Queens College, CUNY, Department of Computer Science
Object-Oriented Programming in C++
CSCI 211/611
Summer 2018
Instructor: Dr. Sateesh Mane

due date: n/a

Homework: compilation and linking of project with multiple files

- **Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.**
- **The source of difficulty is the English (understanding the text).**
- **If you do not understand the words in the lectures or homework, THEN ASK.**
- **If you do not understand the concepts in the lectures or homework, THEN ASK.**
- **Send me an email, explain what you do not understand.**
- **Do not just keep quiet and then produce nonsense in exams.**

- **Consult your lab instructor for assistance.**
- **You may also contact me directly, but I cannot promise a prompt response.**
- Please submit your inquiry via email to `Sateesh.Mane@qc.cuny.edu`.
- **This homework assignment is an exercise in program compilation, for a project with multiple files.**
- **There is nothing to "calculate" in this assignment.**
- **We shall employ the questions from Homework 1 and compile the code in a specific way.**

1 Project with multiple source files

- In most real-life applications, a project is too large for all the code to fit into one file.
- Hence the source code is placed in multiple files.
- We shall learn how to compile a C++ project with multiple source files.
- In this section, we make some general observations.
- **If even a small part of the program code is edited, the entire project must be recompiled, if all the code is in one file.**
    1. This is time consuming, for a large project.
    2. If the code is in multiple source files, we only need to recompile those files which are affected by the change.
    3. The rest of the project need not be recompiled.
- **Multiple developers can work simultaneously on different source files.**
    1. If all the code is in one file, only one person can work on the file at a time.
    2. A large project typically has multiple software developrs, working on different parts of the code.
    3. If the code is in multiple source files, the individual developers can perform their work simultaneously.
    4. This asumes, of course, that the code changes by the individual developers do not affect each other. *This is a serious management problem for large projects.*
- **A source file can be included in multiple different projects.**
    1. Actually, we see this all the time without even thinking about it.
    2. The I/O and math libraries are included in almost every C++ project.
    3. We do not write a separate math library (for example) for every C++ project.
    4. The same concept applies to a source file of our own.
    5. The same file may be useful in many different projects.
    6. There is no need to copy and paste it for each project.
    7. Furthermore, if the file is upgraded, all the projects benefit from the upgrade.
    8. *Convsersely, if a bug is introduced into the file, all the projects are affected by the bug. This is also a serious management problem for large projects, especially when writing heavily used software libraries which will be used by many people.*
- **There is nothing "C++" in this assignment.**
- The entire assignment is about software organization, for a (possibly) large project.

2 Forward declarations

- Functions in one file may call functions defined in a different file.
- Clearly, this is a problem.
- To solve this problem, C++ supports **forward declarations.**
- A forward declaration is just the function signature.
- We place the forward declarations in **header files.**
- A header file is usually denoted using a ".h" suffix.
- For a project with multiple files of source code, each source file includes suitable header files which contain all the relevant forward declarations to compile the source code.
- The syntax to include a header file "header_filename.h" is as follows.

```
#include "header_filename.h"
```

- The overall set of project files consists of a set of source ".cpp" files and header ".h" files.

3 File names

- We shall employ the code in Homework 1 and separate the code into multiple files.
- There are eight files plus one file to hold the main program.
- We shall create files with the following names.
- **Header files** to contain the function declarations.

```
hw1_Q1.h
hw1_Q2.h
hw1_Q3.h
hw1_Q4.h
```

- **Source (cpp) files** to contain the function bodies

```
hw1_Q1.cpp
hw1_Q2.cpp
hw1_Q3.cpp
hw1_Q4.cpp
```

- **Source file with main program.**

```
hw1_main.cpp
```

- **Place all the files in the same file folder.**
- We shall go through the contents of the files in turn.

4 Header files

- The header files `hw1_Q[1,2,3,4].h` contain the function declarations, for each question.
- They are short files with only a few lines each.
- **hw1_Q1.h**

```
#include <string>
using namespace std;

void concat1(int n, string str[], string &s_out);
string concat2(int n, string str[]);
string& concat3(int n, string str[]);
```

- **hw1_Q2.h**

```
void swap1(int u, int v);
void swap2(int &u, int &v);
```

- **hw1_Q3.h** (*Only one line!*)

```
int RPM(int a, int b);
```

- **hw1_Q4.h**

```
#include <string>
using namespace std;

bool word_match(string s1, string s2);
```

5 hw1_Q1.cpp

- This file contains the code for the function bodies of `concat1`, `concat2` and `concat3`.
- Not all of the header files may be necessary, may be useful when performing debugging tests.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
#include "hw1_Q1.h"

using namespace std;

void concat1(int n, string str[], string &s_out)
{
    // etc
}

string concat2(int n, string str[])
{
    // etc
}

string& concat3(int n, string str[])
{
    // etc
}
```

6 hw1_Q2.cpp

- Here is the complete file.

