Last updated 08 January, 2017

Assignment 1 (100 points possible):

Please see Submission Checklist (below) for submission requirements.

It is time to bring together some concepts we have been learning!

(20 points) Remember to create a weekly report!

1. Explain what the weekly project is asking you to do in English, pseudocode, and drawings.

2. Explain, in your own words, what the overall lessons for the week were:

   So, if you had demos or were asked to write the following:
   write a text editor that does x, y, and z,
   write a number guessing game for two players that requires options a, b, and c,
   create a program that takes a file as input, processes it with i and j, and then outputs it to another file,
   …

   Then you might say that the lessons of the week were about learning about input and output.

   Note: This can be a single sentence if the week had one strongly related topic, or it could be a few sentences describing a couple topics and how they might relate to each other.
   "The exercises this week are all related to conditional statements and the project is about using if and case statements to simulate a number guessing game"

3. Design: describe or draw out how the program should behave.
   3.1. Use pseudocode, flowcharts, drawings, or explanations. This could be a combination of techniques.
   3.2. Describe the overall tasks and sub-tasks of your program.
   3.3. This design should at least take into account any input, any processing, and any output that you think you will need to solve the assignment requirements.

4. Testing: Design and describe some tests to verify that your code would be working properly. Perhaps create a table (input, expected output, and actual output) describing the tests that you plan to perform to demonstrate that your program meets the assignment requirements.

   Example (how is this program supposed to behave?):

input	expected output	actual output
0	0	0
1	1	1
2	4	4
100	10,000	10,000
-1	1	-1 (something seems wrong here, time to investigate the code!)

5. Implementation: this will generally be the code you use to achieve your assignment requirements, but may sometimes include provided files or other files you create.

   Note: You do not include the implementation in your report, it plays a much greater role in your assignment grade.

6. Now test each of the cases you came up with in your testing section and note any unexpected results or fixes in your testing or reflection sections.

   Note: modifying designs and tests for your implementation is okay, but make sure you take note and think carefully during the implementation process so that you can include whatever changes you make in your reflection section and improve your later design and test creation processes.

   If you fail any test cases, then you can go back in your code and fix any errors and test it again, but you should note this anytime it happens so you can mention some of it in your reflection.

7. Reflection: now that you are done with your program (even if the program is not complete!) you should discuss the process. You should mention things like:
    7.1. Was your understanding complete and design adequate at the start of the project or did you learn something about the problem as you went. What details might someone have missed during the understanding process that the implementation and testing process reveal?
    7.2. Did all your tests work out the way you expected or did you have to alter your tests because of some design, implementation, or testing details?
    7.3. Did implementation go without any problems, were there details that were difficult to get working the way you wanted them to?
    7.4. What details might someone miss during the implementation process?
    7.5. What techniques helped you approach the problem, from class and from outside class? Does this project seem related to previous projects and do you see any names for future projects that it might be related to?

(20 points) Exercises components

Exercises for this assignment (this assignment is focusing on tools and access to the systems we will be using for this class):

1. Try to form a group of people that you can ask questions of or study with,

   (**Note**: quick responding study groups are a huge benefit in a class like this where little details can take lots of time if you do not have someone to bounce ideas off of or have look at your code with fresh eyes).

2. (participation) Consider viewing videos on some of the tools for the course, especially videos on:
    2.1. how to edit files with a command line text editor such as Vim (try vimtutor or try nano (another text editor) if you cannot stand the shortcuts and bimodal editing in Vi),

       (**Note**: you can choose to use a graphical editor, and be sure to bring questions to class!)

    2.2. how to compile programs with gcc,

       (**Note**: again, you can choose to use a graphical editor, which likely has compiling and execution shortcuts built in)

    2.3. or some videos or documents on C programming (perhaps through thenewboston.com, tutorialspoint.com, or cprogramming.com)

3. (10 points) Draw a picture showing the relationships between the computers you might need, the programs you might need, where those programs are located, and where those programs affect things for your assignment (show us your understanding of where components in your programming process are located and what they affect).

   File must be called: systems.pdf
   Not sure how to create a PDF? Check in your document editor for save and export options.

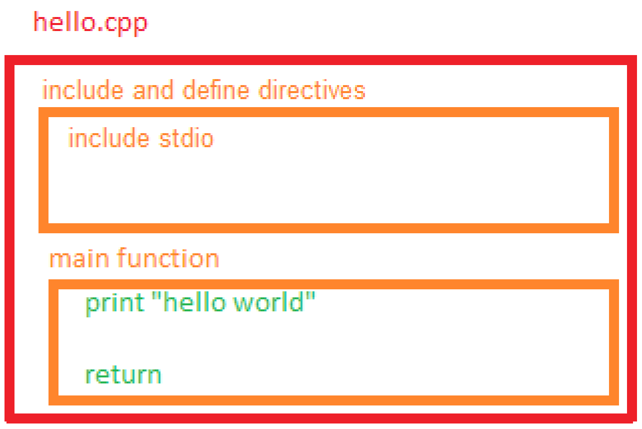
4. (05 points) Before beginning any coding on the project, draw or write out approximately what the following program will look and act like. Be sure to show that you understand the various aspects of the problem, the solution, and that you will be able to test your solution (think of your tests as you are designing your solution. This is called test driven development and it works wonders for writing complete, correct, and maintainable programs).

