Programming Style Guide

NOTE: I will be doing style grading based on these guidelines for all assignments for the rest of the semester. Style will account for 30% of your grade -- note, though, that adhering to the style guidelines will make writing robust code significantly easier. Items in bold are particularly important.

Programming Guidelines:

Functions:

- Care should be taken to try to keep functions to less than the height of a single screen (usually 30-35 lines). **If a function is more than 35 lines, you should consider trying to break the function into smaller functions.**
  - Note: this is not a strict rule. Sometimes a function will have to be more than 35 lines. However, you should try to keep functions as short and concise as possible.
- **If the exact same lines of code are repeated twice or more in a program, that code should be moved into a function**
  - Note: single line statements such as returns or variable assignments are excluded from this rule

Variable declarations:

- Local variables should ideally be declared at the top of a function
- Instance variables should be declared at the top of a class/struct
- A variable should always be given an initial value
- **If a number, word, or piece of data is declared in more than two places in a function or group of functions then it should be made into a variable**
  - **Note: if this piece of data will never change then the variable should be made constant.** If a constant is used in more than one function, it should be declared as a global constant (after preprocessor includes, but before all functions).

White space (spaces, indents, and new lines):

- White space must be used to make code more readable
- **Code inside a block should be indented. This includes single line scopes**
  - Note: see last page for an example of program scope
- (not a strict rule) Declarations of local variables should be separated from the body of a function by a new line when possible.

Comments:

- Comments are used in order to inform programmers what the code is doing
- Comments that span more than three lines should use the multi line comment /* */ rather than the single-line // comment.
- All functions other than main() must have a comment above them explaining what the function does and its role in the program. **Note:** You do not have to include a comment above functions within a class so long as the name of the function clearly describes what it does.
- **There must be a multi-line comment at the top of the file containing main() consisting of your name, the question the code responds to, and a description of what the program does.**
- **For files declaring classes (.h files) include a multi-line comment at the top of the file containing your name, the question the code responds to, and a description of the role of the class in your program.**
- You should do their best to convey what a piece of code is doing by using good variable and function names, but additional comments should be added when a piece of codes function is ambiguous or when a piece of code must perform a complex task

Custom data type declarations (classes, structs, typedefs, extra...):

- Generally classes, structs, and objects that have instance variables should be declared in their own file, unless they are utilized by only a single function or small group of functions in the same file
- Generally enums and typedefs that are used for a single function or group of functions can be declared alongside said functions if they are all in the same file. Declare typedefs and enums above all functions, but below preprocessor include directives.

If and Switch statements:

- Care should be taken to avoid nesting if or switch statements greater than two levels down
  - Note: sometimes if/switch statements will have to be nested inside each other. Nevertheless, it should be avoided if possible

Loops:

- Care should be taken to avoid nested loops greater than two levels down
  - Note: sometimes loops will have to be nested inside each other, but it should be avoided if possible

User input:

- User input should be screened for simple type mismatches (e.i. Letters inside of numbers) **(only for classes beyond CIS 6)**
- User input should be screened for out of bounds inputs
- Note: input should be screened for simple wrong inputs, but don't become obsessed with trying to catch every single wrong input.

<u>Naming Conventions:</u>

General naming convention:

- In general names (function names, variable names) should convey the use of a piece of code. If you can't figure out any sensible name whatsoever for a variable or function, this indicates deep problems in your program design -- figure out how to fix these flaws on paper before writing more code.
- **Do not use single-character variable or function names, ever.**

Functions:

- Function names should be in camelcase
    - Camelcase: first word starts with lowercase and following words start with uppercase, Example: thisIsSomeCamelCase
- Function names should give a good indication of a functions usage.

