Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

«Волгоградский государственный технический университет»

Факультет                                 электроники и вычислительной техники

(наименование факультета)

Кафедра                     Программное обеспечение автоматизированных систем

(наименование кафедры)

**ОТЧЕТ**

**О     учебной      практике на                кафедре ПОАС**

вид практики наименование кафедры или научной

лаборатории университета

Руководитель практики от

университета,

  доцент кафедры ПОАС    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д.В.

должность подпись

Студент гр.      ПрИн-266       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тажибова А.В.

подпись

Студент гр.      ПрИн-266       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шалиевская А.А.

подпись

Отчет защищен с оценкой:

Тажибова А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Шалиевская А.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Волгоград 2016 г

**Содержание**

[Введение 3](#_Toc457635296)

[Формулировка задания 4](#_Toc457635297)

[Содержание практики 5](#_Toc457635298)

[Заключение 6](#_Toc457635299)

[Приложение А. Способ упаковки-распаковки данных 7](#_Toc457635300)

[Приложение Б. Внешняя спецификация программы 8](#_Toc457635301)

[Приложение В. Проект программы 11](#_Toc457635302)

[Приложение Г. Модульные тесты 17](#_Toc457635303)

[Приложение Д. Разделение обязанностей между членами команды 71](#_Toc457635304)

[Приложение Е. Порядок конструирования программы 73](#_Toc457635305)

[Приложение Ж. Код программы 74](#_Toc457635306)

[Приложение З. Оценка полученных результатов 85](#_Toc457635307)

# Введение

Целью учебной практики является закрепление навыков, связанных с процедурно-ориентированным программированием, а именно:

* составление внешней спецификации программы;
* декомпозиция программы на подпрограммы;
* создание модульных тестов;
* программирование с псевдокодом;
* создание собственных структур данных;
* реализация типовых алгоритмических структур на языке программирования;
* форматирование кода в соответствии с единым стилем кодирования;
* проектирование программы «сверху-вниз» и ее отладка – «снизу-вверх»;
* ведение репозитория кода.

Так же учебная практика дает навыки разработки программы в команде.

# Формулировка задания

Для компактного хранения текста замените наиболее часто встречающиеся последовательности из двух символов их кодами. В качестве кода возьмите символы, не встречающиеся в данном тексте. Составьте таблицу кодов. Упакуйте или распакуйте заданный текст.

Архиватор должен работать в консольном режиме, получая параметры через командную строку.

Формат командной строки в режиме упаковки:

packer.exe <пробел> <флаг упаковки> <пробел> <путь к файлу, который надо упаковать>

Флаг упаковки в командной строке должен обозначаться буквой «p».

Пример командной строки при упаковке: packer.exe p C:\folder\text.txt

Формат командной строки в режиме распаковки:

packer.exe <пробел> <флаг распаковки> <пробел> <путь к файлу, который надо распаковать>

Флаг распаковки в командной строке должен обозначаться буквой «u».

Пример командной строки при распаковке: packer.exe u C:\folder\text.txt

Способ упаковки-распаковки данных представлен в Приложении А.

# Содержание практики

В ходе прохождения учебной практики выполнены следующие виды работ:

* составлена внешняя спецификация программы – см. Приложение Б;
* программа декомпозирована на подпрограммы – см. Приложение В;
* созданы собственные типы данных – см. Приложение В;
* созданы модульные тесты к подпрограммам – см. Приложение Г;
* написан псевдокод для подпрограмм – см. Приложение Ж;
* закодирован псевдокод - см. Приложение Ж;
* код отформатирован в соответствии с единым стилем кодирования - см. Приложение Ж.

# Заключение

В ходе прохождения учебной практики закреплены все навыки, необходимые для создания процедурно-ориентированных программ, и как результат создана программа, которая упаковывает/распаковывает данные в соответствии со способом, представленном в приложении А. Порядок конструирования программы представлен в Приложении Е. Объем работ, выполненный каждым членом команды, представлен в Приложении Д. Оценка полученных результатов представлена в Приложении З.

# Приложение А. Способ упаковки-распаковки данных

Для компактного хранения текста замените наиболее часто встречающиеся последовательности из двух символов их кодами. В качестве кода возьмите символы, не встречающиеся в данном тексте. Составьте таблицу кодов. Упакуйте или распакуйте заданный текст.

Алгоритм функции pack

1. Находим символы, которых нет в тексте

2. Находим все пары символов, сортируя их по частоте повторений

3. Составляем словарь, где каждой часто повторяющейся паре соответствует символ, отсутствующий в тексте

4. Заменяем часто встречающиеся символы в тексте на их коды

Алгоритм функции unpack

1. Разделяем входной файл на текст, который надо распаковать и словарь

2. Заменяем коды символов на соответствующие им пары символов

# Приложение Б. Внешняя спецификация программы

1. **Наимен**о**вание программы**

Данная программа является программой упаковки и распаковки по словарю заданного текста (далее packer или программа).

1. **Назначение программы**

Программа предназначена для компактного хранения заданного текста. Упаковка текста производится путем замены наиболее часто встречающихся последовательностей из двух символов их кодами. В качестве кода выбираются символы, не встречающиеся в тексте.

1. **Требования к режиму упаковки**
   1. **Состав и структура входных данных**

Входными данными является текстовый файл, содержащий текст, который нужно упаковать. Текст в файле должен быть представлен в кодировке Windows-1251.

* 1. **Состав и структура выходных данных**

Выходными данными являются текстовый файл, содержащий упакованный текст и файл с логом, содержащий отчет о процессе упаковки (краткий перечень выполнившихся функций). Имя файла с логом всегда «log.txt». Выходной файл должен быть представлен в кодировке Windows-1251.

Имя выходного файла с упакованным текстом и словарем образуется добавлением к имени исходного файла приставки \_packed.

* 1. **Ограничения на выполнение**

В случае если количество часто повторяющихся пар превышает количество свободных символов, программа сжимает те пары символов, на которые хватило свободных символов.

1. **Требования к режиму распаковки**
   1. **Состав и структура входных данных**

Входными данными являются текстовый файл, содержащий упакованный текст. Текст в файле должен быть представлен в кодировке Windows-1251.

* 1. **Состав и структура выходных данных**

Выходными данными является текстовый файл, содержащий исходный текст и файл с логом, содержащим отчет о процессе распаковки (примерный перечень выполнившихся функций). Имя файла с логом всегда «log.txt». Выходной файл должен быть представлен в кодировке Windows-1251.

Имя выходного файла образуется из имени исходного файла прибавлением к нему приставки \_unpacked.

Пример распаковки:

* исходный текст: т\*ет р\*ка
* распакованный текст: течет речка

1. **Ограничения на выполнение**

В случае возникновения ошибок пользователю будут выведены сообщения, приведенные в таблице 1.

Таблица 1. Типы ошибок и соответствующие им сообщения

|  |  |
| --- | --- |
| Тип ошибки | Сообщение об ошибке |
| Файл не открывается | «Ошибка открытия файла» |
| В тексте отсутствуют свободные символы | «Содержимое данного файла невозможно упаковать по причине отсутствия свободных символов» |
| Переданный на распаковку файл невозможно распаковать из-за ошибок в его структуре | «Невозможно произвести распаковку: ошибка в файле» |

Если функция вызывается несколько раз для одного и того же файла, то содержимое выходных файлов (файл с логом, файл, содержащий упакованный или распакованный текст) перезаписывается.

# Приложение В. Проект программы

1. **Декомпозиция программы на подпрограммы**

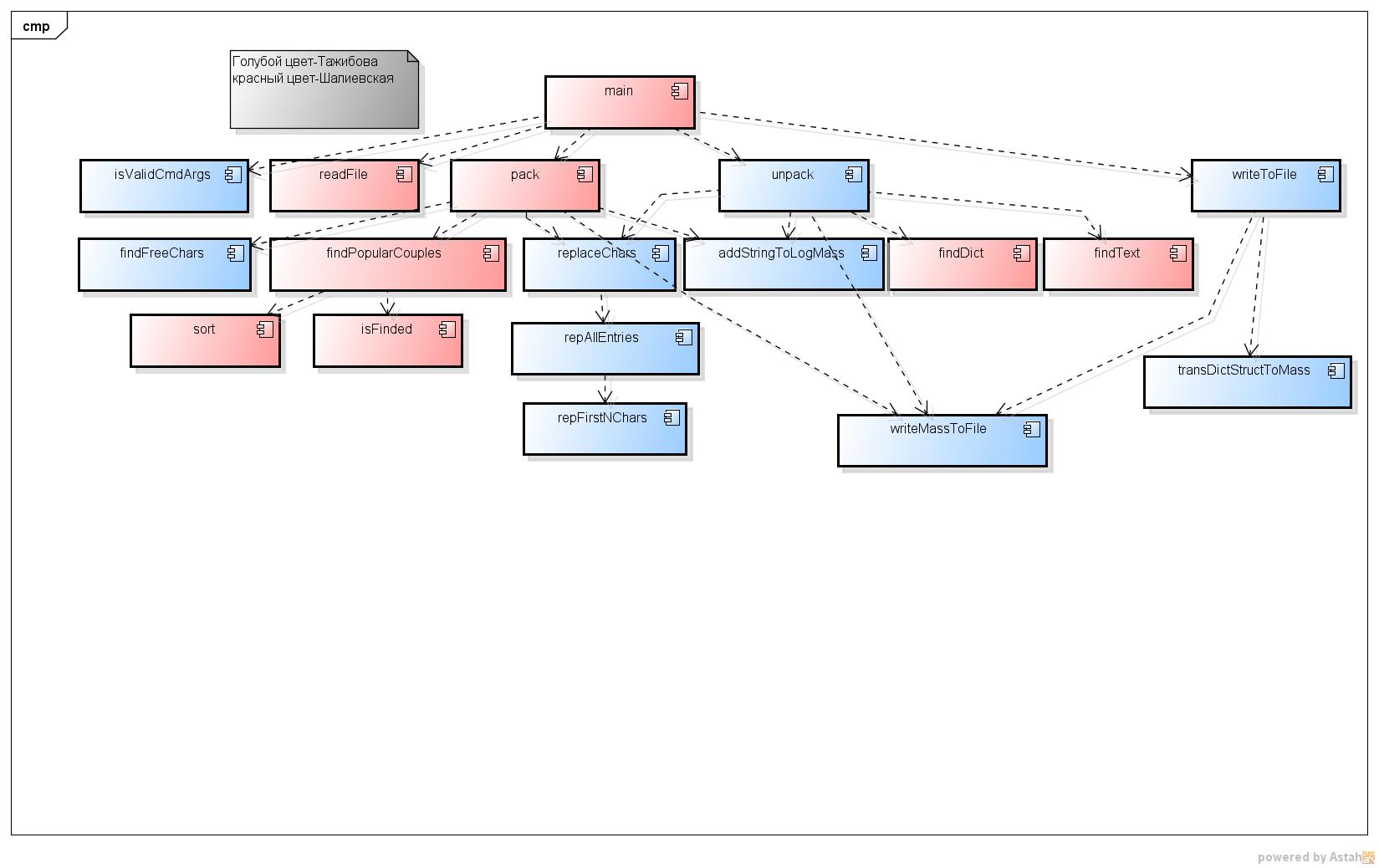


Рисунок В.1 – Иерархия вызова функций

1. **Собственные структуры данных**

/\*!

Структура, содержащая информацию о найденной паре символов

\*/

struct Couple

{

int coupleAmount; //! Количество повторений пары в тексте

char couple[3]; //! Символы, образующие пару

};

/\*!

Структура, содержащая информацию о строке словаря

\*/

struct DictString

{

char oftenRepeatedCouple[3]; //! Часто повторяющаяся пара символов

char coupleCode[2]; //! Код часто повторяющейся пары символов

};

1. **Перечень подпрограмм**

/\*!

Функция разбора параметров командной строки, введенных пользователем при запуске программы

\param[in] argc Количество аргументов командной строки

\param[in] argv Аргументы коммандной строки

\param[out] filename Имя файла, для которого вызвана программа

\return 0, если разбор арргументов командной строки прошел успешно, иначе 1

\*/

int isValidCmdArgs(int argc,char\*\* argv,char\* filename);

/\*!

Заменяет часто повторяющиеся пары символов их кодами согласно словарю

\param[in|out] text Текст, в котором необходимо произвести замену часто повторяющихся пар

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[in] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[in] dictSize Размер словаря (количество часто повторяющихся пар, записанных в массив структур dictionary)

\param[in] isPack Флаг, определяющий режим работы функции и равный 1, если упаковка, 0 если распаковка

\return 1 если все замены согласно словарю произошли успешно, иначе 0

\*/

int replaceChars(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int dictSize,int isPack);

/\*!

