CPE 111 - Algorithms and Data Structures
International Sections - January 2021
Laboratory Exercise 7

Objective

This lab gives you the opportunity to work with data structures that have linked lists as components.

Instructions

Summary

Create a program called **dictionary.c** that stores multiple words and definitions. After building the dictionary, allow the user to request the definitions of words. Each time the user enters a word, look it up in the dictionary to find its definition or definitions and print them.

Details

1. Download the following files from the website (under the Lab 7 entry):

```
linkedListUtil.c
linkedListUtil.h
Makefile
wordlist.txt
```

2. Open or type **wordlist.txt** so you can see its contents. Each line has a single word plus its definition, with a ":" separating the two. Your program will read this file and build a dictionary structure that will be an array of linked lists. The array will have 26 elements, one for each letter of the alphabet. (Do not change this file!)

3. Create a program called **dictionary.c**. Include the usual system header files and also include "**linkedListUtil.h**". If you look at this header file you will see that it defines a number of functions that create and manipulate linked lists. The **newList()** function creates and returns a list, using a data type **LIST_HANDLE**. All the other list functions take a **LIST_HANDLE** as an argument, to tell them which list to work with. So you can have multiple different linked lists at the same time.

4. Inside your main function, create an array of 26 list handles. For example:

```
LIST_HANDLE dictionary[26];
```

Initialize the array to all NULLs using the **memset** function.

```
memset(dictionary,0,sizeof(dictionary));
```

5. Define a structure to hold one word and definition. For example:

```
typedef struct
 {
 char word[32];
 char definition[128];
 } DICT_ENTRY_T;
```

6. Write a function called **buildDictionary** that will read the *wordlist.txt* file and create your dictionary. You will need to pass the **dictionary** array as an argument to this function. The logic of this function will be as follows:

```
open the file wordlist.txt
if  open is not successful
    give error message and return
endif
while there are lines left in the file
    read the next line
    break up the line into the word and the definition
    allocate a new DICT_ENTRY_T and check the result (if NULL give error and return)
    copy word from the line into the 'word' item in the dictionary entry
    copy definition from the line into the 'definition' item in the dictionary entry
    figure out where in the dictionary (0 through 25) this word belongs - call this index 'i'
    if dictionary[i] is NULL
        call newList to create a new list and store the returned value in dictionary[i]
    endif
    use listInsertEnd to add the new entry to the end of the appropriate list (the list in dictionary[i])
end while
close file
return
```

7. Add code to call **buildDictionary** from your main function. Compile and test. (You won't be able to see anything yet, but make sure the program does not crash.)

8. Write a function called **printDictionary** which prints the contents of your dictionary. You will need to pass the **dictionary** array as an argument to this function. The logic of this function will be as follows:

```
for i = 0 to 25
       list = dictionary[i]
       if list is NULL
          print "No words with letter 'x'" (where letter is 'A' for 0, 'B' for 1, etc.)
       else
          call listReset to return to the beginning of the list
          call listGetNext to get the data (a DICT_ENTRY_T*) from the first item in the list
          while data is not NULL
              print word and definition for current data item
              call listGetNext to get the data for the next item in the list
          end while
       end if
 end for
```

Call **printDictionary** from your main function, after you call **buildDictionary**. Compile and test. Now you can see if your dictionary contents are correct.

9. Write a function called **freeDictionary** to release the memory for the dictionary. You will need to pass the **dictionary** array as an argument to this function. The logic of this function will be similar to that of **printDictionary** except that instead of printing the information in the DICT_ENTRY_T*, you will free it. When you get to the last item of the list for a particular letter, you need to free the list handle itself.

Call **freeDictionary** at the end of your main function, before you exit. Compile and test the program. The behavior should not change.

10. Now write a function called **printDefinitions**. It should take two arguments, the **dictionary** array and a character string which is the word you want to look up. The function should look up the word in the dictionary and print out any definition it finds for the word. (Note that a few words in the dictionary have more than one definition.)

This function must figure out which array element corresponds to the first letter of the word you are looking up. Then, like **printDictionary** and **freeDictionary**, you must iterate through the list. Each time you get a new entry, you must check the word part to see if it matches. If it does, print the definition as well.

Keep track of the number of times you match the word. If this count is zero, print "Word not found".

Notice that there are actually two ways a word could be "not found". If there is no list at all for the first letter of the word, we know it's not in the dictionary. If there is a list, we have to look through all the items in that list.

11. Add a loop to your main function, after the call to **printDictionary** but before calling **freeDictionary.** In this loop, ask the user to enter a word. Then call **printDefinitions** to show the definitions of the word (if any). Exit the loop when the user enters a carriage return without any word.

12. Compile your program. Then test it with the following words:

ant	(one definition)
zebra	(one definition)
table	(one definition)
aunt	(not in dictionary)
watch	(two definitions)
paint	(two definitions)
ice	(no words starting with 'i')
eye	(no words starting with 'e')

13. Upload **dictionary.c**

Hints

To figure out which array element to use, based on the first letter of a word, you can use the following code:

```
int getDictIndex(char* word)
{
    char c = tolower(word[0]);
    return (c - 'a');
}
```

To turn a number (index into dictionary) into a character you can print, you can do the following:

```
    printf("Entries under the letter %c\n", i + 'A');  /* this will print as a capital letter */
```

To break up the line you read from the file into word and definition, you can use the function **strpbrk**. This function searches in a string for the first occurrence of any character in another string, and returns a pointer to the location where that character was found (which will be somewhere inside the first string) or NULL if not found. Your code might look like this:

```
    char input[512];      /* buffer to store word and definition from file */
    char* pDelim = NULL;  /* used to find position of ':' */
    DICT_ENTRY_T * pEntry = NULL;
    /* open and check file */
    while(fgets(input,sizeof(input),pFile) != NULL)
        {
        int len = strlen(input);
        if (input[len-1] == '\n')
          {
          input[len-1] = '\0';    /* get rid of newline */
          }
        pDelim = strpbrk(input,":");  /* find the colon */
        if (pDelim != NULL)           /* if no colon, just skip the line */
          {
          /* allocate new dictionary entry */
          pEntry = (DICT_ENTRY_T*) calloc(1,sizeof(DICT_ENTRY_T));
          if (pEntry == NULL) /* allocation error */
            {
            returnVal = 0;
            break;
            }
          *pDelim = '\0';                         /* replace ':' with 0 */
          strcpy(pEntry->word,input);             /* copy the word */
          strcpy(pEntry->definition,(pDelim+1));  /* definition starts at next char */
           /* now continue to add the new entry to the appropriate list */
          ....
          }
        }
```

As another possible approach, you could use the ***strtok()*** function. If you do, be sure to check that the returned value is not NULL before you use it.