Lab 9

Introduction to C
CSCI 112, Spring 2015

Objectives

- Learn to use file pointers in C
- Practice reading from a file
- Practice writing to a file
- Practice with common string functions
- Learn to give arguments to your C program

Overview

For this lab, you are given two text files containing information about chemical elements from the periodic table. One file contains elements in columns 1-12 of the periodic table while the other file contains the elements from columns 13-18. The files are each sorted by element number. You will merge these two files together, maintaining a sorted order, and write the information out to a third file. Next, you will process the arguments given to your C program and print out the information about each chemical element.

Data Format

In this lab, you will process data from two CSV files. The data from your files will be in format as this example:

```
47, 107.8682, Silver, Ag, ancient, 11
```

This line represent the element with an atomic number of 47, an atomic mass of 107.8682, the element name Silver, the element abbreviation Ag, a year in which the element was discovered (or the word "ancient"), and the element's category (column number) from the periodic table.

Global Elements

1. Download files `elements_file1.csv` and `elements_file2.csv` from D2L. Save this in the same directory as your .c file.
2. Include `stdio.h`, `stdlib.h`, and `string.h`.
3. Define a constant `LINE_SIZE` and set it to be 100. You will assume any line read from a file will be less than 100 characters.

4. Define a global variable:

```
const char delim[2] = ",";
```

You will use this string as a list of the deliminators used to split lines of text from your files. Since our data are in CSV (comma separated values) files, we set our deliminator to be a comma.

> PROTIP: A global variable will appear near the top of your program. You do not need to pass a global variable to your functions, but rather it will be accessible to all of your functions in your program automatically.
>
> In general, it is good programming practice to use global variables only rarely as they add to complexity of the code and reduce readability. In this case, we set our global variable to be a constant so that we will not accidentally change it.

5. Give your function prototypes. You will need to read through the lab instructions completely in order to determine the names and signatures of your functions.

`main` function

1. Create a main function with this signature:

```
int main( int argc, char *argv[] )
```

This will pass to your `main` function any arguments given to your program at the command line.

> PROTIP: C allows you to pass a variable number of arguments to your program's `main` function. The array of strings `char *argv[]` contains the individual arguments passed to the program. The `int argc` parameter gives the size of the array `argv[]`. The value at `argv[0]` is the name of your program. The actual arguments passed to your program are stored as `argv[1]` through `argv[argc - 1]`.

2. Declare three file pointers. Open the three files: open files `elements_file1.csv` and `elements_file2.csv` for reading and open file `elements.csv` for writing.

3. Check if any of the file pointers are `NULL`. If so, the file failed to open or the file was not found. In this case, print an error message and exit your program with error.

> PROTIP: Always makes sure your files opened properly before progressing.
>
> ```
> if ( fptr == NULL ){
>     printf("Error loading file.");
>     exit(1);                          // Exit with an error
> }
> ```

4. Call your function which will merge read the input files, sort them, and write the data to the output file.

5. Close your file pointers.

6. Create a for loop to iterate through the program's arguments. For each argument to your program, call your function which will search your element file and display the element's information.

   PROTIP: Remember to start this loop at index 1. The value at `argv[0]` is special and holds the name of your program.

Function to Read, Merge, and Write Files

This function will take in three file pointers as its parameters: two refer to the input files and the third is the output file. You should assume both input files are sorted properly. You will read from both input files and merge their content into a single output file which maintains proper sorted order.

1. Create two string variables both of size `LINE_SIZE`. As you process the files, these variables will hold the content of the current line from file 1 and the current line from file 2.

2. Create two additional string variables, also of size `LINE_SIZE`. You will use these to store copies of your variables mentioned in the previous instruction.

3. Use the function `fgets` to read a line from each of the input files.

   PROTIP: The function `fgets` reads text until a newline or EOF is read.
   http://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm

4. Use the function `strcpy` to make of copy of each of these lines.

   PROTIP: The function `strcpy` makes a copy of a string.
   http://www.tutorialspoint.com/c_standard_library/c_function_strcpy.htm

5. Create a while loop that continues until `EOF` is detected in either input file.

   PROTIP: The function `feof` tests for end-of-file indicator.
   http://www.tutorialspoint.com/c_standard_library/c_function_feof.htm

(a) Use `strtok` to split the line using the deliminator `delim`, and save the result as a string. The first time you call `strtok`, it will return the first substring (the string up until the first deliminator). Note that `strtok` will alter the string you pass in, which is why you made a copy previously. Repeat this for the other line.

> PROTIP: The function `strtok` breaks the string into a series of tokens using a deliminator.
>
> http://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm

You should have two atomic numbers, one taken from each file, each saved as a string variable.

> PROTIP: The `strcmp` method will compare strings lexicographically. This means that value "21" comes before the value "3". If your strings contain numbers, you will want to convert them to numbers and compare numerically.