Заменяет в строке str все вхождения подстроки str1 на содержимое строки str2

\param[in|out] str Строка, в которой нужно произвести замены

\param[in] toBeReplaced Строка, вхождения которой в str нужно заменить на строку replacement

\param[in] replacement Строка, которой нужно заменить вхождения toBeReplaced в str

\return 1 если замена произошла успешно, иначе 0

\*/

int repAllEntries(char\* str,char\* toBeReplaced,char\* replacement);

/\*!

Заменяет первые n символов строки strTo на символы, находящиеся в strFrom

\param[in|out] strTo Строка, в которой нужно произвести замену

\param[in] n Количество символов в строке strTo, которые нужно заменить

\param[in] strFrom Символы, которыми нужно заменить n символов строки strTo

\return 1 если замена произошла успешно, иначе 0

\*/

int repFirstNChars(char\* strTo,int n,char\* strFrom);

/\*!

Ищет частые пары символов в тексте

\param[in] text Текст, в котором необходимо найти частые пары символов

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[in|out] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[in|out] dictCount Размер словаря

\*/

void findPopularCouples(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictCount);

/\*!

Сортирует массив пар по убыванию количетсва их повторений

\param[in|out] couple Пары для сортировки

\param[in] coupleCount Количетсво пар

\*/

void sort(Couple\* couple, int coupleCount);

/\*!

Проверяет, была ли найдена пара ранее

\param[in] couple Проверяемая пара

\param[in] couples Найденны ранее пары

\param[in] coupleCount Количество ранее найденных пар

\return -1, если пара еще не была найдена или индекс пары, если такая уже есть

\*/

int isFinded(Couple couple, Couple\* couples,int coupleCount);

/\*!

Находит символы, отсутствующие в тексте

\param[in] text Текст, в котором проходит поиск

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[out] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[out] dictSize Размер словаря

\*/

void findFreeChars(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int\* dictSize);

/\*!

Считывает содержимое файла в массив

\param[in] filename Имя файла для считывания

\param[out] stringsCount Количество строк в тексте

\param[out] charCount Максимальное количество символов в строке

\return указатель на считанный текст, 0, при ошибке

\*/

char\*\* readFile(char\* filename,int \*stringsCount,int \*charCount);

/\*!

Находит в массиве словарь

\param[in] text Текст, содержащий словарь и запакованный текст

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[out] dictionary Полученный словарь

\param[out] dictCount Количество пар в словаре

\return 1, если успешно, 0 при ошибке

\*/

int findDict(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictCount);

/\*!

Находит в массиве текст, который надо распаковать

\param[in|out] text Текст, содержащий словарь и запакованный текст

\param[in|out] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[in] dictCount Количество пар в словаре

\return 1, если успешно, 0 при ошибке

\*/

int findText(char\*\* text,int \*stringsCount,int charCount,int dictCount);

/\*!

Записывает в файл данные

\param[in] filename Имя файла, в который производится запись

\param[in] text Текст, который необходимо записать в файл

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[in] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[in] dictSize Размер словаря

\param[in] mode Флаг, определяющих тип записи, 0-запись словаря с запакованным текстом, 1-запись распакованного текста

\return 1, если запись завершилась успешно, иначе 0

\*/

int writeToFile(char\* filename,char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int dictSize,int mode);

/\*!

Записывает в файл массив символов

\param[in] filename Имя файла, в который производится запись

\param[in] text Текст, который необходимо записать в файл

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\return 1, если запись завершилась успешно, иначе 0

\*/

int writeMassToFile(char\* filename,char\*\* text,int stringsCount,int charCount);

/\*!

Добавляет строку в массив-лог

\param[in|out] logMass Массив, содержащий текст для лог файла

\param[in] stringsCount Количество строк в массиве

\param[in] isSucceeded Флаг, равный 1 если функция для которой пишется лог завершилась успешно, иначе 0

\param[in] funcDescription Описание задачи функции

\*/

void addStringToLogMass(char\*\* logMass,int stringsCount,int isSucceeded,char\* funcDescription);

/\*!

Переводит словарь-структуру в массив

\param[in] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[in] dictSize Размер словаря

\return словарь-текст при успешном выполнении функции, иначе NULL

\*/

char\*\* transDictStructToMass(DictString\* dictionary,int dictSize);

/\*!

Функция упаковки текста

\param[in|out] text Текст, который необходимо запаковать

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\param[out] dictionary Массив объектов структуры, содержащий все часто повторяющиеся пары и их коды

\param[out] dictSize Число пар в словаре

\*/

void pack(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictSize);

/\*!

Функция распаковки текста по словарю

\param[in|out] text Текст, который необходимо распаковать

\param[in] stringsCount Количество строк в тексте

\param[in] charCount Максимальное количество символов в строке

\return 1, если распаковка выполнилась без ошибок, иначе 0

\*/

int unpack(char\*\* text,int stringsCount,int charCount);

/\*!

Функция для поиска ошибок в работе функции pack

\param[in] text Текст, который необходимо запаковать

\param[in] m Количество строк в тексте

\param[in] count Количество пар в словаре, которые нашла функция

\param[in] expectCount Ожидаемое количество пар в словаре

\param[out] dict Составленный функцией словарь

\param[out] expectDict Ожидаемый словарь

\param[out] errors Ошибки, которые нашла функция

\param[out] errorsCount Количетсво ошибок

\*/

void check\_pack(char\*\* text,int m,int count, int expectCount, DictString\* dict, DictString\* expectDict,char\*\* errors,int \*errorsCount);

/\*!

Функция для поиска ошибок в работе функции findAllCouples

\param[in] couples Найденные пары

\param[in] expectCouples Пары, которые должны были быть найдены

\param[in] couplesCount Количество найденных пар

\param[in] expectСouplesCount Ожидаемое количество найденных пар

\param[out] errors Ошибки, которые нашла функция

\param[out] errorsCount Количетсво ошибок

\*/

void check\_findPopularCouples(DictString\* dictionary,DictString\* expectDictionary,int dictCount,int expectDictCount,char\*\* errors,int\* errorsCount);

/\*!

Функция для поиска ошибок в работе функции sepFile

\param[in] text Текст, который составила функция

\param[in] expectText Ожидаемый текст

\param[in] stringsCount количетсво строк в полученном тексте

\param[in] charCount Максимальное количетсво символов в строке полученного текста

\param[in] expectStringsCount количетсво строк в ожидаемом тексте

\param[in] expectCharCount Максимальное количетсво символов в строке ожидаемого текста

\param[out] errors Ошибки, которые нашла функция

\param[out] errorsCount Количетсво ошибок

\*/

void check\_findText(char\*\* text,int stringsCount,int charCount,char\*\* expectText,int expectStringsCount,int expectCharCount,char\*\* errors,int \*errorsCount);

/\*!

Функция для поиска ошибок в работе функции findDict

\param[in] dictionary Составленный функцией словарь

\param[in] expectDictionary Словарь, который ожидается

\param[in] dictCount Количество найденных пар в словаре

\param[in] expectDictCount Ожидаемое количество найденных пар в словаре

\param[out] errors Ошибки, которые нашла функция

\param[out] errorsCount Количетсво ошибок

\*/

void check\_findDict(DictString\* dictionary,DictString\* expectDictionary,int dictCount,int expectDictCount,char\*\* errors,int \*errorsCount);

# Приложение Г. Модульные тесты

**Модульные тесты для подпрограммы findDict**

void TestNoDict()//! Словаря нет

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int charCount=28;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"...");

strcpy(text[1],"Под разделителем есть текст");

strcpy(text[2],"Но словаря не было");

strcpy(text[3],"Надо вывести ошибку");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

}

void TestSepInText()//! Разделитель находится в самом тексте

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int charCount=22;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"Вот начался текст");

strcpy(text[2],"А разделителя еще нет");

strcpy(text[3],"Должна быть...ошибка");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

}

void TestOneStrInDict()//! В словаре одна строка

{

Check check;

int result;//Возвращаемое значение функции

int expectResult=1;//ожидаемое возвращаемое значение

int dictCount;//количетсво пар в словаре

int expectDictCount=1;//ожидаемое количетсво пар в словаре

int charCount=28;//Максимальное число символов в строке

int stringsCount=4;//Число строк в тексте

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"...");

strcpy(text[2],"Сверху разделитель");

strcpy(text[3],"Ошибки нет");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,"\*");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

if(result!=expectResult)//проверяем, правильно ли функция определила наличие ошибок

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

else

{

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

}

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestManyStrInDict()//! В словаре много строк

{

Check check;

int result;//Возвращаемое значение функции

int expectResult=1;//ожидаемое возвращаемое значение

int stringsCount=6;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int expectDictCount=3;//ожидаемое количетсво пар в словаре

int charCount=20;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"на+");

strcpy(text[2],"а ъ");

strcpy(text[3],"...");

strcpy(text[4],"Обычный текст");

strcpy(text[5],"После разделителя");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,"\*");

strcpy(expectDictionary[1].oftenRepeatedCouple,"на");

strcpy(expectDictionary[1].coupleCode,"+");

strcpy(expectDictionary[2].oftenRepeatedCouple,"а ");

strcpy(expectDictionary[2].coupleCode,"ъ");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

if(result!=expectResult)//проверяем, правильно ли функция определила наличие ошибок

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

else

{

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

}

for(int i=0; i<6; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

free(text);

free(dictionary);

free(expectDictionary);

}

void TestNoSep()//! Разделителя нет вообще

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int charCount=21;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"на+");

strcpy(text[2],"разделителя нет");

strcpy(text[3],"начинается текст");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

}

void TestSeparatorNotAtTheBeginning()//! Разделитель находится не в начале строки

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int charCount=21;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"на1");

strcpy(text[2]," ...");

strcpy(text[3],"тут начинается текст");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

}

void TestNoCode()//! В словаре справа написан не один символ

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount;//количетсво пар в словаре

int charCount=21;//Максимальное число символов в строке

DictString\* dictionary=(DictString\* )malloc(stringsCount\*sizeof(DictString));//Словарь, полученный из текста

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"на-12345");

strcpy(text[2],"...");

strcpy(text[3],"тут начинается текст");

result=findDict(text,stringsCount,charCount,dictionary,&dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

free(dictionary);

}

**Модульные тесты для подпрограммы findFreeChars**

void NotEnoughDictChars() //! Недостаточно свободных символов

{

int i,j=0;

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(3\*sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(131);

strcpy(text[0],"цацикакинанигагишашищащизазихахифафивавипапирарилалидадижажичачицуцекукенунегугешушещущезузехухефуфевувепуперурелуледудежужечучесу");

text[1]=(char\*)malloc(131);

strcpy(text[1],"цыцякыкянынягыгяшышящыщязызяхыхяфыфявывяпыпярырялылядыдяжыжячычяцоцэкокэнонэгогэшошэщощэзозэхохэфофэвовэпопэрорэлолэдодэжожэчочэсо");

text[2]=(char\*)malloc(7);

strcpy(text[2],"эубэуъ");

DictString\* dict=(DictString\*)malloc(229\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

int dictSize=0; // Размер словаря

char charCode; // Символ-код

// Вызываем тестируемую функцию

findFreeChars(text,3,131,dict,&dictSize);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(dictSize==227,L"Размер словаря равен \"%d\", а не 227",dictSize);

for(i=255; i>1; i--)

{

if(i!=0 && i!=224&&i!=225&&i!=226&&i!=227&&i!=228&&i!=229&&i!=230&&i!=231&&i!=232&&i!=234&&i!=235&&i!=237&&i!=238&&i!=239&&i!=240&&i!=241&&i!=243&&i!=244&&i!=245&&i!=246&&i!=247&&i!=248&&i!=249&&i!=250&&i!=251&&i!=253&&i!=255)

{

charCode=(unsigned char)i;

CFIX\_ASSERT\_MESSAGE(dict[j].coupleCode[0]==charCode,L"Вместо кода \"%c\" был получен код \"%s\"",charCode,dict[j].coupleCode[0]);

if(j!=dictSize)

j++;

}

}

// Освобождаем выделенную для тестирования память

for(int i=0; i<2; i++)

free(text[i]);

free(text);

free(dict);

}

void OneStrInText() //! В тексте одна строка

{

// Подготавливаем данные для тестирования

int i,j=0;

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(12);

strcpy(text[0],"течет речка");

DictString\* dict=(DictString\*)malloc(249\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

int dictSize=0; // Размер словаря

char charCode; // Символ-код

// Вызываем тестируемую функцию

findFreeChars(text,1,12,dict,&dictSize);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(dictSize==247,L"Размер словаря равен \"%d\", а не 247",dictSize);

for(i=255; i>1; i--)

