file that is included (like copying/pasting) in a C++ program and inserts statements into that program which will allow the programmer to use many of the facilities included in the C++ language itself
  - Header files can also be custom produced to meet your own needs
    - ex: Functions/Constants you may use in all of your programs not part of the standard C++ implementation - like Student.

Why we need them?

- Speeds up compile time
  - If everything is part of your program source code, it'll get VERY big. Having external libraries pre-compiled and then having include files point to them requires the compiler to only compile your changes to your own source code.
- Keeps your program code more organized
- Allows you to separate interface and implementation
  - ex: Function prototypes from Function definitions

Why we need them?

- Downside? Makes things a bit more complicated
- But, the complications are worth it for the benefits previously described.
- Note: Just do this slowly, and plan for your include design.

Example #1: The Student Structure

- Instead of defining your student structure directly in your main program, define it outside of your main program and include it. This way other programs can use the same definition without having to ‘redefine’ it.

```
// student.hpp
structure student {
    string lname;
    string fname;
    string major;
    int age;
    double gpa;
};
```

Example #1: The Student Structure

- Now, in your main.cpp program source

```
// main.cpp
#include “student.hpp”

int main() {
    student myStudent;
    myStudent.lname = “Komanetsky”;
    myStudent.fname = “Bill”;
    //etc.
}//main
```

The Include Statement

- The Include statement is basically a copy/paste type of operation
- It takes the contents of the included file and simply inserts or pastes it into the position in the program's source code where the include statement is located
- The include file is not compiled on it's own, it is inserted into your main source file and THAT is compiled
- Note that even though the include file is not compiled, it is 'pre-compiled' to assure all dependencies have been met

Types of Files

- For C programs
  - Source file: filename.c
  - Include file: filename.h

- For C++ programs
  - Source file: filename.cpp
  - Include file: filename.hpp (filename.h can also be used)

- Why different file extensions?
  - Depends upon the compiler

Include ifdef Statements

- Compilers are stupid. They don't know what has been included, if enough has been included, or if duplicates have been included. **Duplicates are a BIG problem**.

- If you have multiple include files which might even include other include files, then you may not be sure if an include file has already been included until you see strange 'Already Defined' errors in your compiled source program.

Include ifdef Statements

- This is where the ifdef statement comes into play
- Much like checking to see what operating system the compiler is running on, you can setup your own include variables to make sure your include file is included only one time.

```
//student.hpp
#ifndef VectorTest_student_h
#define VectorTest_student_h
structure student {
...
};
#endif
```

Include Dependencies

- Your included file may not be compiled on its own, but it must have all of the things it uses specified in the include file.
- Example: since ***string*** is part of the standard namespace, if you don't include the ***using namespace std*** statement, you will get a 'precompile' error.

```
// student.hpp
using namespace std;
structure student {
      string lname;
      string fname;
      string major;
      int age;
      double gpa;
};
```

Include Dependencies

- So, make your include file as self contained as possible. Include all dependencies it has inside of your included file. Doing this may even remove include dependencies in your main.cpp file!

- Some programmers say you should not have includes inside of an include file. This is poor programming because then, your main.cpp file must include everything your include files needs. So, if you need to know what your include file needs anyway, just put it in your include file!

What to Include in an Include File

- Object Oriented Programs needs include files and you should begin thinking in terms of objects and including those includes which contain those object definitions.
- So, for our student example, add everything that affects ONLY a student into the student.hpp file.
  - Look at all functions in your program. If a function accepts and/or returns ONLY a student (or some standard data types), then place that function in your include file.

What Should You Do for CS-1?

- Create your own include file, and call if student.hpp
- Place the Student structure in your include file
- Place the enumerated variables that the Student structure uses in your include file
- Delete the Structure and enumerated variables from your main program source code
- Place #include “student.hpp” in your main program source code

What's after simple Include Files?

- If you have program code (that includes functions) that will be used by many programs, but you don't want them to have to compile your functions (or you want them to be kept secret), then you can compile your include files into a library.

- This means creating a .cpp file that includes your functions, but no main function. Then, you compile the program into an object file or 'library'

- Now you have a library that you can add to your linker when you write your actual 'main' program. No Source Code!!

Static Libraries

- Here are the steps to creating a library to be used during ‘Link’ time.
  1. Create an include file which has your function prototypes defined. Example: myfunctions.hpp
  2. Create your source file for your functions and implement those functions. Example: myfunctions.cpp
  3. Compile (do not link) your static library.
- Now you can create your main C++ program, include the .hpp file (myfunctions.hpp), and then link your library.

```
// MathFuncsLib.h

namespace MathFuncs
{
	// Returns a + b
	static double Add(double a, double b);

	// Returns a - b
	static double Subtract(double a, double b);
}
```

```
// MathFuncsLib.cpp

#include "MathFuncsLib.h"

#include <stdexcept>

using namespace std;

namespace MathFuncs
{
    double MyMathFuncs::Add(double a, double b)
    {
        return a + b;
    }//Add

    double MyMathFuncs::Subtract(double a, double b)
    {
        return a - b;
    }//Subtract

}//namespace MathFuncs
```

```
// MyExecRefsLib.cpp

#include <iostream>

#include "MathFuncsLib.h"

using namespace std;

int main()
{
    double a = 7.4;
    int b = 99;

    cout << "a + b = " <<
        MathFuncs::MyMathFuncs::Add(a, b) << endl;
    cout << "a - b = " <<
        MathFuncs::MyMathFuncs::Subtract(a, b) << endl;

    return 0;
}
```