```
############################################################
#           cs315 Week 1
#
#   Welcome to the class, read the following carefully.
#
############################################################

Brief course overview:
    1) lecture covers concept and theory, discussion and labs cover practice and implementation
    2) attendance is required for both (attendance is very important as we learn the material incrementally)
    3) course book:
        "MIPS Assembly Language Programming by Robert Britton" is short but appendices are VERY useful, so get a copy and DO NOT rely on the Internet for MIPS help

Homework:
    1) theory homework is submitted in lecture time
    2) labs and programs are submitted online (through D2L)

Grading:
    1) lab assignments are due within 5 days following the lab day
        Ex: students in Wednesday lab have until 11:59PM on the following Monday to submit the lab
    2) labs and programs are graded separately
    3) each lab is worth 3 points:
        -> 1 point for attendance
        -> 1 point for submitting an attempt at a lab
        -> 1 point for submitting a lab which is mostly correct

Assembly language:
    * low level language
    * translates to machine code (i.e. the binary strings used to control the circuitry/hardware)
    * does not include features of a high-level language (e.g. loops, IF statements, etc.)

Why study assembly language:
    * speed - optimizing instructions (especially in a loop) reduces execution time
    * security - assembly language has a powerful amount of control over a processor
    * compilers - translate high-level languages into assembly language
        Ex: {C, Python} >-- compiler --> {MIPS} >-- assembler --> {Machine code (0's and 1's)}
    * operating systems - manage interface between programs and hardware
    * embedded computing - trend toward mobile/embedded computing is bringing programmers closer to hardware

MIPS:
    * Microprocessor without Interlocking Pipeline Stages
    * 32-bit load-store RISC architecture ('load-store' will be explained later)
    * used in embedded systems (e.g. PIC32 microcontrollers)
    * is similar to ARM, a popular architecture widely used in smartphones

Registers:
    * 32-bits wide storage elements (32 bits is 1 'word' in MIPS)
    * registers are connected to processor
    * used to hold operands and results for very short-term storage
    * assigned names to denote their intended use (e.g. $t3, $v1, $a0)
    * registers are faster than memory but more expensive than main memory

Memory:
    * 32-bit words, byte addressable but usually accessed from the start of a word
    * NOT connected to the processor
    * operands and results must be loaded to/stored from registers (hence the term 'load-store architecture')
```

```
    * used for long-term storage
    * slower but less expensive than registers
    * organized into different parts (e.g. static memory, dynamic memory, program memory)

Instructions:
    * found in appendix C
    * 2-bit words executed by the processor
    * execute in a fixed number of cycles
    * specify operation, operands (registers or immediate values), and result registers
        Ex: addiu $t0, $t1, 4    # add value 4 to the value that is in register $t1 and store result in register $t0
             ^
        add instruction

Macros:
    * found in appendix D
    * translate to instructions
    * may be translated one of several ways
    * are NOT themselves executed by the processor
    * number of cycles may vary depending on how macro is translated
        Ex:  Load immediate, 'li', may translate one of two ways depending on the immediate value

System calls:
    * ask the 'system' to do something for us
    * handles I/O, memory management, etc.

Variables:
    * labels associated with a memory address
    * similar to but not the same as variables in a high-level language

Labels:
    * symbolic (i.e. text) names
    * used for variables, used to mark locations in a program (e.g. jump/branch targets)
        Ex: enterVal_p:  .asciiz "Enter an integer:  "
                ^
              label

Directives:
    * commands to direct the assembler
    * NOT executed by the processor (i.e. they do not turn into MIPS instructions)
        Ex:
            .data
        --> values following .data will be placed in static memory
        --> variables go in .data sections

            .text
        --> values following .text will be placed in program memory
        --> instructions go in .text sections

Comments:
    * comments starts with: #
        Ex: li $v0, 10      # load immediate value 10 into register $v0
    * used for notes, explanations, code organization, etc.
    * discarded by assembler
    * have no impact on the execution of the program
    * comments MUST be used in course work for full credit

Miscellaneous notes:
```

```
    * THIS CLASS BUILDS Material covered each week will be necessary for the duration of the course
    * missing information one week will have a detrimental cascading effect
    * very easy to get behind on the material, not very easy to catch up
    * if a class is missed, then get notes from other students, from the instructor or TA, etc.

Avoid academic dishonesty:
    * helping each other is encouraged, sharing solutions IS NOT
    * if you are struggling, ask for help.  Do NOT plagiarize, copy, 'borrow', etc.
    * DO NOT claim credit for work which is not yours
    * makes notes in comments of other students you may have discussed your work with
    * examples of academic dishonesty:
        -> copying code from another student, from another solution, etc.
        -> giving a solution to another student (even after saying, "Don't copy this")
        -> academic dishonest does not only place you at a disadvantage, it undermines the credibility of all other students and the university in general
        -> academic dishonest can result (and has resulted) in students being expelled

QtSpim:
    * simulator/assembler used for labs
    * one of several MIPS simulators
    * not all simulators are compatible (DO NOT use MARS which is a java based simulator)
    * programs are graded with QtSpim, so using another simulator may cause problems when grading
    * already installed on lab computers
    * can (and should) be downloaded on your own machine
    * reads PLAIN TEXT programs only
    * programs should be written in Notepad, Wordpad, Notepad++, or similar text editors
    * programs MUST be saved as plain text (QtSpim will not recognize text formatting data saved by word processors)
    * save program files with a ".s" file extension
    * QtSpim is not the most user-friendly program, please spend time practicing with it

LEARN HOW TO DEBUG!
    * extremely important skill to have as a Computer Science professional
    * much time and many headaches will be saved by debugging
    * lots of insight into programming will be gained through debugging

Method for debugging in QtSpim:
    * breakpoints - pause program immediately before an instruction is executed
    * right-click on an instruction, click "Set Breakpoint" or "Clear Breakpoint"
    * can be used to pause a program and examine register/memory contents for correct/expected values
    * single stepping - execute one instruction at a time
    * can be used to examine how register/memory changes after each instruction
    * DO NOT waste time using a 'guess-and-check' method

Miscellaneous:
    * programs can be sent as an e-mail attachment
    * include programs when asking instructor or TA for help
    * labs are practice for programs an exams
    * good amount of lab work can be re-used in programming assignments
    * check D2L for lab solutions

Lab assignment - "Hello, world!" program:
    * copy handout into a text editor (e.g. Notepad)
    * load and run program in QtSpim
    * future labs will be more difficult than simply copying code from a handout!
```