```
#include "hw1_Q2.h"

void swap1(int u, int v)
{
  int t = u;
  u = v;
  v = t;
}

void swap2(int &u, int &v)
{
  int t = u;
  u = v;
  v = t;
}
```

7 hw1_Q3.cpp

- This file contains the code for the function body of `RPM`.
- Not all of the header files may be necessary, may be useful when performing debugging tests.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <cmath>
#include "hw1_Q3.h"

using namespace std;

int RPM(int a, int b)
{
    // etc
}
```

8 hw1_Q4.cpp

- This file contains the code for the function body of `word_match`.
- Not all of the header files may be necessary, may be useful when performing debugging tests.

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <cmath>
#include "hw1_Q4.h"

using namespace std;

bool word_match(string s1, string s2)
{
    // etc
}
```

9 hw1_main.cpp

- This file contains the code for the main program.
- The main program contains code to test our functions in `hw1_Q1.cpp`, `hw1_Q2.cpp`, `hw1_Q3.cpp` and `hw1_Q4.cpp`.
- Not all the system header files may be necessary, may be useful when performing debugging tests.
- **We include all four header files** `hw1_Q1.h`, `hw1_Q2.h`, `hw1_Q3.h` and `hw1_Q4.h`.

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <cmath>

#include "hw1_Q1.h"             // include all four header files
#include "hw1_Q3.h"
#include "hw1_Q2.h"
#include "hw1_Q4.h"

using namespace std;


int main()
{
    // code to test Q1, Q2, Q3, Q4

    return 0;
}
```

10 Compiling and linking

- The process of creating an executable file consists of two steps: **compiling** and **linking.**
- As the name suggests, **compiling** is the process of compiling the individual ".cpp" files.
  1. For each cpp file, the compiler produces a ".o" **object file.**
  2. An object file is not a human–readable file. We do not write or edit an object file.
  3. It is perfectly possible, and quite common in real-life projects, to compile only one file and not the entire project.
  4. **The compiler can compile an individual file without needing to know the source code for the whole project.**
  5. The compiler only needs to know the source code in the file being compiled, plus relevant information in the included header files.
- **Linking** is the process of pulling together (or "linking") all the object files, plus the system libraries (such as I/O and math) to produce the final executable file.
  1. Linking is performed by the **linker**.
  2. The linker is a separate program from the compiler.
  3. **The linker must know all the object files of the whole project.**
  4. **The linker must also know all the relevant system libraries which are required by the project.**
  5. *The linker searches through all the object files and system libraries to verify that the compiled code of every function used in the project is found in the list of files.*
  6. The linker issues an error if it cannot find the compiled code for a function, e.g. if we forget to include the name of an object file in the list.
  7. *The linker also generates an error if it finds duplicate compiled code,* e.g. if we accidentally write the code for the same function in two different files of the project. This sometimes happens.
  8. These are is not the same as compilation errors, but nevertheless it means ab executable file will not be built.
  9. If all goes well, the linker produces an executable file.
  10. The executable file is what we actually run.

11 Console commands to compile and link a project

- We shall describe the commands gor the `g++` compiler.
- **Put all the .cpp and .h source and header files in one folder.**
- (It is possible to store the files in multiple folders but this simply creates unnecesary complications.)
- To compile, type the following command into the console and hit `enter`.

```
g++ -c hw1_main.cpp hw1_Q1.cpp hw1_Q2.cpp hw1_Q3.cpp hw1_Q4.cpp
```

- If there are any compilation errors, the compiler will report the errors.
- If there are no errors, the compiler will produce one ".o" object file for each .cpp file.
- To link, type the following command into the console and hit `enter`.

```
g++ -o a.exe hw1_main.o hw1_Q1.o hw1_Q2.o hw1_Q3.o hw1_Q4.o
```

- The file "a.exe" is the name of the executable file.
- You can substitute any other name, e.g. "a.out" is typical on a Unix system.
- If there are any linker errors, the linker will report the errors.
- If there are no errors, the linker will produce the executable file with the name specified in the command line.
- You can run the executable file.
- **Suppose we edit the main program and nothing else.**
  1. We do not need to recompile everything.
  2. Type only the following, to compile only one cpp file.

     ```
     g++ -c hw1_main.cpp
     ```

  3. **To link, we must supply all the object files.**
  4. The link command is the same as before.

     ```
     g++ -o a.exe hw1_main.o hw1_Q1.o hw1_Q2.o hw1_Q3.o hw1_Q4.o
     ```

  5. This will produce a new executable file.
- **Hence we create a fresh executable file without recompiling the entire project.**

12 Different vendors

- There is a weak point in the above procedure, which is that different vendors have different commands for compilation and linking.
- Many vendors have graphical user interfaces and do not employ console commands.
- The commands in Sec. 11 are only for the `g++` compiler.
- **The essential lesson of this assignment is the concept of splitting a project into multiple files, and why this is a good idea.**
- The mechanics of *how* the compile and link steps are performed depends on the vendor.