Important things to remember:

- No lines over 80 characters! If you have a line that is 81+ characters, break it into two lines. Be very careful when writing comments as those tend to be where students get deducted.
- No tabs in C files! Instead, use 2 spaces. The vimrc file we provide is set to expand tabs in C files to be 2 spaces. I highly recommend using this.
- Use tabs in Assembly files. One tab should be equal to 8 spaces. If you use too many tabs your lines could appear longer than 80 characters.
- No Magic Numbers! Any number other than 0, 1, and -1 are counted as magic numbers. In the event you need to use a number (it happens a lot), define a constant at the top of your file.
  IMPORTANT: Give your constants meaningful names. Simply saying `#define TWO 2` is not good enough. Name constants in such a way that someone can reasonably guess what the value is used for. It's best to exclude magic numbers to ensure that your code is both more easily maintained and more readable.
- In addition to the above, use reasonable names for your variables. This greatly helps readability and allows others to more easily follow your code.
- Make sure to send any error output to stderr using fprintf. Normal output usually will go to stdout unless the assignment specifies otherwise.
- Stick to the Assembly looping construct covered in class (if this doesn't make sense it means it hasn't been covered yet). We will be checking for this and failure to follow our directions will result in a deduction.
- Whitespace is your friend! Separate logical blocks of code with a blank line in between. It is a lot harder to debug a wall of text than it is to work with something that is broken up. However, don't go overboard. You don't need to have a newline between every two unrelated lines of code, and definitely don't use too many at once.
- Remove any and all commented out code. Leave only what is relevant.

File Headers:

Every file needs a file header (yes, even the README). This header needs to have the format of:

```
/*
 * Filename: <filename.extension>
 * Author: <Your name>
 * UserId: <Your cs30x account>
 * Description: <A description of what the file is used for, i.e. usually what
 *               the functions defined within are used for>
 * Date: <Date when file is created or last modified date>
 * Sources of help: <Any sources of help you used, be they tutors or Piazza
 *                   posts. Other students should not be listed here as other
 *                   students should not help/look at your code>
 */
```

Method Headers:

We have two different type of method headers. Those for C functions and those for Assembly routines. We are looking for the headers to be EXACTLY in the format we describe.

C Function Header

```
/*
 * Function Name: <function name>()
 * Function Prototype: <Definition of the function> for example, for main this
 *                      would be int main( int argc, char* argv[ ] );
 * Description: <Description of how the function behaves>
 * Parameters: <Name any parameters passed into the function, and how they are
 *              used. If no parameters are used, say None>
 * Side Effects: <Any behaviors the function might exhibit that are not
 *                immediately apparent (related to the return value). Examples
 *                include updating a value pointed to by a parameter, or
 *                printing things to stdout/stderr. If there are no side
 *                effects, say None>
 * Error Conditions: <Explain any potential errors/exceptions that may occur if
 *                    your function is used incorrectly. If there are no error
 *                    conditions, say None>
 * Return Value: <What does the return value of this function represent/what
 *                will it be used for?>
 */
```

Assembly Routine Header

Assembly routine headers require everything that the C Headers do, in addition to listing any registers used.

```
/*
 * <Full C Header>
 *
 * Registers used:
 *     <register> - <use> -- <description of what the value represents>
 * example:
 *     r0 - arg 1 -- number to be squared
 *     r1 - local variable -- used to keep track of loop index
 */
```

Commenting:

Well documented code is good code. Commenting is a skill that you will develop in time, but here are some guidelines to help jump start the process.

Assembly Files

For Assembly files, you'll want to comment almost every line. Instructions can be pretty cryptic and it is easy to forget what you were intending on doing so comment as you go. **If your Assembly is not commented, tutors will NOT help you in the lab.** Sometimes it can also be helpful to include the C code you are translating in your comments.

Use inline comments after your instructions, marked with **@**

```
mov   r5, r0          @ Copies the input parameter into register 5.
add   r6, r5, r1      @ Adds r5 with the second argument.
```

Make sure to align all your inline comments vertically.

C Files

For C files, comments can be more infrequent. You don't need to comment every line, but you'll need to comment logical blocks. If there are a few lines of code that work towards the same goal, start the block off with a comment explaining what will happen in the next few lines.

Keep your commenting style consistent! It is alright to use either inline comments, comments before the code, or block comments. But be consistent! Use one commenting style throughout your program.

Failure to follow anything listed in this document can (and will) result in deductions from your programming assignments!