{

if(i!=0 && i!=32 && i!=242 && i!=229 && i!=247 && i!=240 && i!=234 && i!=224)

{

charCode=(unsigned char)i;

CFIX\_ASSERT\_MESSAGE(dict[j].coupleCode[0]==charCode,L"Вместо кода \"%c\" был получен код \"%s\"",charCode,dict[j].coupleCode[0]);

if(j!=dictSize)

j++;

}

}

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void MoreThanOneStrInText() //! В тексте более одной строки

{

// Подготавливаем данные для тестирования

int i,j=0;

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(12);

strcpy(text[0],"течет речка");

text[1]=(char\*)malloc(12);

strcpy(text[1],"печет печка");

DictString\* dict=(DictString\*)malloc(248\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

int dictSize=0; // Размер словаря

char charCode; // Символ-код

// Вызываем тестируемую функцию

findFreeChars(text,2,12,dict,&dictSize);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(dictSize==246,L"Размер словаря равен \"%d\", а не 246",dictSize);

for(i=255; i>1; i--)

{

if(i!=0 && i!=32 && i!=242 && i!=229 && i!=247 && i!=240 && i!=234 && i!=224 && i!=239)

{

charCode=(unsigned char)i;

CFIX\_ASSERT\_MESSAGE(dict[j].coupleCode[0]==charCode,L"Вместо кода \"%c\" был получен код \"%s\"",charCode,dict[j].coupleCode[0]);

if(j!=dictSize)

j++;

}

}

// Освобождаем выделенную для тестирования память

for(i=0; i<2; i++)

free(text[i]);

free(text);

free(dict);

}

**Модульные тесты для подпрограммы findPopularCouples**

void TestEmptyFile()//! Файл пуст

{

int m=1;// Количество строк в тексте

int n=0;// Количество символов в самой длинной строке

int dictCount=3;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=0;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[1];//Словарь, который должен получиться

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestNoPair()//! В тексте нет ни одной пары

{

int m=4;// Количество строк в тексте

int n=2;// Количество символов в самой длинной строке

int dictCount=3;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=0;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"т");

strcpy(text[1],"а");

strcpy(text[2],"п");

strcpy(text[3],"р");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[1];//пары, которые должны быть найдены

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestPairsWithOnlyOneRepetition()//! Все пары в тексте повторяются ровно 1 раз

{

int m=4;// Количество строк в тексте

int n=3;// Количество символов в самой длинной строке

int dictCount=4;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=4;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"тр");

strcpy(text[1],"ам");

strcpy(text[2],"па");

strcpy(text[3],"ра");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[4];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"тр");

strcpy(expectDictionary[1].oftenRepeatedCouple,"ам");

strcpy(expectDictionary[2].oftenRepeatedCouple,"па");

strcpy(expectDictionary[3].oftenRepeatedCouple,"ра");

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestEquifrequentPairs()//! Все пары в тексте повторяются равное число раз

{

int m=5;// Количество строк в тексте

int n=3;// Количество символов в самой длинной строке

int dictCount=5;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=5;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"па");

strcpy(text[1],"ра");

strcpy(text[2],"та");

strcpy(text[3],"ту");

strcpy(text[4],"ам");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[6];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"па");

strcpy(expectDictionary[1].oftenRepeatedCouple,"ра");

strcpy(expectDictionary[2].oftenRepeatedCouple,"та");

strcpy(expectDictionary[3].oftenRepeatedCouple,"ту");

strcpy(expectDictionary[4].oftenRepeatedCouple,"ам");

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestOnePair()//! В тексте всего одна пара

{

int m=2;// Количество строк в тексте

int n=3;// Количество символов в самой длинной строке

int dictCount=3;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=1;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"да");

strcpy(text[1],"да");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[1];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"да");

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestDifferentFrequencyAllRecorded()//! Все пары повторяются разное число раз, свободных символов хватило на все

{

int m=3;// Количество строк в тексте

int n=9;// Количество символов в самой длинной строке

int dictCount=20;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=3;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"бб");

strcpy(text[1],"жжж");

strcpy(text[2],"ааааааа");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[3];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"аа");

strcpy(expectDictionary[1].oftenRepeatedCouple,"жж");

strcpy(expectDictionary[2].oftenRepeatedCouple,"бб");

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestDifferentFrequencyNotAllRecorded()//! Все пары повторяются разное число раз, свободных символов хватило не на все

{

int m=3;// Количество строк в тексте

int n=9;// Количество символов в самой длинной строке

int dictCount=2;// Изначальный размер словаря, в котором записаны только свободные символы

int expectDictCount=2;// Размер словаря, который должен получиться после записи в него пар

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"бб");

strcpy(text[1],"жжж");

strcpy(text[2],"ааааааа");

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

DictString expectDictionary[2];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"аа");

strcpy(expectDictionary[1].oftenRepeatedCouple,"жж");

findPopularCouples(text,m,n,dictionary,&dictCount);//вызываем тестируемую функцию

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

char \*\*errors = (char \*\*)malloc(6\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

free(dictionary);

}

**Модульные тесты для подпрограммы findText**

void TestNoText()//! Текста нет

{

int result;//Возвращаемое значение функции

int expectResult=0;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount=3;//количетсво пар в словаре

int charCount=5;//Максимальное число символов в строке

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"на+");

strcpy(text[2],"а ъ");

strcpy(text[3],"...");

result=findText(text,&stringsCount,charCount,dictCount);

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

free(text);

}

void TestNoError()//! Ошибок нет

{

Check check;

int expectStringsCount=2;

int expectCharCount=20;

int result;//Возвращаемое значение функции

int expectResult=1;//ожидаемое возвращаемое значение

int stringsCount=4;//Число строк в тексте

int dictCount=1;//количетсво пар в словаре

int charCount=20;//Максимальное число символов в строке

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < stringsCount; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"ая\*");

strcpy(text[1],"...");

strcpy(text[2],"Вот начался текст");

strcpy(text[3],"И закончился");

char \*\*expectText = (char \*\*)malloc(expectStringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

expectText[i] = (char \*)malloc(expectCharCount\*sizeof(char));

}

strcpy(expectText[0],"Вот начался текст");

strcpy(expectText[1],"И закончился");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

result=findText(text,&stringsCount,charCount,dictCount);

if(result!=expectResult)//проверяем, правильно ли функция определила наличие ошибок

CFIX\_ASSERT\_MESSAGE(result==expectResult, L"Неправильно определено наличие ошибок");

else

{

check.check\_findText(text,stringsCount,charCount,expectText,expectStringsCount,expectCharCount,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

}

for(int i=0; i<4; i++) // цикл по строкам

{

free(text[i]);// освобождение памяти под строку

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

free(text);

for(int i=0; i<2; i++) // цикл по строкам

{

free(expectText[i]);// освобождение памяти под строку

}

free(expectText);

}

**Модульные тесты для подпрограммы isFinded**

void TestFirsTime()//! Такая пара найдена впервые

{

Couple couple;// Проверяемая пара

strcpy(couple.couple,"ла");

int coupleCount=4;// Количество уже найденных пар

Couple\* couples=(Couple\*)malloc(coupleCount\*sizeof(Couple));// Массив с ранее найденными парами

strcpy(couples[0].couple,"на");

strcpy(couples[1].couple,"ка");

strcpy(couples[2].couple,"жа");

strcpy(couples[3].couple,"ша");

int result=isFinded(couple,couples,coupleCount);

CFIX\_ASSERT\_MESSAGE(result==-1,L"Ошибка, пара найдена впервые");

}

void TestAlreadyFound()//! Такая пара уже была найдена

{

Couple couple;// Проверяемая пара

strcpy(couple.couple,"ла");

int coupleCount=4;// Количество уже найденных пар

Couple\* couples=(Couple\*)malloc(coupleCount\*sizeof(Couple));// Массив с ранее найденными парами

strcpy(couples[0].couple,"на");

strcpy(couples[1].couple,"ка");

strcpy(couples[2].couple,"ла");

strcpy(couples[3].couple,"ша");

int result=isFinded(couple,couples,coupleCount);

CFIX\_ASSERT\_MESSAGE(result==2,L"Ошибка, пара уже была найдена");

}

**Модульные тесты для подпрограммы isValidCmdArgs**

void LessThan3Args() //! Количество аргументов командной строки меньше 3х

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(100); // Имя файла

char\*\* argv=(char\*\*)malloc(2\*sizeof(char\*)); // Аргументы командной строки

argv[0]=(char\*)malloc(11\*sizeof(char));

strcpy(argv[0],"Packer.exe");

argv[1]=(char\*)malloc(2\*sizeof(char));

strcpy(argv[1],"p");

// Вызываем тестируемую функцию

int success=isValidCmdArgs(2,argv,filename);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Разбор аргументов командной строки прошел неуспешно");

// Освобождаем выделенную для тестирования память

free(filename);

for(int i=0; i<2; i++)

free(argv[i]);

free(argv);

}

void MoreThan3Args() //! Количество аргументов командной строки больше 3х

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(100); // Имя файла

char\*\* argv=(char\*\*)malloc(4\*sizeof(char\*)); // Аргументы командной строки

argv[0]=(char\*)malloc(11\*sizeof(char));

strcpy(argv[0],"Packer.exe");

argv[1]=(char\*)malloc(2\*sizeof(char));

strcpy(argv[1],"p");

argv[2]=(char\*)malloc(21\*sizeof(char));

strcpy(argv[2],"C:\\folder\\text.txt");

argv[3]=(char\*)malloc(6\*sizeof(char));

strcpy(argv[3],"param");

// Вызываем тестируемую функцию

int success=isValidCmdArgs(4,argv,filename);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Разбор аргументов командной строки прошел неуспешно");

// Освобождаем выделенную для тестирования память

free(filename);

for(int i=0; i<4; i++)

free(argv[i]);

free(argv);

}

void Incorrect2Arg() //! Количество аргументов командной строки 3, но 2-ый некорректен

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(100); // Имя файла

char\*\* argv=(char\*\*)malloc(3\*sizeof(char\*)); // Аргументы командной строки

argv[0]=(char\*)malloc(11\*sizeof(char));

strcpy(argv[0],"Packer.exe");

argv[1]=(char\*)malloc(3\*sizeof(char));

strcpy(argv[1],"pp");

argv[2]=(char\*)malloc(21\*sizeof(char));

strcpy(argv[2],"C:\\folder\\text.txt");

// Вызываем тестируемую функцию

int success=isValidCmdArgs(3,argv,filename);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Разбор аргументов командной строки прошел неуспешно");

// Освобождаем выделенную для тестирования память

free(filename);

for(int i=0; i<3; i++)

free(argv[i]);

free(argv);

}

void PackMode() //! 2-ой аргумент - флаг упаковки

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(100); // Имя файла

char\*\* argv=(char\*\*)malloc(3\*sizeof(char\*)); // Аргументы командной строки

argv[0]=(char\*)malloc(11\*sizeof(char));

strcpy(argv[0],"Packer.exe");

argv[1]=(char\*)malloc(2\*sizeof(char));

strcpy(argv[1],"p");

argv[2]=(char\*)malloc(21\*sizeof(char));

strcpy(argv[2],"C:\\folder\\text.txt");

// Вызываем тестируемую функцию

int success=isValidCmdArgs(3,argv,filename);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Разбор аргументов командной строки прошел неуспешно");

CFIX\_ASSERT\_MESSAGE(strcmp("C:\\folder\\text.txt",filename)==0,L"Вместо имени \"C:\\folder\\text.txt\" было получено \"%s\"",filename);

// Освобождаем выделенную для тестирования память

free(filename);

for(int i=0; i<3; i++)

free(argv[i]);

free(argv);

}

void UnpackMode() //! 2-ой аргумент - флаг распаковки

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(100); // Имя файла

char\*\* argv=(char\*\*)malloc(4\*sizeof(char\*)); // Аргументы командной строки

argv[0]=(char\*)malloc(11\*sizeof(char));

strcpy(argv[0],"Packer.exe");

argv[1]=(char\*)malloc(3\*sizeof(char));

strcpy(argv[1],"u");

argv[2]=(char\*)malloc(21\*sizeof(char));

strcpy(argv[2],"C:\\folder\\text.txt");

// Вызываем тестируемую функцию

int success=isValidCmdArgs(3,argv,filename);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Разбор аргументов командной строки прошел неуспешно");

CFIX\_ASSERT\_MESSAGE(strcmp("C:\\folder\\text.txt",filename)==0,L"Вместо имени \"C:\\folder\\text.txt\" было получено \"%s\"",filename);

// Освобождаем выделенную для тестирования память

free(filename);

for(int i=0; i<3; i++)

free(argv[i]);

free(argv);

