CS3841 Lab 3
A UNIX Word counter - Part 1

Dr. Walter Schilling

Due: September 27, 2011 23:00 CDT

1 Lab Objectives

1. Construct software in C using a previously developed library in archive format.
2. Develop software using standard C file input routines.
3. Develop software which writes to the stderr output stream using C syntax.
4. Use command line parameters to pass data between programs.
5. Manage dynamic memory and heap allocation using C methods.

2 Recommended Reference Readings

Topic	Book	Pages
Command Line Parameters in C	C: A Reference Manual Fifth Edition	303-304
C File I/O	C: A Reference Manual Fifth Edition	363-370, 377-385
C Test Processing	C: A Reference Manual Fifth Edition	363-370, 377-385
C Character Processing	C: A Reference Manual Fifth Edition	335-345
C String processing	C: A Reference Manual Fifth Edition	347-351
Making and Using GNU Libraries	`http://www.delorie.com/djgpp/doc/ug/larger/archives.html`	

3 Introduction

The purpose of this lab is to develop a C application which can perform a meaningful task. In particular, the task this program is to perform is count the number of times words occur in a text file. The user will run the program, passing in a command of either a W or a C, which will control whether the output of the file shows the Word followed by the word count. A second command line argument indicates the name of the file that is to be counted.

Once you have completed this step, you are to demonstrate this program by sorting the output of this program using the UNIX sort command. You must be capable of piping the output of this program into the sort command, resulting in the output being sorted either alphabetically by word or word count.

4 Specific Requirements

1. Your program shall use a library version created from the linked list your wrote last week.
2. Your program shall execute under UBUNTU Linux.
3. The program shall accept two command line options. The first option shall be W if the output is to show the word first or C if the output is to show the count first. The second option shall name the file in which the words are to be counted.

4. Your program shall consist of at least two new C files. One C file shall contain the main logic of the program (aka reading the file, splitting the words out, etc.), and the second file shall contain management of the words and word counts.

5. The word which is being stored, as well as the word count, shall be stored in a C structure.

6. In order to simplify implementation, the word may be fixed in length at 30 characters. It is safe to assume no word will be longer than 32 characters[1].

7. Your program shall use the linked list from last week to store word structures.

8. Immediately prior to program exit, your program shall print to the stderr stream of the console the remaining dynamic memory allocated by including the following code. This should always be zero at program exit. If not, there is a memory leak within your program.

```
struct mallinfo veryend = mallinfo();
fprintf(stderr, "Final Dynamic Memory used: %d\n", veryend.uordblks);
```

9. All words shall be converted to upper case before being counted.

10. All words shall be stripped of non-alphabetic characters before being counted. For example, "it's" shall be counted as "IT" and "Drs." would be counted as "DRS".

11. Your code shall not contain any magic constants (i.e. random numbers assigned to the source code.)

12. All functions which are externally visible shall be defined in an appropriate header file.

13. All methods and variables which are limited in scope to the given file shall be declared with a "static" prefix.

14. All code must be fully commented. At the top of each file shall contain a comment block with your name, your course, the assignment name, the date, and what the file is responsible for doing. Each function shall have a function block declaring the purpose for the function, the parameters accepted, and the return value (if one exists).

15. Header files must be protected against multiple inclusion using an appropriately constructed ifdef or ifndef construct.

16. Output to the screen shall be manipulated using the C printf function.

17. File input and output shall be handled using the stdio.h library and fscanf.

5 Design

The exact details of the design are left up to you as the designer. However, a sample class diagram is included in Figure 1 which shows one possible solution.

The basic element of the design is the worddata structure. This structure contains a word definition (up to 32 characters in length) and a 32 bit integer count of the occurrences of the word. The initialize method will initialize a linked list of words to have no words present. The add word will add a new word to the list and provide it with an initial count of 1. The find word will search the linked list to find a word matching the text passed in. If a match is found, a pointer to the structure is returned. If it is not found, then NULL is returned. The combine counts method will combine the data counts in two worddata structures. For instance, if file a word is found 8 times and in another it is found 5 times, calling this function will result in a count of 13.[2] The clean up function will free all dynamically allocated memory. The gettotalwordcount method will return the total number of unique words which have been found (aka the length of the linked list.) The getWord method will return the word at a given index from the linked list.

[1] Trivia note: According to wikipedia, the most accurate source for this type of information (or maybe not...), the longest nontechnical word in the dictionary is 29 characters in length, making 32 an appropriate upper bound for the length of the dictionary words.

[2] Note: This function will be used more next week. You will implement it this week, though it may be buggy until you actually use it next week. However, the closer to correct it is from the beginning, the easier your task will be next week.

