/\*\*

\* dictionary.c

\*

\* Computer Science 50

\* Problem Set 5

\*

\* Implements a dictionary's functionality.

\*/

#include <stdbool.h>

#include <stdio.h>

#include <cs50.h>

#include <string.h>

#include <ctype.h>

#include "dictionary.h"

#define HASHSIZE 26

typedef struct sllist

{

char val [45];

struct sllist\* next;

}

sllnode;

void del(sllnode\* HashTable[], char\* key);

unsigned int counttable (sllnode\* HashTable[]);

void create(sllnode\* HashTable[]);

void destroy(sllnode\* HashTable[]);

unsigned int hash(const char\* string);

bool find(sllnode\* HashTable[], char\* key);

char s [45];

sllnode\* HashTable[HASHSIZE];

/\*\*

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

\*/

bool check(const char\* word)

{

// выделяем массив символов для копирования переданного слова

char words [45];

// копируем

strcpy(words, word);

// переводим все буквы в нижний регистр

for (int i = 0; i < strlen(words); i++)

{

words[i] = tolower(words[i]);

}

// определили индекс в хэш-таблице

int index = hash(words);

// поставили указатель на первый элемент связанного списка в таблице

sllnode\* ptr = HashTable[index];

ptr = ptr->next;

// проверяем, есть ли слово в списке

while (ptr != NULL)

{

if (strcmp(words, ptr->val) == 0)

{

return true;

}

ptr = ptr->next;

}

// TODO

return false;

}

/\*\*

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

\*/

bool load(const char\* dictionary)

{

// открыли словарь для загрзки и проверили на ошибки

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

if (pFile == NULL)

{

printf("Could not open\n");

unload();

return false;

}

// создали хэш-таблицу

create(HashTable);

// считывамем из файла в хэш-таблицу

while (fscanf (pFile, "%s", s) != EOF)

{

// hash-function

int index = (toupper (s[0]) - 'A') % HASHSIZE;

// выделили память

sllnode\* ptr = malloc(sizeof(sllnode));

// проверка на ошибку

if (ptr == NULL)

{

printf("Error1\n");

return false;

}

// скопировали значение в хэш-таблицу

strcpy(ptr->val, s);

ptr->next = HashTable[index]->next;

HashTable[index]->next = ptr;

}

// закрыли файл

fclose(pFile);

// TODO

return true;

}

/\*\*

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

\*/

unsigned int size(void)

{

// посчитали число элементов в таблице

return counttable (HashTable);

}

/\*\*

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

\*/

bool unload(void)

{

// удалили таблицу

destroy(HashTable);

return true;

}

// удаляем таблицу и очищаем память

void destroy(sllnode\* HashTable[])

{

// проходимся по каждому связанному списку и удаляем его элементы

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

{

sllnode\* ptr = HashTable[i];

while (ptr->next != NULL)

{

del(HashTable, ptr->next->val);

}

}

// удаляем основу хэш-таблицы

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

{

free(HashTable[i]);

}

}

// удаляет из таблицы значение по ключу key

void del(sllnode\* HashTable[], char\* key)

{

// проверка, существует ли значение в таблице

if (find(HashTable, key))

{

int index = hash(key);

sllnode\* ptrprev = HashTable[index];

sllnode\* ptrcurr = HashTable[index]->next;

while (ptrcurr->val != key)

{

ptrcurr = ptrcurr->next;

ptrprev = ptrprev->next;

}

ptrprev->next = ptrcurr->next;

free(ptrcurr);

}

}

// выводит всю хеш-таблицу

unsigned int counttable (sllnode\* HashTable[])

{

int counter = 0;

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

{

sllnode\* ptr = HashTable[i];

while (ptr->next != NULL)

{

ptr = ptr->next;

counter++;

}

}

return counter;

}

// hash function по первой букве слова

unsigned int hash(const char\* string)

{

int hash = toupper (string[0]) - 'A';

return hash % HASHSIZE;

}

// ищет значение в хэш таблице

bool find(sllnode\* HashTable[], char\* key)

{

int index = hash(key);

sllnode\* ptr = HashTable[index];

// проверка первого элемента списка

if (ptr->val == key)

{

return true;

}

// цикл по элементам

while (ptr->next != NULL)

{

if (ptr->val == key)

{

return true;

}

else

{

ptr = ptr->next;

}

}

// проверка последнего элемента списка

if (ptr->val == key)

{

return true;

}

return false;

}

// выделяем память под таблицу

void create(sllnode\* HashTable[])

{

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

{

HashTable[i] = malloc(sizeof(sllnode));

// проверяем на ошибку

if (HashTable[i] == NULL)

{

printf("Error1\n");

exit(1);

}

HashTable[i]->next = NULL;

}

}