}

**Модульные тесты для подпрограммы pack**

void TestOneString()//! Текст состоит из одной строки

{

int m=1;// Количество строк в тексте

int n=30;// Максимальное количество символов в строке

int count=0;//количество пар в словаре=0;

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нанала");

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"нанала");

int expectCount=4;

DictString expectDict[4];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"на");

expectDict[0]=dict;

strcpy(dict.oftenRepeatedCouple,"ла");

expectDict[1]=dict;

strcpy(dict.oftenRepeatedCouple,"ан");

expectDict[2]=dict;

strcpy(dict.oftenRepeatedCouple,"ал");

expectDict[3]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,1,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestFewStrings()//! Текст состоит из нескольких строк

{

int m=2;//количество строк

int n=15;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"Надо");

strcpy(text[1],"надо");//текст, который необходимо упаковать

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"Надо");

strcpy(copyText[1],"надо");//текст, который необходимо упаковать

int expectCount=4;

DictString expectDict[4];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"На");

expectDict[0]=dict;

strcpy(dict.oftenRepeatedCouple,"ад");

expectDict[1]=dict;

strcpy(dict.oftenRepeatedCouple,"до");

expectDict[2]=dict;

strcpy(dict.oftenRepeatedCouple,"на");

expectDict[3]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,2,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestFewEmptyStrings()//! В тексте несколько пустых строк

{

int m=5;//количество строк

int n=15;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"Надо");

strcpy(text[1],"надо");

strcpy(text[2],"");

strcpy(text[3],"Надо");

strcpy(text[4],"");

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"Надо");

strcpy(copyText[1],"надо");

strcpy(copyText[2],"");

strcpy(copyText[3],"Надо");

strcpy(copyText[4],"");

int expectCount=4;

DictString expectDict[4];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"На");

expectDict[0]=dict;

strcpy(dict.oftenRepeatedCouple,"ад");

expectDict[1]=dict;

strcpy(dict.oftenRepeatedCouple,"до");

expectDict[2]=dict;

strcpy(dict.oftenRepeatedCouple,"на");

expectDict[3]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,5,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestNoText()//! Текста нет

{

int m=0;//количество строк

int n=0;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

int expectCount=0;

pack(text,m,n,dictionary,&count);

if(count>expectCount)// Проверяем, что пар найдено не больше, чем нужно

CFIX\_ASSERT\_MESSAGE(count==expectCount, L"Заменено слишком много пар");

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestTextConsistsOfOnePair()//! Текст содержит только одну пару

{

int m=1;//количество строк

int n=3;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре=(DictString\* )malloc(m\*n\*sizeof(DictString\*));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"Бу");//текст, который необходимо упаковать

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"Бу");//текст, который необходимо упаковать

int expectCount=1;

DictString expectDict[1];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"Бу");

expectDict[0]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,1,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestEquifrequentPairs()//! Все пары в тексте встречаются равное число раз

{

int m=5;//количество строк

int n=6;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"па");

strcpy(text[1],"ра");

strcpy(text[2],"та");

strcpy(text[3],"ту");

strcpy(text[4],"ам");

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"па");

strcpy(copyText[1],"ра");

strcpy(copyText[2],"та");

strcpy(copyText[3],"ту");

strcpy(copyText[4],"ам");

int expectCount=5;

DictString expectDict[5];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"па");

expectDict[0]=dict;

strcpy(dict.oftenRepeatedCouple,"ра");

expectDict[1]=dict;

strcpy(dict.oftenRepeatedCouple,"та");

expectDict[2]=dict;

strcpy(dict.oftenRepeatedCouple,"ту");

expectDict[3]=dict;

strcpy(dict.oftenRepeatedCouple,"ам");

expectDict[4]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,5,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestManySymbols()//! В тексте отсутствуют многие символы, их количество больше, чем количество популярных пар

{

int m=1;//количество строк

int n=26;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"мало");

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"мало");

int expectCount=3;

DictString expectDict[3];//пары в словаре, которые должны были быть найдены

DictString dict;

strcpy(dict.oftenRepeatedCouple,"ма");

expectDict[0]=dict;

strcpy(dict.oftenRepeatedCouple,"ал");

expectDict[1]=dict;

strcpy(dict.oftenRepeatedCouple,"ло");

expectDict[2]=dict;

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check\_pack(copyText,1,count,expectCount,dictionary,expectDict,errors,&errorsCount);

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

void TestPairsMoreThanCharacters()//! Пар больше, чем отсутствующих символов

{

int m=1;//количество строк

int n=329;//количетсво символов в максимальной строке

int count=0;//количество пар в словаре

DictString\* dictionary=(DictString\* )malloc(256\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре;//Массив объектов структуры DictString, содерщащий информацию о словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++) //выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"ааабббвввгггдддеееёёёжжжзззииийййккклллмммнннооопппрррссстттуууфффхххцццчччшшшщщщъъъыыыьььэээюююяяяАААБББВВВГГГДДДЕЕЕЁЁЁЖЖЖЗЗЗИИИЙЙЙКККЛЛЛМММНННОООПППРРРССТТУФУХЦФЧШХЩЪЫЬЭЮЯЇЅЅј»Є№Ё·¶µґіІ±°Ї®¬«Є©§¦Ґ¤ЈўЎџќњ›љ™—–•”“’‘ЂЏћЋЌЊ‹Љ‰€‡†…„ѓ‚ЃЂ~}|{QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm`\_^]\

[@?>=<;:1234567890/.—,+\*)(‘&%$#«! \n");

char \*\*copyText = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

copyText[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(copyText[0],"ааабббвввгггдддеееёёёжжжзззииийййккклллмммнннооопппрррссстттуууфффхххцццчччшшшщщщъъъыыыьььэээюююяяяАААБББВВВГГГДДДЕЕЕЁЁЁЖЖЖЗЗЗИИИЙЙЙКККЛЛЛМММНННОООПППРРРССТТУФУХЦФЧШХЩЪЫЬЭЮЯЇЅЅј»Є№Ё·¶µґіІ±°Ї®¬«Є©§¦Ґ¤ЈўЎџќњ›љ™—–•”“’‘ЂЏћЋЌЊ‹Љ‰€‡†…„ѓ‚ЃЂ~}|{QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm`\_^]\

[@?>=<;:1234567890/.—,+\*)(‘&%$#«! \n");

pack(text,m,n,dictionary,&count);

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;

for(int i=0;i<count;i++)// Проверяем, что ключей, использованных в словаре нет в тексте

{//для каждого ключа

int error=0;

for(int j=0;j<m && error==0;j++)//для каждой строки текста

if(strchr(copyText[j],\*dictionary[i].coupleCode)!=0)//если ключ и символ текста равны

{

error=1;

char message[95]={"Как ключ использован символ, который есть в тексте: "};

strncat(message,dictionary[i].coupleCode,1);

strcpy(errors[errorsCount],message);

errorsCount++;

}

}

for(int i=0;i<count;i++)// Проверяем, что ключи в словаре не повторяются

{

int error=0;

for(int j=0;j<count && error==0;j++)

{

if(strncmp(dictionary[i].coupleCode,dictionary[j].coupleCode,1)==0 && i!=j)

{

error=1;

char message[50]={"Ключ "};

strncat(message,dictionary[i].coupleCode,1);

strcat(message," используется в словаре несколько раз");

strcpy(errors[errorsCount],message);

errorsCount++;

}

}

}

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

{

WCHAR Mes[100];

MultiByteToWideChar(CP\_ACP,0,errors[i],strlen(errors[i])+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(1==2, Mes);

}

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(copyText[i]); // освобождение памяти под строку

}

free(copyText);

}

**Модульные тесты для подпрограммы readFile**

void TestOneStr()//! Файл содержит только одну строку

{

char filename[]={"OneString.txt"};// Имя файла, с которого считываем

int expectM=1;//ожидаемое число строк

int expectN=43;//ожидаемая длина строки

char \*\*text;// Текст, который считали с файла

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

char expectText[1][44]={{"Вот текст, который состоит из одной строки."}};

CFIX\_ASSERT\_MESSAGE(text!=0, L"Неправильно определено наличие ошибки");

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

for (int i=0;i<expectM;i++)//для каждой строки

{

char message[30]={"Неправильно считана строка "};

char str[3];

itoa(i+1,str,10);

strcat(message,str);

WCHAR Mes[30];

MultiByteToWideChar(CP\_ACP,0,message,strlen(message)+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(strncmp(text[i],expectText[i],strlen(expectText[i]))==0, Mes);

}

free(text[0]); // освобождение памяти под строку

free(text);

}

void TestNoFile()//! Такого файла нет

{

char filename[]={"NoFile.txt"};// Имя файла, с которого считываем

int expectM=1;//ожидаемое число строк

int expectN=1;//ожидаемая длина строки

char \*\*text;// Текст, который считали с файла

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

CFIX\_ASSERT\_MESSAGE(text==0, L"Неправильно определено наличие ошибки");

free(text);

}

void TestFileGreaterThan10000()//! Файл больше 10000 байт

{

char filename[]={"FileGreaterThan10000.txt"};// Имя файла, с которого считываем

int expectM=521;//ожидаемое число строк

int expectN=65;//ожидаемая длина строки

char \*\*text;// Текст, который считали с файла

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

char expectText[4][65]={{"Очень большой текст\n"},{"Бывало, он еще в постеле:\n"},{"Двойной лорнет скосясь наводит\n"},{"На ложи незнакомых дам;\n"}};

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

CFIX\_ASSERT\_MESSAGE(strncmp(text[0],expectText[0],strlen(expectText[0]))==0, L"Неправильно считана строка 1");

CFIX\_ASSERT\_MESSAGE(strncmp(text[193],expectText[1],strlen(expectText[1]))==0, L"Неправильно считана строка 194");

CFIX\_ASSERT\_MESSAGE(strncmp(text[384],expectText[2],strlen(expectText[2]))==0, L"Неправильно считана строка 385");

CFIX\_ASSERT\_MESSAGE(strncmp(text[371],expectText[3],strlen(expectText[3]))==0, L"Неправильно считана строка 468");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

}

void TestFileLessThan10000()//! Файл весит меньше 10000 байт

{

char filename[]={"FileLessThan10000.txt"};

int expectM=4;//ожидаемое число строк

int expectN=32;//ожидаемая длина строки

char \*\*text;// Текст, который считали с файла

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

char expectText[4][35]={{"«Мой дядя самых честных правил,\n"},{"Когда не в шутку занемог,\n"},{"Он уважать себя заставил\n"},{"И лучше выдумать не мог.!»"}};

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

for (int i=0;i<expectM;i++)//для каждой строки

{

char message[30]={"Неправильно считана строка "};

char str[3];

itoa(i+1,str,10);

strcat(message,str);

WCHAR Mes[30];

MultiByteToWideChar(CP\_ACP,0,message,strlen(message)+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(strncmp(text[i],expectText[i],strlen(expectText[i]))==0, Mes);

}

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

}

void TestEmptyFile()//! Файл пуст

{

char filename[]={"EmptyFile.txt"};

int expectM=0;//ожидаемое число строк

int expectN=0;//ожидаемая длина строки

char \*\*text;// Текст, который считали с файла

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

}

void TestFewStrings()//! Файл содержит несколько строк

{

char filename[]={"FewStrings.txt"};

int expectM=4;//ожидаемое число строк

int expectN=48;//ожидаемая длина строки

char\*\* text=(char\*\*)malloc(sizeof(char\*));

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

char expectText[4][50]={{"Это текст, который состоит из нескольких строк.\n"},{"Это вторая строка.\n"},{"А это третья строка.\n"},{"Конец многострочного текста."}};

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

for (int i=0;i<expectM;i++)//для каждой строки

{

char message[30]={"Неправильно считана строка "};

char str[3];

itoa(i+1,str,10);

strcat(message,str);

WCHAR Mes[30];

MultiByteToWideChar(CP\_ACP,0,message,strlen(message)+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(strncmp(text[i],expectText[i],strlen(expectText[i]))==0, Mes);

}

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

}

void TestCppFile()//! Текстовый файл с расширением cpp

{

char filename[]={"CppFile.cpp"};

int expectM=2;//ожидаемое число строк

int expectN=46;//ожидаемая длина строки

char\*\* text=(char\*\*)malloc(sizeof(char\*));

int stringsCount;// Считанное количество строк

int charCount;// Максимальная длина считанной строки

text=readFile(filename,&stringsCount,&charCount);

char expectText[2][48]={{"Обычный текстовый файл, но с расширением cpp.\n"},{"Конец файла"}};

CFIX\_ASSERT\_MESSAGE(stringsCount==expectM, L"Неправильное число считанных строк");