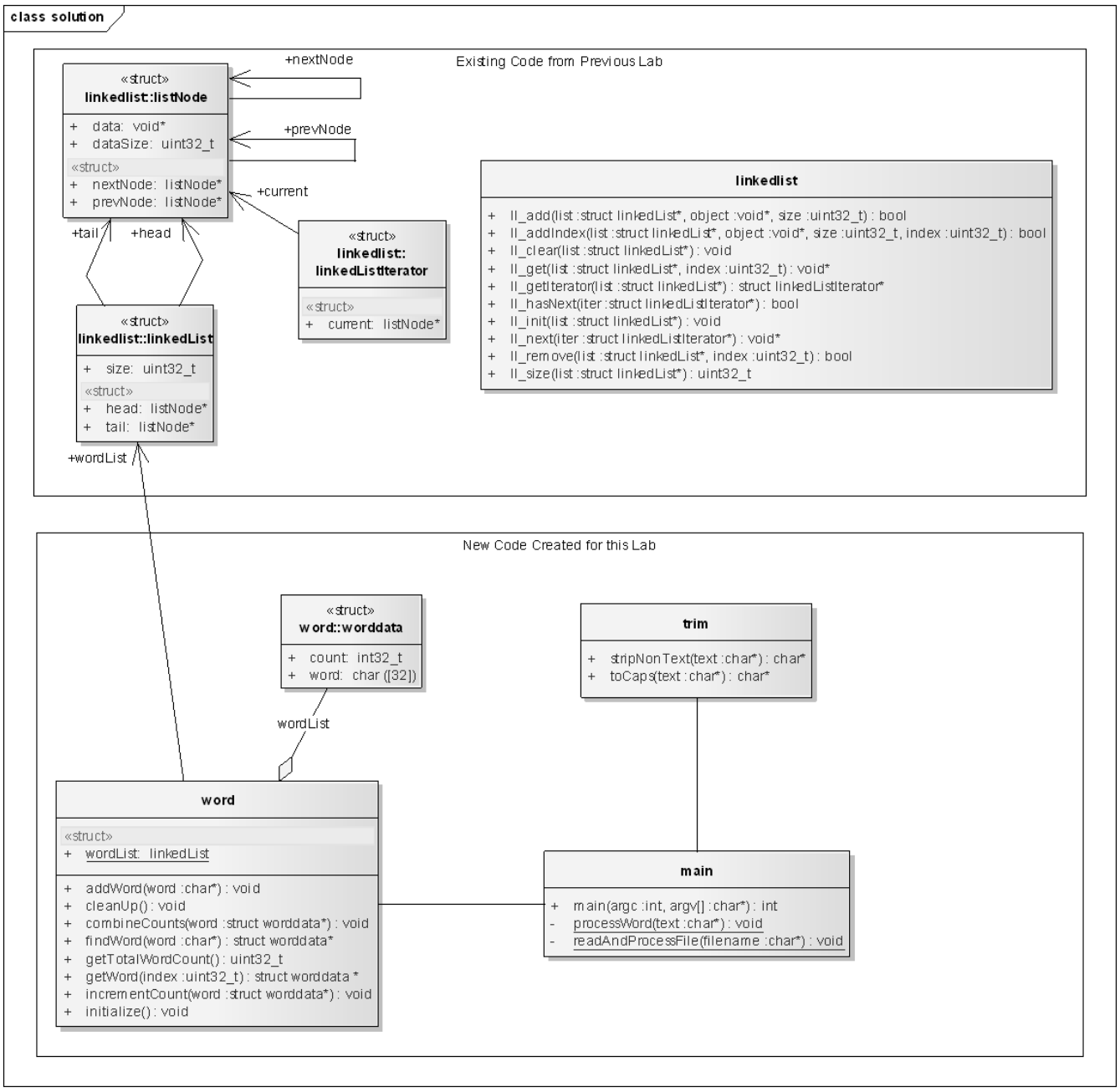

Figure 1: Sample UML diagram showing a potential design solution.

6 Sample Program Executions

Sample program executions are given in Figure 2 - 4.

```
student@student-VirtualBox:/media/sf_Dropbox/ClassPreps/Fall2011/CS3841/Labs/WordCounterPart1/solution$ ./wordCounter W WarAndPeace.txt
THE 34253
PROJECT 95
GUTENBERG 87
EBOOK 10
OF 14961
WAR 302
AND 21811
PEACE 114
BY 2389
LEO 4
TOLSTOY 9
THIS 2048
IS 3259
FOR 3485
USE 81
ANYONE 192
ANYWHERE 26
AT 4509
NO 1186
COST 26
WITH 5676
ALMOST 153
.
.
.
LICENSED 1
OUTDATED 1
IRS 1
REGULATING 1
CHARITIES 1
PAPERWORK 1
SOLICITATION 1
PROHIBITION 1
DONATION 1
ADDRESSES 1
CHECKS 1
EDITION 1
PG 1
NEWSLETTER 1
student@student-VirtualBox:/media/sf_Dropbox/ClassPreps/Fall2011/CS3841/Labs/WordCounterPart1/solution$
```

