$Id: asg4c-edfile-dllist.mm,v 1.25 2012-11-15 16:43:17-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Assignments/asg4c-edfile-dllist
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Assignments/asg4c-edfile-dllist

1. Overview

This assignment will make use of doubly-linked lists in order to implement a very simple line editor in the style of **ed**(1). Data Structures goals : Experience in ANSI C with linked lists and pointers, and file handling.

2. Program Specification

NAME

edfile – list text editor

SYNOPSIS

edfile *filename*

DESCRIPTION

The **edfile** utility reads in lines from a file and stores them in a list. Editing operations make changes to this list. Eventually the lines are written out to a file.

OPTIONS

None.

OPERANDS

The operand is a filename. When the program begins, all of the lines from the file given on the command line are read in sequence and inserted into the list. The current line becomes the last line in the list. It is an error for there to be no filenames.

COMMANDS

After all of the files (if any) have been read in, **stdin** is read. Each line of **stdin** contains one command which is applied to the list in various ways. The editor can always get to the first line and last line in the file in O(1) time. It also maintains a pointer to the 'current' line. Note that there are ***no*** spaces between the command letter and its operand when an operand is permitted. Any time an attempt is made to print the current line and there is none, an error message is printed.

Before each command is read, a prompt is printed, giving the basename of the program. If either **stdin** or **stdout** is not connected to a terminal, all input lines are echoed to the output. Use **isatty**(3) to determine if they are attached to a terminal.

Any line beginning with any other character than those mentioned here. is in error. It is also an error if the single character commands are followed by any characters except spaces.

#*anything*
: A hash indicates a comment. Any line beginning with a hash is ignored.

$ The current line is set to the last line in the list. The new current line is then printed.

***** All of the lines in the list are printed. The current line becomes the last line in the list.

. The current line is printed.

0 The current line is set to the first line in the list. The new current line is then printed.

< The current line is set to the previous line. The new current line is then printed.

> The current line is set to the following line. The new current line is then printed.

a*inputline*
: The text following the letter '**a**' is inserted ***after*** the current line. The line just inserted becomes the current line, which is then printed.

`d` The current line in the list is deleted. The next line becomes the current line, if any. Otherwise the last line becomes the current line.

`i`*inputline*
The text following the letter '`i`' is inserted ***before*** the current line. The line just inserted becomes the current line, which is then printed.

`r`*filename*
The contents of the specified file are read in and inserted ***after*** the current line. The current line becomes the last line inserted. An error message is printed if the file can not be accessed. If the operation succeeded, the number of lines read in is printed.

`w`*filename*
All of the lines in the list are written to the specified file. The number of lines written is printed. An error message is printed if the file can not be created. The current line becomes the last line in the list.

`w` All of the lines in the list are written to the file specified on the command line. The number of lines written is printed. An error message is printed if the file can not be created. The current line becomes the last line in the list.

`^D` At end of file, whether at a terminal or controlled by a file, the program stops and writes the list to the file specified on the command line. It then prints the string `"^D\n"` to `stdout`.

— An empty line or a line consisting only of white space (`isspace`(3c)) is ignored.

EXIT STATUS

0 No errors were detected.

1 Some invalid commands or options were detected, or file access errors were detected.

3. Implementation Sequence

Following is a suggested implementation sequence.

(α) Study the behavior of `test/edfile.perl` and compare it to the specifications above.

(β) Read the instructions to the graders in the `.score/` subdirectory.

(γ) Begin with the files in the `code/` subdirectory. This contains a working program, but one which does practically nothing.

(δ) Look at `list.[hc]`. This contains your ADT to be implemented. You must eventually replace all the calls to the stub printing function. General rule: The list ADT maintains the list and does not do any I/O at any time for any reason, except for debug output. All I/O must be done in `edfile.c`.

(ε) Study the function `editfile` to see how it reads lines, chomps the trailing newline, and switches to the particular line type. Replace each of the cases by a call to a newly-written function to do the actual work. The function `editfile` is allready large enough to be confusing.

(ζ) For each of the commands that should not have arguments, but do have them, print an error message. Otherwise make an appropriate call to one of the list functions. Since there are 11 commands, you should write 11 functions.

(η) Do not make any line of the `switch` statement longer than one line. Each item in the switch should be a single function call, not inline code.

(θ) All error messages are written to `stderr`, with `Exit_Status` set to `EXIT_FAILURE`. Do an `fflush(NULL)` before and after each write to `stderr`.

(ι) Replace each of the stubs in the list ADT by appropriate code. Note that each of these functions may manipulate the list, but may not do any I/O.

(κ) In the above development, just comment out any calls to **free**(3c), or any associated functions. Get your program working without worrying about memory leak. There will be a significant penalty for any program that crashes on an assertion failure or halts and dumps **core**.

(λ) Check your program using **valgrind**. Eliminate references to uninitialized memory and references to memory outside of allocated nodes.

(μ) Submit your program. You should have a **submit** target in your **Makefile**. Submit the **Makefile** as well as all of your **.h** and **.c** files.

(ν) Verify that **checksource** does not complain.

(ξ) At this point decide if you are done or not. If not, eliminate all memory leak and verify using **valgrind --leak-check=full** that you have done so. If today is the due date, don't bother, you don't have time. If the due date is still a week away and/or you won't be satisfied with less than a perfect score, then you must eliminate both dangling pointers and memory leak.