CFIX\_ASSERT\_MESSAGE(charCount==expectN, L"Неверно найдена длина максимальной строки");

for (int i=0;i<expectM;i++)//для каждой строки

{

char message[30]={"Неправильно считана строка "};

char str[3];

itoa(i+1,str,10);

strcat(message,str);

WCHAR Mes[30];

MultiByteToWideChar(CP\_ACP,0,message,strlen(message)+1,(LPWSTR)Mes,sizeof(Mes)/sizeof(Mes[0]));

CFIX\_ASSERT\_MESSAGE(strncmp(text[i],expectText[i],strlen(expectText[i]))==0, Mes);

}

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

}

**Модульные тесты для подпрограммы repAllEntries**

void EmptyStr() //! Строка str пустая, toBeReplaced и replacement не пустые

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(1); // Входная строка, в которой необходимо произвести замену

strcpy(str,"\0");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"еч");

char\* replacement=(char\*)malloc(2);

strcpy(replacement,"\*");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"\0")==0,L"Вместо пустой строки была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void EmptyToBeReplaced() //! Строка toBeReplaced пустая, str и replacement не пустые

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(12); // Входная строка, в которой необходимо произвести замену

strcpy(str,"течет речка");

char\* toBeReplaced=(char\*)malloc(1);

strcpy(toBeReplaced,"\0");

char\* replacement=(char\*)malloc(2);

strcpy(replacement,"\*");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"течет речка")==0,L"Вместо строки \"течет речка\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void EmptyReplacement() //! Строка replacement пустая, str и toBeReplaced не пустые

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(12); // Входная строка, в которой необходимо произвести замену

strcpy(str,"течет речка");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"еч");

char\* replacement=(char\*)malloc(1);

strcpy(replacement,"\0");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"течет речка")==0,L"Вместо строки \"течет речка\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void NoToBeReplacedInStr() //! Нет ни одного вхождения строки toBeReplaced

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(12); // Входная строка, в которой необходимо произвести замену

strcpy(str,"течет речка");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"гы");

char\* replacement=(char\*)malloc(1);

strcpy(replacement,"\0");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"течет речка")==0,L"Вместо строки \"течет речка\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void OneEntryInStr() //! Одно вхождение toBeReplaced в переданной строке

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(12); // Входная строка, в которой необходимо произвести замену

strcpy(str,"течет речка");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"те");

char\* replacement=(char\*)malloc(2);

strcpy(replacement,"\*");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"\*чет речка")==0,L"Вместо строки \"\*чет речка\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void MoreThanOneEntryInStr() //! Несколько вхождений toBeReplaced в переданной строке

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(25); // Входная строка, в которой необходимо произвести замену

strcpy(str,"течет речка, печет печка");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"еч");

char\* replacement=(char\*)malloc(2);

strcpy(replacement,"\*");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"т\*ет р\*ка, п\*ет п\*ка")==0,L"Вместо строки \"т\*ет р\*ка, п\*ет п\*ка\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

void OneEntryAndNothingElse() //! Строка состоит только из одного вхождения toBeReplaced

{

// Подготавливаем данные для тестирования

char\* str=(char\*)malloc(3); // Входная строка, в которой необходимо произвести замену

strcpy(str,"еч");

char\* toBeReplaced=(char\*)malloc(3);

strcpy(toBeReplaced,"еч");

char\* replacement=(char\*)malloc(2);

strcpy(replacement,"\*");

// Вызываем тестируемую функцию

int success=repAllEntries(str, toBeReplaced, replacement);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(str,"\*")==0,L"Вместо строки \"\*\" была получена строка \"%s\"",str);

// Освобождаем выделенную для тестирования память

free(str);

free(toBeReplaced);

free(replacement);

}

**Модульные тесты для подпрограммы repFirstNChars**

void EmptyStrToNGreater0() //! Строка strTo пустая, n > 0

{

// Подготавливаем данные для тестирования

char\* strTo=NULL; // Входная строка, в которой надо заменить символы

char strFrom[4]="str"; // Строка, которой заменяются символы в strTo

int n=2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

}

void NLess0() //! n < 0

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(5\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str1");

char strFrom[5]="str2"; // Строка, которой заменяются символы в strTo

int n=-2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(strTo);

}

void NGreaterStrToLen() //! n больше длины строки strTo

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(5\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str1");

char strFrom[5]="str2"; // Строка, которой заменяются символы в strTo

int n=6; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(strTo);

}

void EmptyStrFrom() //! strFrom пустая при нормальных strTo и n

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(5\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str1");

char\* strFrom=NULL; // Строка, которой заменяются символы в strTo

int n=3; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(strTo);

}

void NEq0FullStrToAndStrFrom() //! n == 0 при непустых strTo и strFrom

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(5\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str1");

char strFrom[5]="str2"; // Строка, которой заменяются символы в strTo

int n=0; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(strTo);

}

void NEq0EmptyStrToAndStrFrom() //! n == 0 при пустых strTo и strFrom

{

// Подготавливаем данные для тестирования

char\* strTo=NULL; // Входная строка, в которой надо заменить символы

char\* strFrom=NULL; // Строка, которой заменяются символы в strTo

int n=0; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

}

void NG0EmptyStrFromFullStrTo() //! Строка strFrom пустая, строка strTo непустая и n > 0

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(5\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str1");

char\* strFrom=NULL; // Строка, которой заменяются символы в strTo

int n=2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(strTo);

}

void NG0EmptyStrFromAndStrTo() //! Строка strFrom пустая, строка strTo пустая и n > 0

{

// Подготавливаем данные для тестирования

char\* strTo=NULL; // Входная строка, в которой надо заменить символы

char\* strFrom=NULL; // Строка, которой заменяются символы в strTo

int n=2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция произвела замену, хотя не должна была");

}

void NRepN() //! Замена n символов на n символов

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(4\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str");

char strFrom[3]="in"; // Строка, которой заменяются символы в strTo

int n=2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(strTo,"inr")==0,L"Вместо строки \"inr\" была получена \"%s\"",strTo);

// Освобождаем выделенную для тестирования память

free(strTo);

}

void GreaterNumRepSmaller() //! Замена большего количества символов на меньшее

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(7\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"string");

char strFrom[2]="w"; // Строка, которой заменяются символы в strTo

int n=3; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(strTo,"wing")==0,L"Вместо строки \"wing\" была получена \"%s\"",strTo);

// Освобождаем выделенную для тестирования память

free(strTo);

}

void SmallerNumRepGreater() //! Замена меньшего количества символов на большее

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(7\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str");

char strFrom[6]="peppa"; // Строка, которой заменяются символы в strTo

int n=2; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(strTo,"peppar")==0,L"Вместо строки \"peppar\" была получена \"%s\"",strTo);

// Освобождаем выделенную для тестирования память

free(strTo);

}

void RepAllChars() //! Замена всех символов строки strTo

{

// Подготавливаем данные для тестирования

char\* strTo=(char\*)malloc(6\*sizeof(char)); // Входная строка, в которой надо заменить символы

strcpy(strTo,"str");

char strFrom[6]="peppa"; // Строка, которой заменяются символы в strTo

int n=3; // Количество заменяемых символов

// Вызываем тестируемую функцию

int success=repFirstNChars(strTo,n,strFrom);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Замена не была произведена");

CFIX\_ASSERT\_MESSAGE(strcmp(strTo,"peppa")==0,L"Вместо строки \"peppar\" была получена \"%s\"",strTo);

// Освобождаем выделенную для тестирования память

free(strTo);

}

**Модульные тесты для подпрограммы replaceChars**

void EmptyText() //! Входной текст и словарь пустые

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(sizeof(char));

strcpy(text[0],"\0");

DictString\* dict=(DictString\*)malloc(sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].coupleCode,"\0"); strcpy(dict[0].oftenRepeatedCouple,"\0");

// Вызываем тестируемую функцию

int success=replaceChars(text,0,0,dict,0,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("\0",text[0])==0,L"Вместо пустого текста была получена строка \"%s\"",text[0]);

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void EmptyTextFilledDict() //! Словарь непустой, текст пустой

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(sizeof(char));

strcpy(text[0],"\0");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int dictCount=3;

int success=replaceChars(text,0,0,dict,dictCount,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("\0",text[0])==0,L"Вместо пустого текста была получена строка \"%s\"",text[0]);

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void FilledTextEmptyDict() //! Текст непустой, словарь пустой

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(12\*sizeof(char));

strcpy(text[0],"течет речка");

DictString\* dict=(DictString\*)malloc(sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].coupleCode,"\0"); strcpy(dict[0].oftenRepeatedCouple,"\0");

// Вызываем тестируемую функцию

int success=replaceChars(text,1,12,dict,0,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("течет речка",text[0])==0,L"Вместо строки \"течет речка\" была получена \"%s\"",text[0]);

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void NoDictCouplesInText() //! В тексте отсутствуют пары из словаря

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(22\*sizeof(char));

strcpy(text[0],"течет речка");

DictString\* dict=(DictString\*)malloc(3\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"пи"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"гы"); strcpy(dict[1].coupleCode,"+");

// Вызываем тестируемую функцию

int dictCount=2;

int success=replaceChars(text,0,0,dict,dictCount,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("течет речка",text[0])==0,L"Вместо строки \"течет речка\" была получена \"%s\"",text[0]);

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void SomeDictCouplesNotInText() //! Не все пары из словаря присутствуютт в тексте

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(23\*sizeof(char));

strcpy(text[0],"течет речка");

text[1]=(char\*)malloc(23\*sizeof(char));

strcpy(text[1],"печет печка");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"пи"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int dictCount=3;

int success=replaceChars(text,2,23,dict,dictCount,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("т\*+ р\*ка",text[0])==0,L"Вместо строки \"т\*+ р\*ка\" была получена \"%s\"",text[0]);

CFIX\_ASSERT\_MESSAGE(strcmp("п\*+ п\*ка",text[1])==0,L"Вместо строки \"п\*+ п\*ка\" была получена \"%s\"",text[1]);

// Освобождаем выделенную для тестирования память

for(int i=0; i<2; i++)

free(text[i]);

free(text);

free(dict);

}

void PackMode() //! Режим упаковки

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(12\*sizeof(char));

strcpy(text[0],"течет речка");

text[1]=(char\*)malloc(12\*sizeof(char));

strcpy(text[1],"печет печка");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int dictCount=3;

int success=replaceChars(text,2,12,dict,dictCount,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("т\*+ р\*,",text[0])==0,L"Вместо строки \"т\*+ р\*,\" была получена \"%s\"",text[0]);

CFIX\_ASSERT\_MESSAGE(strcmp("п\*+ п\*,",text[1])==0,L"Вместо строки \"п\*+ п\*,\" была получена \"%s\"",text[1]);

// Освобождаем выделенную для тестирования память

for(int i=0; i<2; i++)

free(text[i]);

free(text);

}

void UnpackMode() //! Режим распаковки

{

// Подготавливаем данные для тестирования

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Входной текст

text[0]=(char\*)malloc(12\*sizeof(char));

strcpy(text[0],"т\*+ р\*,");

text[1]=(char\*)malloc(12\*sizeof(char));

strcpy(text[1],"п\*+ п\*,");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int dictCount=3;

int success=replaceChars(text,2,12,dict,dictCount,0);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"В текст были внесены ненужные изменения");

CFIX\_ASSERT\_MESSAGE(strcmp("течет речка",text[0])==0,L"Вместо строки \"течет речка\" была получена \"%s\"",text[0]);

CFIX\_ASSERT\_MESSAGE(strcmp("печет печка",text[1])==0,L"Вместо строки \"печет печка\" была получена \"%s\"",text[1]);

// Освобождаем выделенную для тестирования память

for(int i=0; i<2; i++)

free(text[i]);

free(text);

}

**Модульные тесты для подпрограммы sort**

void TestChanges()//! Функция вносила изменения в порядок массива

{

int coupleCount=4;// Количество пар в массиве

Couple\* couples=(Couple\*)malloc(coupleCount\*sizeof(Couple)); // Массив найденных пар, который необходимо упорядочить

strcpy(couples[0].couple,"на");

couples[0].coupleAmount=7;

strcpy(couples[1].couple,"ка");

couples[1].coupleAmount=6;

strcpy(couples[2].couple,"ла");

couples[2].coupleAmount=11;

strcpy(couples[3].couple,"ша");

couples[3].coupleAmount=8;

sort(couples,coupleCount);

CFIX\_ASSERT\_MESSAGE(strcmp(couples[0].couple,"ла")==0,L"Ошибка, первая пара должна быть- ла");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[1].couple,"ша")==0,L"Ошибка, вторая пара должна быть- ша");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[2].couple,"на")==0,L"Ошибка, третья пара должна быть- на");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[3].couple,"ка")==0,L"Ошибка, последняя пара должна быть- ка");

