CIT 593 Homework 12: C & File I/O

Due Date: Wednesday 11/26/2014 via canvas upload ONLY

Note: Because LC-4's C only has I/O functions we create (example: lc4_getc(), lc4_putc()), we need to switch to a more mature C-compiler that has access to the C standard I/O library of functions we've discussed in lecture. The book will properly discuss these functions in ***Chapter 16 & 18*** *of the book.* While these more mature functions may operate differently internally on different OS's, they all behave the same way across the C-language. So the book is definitely a good source for learning these I/O functions for C.

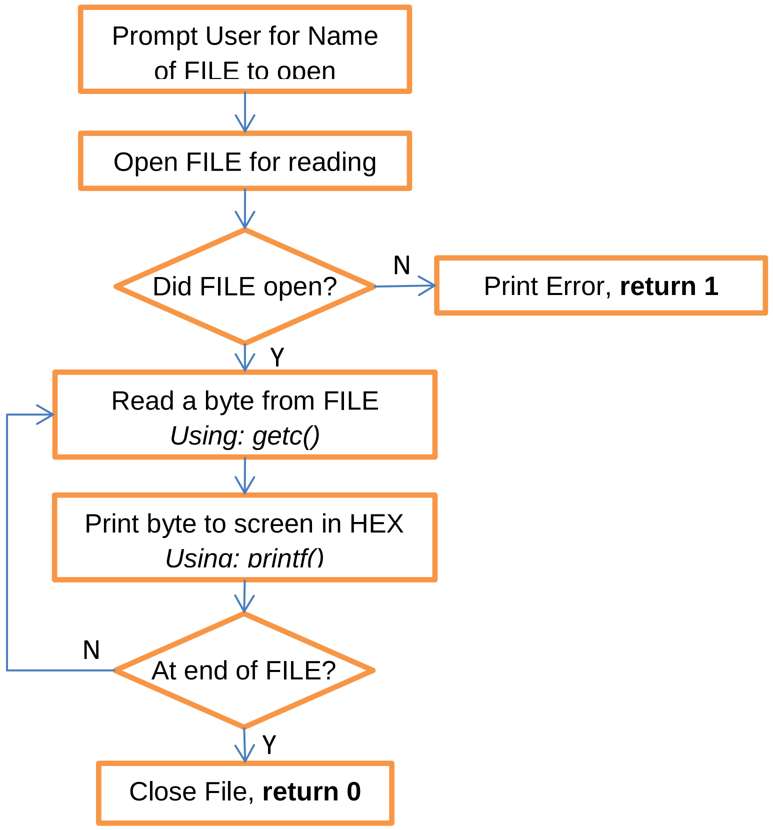
Assigned Problems (to be done individually, NOT group work):

For this assignment you will need to work & compile your C-programs on the university's server: eniac. This is necessary to access the functions published in the header file: stdio.h
How to author C programs on eniac, compile, and transfer files back and fourth to your computer will be covered in recitation. However, a tutorial is included with this HW (on canvas) to learn this material on your own if you prefer!

1) Working with ASCII files and the ASCII display

Overview: For this problem you will write a C program to prompt the user for the name of an ASCII file to open. Next you will open the file, read in its ASCII contents and display it as hex on the screen.

The following flowchart for your program will be referenced in the problem description:


Prompt User for Name of FILE to open
Open FILE for reading
Did FILE open?
N
Print Error, return 1
Y
Read a byte from FILE Using: getc()
Print byte to screen in HEX Using: printf()
At end of FILE?
N
Y
Close File, return 0

CIT 593 Homework 12: C & File I/O

Author your C-program using a text editor, and name your program: **my_hexdump.c** Your program should begin by outputting the following message to the screen using the `printf()` function:

```
Please enter the name of the file you would like to convert:
```

Next, use a function like `scanf()` to take input from the user into an char array. Your user will enter the name of a file that is in ASCII format (aka a "text" file). Using a function like fopen(), open the file named by the user. Be careful to inspect the output of fopen() to ensure that the file existed and actually opened. If it does not, print an error to "standard error" and return a 1 from main() as shown in the flowchart above.

If the file is valid and you receive a valid pointer (aka "handle") back to the file, use any function you like (as shown in lecture) to read the first byte of the file. As this is an ASCII file, the first byte will be the first letter in that file. Print the byte out to the screen in HEX and then put a space (using printf()). Repeat this process of reading a single byte at a time and printing its hex representation to the screen, until you reach the end of the file. Afterwards, close the file, and return with a 0 from main() to indicate success to the operating system.