Figure 2: Example output 1 without piping for the file WarAndPeace.txt

```
wws@WWS-Ubunto:~/CS3841/Labs/WordCounterPart1/solution$ ./wordCounter C test.txt
1 FOUR
1 SCORE
5 AND
1 SEVEN
1 YEARS
1 AGO
.
.
.
8 TO
11 THE
1 PROPOSITION
10 THAT
.
.
.
1 GOVERNMENT
3 PEOPLE
1 BY
1 PERISH
2 EARTH
Final Dynamic Memory Used after program completion: 0
wws@WWS-Ubunto:~/Dropbox/ClassPreps/Fall2010/CS3841/Labs/WordCounterPart1/solution$
```

Figure 3: Example output 2 without sorting

7 Development Process

The first thing you will need to do is to make certain that your makefile from last week is actually generating a library file. The makefile should be creating the file liblinkedlist.a. The GNU archiver ar combines a collection of object files into a single archive file, also known as a library. An archive file is a convenient way of distributing a large number of related object files together. While we are only doing this with a single source code module, the concept is exactly the same as you would use for a large number of files (for example, libstdc).

```
wws@WWS-Ubunto:~/CS3841/Labs/WordCounterPart1/solution$ ./wordCounter W test.txt
FOUR 1
SCORE 1
AND 5
SEVEN 1
YEARS 1
AGO 1
OUR 2
.
.
.
THE 11
PROPOSITION 1
THAT 10
ALL 1
MEN 2
.
.
.
UNDER 1
GOD 1
BIRTH 1
FREEDOM 1
GOVERNMENT 1
PEOPLE 3
BY 1
PERISH 1
EARTH 2
Final Dynamic Memory Used after program completion: 0
wws@WWS-Ubunto:~/Dropbox/ClassPreps/Fall2010/CS3841/Labs/WordCounterPart1/solution$
```

Figure 4: Example output 3 without sorting

Now you will need to create the rest of your word counter program. How you go about doing the implementation is up to you. However, in your makefile, you will need to make certain you include the library which defines your linked list functions. To do this, you will need to customize your makefile. Details on using archive files and working with libraries can be found at `http://www.delorie.com/djgpp/doc/ug/larger/archives.html`.

8 Deliverables

1. Tar file including all of your source code, makefile, etc. Your source code must compile without any compiler warnings or errors by simply issuing the command "make all" at the command line prompt. "make clean" also must function.

2. Lab report, submitted in pdf format, with the following

   (a) Name, date, and lab title

   (b) What things went right and what things went wrong with this lab.

   (c) What did you learn from this lab.

   (d) Output captures of the program executing properly.

   (e) Captures of the program operating with the results sorted and using the input file test.txt. One capture shall show the sort being done alphabetically. One capture shall show the sort being done by the word frequency. Also include an explanation of how you accomplished this task. (Hint: It probably will be with a pipe operation and a shell command.)

   (f) An analysis of the novel War and Peace. What are the 10 most commonly occurring words and what are the 10 least commonly occuring words. (Hint: This may be best expressed as a table.)

   (g) Conclusions What did you learn by doing this lab, and how will it help you in the future?

   (h) Source code as an appendix (In order that it can be commented)

Deliverables are to be uploaded to the course website.

9 Preliminary grading rubric

Grading Details

	Weight Factor		
Source Code			
	4		Code Quality - Code properly commented - Code indicates name, course, and date in comment block - Multiple C files used for application development - No Magic constants in source code - Header files protected against multiple inclusion - Static prefix used on file scope variables - Protection against NULL pointer dereferencing.
	3		Code Functionality - Data stored properly in linked list. - Code properly converts to upper case and strips non-alphabetic characters from words. - Printf used for code output - File I/O handled using stdio.h
	2		Run Quality - No Segmentation faults when program runs - Memory leaks not present within source code and demonstrated through dynamic memory usage printout. - Program executes efficiently and quickly - Command line parameters work correctly - Invalid command line rejected
	2		Source Code Submission - All code included in tar file - All source files and header files included - Makefile included - Code can be compiled from submitted code without warning or error
Report			
	2		Format - Name, Date, Lab Title - File submitted in pdf format
	2		Things gone right and things gone wrong clearly described - Complete sentences - All aspects covered
	2		Screen capture / output of program operating shown - Program operates in both modes - Program operates with sort enabled, both by word and by number - Capture shows pipe operation occurring
	2		What was learned thoroughly discussed - Complete Sentences
	2		Conclusions - Complete sentences
	1		Source code included in Appendix