Washington State University Vancouver

Systems Programming - CS 360

Assignment 2

Instructor:
Ben McCamish

Overall Assignment - 100 points

Write a program (in C) targeted at the Linux platform which checks whether a specified word exists in a dictionary or not. There are two possible dictionaries, both included in the handout.

Example of webster dictionary format:

line	offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8143	130288	g	o	u	r	m	e	t									\n
8144	130304	g	o	u	t												\n
8145	130320	g	o	v	e	r	n										\n
8146	130336	**g**	**o**	**v**	**e**	**r**	**n**	**a**	**n**	**c**	**e**						**\n**
8147	130352	g	o	v	e	r	n	e	s	s							\n
8148	130368	g	o	v	e	r	n	o	r								\n
8149	130384	g	o	w	n												\n
8150	130400	g	r	a	b												\n

- Format is 1 word per line
- Lines are in ascending sorted order
- Each line is x characters long
- Use binary search

Marks are shown below. Any requirement marked with "(Required)" will result in a score of 0 if not implemented.

Program Interface (Required)

```
int lineNum(char *dictionaryName, char *word, int dictWidth)

Where: dictionaryName is a path to the dictionary location,
word is the string you want to search, and dictWidth is the width
of the dictionary in characters.

Return: Your function should return the line number of the
word, if present. If the word is not present, it should return
the negative number of the line last searched.
```

1 Specifications and Restrictions

- (Required) You may only use the i/o methods covered in class to interface with the file. This includes `lseek`, `read`, `write`, `open`, and `close`. However, feel free to use other libraries for debugging (such as `printf`) or for string comparison (such as `strcmp`).
- (60 points) I will test 12 different words on your program using a variety of dictionaries. Each word will be worth 5 points. You must truncate all entered words to the specified width of characters in the dictionary, since my dictionary does not contain anything longer then that. Your function must return the line number the word is found on, the negative of the last line that was searched (if it wasn't found), or the value of the error code.
- (20 points) Error catching. You must catch errors and print out an appropriate error message containing the `errno` and the message produced by that error. This means you will need to `errno.h` and `string.h`, libraries at least. Your function should return the error number if an error occurs.
- (20 points) Style. 20 points will be dedicated to the style of your code. This includes indentation, commenting, and naming conventions.
- (Required) You should utilize lseek() to navigate the file. The entire file should never be in memory at once.
- (Required) You should create your own main function in a seperate `.c` file to test your lineNum function. Only upload the `lineNum.c` file that has the interface defined above.

What to turn in (on Autolab):

- lineNum.c (no header files)