As an example…if you created a simple ASCII file that contained the following text:

```
I love CIT 593!
```

← Filename: **`your_ascii_file.txt`**

Then when the program runs, the following interaction should be shown on the screen:

```
Please enter the name of the file you would like to convert:
your_ascii_file.txt
```

← *Note, the user must type this and press enter*

```
49 20 4C 6F 76 65 20 43 49 54 20 35 39 33 21
```

Remember, you must compile on eniac. To compile a C-program on eniac, type:

```
clang -o my_hexdump my_hexdump.c
```

To run your program on eniac type:

```
./my_hexdump
```

Your "input" ASCII file must be in the same directory as your .C program.

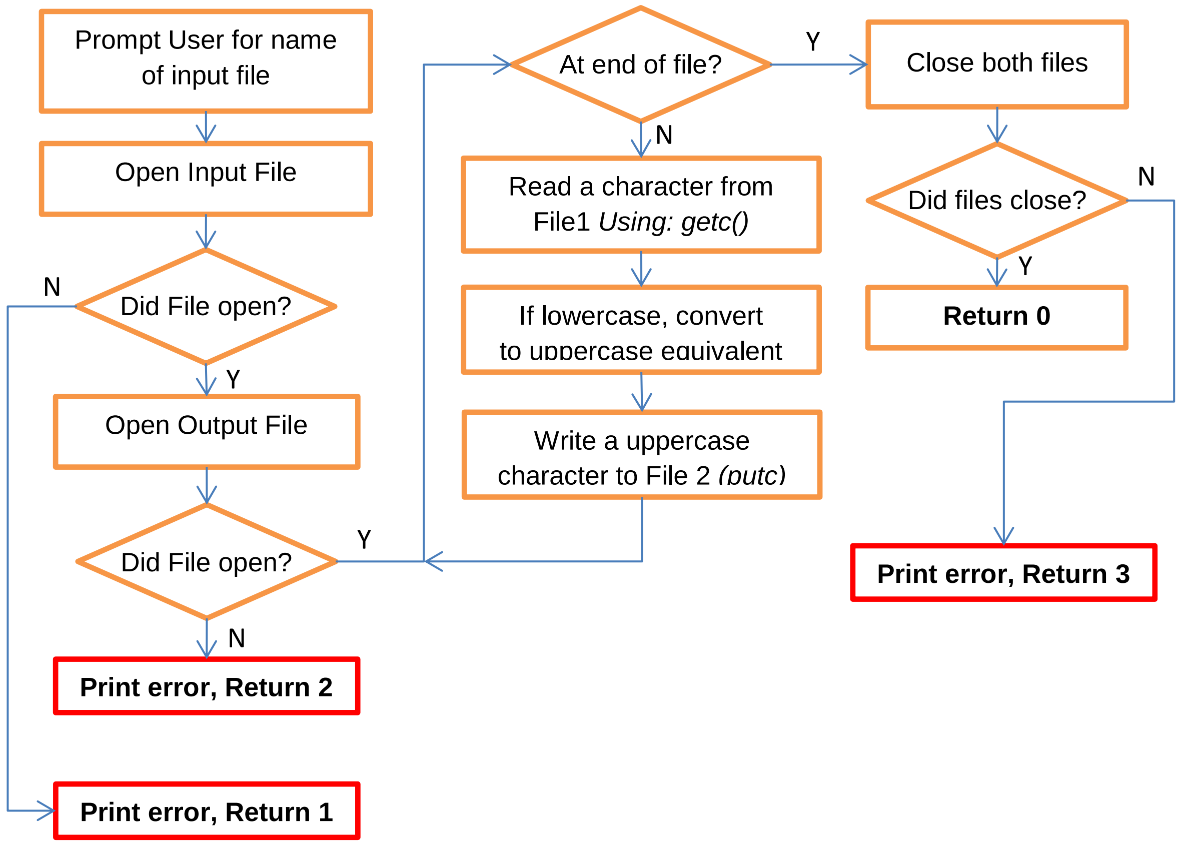
For this part you should turn in 2 files: my_hexdump.c , your_ascii_file.txt
Note: we will test with a 200 character ASCII file of our own!

CIT 593 Homework 12: C & File I/O

2) Working with ASCII files & strings in C

Overview: For this problem you will write a C program to prompt the user for the name of an ASCII file to open. Next you will open the file, read in its ASCII contents, convert any letter in lowercase to UPPERCASE, and write it out to a new file. In essence, you are writing an ASCII file copying program, except the "copied file" will contain the uppercase equivalent of the original.

The following flowchart for your program will be referenced in the problem description:



Author your C-program using a text editor, and name your program: **converter.c** Similar to problem 1), prompt the user for an ASCII file to use as your "input" file. Append "cp_" to the start of the name of the input file (using a function like: strcat), to create the name of your output file. Create/open the output file for writing. Carefully follow the flowchart above to handle all possible errors.

