For the purposes of this project the following issues should be known:

STeem emulator:

When a file is modified, whether in the emulator or in Windows, the file's timestamp is not updated. This means that make will not work correctly. You either have to 1) manually delete files or 2) use the provided script run_make.g

---

The C language:

All variable definitions must come before any executable code. If definitions and code are mixed the compiler will generate an error message about an "illegal storage class" for the definition line.

This restriction holds for ALL variable definitions, i.e. you can't define a loop counter variable in the for loop for a counted loop!

---

Global variables are common in system software - and therefore allowed in this course, if used judiciously and correctly! For example input routines and the keyboard device driver will need to share an input buffer. A variables can only be defined (have memory allocated for it) in one module, but if the definition (int x) is put into a header file this will result in a unique variable being defined in each module that includes this header file. Therefore, a method that declares the characteristics of this variable, but doesn't allocate space is required -- in C this is achieved by the extern keyword. Thus, the variable will be defined in one module and in the corresponding header file you will put the following declaration:

```
extern int x;
```

---

The c68 compiler:

C68 uses DRI linkable format and in this format labels are limited to 8 meaningful characters. Therefore, make the first 8 characters unique. Additional characters are possible but they will be eliminated by the compiler, so if you are trying to identify items use a common suffix instead of a prefix.

---

Since the 68000's normal size is a word (16 bits) when assigning a number constant the compiler automatically uses this size, regardless of the data type! So when using long variables you need to suffix the constant value with'L'.

```
e.g.    long *ptr = 0x88FF02L;
```

---

If you omit the #include of a library header file c68 will generate a WARNING about an "implicitly declared function" for each function call from this library. For any undeclared function the compiler assumes the return type of each call to be of type "int" regardless of what the actual return type is.

---

Converting from UINT8 to int doesn't work as expected, since it widens before becoming signed (this can be an issue when handling mouse packet delta x/ys). The correct procedure is to convert to signed first, and then to widen.

---

To compile and link with the math library the -f flag must be included on the cc68x command line, i.e.

```
cc68x -f math_pgm.c
```

---

Combining the definition and initializaton of a character array together can cause problems IF the array size is odd. The initial process involves a move.w that can result in an address error

Note: really found this to be a problem with a 2-D array of chars, i.e. an array of strings with the max string length being odd. For example an array of messages.

---

Combining C and assembly modules:

The C compiler prefixes global identifiers with an underscore. In other words, if you have an `add` function in a C module it will be renamed to `_add` by the compiler. Similarly, `main` is renamed to `_main`, etc. Thus, functions and variables in an assembly module that are to be visible in a C module must have names that start with an underscore.

---

Items in an assembly module that are to be visible in a C module must be XDEF'ed in the assembly module and a C header file must be constructed with the appropriate declarations (prototypes and externs)

---

If combining an assembly language value returning function with a C caller – the C compiler always uses global registers for returning values, specifically it uses d0 for the return value. How it returns addresses, i.e. structures or pointers have not been tested.

It is doubtful that the assembly language routines done in this class will use a return value. However, if needed it is probably better to use pass by reference.

---

The compiler uses some of the registers, possibly a0-a2/d0-d2, as scratch registers in functions. Since this is done consistently if your code is ALL C there is no issue. However, when calling a C function from an assembly routine this can possibly lead to the corruption of a register AND is especially true when working with interrupts. Thus, when combining C and ASM the assembly routine should save and restore all registers.

---