(b) Convert each of these strings to an integer, using `atoi` function. Now, you can compare the two atomic numbers using $<, <=, >, >=$ and $==$.

> PROTIP: The function `atoi` converts a string to an integer.
>
> http://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm

(c) Determine which atomic number is smaller. Print out the respective line to your output file. Use your copy of the line variable so that you print the entire line (recall the variable you passed to `strtok` was altered). Next, reset your string variable for this line to empty (e.g., `str[0] = '\0'`). Read in another line from that file. Update your copy of this line.

> PROTIP: Be sure to only read a new line from an input file after writing it to the output file. Be careful not to read in a new line from the other file, as this will cause you to accidentally skip some lines. The value you do not use will be used in the comparison on the next loop iteration.

(6) Once your loop ends, this indicates that one of the two input files is exhausted. Determine which file ended. From the other file, continue to read in all lines and print them out to your output file until this file reaches EOF.

(7) Lastly, print out the statement:

```
File merging complete.
```

Function to Find an Element

This function will take in a string as input and will return nothing. The input argument will be element's symbol to find, such as "C" or "Au".

1. Create a file pointer. Open your newly combined file `"elements.csv"`; this time as an input file. If the file pointer is `NULL`, print an error and `exit(1)`.
2. Create two string variables: one to store the line of text and the other to store a copy of the line of text. Again, use `LINE_SIZE` as the size of both arrays.
3. Create a while loop that continues until `EOF` is detected.
   a. Use `fgets` to read in line from the file.
   b. Check if the line is empty. If the result of your `fgets` call is `NULL`, use the command `continue;` to skip this iteration of the loop.
   c. Next, extract the tokenized strings from the line. Again use a call to `strtok` to get the first deliminated element. Save this string as `atomic_number`.

      > PROTIP: The first call `strtok(str, delim)` will return the first tokenized substring and proceed to split the input string. For subsequent elements (i.e., not the first), use the call `strtok(NULL, delim)`. Each time you call this, it will give you the next substring from your original string.

   d. Make repeated calls to `strtok(NULL, delim)` to save: atomic weight, element's name, element's symbol, year the element was discovered, and the element's category in the periodic table. All the these values can be stored as strings.
   e. Compare the elements element's symbol to the target symbol to find. If it matches, print out the element's name, symbol, atomic number, and atomic weight.

      > PROTIP: The function `strcmp` compares two string lexicographically.
      >
      > http://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm

   f. If the year discovered has the value of "ancient", do not print the element's year string. Otherwise, if the year is numeric, print out the year discovered.
4. After the loop, be sure to close your file pointer before the end of the function.

Example Execution

```
$ ./program9 C O Au

File merging complete.

Element: Carbon
 Symbol: C
 Atomic Number: 6
 Atomic Weight: 12.0107

Element: Oxygen
 Symbol: O
 Atomic Number: 8
 Atomic Weight: 15.9994
 Discovered in: 1774

Element: Gold
 Symbol: Au
 Atomic Number: 79
 Atomic Weight: 196.9665
```

Compile & Test

Compile your program using this `gcc` command. `c99` is a shortcut for running `gcc -std=c99`, which uses the C99 standard instead of the default C89 standard.

```
$ c99 lab9.c -o program9
```

Run your code, supplying certain element symbols as arguments to your program.
For example, supplying symbols for Carbon, Oxygen, and Gold:

```
$ ./program9 C O Au
```

PROTIP: In this lab, there are many opportunities to make a memory mistake. To cope with this, we **recommend** you write small portions of your program and test them before moving on. This makes it easier to find your errors. It is also just good programming practice.

Resources

- You should attend the Friday Lab session to seek assistance from the TAs and CAs.
- For general questions, check the D2L `Lab 9 Help Forum` to see if another student has already asked your question. Otherwise, post your question on the D2L forum. This forum is intended for general questions that will benefit all students. Do not paste your code nor give away any specific answers to the lab on this forum.
- For specific questions, attend the weekly lab session or attend the TA's office hours.
- You are encouraged to use resources or tutorials on the internet to learn unix or C. Check the class resource list on D2L for some links to useful resources.

Submission

- Assigned: April 21st
- Lab Day: April 24th
- Due Date: May 1, 8:00am

1. Make sure you included ample and informative comments – it is 20% of your grade!
2. Rename your C file to `last_first_lab9.c` and substitute your last and first name.

   Protip: If you fail to follow the above file naming conventions, your program may not be graded automatically and you will lose points.

3. Submit your `.c` file to the D2L dropbox Lab9

   Protip: Submit only your C file. Do not submit your object file or your executable program. Do not archive (e.g., `zip`) your file.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work will receive a 50% penalty. Labs submitted more than 24 hours late will not be accepted. The late deadline is Saturday, May 2nd, 8:00am.