In a loop, read the first character of the user's input file. Inspect the character to see if it is lowercase. If it is lowercase, convert it to uppercase. If it isn't a lowercase letter or is some other ASCII character, leave it as is. Finally write the character, in either case, to the output file. Repeat this process until you reach the end of the file.

After exiting the loop, close the files (check for errors) and return from main().

CIT 593 Homework 12: C & File I/O

If a user provides the following input file:

```
I love CIT 593!
```

← Filename: **`your_ascii_file.txt`**

Your program should create the following output file:

```
I LOVE CIT 593!
```

← Filename: **`cp_your_ascii_file.txt`**

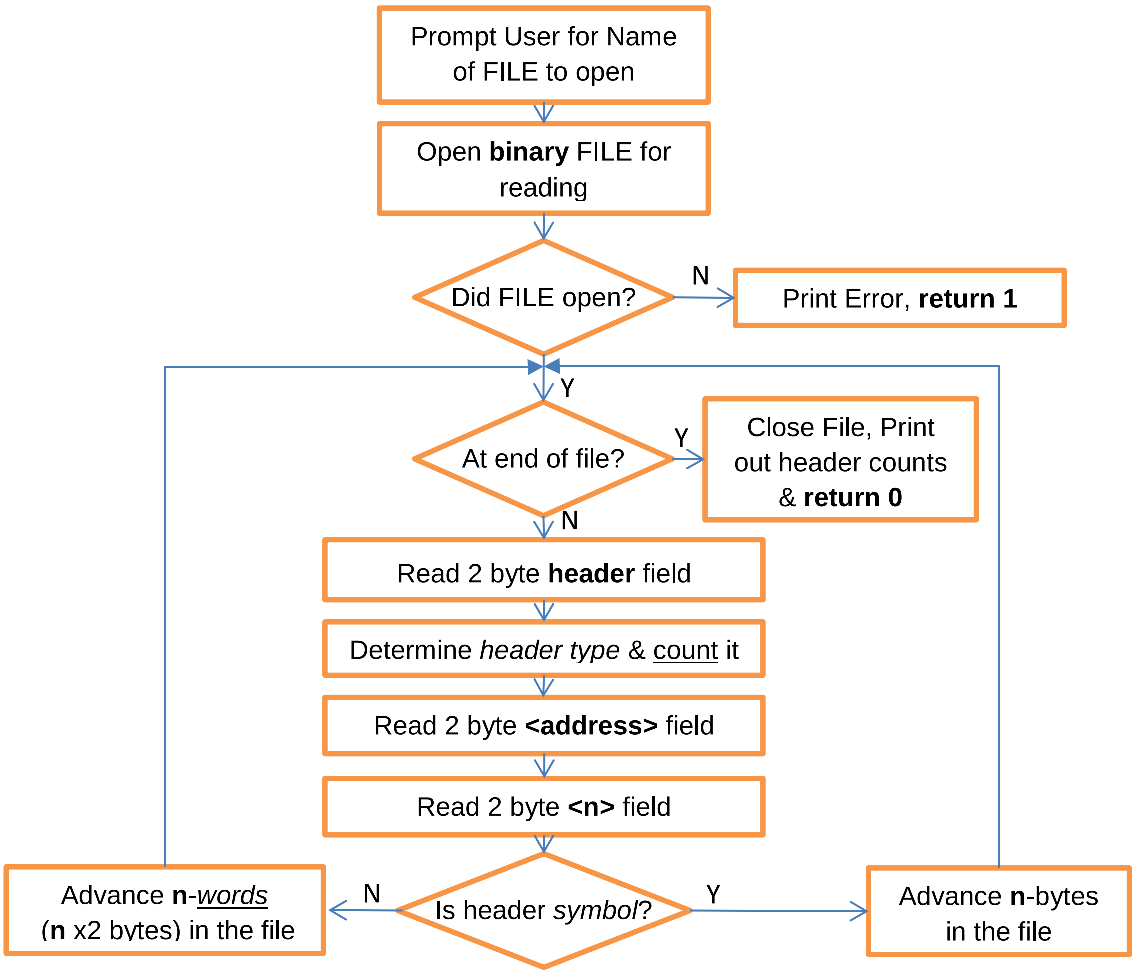
You may use any C-functions in stdio.h that you like. My suggestions in the flowchart are simply suggestions!