}

void TestNoChanges()//! Функция не вносила изменения в порядок массива

{

int coupleCount=4;// Количество пар в массиве

Couple\* couples=(Couple\*)malloc(coupleCount\*sizeof(Couple)); // Массив найденных пар, который необходимо упорядочить

strcpy(couples[0].couple,"на");

couples[0].coupleAmount=12;

strcpy(couples[1].couple,"ка");

couples[1].coupleAmount=11;

strcpy(couples[2].couple,"ла");

couples[2].coupleAmount=8;

strcpy(couples[3].couple,"ша");

couples[3].coupleAmount=7;

sort(couples,coupleCount);

CFIX\_ASSERT\_MESSAGE(strcmp(couples[0].couple,"на")==0,L"Ошибка, первая пара должна быть- на");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[1].couple,"ка")==0,L"Ошибка, вторая пара должна быть- ка");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[2].couple,"ла")==0,L"Ошибка, третья пара должна быть- ла");

CFIX\_ASSERT\_MESSAGE(strcmp(couples[3].couple,"ша")==0,L"Ошибка, последняя пара должна быть- ша");

}

**Модульные тесты для подпрограммы transDictStructToMass**

void EmptyDict() //! Пустой словарь

{

// Подготавливаем данные для тестирования

DictString\* dict=(DictString\*)malloc(2\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].coupleCode,"\0"); strcpy(dict[0].oftenRepeatedCouple,"\0");

char\*\* textDict=(char\*\*)malloc(sizeof(char\*)); // Словарь-массив

// Вызываем тестируемую функцию

textDict=transDictStructToMass(dict,0);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(textDict==NULL,L"Возвращаемый массив не пуст");

// Освобождаем выделенную для тестирования память

free(textDict);

free(dict);

}

void DictSizeIsOne() //! В словаре одна пара и один код

{

// Подготавливаем данные для тестирования

DictString\* dict=(DictString\*)malloc(2\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

char\*\* textDict=(char\*\*)malloc(sizeof(char\*)); // Словарь-массив

textDict[0]=(char\*)malloc(4\*sizeof(char));

// Вызываем тестируемую функцию

textDict=transDictStructToMass(dict,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(strcmp(textDict[0],"еч\*")==0,L"Вместо строки \"еч\*\" была получена строка \"%s\"",textDict[0]);

// Освобождаем выделенную для тестирования память

free(textDict[0]);

free(textDict);

free(dict);

}

void DictSizeIsMoreThanOne() //! В словаре несколько пар и кодов

{

// Подготавливаем данные для тестирования

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

char\*\* textDict=(char\*\*)malloc(4\*sizeof(char\*)); // Словарь-массив

textDict[0]=(char\*)malloc(4\*sizeof(char));

textDict[1]=(char\*)malloc(4\*sizeof(char));

textDict[2]=(char\*)malloc(4\*sizeof(char));

// Вызываем тестируемую функцию

textDict=transDictStructToMass(dict,3);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(strcmp(textDict[0],"еч\*")==0,L"Вместо строки \"еч\*\" была получена строка \"%s\"",textDict[0]);

CFIX\_ASSERT\_MESSAGE(strcmp(textDict[1],"ет+")==0,L"Вместо строки \"ет+\" была получена строка \"%s\"",textDict[1]);

CFIX\_ASSERT\_MESSAGE(strcmp(textDict[2],"ка,")==0,L"Вместо строки \"ка,\" была получена строка \"%s\"",textDict[2]);

// Освобождаем выделенную для тестирования память

for(int i=0; i<3; i++)

free(textDict[i]);

free(textDict);

free(dict);}

**Модульные тесты для подпрограммы unpack**

void EpmtyInput() //! Входной текст пуст

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(0\*sizeof(char));

strcpy(incorrectInput[0],"");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,0,0);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

free(incorrectInput);

}

void NoDictionary() //! Отсутствует словарь

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"т\*ет р\*ка");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,1,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

free(incorrectInput[0]);

free(incorrectInput);

}

void NoPackedText() //! Отсутствуеет упакованный текст

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(4\*sizeof(char));

strcpy(incorrectInput[0],"еч\*");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,1,3);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

free(incorrectInput[0]);

free(incorrectInput);

}

void NoSeparator() //! Между словарем и текстом нет специального разделителя

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(2\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"еч\*");

incorrectInput[1]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[1],"т\*ет р\*ка");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,2,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

for(int i=0; i<2; i++)

free(incorrectInput[i]);

free(incorrectInput);

}

void IncorrectSeparator() //! Между словарем и текстом неправильный разделитель

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(3\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"еч\*");

incorrectInput[1]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[1],"!!!");

incorrectInput[2]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[2],"т\*ет р\*ка");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,3,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

for(int i=0; i<3; i++)

free(incorrectInput[i]);

free(incorrectInput);

}

void PackedTextFirst() //! Сначала упакованный текст, а не словарь

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(3\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"т\*ет р\*ка");

incorrectInput[1]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[1],"...");

incorrectInput[2]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[2],"еч\*");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,3,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

for(int i=0; i<3; i++)

free(incorrectInput[i]);

free(incorrectInput);

}

void SpacerBetweenCoupleAndCode() //! В словаре между парой символов и кодом стоит пробел

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(3\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"еч \*");

incorrectInput[1]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[1],"...");

incorrectInput[2]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[2],"т\*ет р\*ка");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,3,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

for(int i=0; i<3; i++)

free(incorrectInput[i]);

free(incorrectInput);

}

void IcorrectCode() //! В словаре код состоит более, чем из одного символа

{

// Подготавливаем данные для тестирования

char\*\* incorrectInput=(char\*\*)malloc(3\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

incorrectInput[0]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[0],"еч\*\*");

incorrectInput[1]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[1],"...");

incorrectInput[2]=(char\*)malloc(10\*sizeof(char));

strcpy(incorrectInput[2],"т\*ет р\*ка");

// Вызываем тестируемую функцию

int success=unpack(incorrectInput,3,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"Функция unpack была завершена некорректно");

// Освобождаем выделенную память

for(int i=0; i<3; i++)

free(incorrectInput[i]);

free(incorrectInput);

}

void OneDictString() //! В словаре одна строка

{

// Подготавливаем данные для тестирования

char\*\* realOutput=(char\*\*)malloc(3\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

realOutput[0]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[0],"еч\*");

realOutput[1]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[1],"...");

realOutput[2]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[2],"т\*ет р\*ка");

char ExpectedOutput[22]={"течет речка"}; // Массив, содержащий распакованный текст

// Вызываем тестируемую функцию

int success=unpack(realOutput,3,9);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"Функция unpack была завершена некорректно");

CFIX\_ASSERT\_MESSAGE(strcmp(ExpectedOutput,realOutput[0])==0,L"Вместо строки \"течет речка\" была получена \"%s\"",realOutput);

// Освобождаем выделенную память

for(int i=0; i<3; i++)

free(realOutput[i]);

free(realOutput);

}

void SeveralDictStrings() //! В словаре несколько строк

{

// Подготавливаем данные для тестирования

char\*\* realOutput=(char\*\*)malloc(6\*sizeof(char\*)); // Массив, содержащий словарь и упакованный текст

realOutput[0]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[0],"еч\*");

realOutput[1]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[1],"ет+");

realOutput[2]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[2],"ка,");

realOutput[3]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[3],"...");

realOutput[4]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[4],"т\*+ р\*,");

realOutput[5]=(char\*)malloc(12\*sizeof(char));

strcpy(realOutput[5],"п\*+ п\*,");

char ExpectedOutput[2][22]= // Массив, содержащий распакованный текст

{"течет речка",

"печет печка"};

// Вызываем тестируемую функцию

int success=unpack(realOutput,6,7);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success!=0,L"Функция unpack была завершена некорректно");

for(int i=0; i<2; i++)

CFIX\_ASSERT\_MESSAGE(strcmp(ExpectedOutput[i],realOutput[i])==0,L"Вместо строки \"течет речка\" была получена \"%s\"",realOutput[i]);

// Освобождаем выделенную память

for(int i=0; i<6; i++)

free(realOutput[i]);

free(realOutput);

}

**Модульные тесты для подпрограммы check\_findDict**

void TestTooManyPairs()//! Пар больше чем нужно

{

Check check;

int dictCount=2;//количетсво пар в словаре

int expectDictCount=1;//количетсво пар в словаре

DictString\* dictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(dictionary[0].oftenRepeatedCouple,"ая");

strcpy(dictionary[0].coupleCode,".");

strcpy(dictionary[1].oftenRepeatedCouple,"ба");

strcpy(dictionary[1].coupleCode,"э");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,".");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Найдено слишком много пар\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestTooFewPairs()//! Пар меньше чем нужно

{

Check check;

int dictCount=1;//количетсво пар в словаре

int expectDictCount=2;//количетсво пар в словаре

DictString\* dictionary=(DictString\* )malloc(2\*sizeof(DictString));//Словарь, полученный из текста

strcpy(dictionary[0].oftenRepeatedCouple,"ая");

strcpy(dictionary[0].coupleCode,".");

DictString\* expectDictionary=(DictString\* )malloc(2\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,".");

strcpy(expectDictionary[1].oftenRepeatedCouple,"ба");

strcpy(expectDictionary[1].coupleCode,"э");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==2, L"Неверное число найденных ошибок");

char expectErrors[2][100]={{"Найдены не все пары\0"},{"Не найдена пара ба\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestWrongCode()//! У пары в словаре неправильный код

{

Check check;

int dictCount=1;//количетсво пар в словаре

int expectDictCount=1;//количетсво пар в словаре

DictString\* dictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(dictionary[0].oftenRepeatedCouple,"ая");

strcpy(dictionary[0].coupleCode,"\*");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,".");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Неверный код у пары ая\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestMissingPair()//! Одна пара не найдена функцией

{

Check check;

int dictCount=1;//количетсво пар в словаре

int expectDictCount=1;//количетсво пар в словаре

DictString\* dictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(dictionary[0].oftenRepeatedCouple,"ая");

strcpy(dictionary[0].coupleCode,".");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ла");

strcpy(expectDictionary[0].coupleCode,"ю");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Не найдена пара ла\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestNoError()//! Ошибок нет

{

Check check;

int dictCount=1;//количетсво пар в словаре

int expectDictCount=1;//количетсво пар в словаре

DictString\* dictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(dictionary[0].oftenRepeatedCouple,"ая");

strcpy(dictionary[0].coupleCode,".");

DictString\* expectDictionary=(DictString\* )malloc(10\*sizeof(DictString));//Словарь, полученный из текста

strcpy(expectDictionary[0].oftenRepeatedCouple,"ая");

strcpy(expectDictionary[0].coupleCode,".");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findDict(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==0, L"Неверное число найденных ошибок");

free(dictionary);

free(expectDictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

**Модульные тесты для подпрограммы check\_findPopularCouples**

void TestTooManyPairs()//! Пар больше чем нужно

{

int dictCount=3;// Число найденных пар

int expectDictCount=2;// Ожидаемое число пар

Check check;

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

strcpy(dictionary[0].oftenRepeatedCouple,"тр");

strcpy(dictionary[1].oftenRepeatedCouple,"мя");

strcpy(dictionary[2].oftenRepeatedCouple,"мя");

DictString expectDictionary[2];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"тр");

strcpy(expectDictionary[1].oftenRepeatedCouple,"мя");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество ошибок

check.check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Словарь слишком большой\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestTooFewPairs()//! Пар меньше чем нужно

{

int dictCount=2;// Число найденных пар

int expectDictCount=3;// Ожидаемое число пар

Check check;

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

strcpy(dictionary[0].oftenRepeatedCouple,"тр");

strcpy(dictionary[1].oftenRepeatedCouple,"мя");

DictString expectDictionary[3];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"тр");

strcpy(expectDictionary[1].oftenRepeatedCouple,"мя");

strcpy(expectDictionary[2].oftenRepeatedCouple,"ро");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество ошибок

check.check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==2, L"Неверное число найденных ошибок");

char expectErrors[2][100]={{"Словарь слишком маленький\0"},{"Не найдена пара ро"}};

int find=0;

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

{

for(int j=0;j<2;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestNoPair()//! Не найдена пара

{

int dictCount=2;// Число найденных пар

int expectDictCount=2;// Ожидаемое число пар

Check check;

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

strcpy(dictionary[0].oftenRepeatedCouple,"тр");

strcpy(dictionary[1].oftenRepeatedCouple,"ло");

DictString expectDictionary[2];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"тр");

strcpy(expectDictionary[1].oftenRepeatedCouple,"мя");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество ошибок

check.check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Не найдена пара мя\0"}};

int find=0;

