

## Assignment 9: Matching Words

**Complete assignment due: Thursday, Oct 30th, 9:00 p.m.**

Write a C program named **match**. The program will take at least two file names. There is one optional command-line argument that may be present:

```
match [-i] wordsFile file [file ...]
```

The **wordsFile** will contain zero or more words, one to a line. Each word is guaranteed to have at most 40 characters, not counting the newline character.

Each **file** will contain zero or more lines of text. There will be at most 160 characters, not counting the newline character, on each line.

### Output:

The output for this program will be to **stdout**. Error messages and usage messages, when needed, will be to **stderr**.

For each file listed, print the file name on the first line of output. Search the file for each of the words in **wordsFile**. For each word, print the word followed by a list of line numbers that contain the word. If the word appears more than once on a line, then print an asterisk after the line number. Put one blank space after the word, and one blank space in-between the numbers. If the word does not appear in the file, do not print the word. Once the file has been searched for all the desired matches, print a blank line.

We are looking for matches of complete words, not partial matches. Thus, if you are looking for the word **able**, then **reasonable**, and **tablespoon** will not be a match. For this assignment a word is defined as a sequence of alphabetic characters. Thus, the following

```
Johnny was an able student. Tables were his specialty.  
Bill was a capable student. Able72's and 473able's were his specialty.
```

The first sentence matches **able**, but not **Tables**. So the line number would appear but without an asterisk. The second sentence matches **473able's** since the **3** and the apostrophe are not alphabetic characters. If **-i** were used, then the second line would have an asterisk since it would also match **Able72's**.

### Example:

A short example,

```
lectura-> books="/home/cs352/fall08/books"  
lectura-> cat words1.txt  
free  
nation  
consecrate  
lectura-> match words1.txt $books/gettysburg.txt $books/raven.txt  
/home/cs352/fall08/books/gettysburg.txt  
nation 4 6* 9 21
```

```
/home/cs352/fall08/books/raven.txt  
never 19 122
```

Notes:

The word “consecrated” appears on line 13 of `gettysburg.txt`, but 13 does not appear as a line number for `consecrate`. The word “freedom” appears in `gettysburg.txt`, but “free” does not. Thus, `free` is not listed in the output. The word “nation” appears twice on line 6 in `gettysburg.txt`, so an asterisk is printed after the line number.

The word “never” appears only twice in `raven.txt`. There are many occurrences of “Nevermore” and “nevermore”, but these do not count here. “Never” appears as well, but `-i` was not specified.

### Errors and Exit Status:

There should be at least two command-line arguments: the file name for the words file and one file to search. If `-i` is present as the first argument, then there should be at least three command-line arguments. Report an error if there are not enough arguments, along with a usage statement. Exit with a status of **2**.

If **fopen** on the **wordsFile** fails, report the error and print a usage statement. Exit with a status of **2**.

If **fopen** on any of the file command-line arguments fails, report the error and print a usage statement. Do not continue processing command-line arguments in this case. Note that you may have successful output for some of the file names listed. If you reach a file name that does not open, report the error and exit. Use a status of **2**.

If the first argument starts with a dash, but is not `-i`, report the error and print a usage statement. Exit with a status of **2**.

If your program searches the specified file(s), but does not find any matches, exit with a status of **1**. If you find at least one match somewhere in the specified file(s), exit with a status of **0**.

### File Handling:

With the exception of the optional `-i` argument, the command-line arguments will be filenames. You will need to open these files and read the contents. For the **wordFile**, you will need to read the contents once for each of the other files you are processing. For the other files, you will need to read them once for each word that you are searching for.

The implication of the previous paragraph is that we are not specifying any upper bounds on the number of words that can appear in **wordFile**, so that you do not know how large an array would be needed. Also, the number of lines in each of the other files can be arbitrarily many.

Use the **fopen** command to open the specified file for input. **fopen** takes two arguments. The first is a string that contains the filename. The filename can be the name of the file or it can be the absolute or relative pathname for the file. The second argument is a string that indicates for what

purpose the file will be used. Each purpose has a single letter. The most common purposes are read, write, and append, which use **r**, **w**, and **a**, respectively. To open a file for reading only, you would use **"r"**, to open a file for reading and writing, you would use **"rw"**. A file opened for writing will start writing at the beginning of the file, which implies it would overwrite the existing contents. A file opened for append will start writing at the end of the current contents.

**fopen** returns a **FILE \***. You will need to declare a variable of this type. You need to insure that **fopen** succeeded; that is, that it did not return **NULL**. An example:

```
FILE *inputFile;  
inputFile = fopen( "sample.txt", "r");  
if ( inputFile == NULL ) {  
    report error here  
    exit with non-zero status or call the usage function  
}
```

Once the file is opened, you can use the **f** version of the various functions that read from input. These include **fscanf** and **fgets**. For this program, **fgets** will work to read each line of the input file. Put the **FILE \*** variable as the third argument for **fgets**. An example, continuing from the previous example:

```
char buf[MAX_CHARS];  
if ( fgets(buf, MAX_CHARS, inputFile) == NULL )  
    handle end-of-file here  
else  
    handle normal input here
```

You can read a file again by first using the rewind command:

```
void rewind(FILE *stream);
```

**rewind** will set the file to start reading from the beginning of the file.

When you are done using a file, it should be closed using **fclose**. Continuing the example from above, the command would be:

```
int result;  
result = fclose(inputFile);  
if ( result != 0 ) {  
    fprintf(stderr, "An error occurred when closing the file\n");  
    ...call the usage function here...  
}
```

There are manpages for **fopen**, **fclose**, **fgets**, **rewind**, and **fscanf**. Your textbook covers this material; see Chapter 22 (both editions).

## Functions:

You should use functions for this program. It should be possible for us to easily understand your program from its organization into functions and the comments you provide. Examples include

using a **usage** function to print the usage error message, and a **processFile** function to read a file looking for matches. How you divide your code into functions is up to you.

Do not use global variables. Pass all needed arguments to functions as parameters. The one allowable exception is if you use the debug printing technique described in class (or a similar technique). The debug variable(s) can be global variables.

**Turnin:** Use **turnin** to turn in your completed **match.c** program. The command is:

```
turnin 352assign9 match.c
```

See the man page for the **turnin** program for details on what **turnin** can do and how you can confirm that your file was turned in.