Note: we will test your program with any length ASCII file we wish! ***Also, carefully look at the ASCII chart, notice anything interesting at the hex codes for A and a ?***

For this part you should turn in 2 files: converter.c , your_ascii_file.txt

3) Working with binary files in C

Overview: For this problem you will write a C program to prompt the user for the name of a binary encoded object file generated by PennSim to open (we know them as .OBJ). Next you'll read in the object file and count the number of headers in the file and output that to the user. Recall, we discussed .OBJ headers in lecture!


Prompt User for Name of FILE to open
Open binary FILE for reading
Did FILE open?
N
Print Error, return 1
Y
At end of file?
Y
Close File, Print out header counts & return 0
N
Read 2 byte header field
Determine header type & count it
Read 2 byte <address> field
Read 2 byte <n> field
Is header symbol?
N
Advance n-words (n x2 bytes) in the file
Y
Advance n-bytes in the file

The following is the format for the binary .OBJ files created by PennSim from your .ASM files. It represents the contents of memory (both program and data) for your assembled LC-4 Assembly programs. In a .OBJ file, there are 3 basic sections indicated by 3 header "types" = CODE, DATA, SYMBOL.

- *Code:* 3-word header (xCADE, <address>, <n>), n-word body comprising the instructions. This corresponds to the .CODE directive in assembly.
- *Data:* 3-word header (xDADA, <address>, <n>), n-word body comprising the initial data values. This corresponds to the .DATA directive in assembly.
- *Symbol:* 3-word header (xC3B7, <address>, <n>), n-character body comprising the symbol string. Note, each character in the file is 1 byte, not 2. There is no null terminator. Each symbol is its own section. These are generated when you create labels (such as "END") in assembly.

CIT 593 Homework 12: C & File I/O

Author your C-program using a text editor, and name your program: lc4_inspect.c Similar to problem 1), prompt the user for the name of the .OBJ "input" file and open the file in **binary** format for reading. Create 3 variables to count the 3 categories of headers: code, data, symbol. Initialize them all to 0 to begin.

As shown in the flowchart, read in the first 2 bytes of the file. It will be the first header in the file. Determine the type of header based of the hexadecimal byte you have read in. If you encounter a "code" header, increment your "code" variable by 1. If it is a "data" header, increment your "data" variable by 1, and finally if it is a "symbol" variable increment your "symbol" variable by 1.

Read the next 2 bytes in the file, this will be the "**address**" field. You can discard this information, as we are only counting the "header" types in your program. Read the next 2 bytes in the file, this will be the "**n**" field. Now, if the "n" field corresponds to a "symbol" header, advance (seek), n-bytes in the file. For all others header types, advance (seek), n-words in the file (aka ***n*2*** bytes, as a word is 16-bits in the LC-4). By properly advancing in the file, you will now be at the next header (or at the end of the file itself). Loop as shown in the flowchart and repeat the header-reading process once again.

Once you have read & counted all the headers in the field, your program should print out:

```
Number of Code headers: [number of Code headers]
Number of Data headers: [number of Data headers]
Number of Symbol headers: [number of Symbol headers]
```

Included in this homework is the example discussed in lecture: file_format_example.obj
Its hex contents are as follows:

```
CA DE 00 00 00 02 90 02 10 00 DA DA 40 00 00 01 00 00 C3 B7 40
00 00 05 4D 59 56 41 52 C3 B7 00 00 00 06 4C 41 42 45 4C 31
```

The header type is in RED, the address is in GREEN, and the "n" field is in purple. Notice the "n-field" corresponds to how many words (or bytes for symbols) should follow it.

After inspecting this file, your program should print out:

```
Number of Code headers: 1
Number of Data headers: 1
Number of Symbol headers: 2
```

For extra credit, you may display the labels themselves!

We will test with our own .OBJ files! You can generate your own .OBJ files by compiling any .ASM file in PennSim! For this part you should turn in 1 file: lc4_inspect.c.

CIT 593 Homework 12: C & File I/O

Directions on how to submit your work:

- Create a single zip file called: **LAST_FIRST_HW12.zip**
- The zip file should contain the 4 files named in this assignment.
- There should not be any sub-directories within your zip file.

As an example, I would turn in the following SINGLE zip file:

FARMER_THOMAS_HW12.zip

This single zip file would contain 4 files only*:*

> **my_hexdump.c**
> **converter.c**
> **your_ascii_file.txt**
> **lc4_inspect.c**

You will then upload ONLY 1 file to canvas: **FARMER_THOMAS_HW12.zip**

- DO NOT TURN IN ANY files **created by the compiler**, **.obj**, or **PennSim.jar**

- Make certain that you submit the latest version of your code.

- Submitting using any other compression type (.RAR, TAR, GZIP) will be rejected.

Paper/Email submissions will not be accepted for this assignment.