Coding Part 2 – Binary Files

In this assignment, **you must use C language**. This is a project that must be done with a group. I will assign you a partner and you **must** work with that group. Part of your grade will be how well **each of you** work on this project. If one person does all the work the rest of the group will get zeros – I have no time or patience for those who steal others work and call it their own.

You were asked, in the previous project, to read a *text* file and encode the MIPS instructions and display them as hexadecimal. In this part of the project, you will repeat your work, but this time, you'll write the values to a *binary data file.*

If your group was unable to complete the first part of the project, please see me and I will release my code to you after the first project is due.

A Binary File

C language provides two ways to work with binary files. Both of these code examples will create a file "data.dat" if it doesn't already exist, truncate it to be a zero-length file, and then write *n* bytes to the file, where *n* is the size of the data value:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int f = open("data.dat",O_CREAT | O_RDWR, 0777);
int rc = write(f, data, sizeof(data));
close(f);
```

```
#include <stdio.h>
FILE *fp = fopen("data.dat", "w");
fwrite(data, sizeof(data), 1, fp);
fclose(fp);
```

An "Executable" File

Modify your program from part 1 to become an "Assembler." Read the assembly file, and write the converted instructions to your binary file. There are two rules:

The first 32-bit value must be the "magic number" for an executable file, which for our file will be the 32-bit value "0x1b162d00." So, that gets written first. If the file doesn't start with the magic, then it won't be "executable" in the 3rd part of the project.

The second 32-bit value must be the number of instructions that are in the executable file. Of course, you won't know how many instructions there are until you're done reading from input! So, what you need to do is to write a 32-bit / integer 0 at that position. Then, after you're finished reading (and writing), use the "lseek" or "fseek" functions to go back to the beginning of the file (or even better, 4 bytes away from the beginning) and the write the number of instructions.

An Example "Executable" File

Input:
add $t1, $t2, $t3
add $t4, $t5, $t6

This would create a 16-byte file. The output of the "od -h data.dat" command would be (note its written in little endian since its running on an Intel machine):

```
0000000 2d00 1b16 0002 0000 4850 014b 6050 01cf
0000020
```

Not every executable is named "data.dat"

Use the *command line arguments* in C to accept the name of the output file from the command line. If you forgot (or never learnt), they are the argv and argc of: "int main(int argc, char **argv)"

Deliverables

Submit your source code to the autograder for the project. The autograder will take a Zip file of your code. The autograder will unzip the contents of your file. The autograder will do "clean" to remove any old build files, and then it will rebuild the file using "make." So, you need to have a "Makefile" as part of your project. Typing in "make" must build the "project2" executable.

Each test will be run as follows:

```
make 2>&1 /dev/null
./project2 test1.dat < testfile1
diff expected.dat test1.dat
```