Assignment 5: Speller

Jordan Mandel

2021_08_04

Speller Overview

- `speller.c` has the implementation of a script that checks against the hash table.
- `dictionary.c` has the function implementations from `dictionary.h`.
- `dictionaries/` has two dictionaries
- `texts/` has example texts
- `keys` has the solutions

In `dictionary.c` there are - `load`: load the words in the dictionary - `hash`: takes a word and runs a hash function on it - `size`: how many words are in the dictionary - `check`: is a given word in the dictionary - `unload`: free all the memory

load

```
bool load(const char *dictionary)
{
// TODO
}
```

- hash table is array of linked list
- hash function decides which linked list to insert into

We have to

- open dictionary file
  - use `fopen` to open the file
  - check to make sure it's not `null`

Definition of node struct:

```
typedef struct node
{
    char word[LENGTH + 1];
    struct node *next;
}
node;
```

N is the number of fields in the hash table.

```
const int N = 1;
node *table[N];
```

- Open dictionary file
  - use `fopen` and make sure return value is `NULL`
- read strings from the file one at a time
  - `fscanf(file_pointer,  "%s", word)` `file_pointer` is from `fopen`, `%s` is for string, `word` is a character array to read into
  - fscanf will return `EOF` at end of file. I think it automatically increments the pointer
- create a new node for each word
  - use malloc to store new node
  - check for null
  - use `strcpy` function to copy word into node

```
node *n = malloc(sizeof(node));
strcpy(n->word, "Hello");
```

```
n->next = NULL;
```

- insert word into a hash table at that location
  - use hash function to insert into index
  - careful not to orphan any nodes

hash

- hash the word to get a hash value
  - use the `hash` funtion to determine which index; alphabetic and apostrophe
  - output between 0 and N - 1
  - get the right linked list; going to have to add a node
  - will have to decide on value of N
  - math using some or all of the letters
  - or find an existing hash function

size

```
- return the number of words in the dictionary
- Either go through the entire hash table and get the number of words, or keep track of the number of words as y
```

check

```
bool check(const char *word)
{
// TODO
}
```

- check function
  - check if word is in dictionary (case insensitive)
  - hash word to obtain hash val; access that linked list
  - traverse linked list; use strcasecmp
  - start with cursor pointing to first node, increment it along linked list, comparing each time until you reach null.

unload

Unload all memory that we used

```
bool unload(void)
{
    // TODO
}
```

- `tmp`, `head`, and `cursor`; `tmp` and `cursor` point at the same place. advance cursor, then free temp