1. Министерство высшего образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 1**

«Работа с файлами»

1. по дисциплине «Структуры данных»
2. Выполнил
3. студент гр. Белоконь Д. А., Солуянов А. Д.
4. <*подпись*>

Проверил Вагисаров В. Б.

1. <*подпись*>

1. Санкт-Петербург
2. 2023
3. **Цель работы**

Необходимо реализовать программу, осуществляющую замену слов в тексте и вывод полученного текста в выходной файл.

Формулировка задания

Входные данные:  
--- путь к файлу1 с текстом  
--- путь к файлу2 со словами, которые требуется заменить  
--- путь к файлу3 со словами, на которые требуется заменить  
--- путь к выходному файлу4, в который будет записан результирующий текст.  
Слова в файлах 2 и 3 разделяются переносом строки. Слово на первой строке файла 2 должно заменяться на слово первой строки файла 3, слово на второй строке — на соответствующее слово второй строки и т.д.

Текст может включать английские буквы, различные знаки препинания и специальные символы (в т.ч. пробелы, табуляции, и т.д.) Результирующий текст должен повторять все знаки препинания и специальные символы оригинального текста. Программа должна эффективно управлять памятью (используем динамическую память). Входные данные в программу должны передаваться с использованием аргументов командной строки и входных параметров main (argv, argc)

Результаты

Для достижения поставленной цели необходимо реализовать структуру данных односвязный линейный список, который будет динамически расширятся и освобождаться, под нужды текста. С помощью функции createList был создан линейный список, с помощью getListLength проверялась длина списки, addElement использовалось для добавления нового элемента, removeList удаляет список compileWord

Программа сначало просматривает количество введённых аргументов и если их меньше нужного количества, а именно 4, то выводит сообщение ошибки. Иначе продолжает выполнение программы, открывает 1 и 3 для чтения и при возникновении ошибки завершает работу, далее создаются два линейных списка для хранения исходного текста и слов исправления, соответственно, в которые по средствам функции readWords записываются слова(функция readWords поочерёдно парсит файл и записывает в него считанные слова)

Далее по средствам функции ltos (получает на вход список, а возвращает стринг) записывает текст в ссылку.

Далее идёт цикл просматривающий, подменяющий и выводящий слова в 4 файл, посредством множественных вложенных условий.

Вывод

По итогу проведённой работы были приобретены навыки использования линейных списков, а также применения динамической памяти. Программу получилось реализовать в полном объёме.

**Приложение**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct list {

void \*data;

struct list \*nextElement;

};

struct list \*createList() {

struct list \*newList = (struct list \*)malloc(sizeof(struct list) \* 1);

newList->data = NULL;

newList->nextElement = NULL;

}

int getListLength(struct list \*list) {

if (list == NULL) {

return -1;

}

int result = 0;

while (list != NULL) {

result += 1;

list = list->nextElement;

}

return result;

}

void addElement(void \*data, struct list \*list) {

if (list == NULL) {

list = createList();

}

while (list->nextElement != 0x0) {

if (list->data == 0x0) {

list->data = data;

return;

}

list = list->nextElement;

}

if (list->data == 0x0) {

list->data = data;

return;

}

struct list \*newElement = createList();

if (newElement == NULL) {

return;

}

list->nextElement = newElement;

newElement->data = data;

}

char \*compileWord(struct list \*list) {

if (list == NULL) {

return NULL;

}

char \*newWord = (char \*)malloc(sizeof(char) \* getListLength(list));

int counter = 0;

while (list != NULL) {

\*(newWord + counter++) = \*((char \*)(list->data));

list = list->nextElement;

}

return newWord;

}

void removeList(struct list \*list) {

struct list \*nextElement;

while (list != NULL) {

nextElement = list->nextElement;

free(list);

list = nextElement;

}

}

struct list \*readWords(FILE \*file) {

char \*symbol = (char \*)malloc(sizeof(char) \* 1);

struct list \*currentWord = createList();

struct list \*words = createList();

if (currentWord == NULL || words == NULL) {

fprintf(stderr, "An error occured while creating a list\n");

return NULL;

