Source code will have the following elements ***in the following order***:

1. Block comments shall be enclosed in /* and */ pairs.
2. Single in-line comments shall use the // comment notation
3. Top level block comment with your name, date, purpose and high level description of the program.
4. `#include` <>. System includes enclosed in wakkas
5. `#include` "". Custom include files enclosed in quotes
6. `#define`. All manifest constants and macros. Their names shall be in UPPER CASE.
7. `struct, union` and `typedef` declarations.
8. Function declarations (you MUST have these)
9. Global variables (avoid like the plague)
10. Function bodies with main() as the LAST function.

If you have an ordering conflict between the #define and the typedef and struct elements, then create a custom include file (filename.h).

Indenting. No less than 3 and no more than 5 spaces shall be used. TAB shall NOT be used for indenting (not on purpose anyway). Indenting will be consistent throughout the program.

Curly braces. The PASCAL convention shall be used simply because it makes the code easier for me to read. For example

```
double square(double x)
   {
   return(x*x);
   }
```

Functions: All function shall be pre-declared in the function declaration section of the program. Function names shall start in lower case and each new word will start with an upper case letter. For example `readFile(handle hFile);`
Functions will have their left parenthesis immediately following the name. No space.

Variable names: Same as the function naming convention, except that variables can start with an upper case letter (but are not required to).

struct, union and typedef declarations names: No restrictions except in terms of the use of single character names (see next item).

The variables i, j, k should only be used for indexing, except on those occasions where the code is clearer to use them otherwise. For example, i for current, k for a spring constant, *etc.*

For, while, if and other elements shall have a space between the expression and the left parenthesis. For example `for (i=0; i<100; ++i).` The `break` statement shall **never ever be used to exit a loop**.

All variables shall be declared prior to the first line of executable code in all functions. **DO NOT declare variables "in-line" at the time of their first use**. Doing so will hurt your grade. This construct is the same as in FORTRAN.

All functions shall have a block comment that precedes the function. This comment shall describe the purpose of the function, the algorithm, all arguments and the return value.

All functions shall have **one and ONLY one return statement** that will be the last line of the function. The return statement shall be formatted as a function with parenthesis, such as `return(0);`, unless it is a void function. Void functions shall have a single `return;` as their last executable line.

Variables and function shall have descriptive names and shall not be given pointless, cute, or flippant names such as `womBats` or `Bob`.

main() shall be fully declared such as `int main(int argc, char* argv[])`

main() will have a single exit() statement and exit() shall ONLY be use inside of main().

Do not use magic numbers! Use manifest constants (`#define`).

The code shall compile with no warnings when using the `-Wall` compilation switch.

EXAMPLE:

```
/*
**
** C example (doesn't really do very much)
**
** Eric R. Nelson
** March 1, 2010
**
**/
#include <stdio.h>
#include <stdlib.h>

#include "atcs.h"

#define PI 3.14159265358979
#define CONGRESS_PI 3.

struct cmplx {double x; double y;};

typedef struct cmplx complex;

void printMyMessage(PSTR message);

/*
** Simple function to print the passed string in a single line
**
** arguments:
** PSTR message - the message to be printed (null terminated string)
**
** returns void
*/
void printMyMessage(PSTR message)
   {
   printf("%s\n", message);
   return;
   }

/*
** main entry function.
** Arguments are accepted but are not used at this time.
** always exits with zero.
*/
int main(int argc, char *argv[])
   {
   printMyMessage("Hello World!");
   exit(0);
   }
```