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

{

for(int j=0;j<2;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

void TestNoError()//! Без ошибок

{

int dictCount=2;// Число найденных пар

int expectDictCount=2;// Ожидаемое число пар

Check check;

DictString\* dictionary=(DictString\*)malloc(dictCount\*sizeof(DictString));// Массив объектов структуры, содержащий все пары символов и количество повторений каждой пары

strcpy(dictionary[0].oftenRepeatedCouple,"тр");

strcpy(dictionary[1].oftenRepeatedCouple,"мя");

DictString expectDictionary[2];//пары, которые должны быть найдены

strcpy(expectDictionary[0].oftenRepeatedCouple,"тр");

strcpy(expectDictionary[1].oftenRepeatedCouple,"мя");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество ошибок

check.check\_findPopularCouples(dictionary,expectDictionary,dictCount,expectDictCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==0, L"Неверное число найденных ошибок");

free(dictionary);

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

}

**Модульные тесты для подпрограммы check\_findText**

void TestTooFewStr()//! В тексте оказалось слишком мало строк

{

Check check;

int stringsCount=1;//Число строк в тексте

int charCount=28;//Максимальное число символов в строке

int expectStringsCount=2;

int expectCharCount=28;

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"Под разделителем есть текст");

char \*\*expectText = (char \*\*)malloc(expectStringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

expectText[i] = (char \*)malloc(expectCharCount\*sizeof(char));

}

strcpy(expectText[0],"Тут текст");

strcpy(expectText[1],"Под разделителем есть текст");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findText(text,stringsCount,charCount,expectText,expectStringsCount,expectCharCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==2, L"Неверное число найденных ошибок");

char expectErrors[2][100]={{"Строк слишком мало\0"},{"Отсутствует строка под номером 0\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

for(int i=0; i<expectStringsCount; i++) // цикл по строкам

{

free(expectText[i]); // освобождение памяти под строку

}

free(expectText);

}

void TestTooManyStr()//! В тексте оказалось слишком много строк

{

Check check;

int stringsCount=2;//Число строк в тексте

int charCount=28;//Максимальное число символов в строке

int expectStringsCount=1;

int expectCharCount=28;

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"Тут текст");

strcpy(text[1],"Под разделителем есть текст");

char \*\*expectText = (char \*\*)malloc(expectStringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

expectText[i] = (char \*)malloc(expectCharCount\*sizeof(char));

}

strcpy(expectText[0],"Тут текст");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findText(text,stringsCount,charCount,expectText,expectStringsCount,expectCharCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Строк слишком много\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

for(int i=0; i<expectStringsCount; i++) // цикл по строкам

{

free(expectText[i]); // освобождение памяти под строку

}

free(expectText);

}

void TestNoStr()//! Ожидаемая строка текста не найдена

{

Check check;

int stringsCount=2;//Число строк в тексте

int charCount=28;//Максимальное число символов в строке

int expectStringsCount=2;

int expectCharCount=28;

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"Тут текст");

strcpy(text[1],"Под разделителем нет текста");

char \*\*expectText = (char \*\*)malloc(expectStringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

expectText[i] = (char \*)malloc(expectCharCount\*sizeof(char));

}

strcpy(expectText[0],"Тут текст");

strcpy(expectText[1],"Под разделителем есть текст");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findText(text,stringsCount,charCount,expectText,expectStringsCount,expectCharCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

char expectErrors[1][100]={{"Отсутствует строка под номером 1\0"}};

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

for(int i=0; i<expectStringsCount; i++) // цикл по строкам

{

free(expectText[i]); // освобождение памяти под строку

}

free(expectText);

}

void TestNoError()//! Ошибок нет

{

Check check;

int stringsCount=2;//Число строк в тексте

int charCount=28;//Максимальное число символов в строке

int expectStringsCount=2;

int expectCharCount=28;

char \*\*text = (char \*\*)malloc(stringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

text[i] = (char \*)malloc(charCount\*sizeof(char));

}

strcpy(text[0],"Тут текст");

strcpy(text[1],"Под разделителем есть текст");

char \*\*expectText = (char \*\*)malloc(expectStringsCount\*sizeof(char \*));// Текст, который считали с файла

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

{

expectText[i] = (char \*)malloc(expectCharCount\*sizeof(char));

}

strcpy(expectText[0],"Тут текст");

strcpy(expectText[1],"Под разделителем есть текст");

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));//Найденные ошибки

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount=0;// Количество найденных ошибок

check.check\_findText(text,stringsCount,charCount,expectText,expectStringsCount,expectCharCount,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==0, L"Неверное число найденных ошибок");

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<stringsCount; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

for(int i=0; i<expectStringsCount; i++) // цикл по строкам

{

free(expectText[i]); // освобождение памяти под строку

}

free(expectText);

}

**Модульные тесты для подпрограммы check\_pack**

void TestTooManyPairs()//! Найдено слишком много пар

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=3;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,"-");

strcpy(dict[1].oftenRepeatedCouple,"ан");

strcpy(dict[1].coupleCode,".");

strcpy(dict[2].oftenRepeatedCouple," ");

strcpy(dict[2].coupleCode,"'");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].coupleCode,"-");

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[1].coupleCode,".");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

char expectErrors[2][100]={{"Заменено слишком много пар\0"},{"В словаре найдена лишняя пара \0"}};

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==2, L"Неверное число найденных ошибок");

int find=0;

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

{

for(int j=0;j<2;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

void TestExtraPair()//! Найдена пара, которой не должно было быть

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=2;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,"-");

strcpy(dict[1].oftenRepeatedCouple,"ая");

strcpy(dict[1].coupleCode,".");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].coupleCode,"-");

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[1].coupleCode,".");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

char expectErrors[1][100]={{"В словаре найдена лишняя пара ая\0"}};

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

void TestKeyInText()//! Ключ в словаре был в тексте

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=2;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,"н");

strcpy(dict[1].oftenRepeatedCouple,"ан");

strcpy(dict[1].coupleCode,".");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].coupleCode,"-");

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[1].coupleCode,".");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

char expectErrors[1][100]={{"Как ключ использован символ, который есть в тексте: н\0"}};

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==1, L"Неверное число найденных ошибок");

int find=0;

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

{

for(int j=0;j<1;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

void TestTheSameKeys()//! В словаре встречаются несколько одинаковых ключей

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=2;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,",");

strcpy(dict[1].oftenRepeatedCouple,"ан");

strcpy(dict[1].coupleCode,",");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].coupleCode,"-");

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[1].coupleCode,".");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

char expectErrors[2][100]={{"Ключ , используется в словаре несколько раз\0"},{"Ключ , используется в словаре несколько раз\0"}};

char \*\*errors = (char \*\*)malloc(5\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 5; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==2, L"Неверное число найденных ошибок");

int find=0;

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

{

for(int j=0;j<2;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

void TestManyErrors()//! Сразу несколько ошибок в работе функции

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=3;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,"н");

strcpy(dict[1].oftenRepeatedCouple,"ан");

strcpy(dict[1].coupleCode,"н");

strcpy(dict[2].oftenRepeatedCouple,"па");

strcpy(dict[2].coupleCode,"р");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].coupleCode,"-");

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[1].coupleCode,".");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

char expectErrors[6][100]={{"Ключ н используется в словаре несколько раз\0"},{"Ключ н используется в словаре несколько раз\0"},{"Как ключ использован символ, который есть в тексте: н\0"},{"Как ключ использован символ, который есть в тексте: н\0"},{"В словаре найдена лишняя пара па\0"},{"Заменено слишком много пар\0"}};

char \*\*errors = (char \*\*)malloc(6\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 6; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==6, L"Неверное число найденных ошибок");

int find=0;

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

{

for(int j=0;j<6;j++)

if(strcmp(errors[i],expectErrors[j])==0)

find=1;

if (find!=1)

CFIX\_ASSERT\_MESSAGE(find==1,L"Ошибка определена неверно");

}

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

void TestNoError()//! Ошибок нет вообще

{

Check check;

int m=1;// Количество строк в тексте

int n=5;// Максимальное количество символов в строке

int count=2;//количество пар в словаре, которое нашла функция

int expectCount=2;//Ожидаемое количетсв пар в словаре

char \*\*text = (char \*\*)malloc(m\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < m; i++)//выделяем память для текста

{

text[i] = (char \*)malloc(n\*sizeof(char));

}

strcpy(text[0],"нана");

DictString\* dict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(dict[0].oftenRepeatedCouple,"на");

strcpy(dict[0].coupleCode,"е");

strcpy(dict[1].oftenRepeatedCouple,"ан");

strcpy(dict[1].coupleCode,"з");

DictString\* expectDict=(DictString\* )malloc(m\*n\*sizeof(DictString));//Массив объектов структуры DictString, содерщащий информацию о словаре

strcpy(expectDict[0].oftenRepeatedCouple,"на");

strcpy(expectDict[0].coupleCode,"е");

strcpy(expectDict[1].oftenRepeatedCouple,"ан");

strcpy(expectDict[1].coupleCode,"з");

char \*\*errors = (char \*\*)malloc(6\*sizeof(char \*));// Текст, который необходимо упаковать

for(int i = 0; i < 6; i++)//выделяем память для текста

{

errors[i] = (char \*)malloc(100\*sizeof(char));

}

int errorsCount;

check.check\_pack(text,m,count,expectCount,dict,expectDict,errors,&errorsCount);

CFIX\_ASSERT\_MESSAGE(errorsCount==0, L"Неверное число найденных ошибок");

for(int i=0; i<5; i++) // цикл по строкам

{

free(errors[i]); // освобождение памяти под строку

}

free(errors);

for(int i=0; i<m; i++) // цикл по строкам

{

free(text[i]); // освобождение памяти под строку

}

free(text);

free(dict);

free(expectDict);

}

**Модульные тесты для подпрограммы findFreeChars**

void NoFilename() //! Не задано имя файла

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(1); // Имя файла

strcpy(filename,"\0");

char\*\* text=(char\*\*)malloc(sizeof(char\*)); // Текст

text[0]=(char\*)malloc(9);

strcpy(text[0],"лалалала");

DictString\* dict=(DictString\*)malloc(2\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"ла"); strcpy(dict[0].coupleCode,"\*");

// Вызываем тестируемую функцию

int success=writeToFile(filename,text,1,9,dict,1,0);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==0,L"В файл была совершена запись, хотя не должна была");

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text);

free(dict);

}

void writePackedTextAndDict() //! Запись упакованного текста и словаря

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(16); // Имя файла

strcpy(filename,"file\_packed.txt");

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Текст

text[0]=(char\*)malloc(10);

strcpy(text[0],"т\*+ р\*,");

text[1]=(char\*)malloc(10);

strcpy(text[1],"п\*+ п\*,");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int success=writeToFile(filename,text,2,7,dict,3,0);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"В файл не была совершена запись");

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text[1]);

free(text);

free(dict);

}

void writeUnpackedText() //! Запись распакованного текста

{

// Подготавливаем данные для тестирования

char\* filename=(char\*)malloc(18); // Имя файла

strcpy(filename,"file\_unpacked.txt");

char\*\* text=(char\*\*)malloc(2\*sizeof(char\*)); // Текст

text[0]=(char\*)malloc(13);

strcpy(text[0],"течет речка");

text[1]=(char\*)malloc(13);

strcpy(text[1],"печет печка");

DictString\* dict=(DictString\*)malloc(4\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

strcpy(dict[0].oftenRepeatedCouple,"еч"); strcpy(dict[0].coupleCode,"\*");

strcpy(dict[1].oftenRepeatedCouple,"ет"); strcpy(dict[1].coupleCode,"+");

strcpy(dict[2].oftenRepeatedCouple,"ка"); strcpy(dict[2].coupleCode,",");

// Вызываем тестируемую функцию

int success=writeToFile(filename,text,2,7,dict,3,1);

// Проверяем корректность полученных в ходе запуска функции данных

CFIX\_ASSERT\_MESSAGE(success==1,L"В файл не была совершена запись");

// Освобождаем выделенную для тестирования память

free(text[0]);

free(text[1]);

free(text);

free(dict);

}

# Приложение Д. Разделение обязанностей между членами команды

Таблица Д.1. Реализация подпрограмм членами команд

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Подпрограмма | Заголовок | Модульные тесты | Псевдокод | Код | Тестирование и отладка |
| main | Шалиевская А. | Шалиевская А. | Тажибова А.-50%  Шалиевская А.-50% | Шалиевская А. | Шалиевская А. |
| isValidCmdArgs | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А.-70%, Шалиевская А.-30% |
| replaceChars | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| repAllEntries | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| repFirstNChars | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| findPopularCouples | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| sort | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| isFinded | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| findFreeChars | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| readFile | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| findDict | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| findText | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| writeToFile | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А.-70%, Шалиевская А.-30% |
| writeMassToFile | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| addStringToLogMass | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| transDictStructToMass | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. |
| pack | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. | Шалиевская А. |
| unpack | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А. | Тажибова А.-70%, Шалиевская А.-30% |

Дополнительные обязанности Шалиевской А.:

* Создание функций класса Check, упрощающих тестирование функций со сложными параметрами и тесты на функции класса Check;
* Создание функциональных тестов на программу.

