Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**Лабораторна робота №12**

на тему:

«Словники»

з курсу:

«Алгоритмізація та програмування»

Виконав:

ст. гр. КН-110

Король Орест

Прийняв:

Кривенчук Ю.П.

Львів – 2017 р.

*Лабораторна робота №12*

***Тема роботи:*** *розробити програму, що вмітиме працювати з словниками і розрізнятиме орфографічно правильні слова.*

***Мета роботи:*** покращити свої навички у роботі з циклами і масивами в мові програмування С. Навчитись правильно зчитувати і виводити елементи, навчитись працювати з словниками.

**Завдання**

​

Розробити програму, яка зможе перевіряти слова на орфографію і наявність в файлі.

**Текст програми**

#

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <math.h>

#include "dictionary.h"

#define HASHTABLE\_SIZE 65536

// define a node type

typedef struct node

{

char word[LENGTH+1];

struct node\* next;

}

node;

// define a variable to count the words loaded from the dictionary

int words;

// define a hashtable

node\* hashtable[HASHTABLE\_SIZE] = {NULL};

// hashing function

int hashing\_function(char\* word)

{

unsigned int hash = 0;

for (int i = 0, n = strlen(word); i < n; i++)

hash = (hash << 2) ^ word[i];

return hash % HASHTABLE\_SIZE;

}

/\*\*

\* Returns true if word is in dictionary else false.

\*/

bool check(const char\* word)

{

// copy the word ( because it is constant type and cannot be used)

int l = strlen(word);

char \*copy = malloc(l);

// copy the characters and make them lower case

for (int i = 0; i < l; i++)

copy[i] = tolower(word[i]);

// add the end of string character

copy[l] = '\0';

// hash the word

int index = hashing\_function(copy);

// check the table for a node at that index

node\* entry = hashtable[index];

if (entry)

{

// point a cursor node to the head node

node\* cursor = entry;

// loop through the nodes while a next pointer isn't null

while (cursor->next != NULL)

{

if (strcmp(copy, cursor->word) == 0)

return true;

cursor = cursor->next;

};

if (strcmp(copy, cursor->word) == 0)

return true;

cursor = cursor->next;

}

return false;

}

/\*\*

\* Loads dictionary into memory. Returns true if successful else false.

\*/

bool load(const char\* dictionary)

{

// open the input dictionary file

FILE\* fp = fopen(dictionary, "r");

// check for an unsuccessful open

if (fp == NULL)

{

printf("Could not open %s.\n", dictionary);

return false;

}

// set a buffer to store an output word of length = LENGTH + 1 for null terminator

char buffer[LENGTH+2];

// loop through the dictionary until a null character

while (fgets(buffer, sizeof(buffer), fp))

{

// overwrite the \n with \0

buffer[strlen(buffer) - 1] = '\0';

// create a temporary node

node\* temp = malloc(sizeof(node));

// set the node's pointers to the word and to the next node

// copies buffer's contents into the location of temp's word pointer

strncpy(temp->word, buffer, LENGTH + 1);

temp->next = NULL;

// hash the word to get the index

int index = hashing\_function(buffer);

// printf("%i" %(index));

// if the hashtable contains no value at that index then assign the temp node there

// (the entry takes the pointers of temp)

if (hashtable[index] == NULL)

{

hashtable[index] = temp;

}

// otherwise append the node to the end of the linked list

else

{

// point a cursor node to the index, (i.e the first entry in the linked list)

node\* cursor = hashtable[index];

// loop through the nodes while the next pointer isn't null

while (cursor->next != NULL)

{

cursor = cursor->next;

}

// the end of the linked list has been found, append the node by changing the shared pointer

cursor->next = temp;

}

words++;

}

// close the file

fclose(fp);

return true;

}

/\*\*

\* Returns number of words in dictionary if loaded else 0 if not yet loaded.

\*/

unsigned int size(void)

{

// return the global variable that was updated in load

return words;

}

/\*\*

\* Unloads dictionary from memory. Returns true if successful else false.

\*/

bool unload(void)

{

// for each node in the hashtable

for (int i = 0; i < HASHTABLE\_SIZE; i++)

{

// check the table for a node at that index

node\* cursor = hashtable[i];

if (cursor)

{

// create a temporary node to save the position of the next node

node\* temp = cursor->next;

// free the current node

free(cursor);

// move the cursor to the next node

cursor = temp;

}

}

return true;

}

***Висновок:*** завдяки набутим знанням з програмування, ми навчились реалізовувати складні програми.