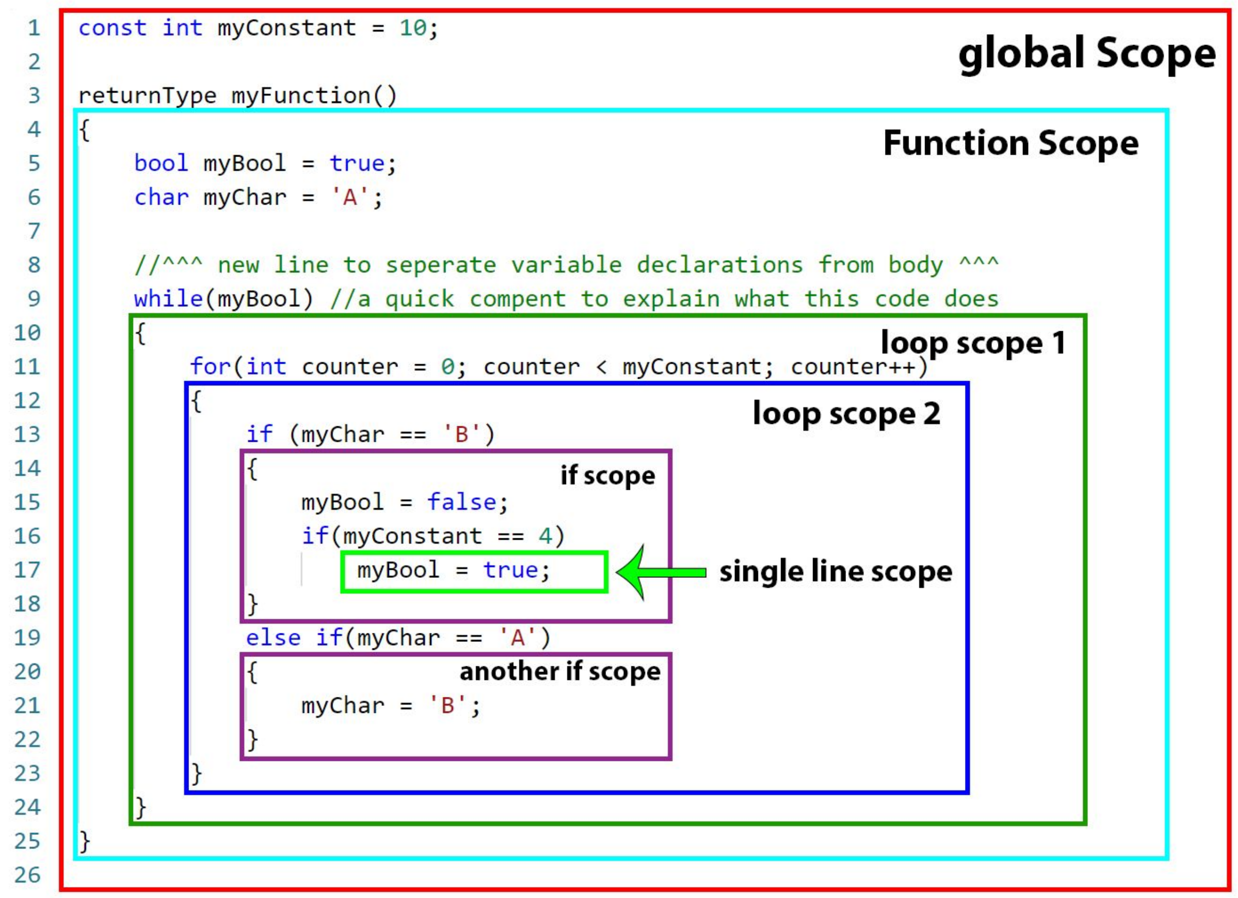
Variables:

- Constants should be in ALL CAPITALS and use snake_case
    - Snake case: each word is split by an underscore instead of a space, Example: this_is_some_snake_case
- **Non-constant global variables should not be used**. However, if you run into a case where one is **ABSOLUTELY** necessary they should start with g_ and then be camelcase
- **It's worth repeating: Non-constant global variables should not be used.** If it is absolutely necessary to use a global variable, **please include a comment above the variable explaining why it is necessary.** "I'm using the variable in more than one function" is not a good explanation -- in this case, define the functions to take the variable as a parameter. The use of global variables makes it much, much more difficult for you to reuse functions in multiple programs, makes it unclear which functions act on what variables, and is considered terrible programming practice. (in software development jargon, globals reduce the *cohesion* of your program)
- Local variables should use camelCase
- Variable names should give a good indication of what the variable is used for and what type of data it stores
- While designing classes, take care not to use unnecessary instance variables. If it is possible to move an instance variable into one of the functions defined within the class, please do so.

Custom data types:

- **Data type names (classes, structs, enums, typedefs) should use Pascalcase**
    - Pascalcase: the first letter of each word is capitalized, Example: ThisIsSomePascalCase

Scope diagram:



```
const int myConstant = 10;

returnType myFunction()
{
    bool myBool = true;
    char myChar = 'A';

    //^^^ new line to seperate variable declarations from body ^^^
    while(myBool) //a quick compent to explain what this code does
    {
        for(int counter = 0; counter < myConstant; counter++)
        {
            if (myChar == 'B')
            {
                myBool = false;
                if(myConstant == 4)
                    myBool = true;
            }
            else if(myChar == 'A')
            {
                myChar = 'B';
            }
        }
    }
}
```