}

while (fscanf(file, "%c", symbol) != EOF) {

if (64 < \*symbol && \*symbol < 91 || 96 < \*symbol && \*symbol < 123) {

addElement(symbol, currentWord);

} else if (\*symbol == 10) {

free(symbol);

symbol = (char \*)malloc(sizeof(char) \* 1);

\*symbol = 0;

addElement(symbol, currentWord);

addElement(compileWord(currentWord), words);

removeList(currentWord);

currentWord = createList();

} else {

fprintf(stderr, "Not a letter or NL symbol in word-correction files");

removeList(currentWord);

removeList(words);

return NULL;

}

symbol = (char \*)malloc(sizeof(char) \* 1);

}

addElement(compileWord(currentWord), words);

removeList(currentWord);

return words;

}

char \*\*ltos(struct list \*words) {

char \*\*result = (char \*\*)malloc(sizeof(char \*) \* getListLength(words));

struct list \*temp = words;

int counter = 0;

while (temp != NULL) {

result[counter++] = temp->data;

temp = temp->nextElement;

}

removeList(words);

return result;

}

int search(char \*word, char \*\*list, int size) {

for (int i = 0; i < size; i++) {

if (!strcmp(word, list[i]))

return i;

}

return -1;

}

int main(int argc, char \*argv[]) {

if (argc < 5) {

fprintf(stderr, "Less than 4 arguments specified\n");

return 1;

}

FILE \*wordsFile = fopen(argv[2], "r");

FILE \*correctionsFile = fopen(argv[3], "r");

if (correctionsFile == NULL || wordsFile == NULL) {

fprintf(stderr, "Could not open file to read words from\n");

return 2;

}

struct list \*wordsList = readWords(wordsFile);

struct list \*correctionsList = readWords(correctionsFile);

fclose(wordsFile);

fclose(correctionsFile);

int wordsCount = getListLength(wordsList);

int correctionsCount = getListLength(correctionsList);

int matrixHeight = wordsCount < correctionsCount ? wordsCount : correctionsCount;

char \*\*words = ltos(wordsList);

char \*\*corrections = ltos(correctionsList);

printf("Changes matrix looks like this:\n");

for (int i = 0; i < 5; i++) {

printf("%s\t\t%s\n", words[i], corrections[i]);

}

FILE \*textFile = fopen(argv[1], "r");

FILE \*outputFile = fopen(argv[4], "w");

char \*symbol = (char \*)malloc(sizeof(char) \* 1), \*temp;

int result;

struct list \*currentWord = createList();

if (currentWord == NULL) {

fprintf(stderr, "An error occured while creating a list\n");

return 3;

}

while (fscanf(textFile, "%c", symbol) != EOF) {

if (64 < \*symbol && \*symbol < 91 || 96 < \*symbol && \*symbol < 123) {

addElement(symbol, currentWord);

} else {

temp = symbol;

symbol = (char \*)malloc(sizeof(char) \* 1);

\*symbol = 0;

addElement(symbol, currentWord);

result = search(compileWord(currentWord), words, matrixHeight);

if (result > -1) {

fprintf(outputFile, "%s", corrections[result]);

} else {

fprintf(outputFile, "%s", compileWord(currentWord));

}

fprintf(outputFile, "%c", \*temp);

free(temp);

removeList(currentWord);

currentWord = createList();

}

symbol = (char \*)malloc(sizeof(char) \* 1);

}

temp = symbol;

symbol = (char \*)malloc(sizeof(char) \* 1);

\*symbol = 0;

addElement(symbol, currentWord);

result = search(compileWord(currentWord), words, matrixHeight);

if (result > -1) {

fprintf(outputFile, "%s", corrections[result]);

} else {

fprintf(outputFile, "%s", compileWord(currentWord));

}

removeList(currentWord);

free(temp);

fclose(textFile);

fclose(outputFile);

return 0;

}