   File must be called: helloDesign.pdf

   In this case the design will be for a *hello world* program (just needs to print hello world).

   (**Note**: You will normally make these design elements for projects but this week it will be an exercise component as well)

4.1. An example design might be like this: draw a box to represent the whole program, draw a smaller box for the "main" function, and fill in the main box with the steps of the program (this drawing will not feel super useful right now, but as we add more capabilities or move functionality out into functions I will ask you to draw more details in your boxes). For the hello world program this will just be something like this (I will try to color code the depth of nested scope):



5. (05 points) Write, compile, and execute a simple hello world program.

File must be called: hello.c

5.1. Your program may look something like this when you are done coding it:
File: hello.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
}
```

5.2. Use **gcc** to compile your hello world program, you can use the following while in the same directory as your source file to compile the source file into an executable file:

```
gcc [-o <output file name>] <source file name>
```

(**Note**: angle brackets denote a required argument, so **<source file name>** could be **hello.c** and **<output file name>** could be **hello**)

(**Note**: square brackets denote an optional argument, you could drop the **-o <output file name>**, and get **a.out** or **a.exe** as the default filename on many systems)

(**Note very well!**: be careful when compiling your files, your **-o** option should not be the same as the source file (if I write **hello.c** for my **-o** option, then I will overwrite my source file with the executable file and lose all my source code in the process))

(**Note**: if there is no output from gcc, then it probably executed without error, check for your output file in the directory with **ls** on linux or **dir** on windows)

Discussion ideas: if there are compiler errors, be sure to check that your code is correct, but also ask about these in class! Remember to write down the error text and any code you think is related (this is a skill to learn too, so do not feel bad if I ask you to include more or less related code).

5.3. Run your hello world program, I would run mine from the directory with the executable with the command **./hello**, or **hello** (depending on your system)

(**Note**: a common way to run executables on a Unix-like system is to type **./<executable file name>**; that is to say, a dot (.), then a slash (/), then the executable file name, so to run my program called hello, I would type: **./hello**

(**Note very well!**: **you should compile and run every line or few lines that you write, which can easily save you debugging time later!**)

5.4. **Discussion ideas**:
- 5.4.1. What exactly is a source file?
- 5.4.2. What is an executable (also called a binary)?
- 5.4.3. What is a process?
- 5.4.4. How do the above differ from each other?
- 5.4.5. Why might I ask you to use an old command line editor to write some source code?

6. **(participation)** Discuss what your report sections looked like (do not worry if you think your work is not very useful right now, this is something we need to learn as well!), how you went from problem description to design to program, what you did to make sure your program was working correctly (you really do need to test each part of your program both separately and as a whole), and whether anybody else has shown any work that could improve your code (this **could** be as easy as copying a portion of their code, or it may be a bit of transformation before using it!).

7. Submit your personal **assignment report** (in pdf format), and **source files** (not executable files, I want the .c files!) before the end of Monday:

   (**Note very well**: late work will not be accepted, so be sure things are in before the due date)

   7.1. Double check that the files you submitted were the report and source files for this assignment just before you submit them!

8. Remember to discuss the design, implementation, and testing process on the discussion boards, these skills will be **very** important later on!

(60 points) Project components (data types and printing).

1. Using whatever means you like, write source code for a program that will create a variable of each of the common primitive data types and print it out using the printf function.

   File must be called: dataTypes.c

   Note: this will likely need at a minimum:

   1.1. a source file called **dataTypes.c**,

   1.2. **#include <stdio.h>** (no need to understand all about this part yet, just be aware that it allows us access to some input and output related capabilities),

   1.3. have a **main** function, similar to the one we used for the hello world program, which looks like:

```
int main()
{
    //code goes here

    return 0;
}
```

1.4. a variable for each primitive data type (you will need to look up what each of the primitive data types are),

1.5. print an indicator message for each variable to the user so that they know what is being printed, (something like: **printf("The value of myShort is: %d", myShort);**)

(N**ote:** you should always remember to include indicator messages for users so that they know what is happening in the program, especially when you have data being printed)

1.6. **Discussion ideas** (discuss other similar things if these have been brought up already!):

1.6.1. You will likely use a short, int, or long data type for this program. In what situations might you use each of these data types over the others?

1.6.2. You are likely to use **printf** with some format specifiers. What specifier do you need for each of these statements to work the way you want, do they work interchangeably? Does using different format specifiers change how your data is displayed?

Remember to submit your report (in PDF) and source files (in .c format) before the end of Monday!

Any interesting discussion items that come up should be included in your report and shared in class (outside of class too!).

Submission Checklist:

Report, ***in PDF format***, must address these four sections (20 pts):
(**Note**: recall that these can be just a couple sentences each, and should be from the point of view of the whole assignment; not just the project and it is not necessary to do this individually for each exercise)

Understanding
Design
Testing
Reflection

(**Note**: The implementation part will be the .c files you submit and does not need to be in the report)

Exercises (20 pts)
systems.pdf
helloDesign.pdf
hello.c
(parts of the report on design drawing and software relationships)

Project (60 pts)
dataTypes.c