ECE 429 / 629 Programming Assignment #1

In this assignment you will write a program that converts MIPS assembly language into MIPS machine code. The program will run from the command line, with a single argument indicating the name of the input file. Thus, if your executable is named *mconvert*, the syntax will be:

mconvert file.txt

where *file.txt* specifies the input file.

The program should parse the input file and create an output file called *out_code.txt*. If the file already exists, the program should overwrite it. The output file should contain one line per assembly instruction in the input file. Each line should consist of 32 bytes followed by a line feed + carriage return (Windows-style "\r\n"). Each of the 32 bytes should be either an ASCII 0 or an ASCII 1. Together, the 32 bytes specify the 32-bit machine code for the instruction, starting with the most significant bit. An example line of the output file, when viewed by a text editor, would look like this:

01000101110101101100100011101110\r\n

except that the \r\n would not be visible, of course. The input file should also contain one instruction per line. Each line should contain the mnemonic for the instruction in lowercase characters followed by whitespace, then the argument(s). The arguments should be separated by commas with no whitespace. Leading and trailing whitespace should be ignored until the end of the line (indicated by "\r\n"). Registers should be specified as the dollar sign ($) followed immediately by a number between 0 and 31, inclusive. Immediate values should be in decimal form. As an example,

```
addi    $1,$2,100    # This adds R1 <= R2 + 100
```

For reference, see the table 'MIPS machine language' on the back cover of *COD*. Only the following instructions need to be supported: add, sub, addi, beq, bne.

If the program ever encounters an invalid input line, it should append the following line to the output file:

!!! invalid input !!!\r\n

exactly as written, with no leading or trailing spaces. The program should then close the output file and terminate.

The output of the program should be exactly as specified above, byte for byte.

Notes

Under no circumstances whatsoever should the program crash. The program should allow for wrong parameters being passed on the command line, as well as a corrupt input file. Buffer overflow should be guarded against.

Never use strcpy. Use strncpy instead, which is safer. Even better, use std::string or CString rather than C-style char arrays whenever possible to minimize the chance of low-level memory mistakes. Similarly, never use sscanf with a "%s" parameter. Instead, use

"%ns", where n is the length of the allocated string, e.g., "%100s". For this assignment, you may want to get lines of the file using fgets, then parse using sscanf and strtok.

Avoid magic numbers. All constants should be defined at the top of your file in an obvious place.

By default, fopen opens a file in text mode on Windows. In text mode, "\n" will automatically be converted to "\r\n". If you would like to avoid such under-the-hood translations, you should open the file in binary mode. This is done by inserting a 'b' into the second string, e.g., fopen("file.txt", "wb").

Never share your code with another student by way of email, copying files, etc. While helpful collaboration is allowed and encouraged insofar as it supports learning, copying code is strictly prohibited. Please protect your work.