Дополнительные обязанности Тажибовой А.:

* Создание функции для ведения файла-лога;
* Разбиение программы на модули.

# Приложение Е. Порядок конструирования программы

Рисунок Е.1 – Порядок реализации подпрограмм членами команд

# Приложение Ж. Код программы

int main(int argc,CHAR\* argv[])

{

setlocale(LC\_CTYPE,"rus");

char filename[100];

int isError=isValidCmdArgs(argc,argv,filename);// Разбираем командную строку

if (isError==0)//Если аргументы считаны успешно

{

int stringsCount,charCount=0;// Количество строк и их длина

char \*\*text=readFile(filename,&stringsCount,&charCount);// Считываем с файла весь текст

if(text!=0)

{

DictString\* dictionary=(DictString\*)malloc(256\*sizeof(DictString));// Словарь

int dictSize=0;// Размер словаря

if(strcmp(argv[1],"p")==0)// Если задан режим упаковки

{

pack(text,stringsCount,charCount,dictionary,&dictSize);

char packFile[100];

strcpy(packFile,filename);

char\* dot=strrchr(packFile,'.');

strcpy(dot,"\_pack");

dot=strrchr(filename,'.');

strcat(packFile,dot);

isError=writeToFile(packFile,text,stringsCount,charCount,dictionary,dictSize,0);

}

else

{

isError=unpack(text,stringsCount,charCount);

if(isError)

{

char unpackFile[100];

strcpy(unpackFile,filename);

char\* dot=strrchr(unpackFile,'.');

strcpy(dot,"\_unpack");

dot=strrchr(filename,'.');

strcat(unpackFile,dot);

isError=writeToFile(unpackFile,text,isError,charCount,dictionary,dictSize,1);

}

}

free(dictionary);

}

else printf("Ошибка открытия файла\n");

for(int i=0; i<stringsCount; i++) // цикл по строкам

free(text[i]); // освобождение памяти под строку

free(text);

}

system("pause");

return 0;

}

int replaceChars(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int dictSize,int isPack)

{

int i,j;

char\* toBeReplaced=(char\*)malloc(3\*sizeof(char)); // Символ или пара символов, которые должны быть заменены

char\* replacement=(char\*)malloc(3\*sizeof(char)); // Символ или пара символов, которыми происходит замена

int success=1; // Успешность выполнения функции repAllEntries

for(i=0; i<dictSize; i++) // Для каждой пары словаря...

{

if(isPack) // Если режим упаковки

{

strncpy(toBeReplaced,dictionary[i].oftenRepeatedCouple,2); // Будем искать вхождение пары символов

toBeReplaced[2]='\0';

strncpy(replacement,dictionary[i].coupleCode,1); // Замена будет происходить одиночным символом

replacement[1]='\0';

}

else // Иначе

{

strncpy(toBeReplaced,dictionary[i].coupleCode,1); // Будем искать вхождение кода

toBeReplaced[1]='\0';

strncpy(replacement,dictionary[i].oftenRepeatedCouple,2); // Замена будет происходить парой символов

replacement[2]='\0';

}

for(j=0; j<stringsCount; j++) // Для каждой строки текста...

{

success\*=repAllEntries(text[j], toBeReplaced, replacement); // Производим замену всех вхождений текущей пары словаря (или кода словаря)

}

}

// Освобождаем выделенную память

free(toBeReplaced);

free(replacement);

if(dictSize>0 && stringsCount>0) // Словарь и текст не пустые

return success;

return 0;

}

int repAllEntries(char\* str,char\* toBeReplaced,char\* replacement)

{

if(strlen(str)!=0 && strlen(toBeReplaced)!=0 && strlen(replacement)!=0) // Если переданные строки не пустые

{

char\* coupleEntry=(char\*)malloc(strlen(str)+1); // Вхождение строки toBeReplaced в строке str

char\* changedStr=(char\*)malloc(strlen(replacement)\*strlen(str)+1); // Строка, содержащая полностью строку str в измененном виде

strcpy(changedStr,str);

coupleEntry=str;

int repIndex; // Индекс символа, с которого начинается замена

int success=1; // Успешность выполнения функции repFirstNChars

do // Делать...

{

coupleEntry=strstr(coupleEntry,toBeReplaced); // Находим очередное вхождение toBeReplaced

if(coupleEntry!=NULL) // Если вхождение было найдено

{

repIndex=strlen(changedStr)-strlen(coupleEntry); // Вычисляем индекс символа, с которого начинается замена

changedStr[repIndex]='\0'; // Ставим нуль-символ на месте начала замены

success\*=repFirstNChars(coupleEntry,strlen(toBeReplaced),replacement); // Производим замену

strcat(changedStr,coupleEntry); // Добавляем остаток строки

}

}while(coupleEntry!=NULL); // Пока не найдены все вхождения строки toBeReplaced в строке str

// Освобождаем выделенную память

free(changedStr);

free(coupleEntry);

if(success==1) // Если вхождения были найдены и замена всех вхождений произошла успешно

return 1; // Замены выполнены успешно

}

return 0; // Замены выполнены неуспешно

}

int repFirstNChars(char\* strTo,int n,char\* strFrom)

{

char\* bufStr; // Буферная строка, хранящая в себе все символы strTo, кроме тех, которые надо заменить

if(strTo!=NULL && strFrom!=NULL && n>0) // Если входные строки не пустые и количество заменяемых символов положительно

{

if(strlen(strTo)>=n) // Если длина строки, в которой нужно заменить символы не меньше количества символов, клоторые нужно заменить в ней

{

bufStr=(char\*)malloc((strlen(strTo)-n+1)\*sizeof(char)); // Выделяем память под буферную строку

strcpy(bufStr,strTo+n); // Сохраняем в буфер строку strTo без символов, которые нужно заменить

strTo[0]='\0'; // Обнуляем strTo

strcpy(strTo,strFrom); // Добавляем в обнуленную строку содержимое strFrom

strcat(strTo,bufStr); // Добавляем сохраненные в буфер символы

free(bufStr); // Освобождаем выделенную память

return 1; // Замена произошла успешно

}

}

return 0; // Замены не произошло

}

int isValidCmdArgs(int argc,char\*\* argv,char\* filename)

{

setlocale(LC\_CTYPE,"rus");

int success=1; // Предполагаем, что возвращаемое значение функции равно 1

if(argc==1) // Если количество аргументов равно 1

printf("Аргументы командной строки не найдены\n"); // Выводим сообщение о том, что пользователь не указал никаких параметров

else if(argc>3) // Иначе если количество аргументов больше 3

printf("Слишком много аргументов командной строки\n"); // Выводим сообщение о том, что введено слишком много аргументов

else if(argc>1 && argc<3) // Иначе если количество аргументов больше 1, но меньше 3

printf("Слишком мало аргументов командной строки\n"); // Выводим сообщение о том, что введено недостаточное количество аргументов

if(argc==3) // Если количество аргументов равно 3

{

if(strcmp(argv[1],"p")&&strcmp(argv[1],"u")) // Иначе если 2-ой параметр, введенный пользователем не равен "p" или "u"

printf("Неверно введен режим упаковки\n"); // Флаг режима упаковки введен неверно

else // Иначе

{

success=0; // Изменяем возвращаемое значение функции на 0

strcpy(filename,argv[2]); // Сохраняем имя файла

}

}

// Возвращаем флаг

return success;

}

void findPopularCouples(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictCount)

{

int countCouples=0;// Количество найденных пар

Couple\* couples=(Couple\*)malloc(2\*stringsCount\*charCount\*sizeof(Couple\*));// Все пары текста

for(int i=0;i<stringsCount;i++) // Для каждой строки заданного текста…

for(int j=0;j<charCount && text[i][j]!='\0' && text[i][j+1]!='\0' && text[i][j+1]!='\n';j++)// Для каждого символа строки, кроме последнего...

{

Couple current;// Определяем текущую пару

current.couple[0]=text[i][j];

current.couple[1]=text[i][j+1];

current.couple[2]='\0';

int find=isFinded(current,couples,countCouples);// Была ли найдена такая пара

if(find==-1)// Если текущая пара еще не была найдена

{

strncpy(couples[countCouples].couple,current.couple,2);//СОхраняем ее

couples[countCouples].couple[2]='\0';

couples[countCouples].coupleAmount=1;

countCouples++;

}

else

couples[find].coupleAmount++;//Увеличиваем количество повторений этой пары в массиве

}

sort(couples,countCouples);// Сортируем пары в порядке убывания количества их повторений

int i;

for (i=0;i<(\*dictCount) && i<countCouples;i++)//Добавляем частые пары в словарь

{

strncpy(dictionary[i].oftenRepeatedCouple,couples[i].couple,2);

dictionary[i].oftenRepeatedCouple[2]='\0';

}

(\*dictCount)=i;

free(couples);

}

void sort(Couple\* couple,int coupleCount)

{

for(int i=0;i<coupleCount-1;i++)

// сравниваем два соседних элемента.

for(int j=0;j<coupleCount-i-1;j++)

if(couple[j].coupleAmount<couple[j+1].coupleAmount)

{

// если они идут в неправильном порядке, то меняем их местами.

Couple tmp=couple[j];

couple[j]=couple[j+1];

couple[j+1]=tmp;

}

}

int isFinded(Couple couple, Couple\* couples,int coupleCount)

{

int find=-1;// Была ли найдена такая пара

// Для каждой уже существующей пары, пока не найдем среди них текущую

for(int k=0;k<coupleCount && find==-1;k++)

{

// Если уже существующая пара равна текущей

if(strncmp(couple.couple,couples[k].couple,2)==NULL)

find=k;

}

return find;

}

void findFreeChars(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int\* dictSize)

{

int i,j;

int curStrLen; // Длина текущей строки текста

int freeCharCount=0; // Количество записанных в словарь свободных символов

int charEntryCounts[256]={0}; // Массив с количеством вхождений символов в тексте

char charCode; // Код-символ

int intCode; // Код-число

for(i=0; i<stringsCount; i++) // Для каждой строки текста

{

curStrLen=strlen(text[i]); // Находим длину текущей строки

for(j=0; j<curStrLen+1; j++) // Для каждого символа текущей строки

{

// Получаем код текущего символа

intCode=(unsigned int)text[i][j];

if(intCode<0)

intCode=256+intCode;

charEntryCounts[intCode]+=1; // Помечаем, что текущий символ текста несвободен

}

}

for(i=255; i>1; i--) // Для каждого символа массива

{

if(!charEntryCounts[i]) // Если количество вхождений текущего символа в тексте ==0 (т.е. символ свободен)

{

charCode=(unsigned char)i; // Получаем символ по коду

dictionary[freeCharCount].coupleCode[0]=charCode; // Записываем код в словарь

//dictionary[freeCharCount].coupleCode[1]='\0';

freeCharCount++; // Увеличиваем количество записанных в словарь свободных символов

}

}

\*dictSize=freeCharCount; // Изменяем размер словаря на количество записанных кодов

}

char\*\* readFile(char\* filename,int \*stringsCount,int \*charCount)

{

char\*\* text=(char\*\*)malloc(sizeof(char\*));

(\*stringsCount)=0;

(\*charCount)=0;

int isOk=0;

FILE \*f;// Дескриптор файла

f = fopen(filename, "r");// Открыть файл

if (f!=NULL)// Если файл открылся без ошибок...

{

char \*stroka =(char \*)malloc(10000);

while(fgets(stroka, 10000, f) != NULL)// пока есть строки читаем их в stroka

{

text=(char\*\*)realloc(text, (\*stringsCount+1)\*sizeof(char\*));// Выделяем память для текста

text[\*stringsCount] = (char\*)malloc(10000\*sizeof(char)); // Выделяем память для содержимого строки

strncpy(text[\*stringsCount],stroka,strlen(stroka));// Сохранить строку в массив с текстом

text[\*stringsCount][strlen(stroka)]='\0';

(\*stringsCount)++;// Увеличить количетсво считанных строк

if ((\*charCount)<strlen(stroka))

(\*charCount)=strlen(stroka);// Записываем длину максимальной строки

}

fclose (f);

}

if((\*stringsCount)==0) text=0;

return text;

}

int findDict(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictCount)

{

int isValidDict=1; // Корректный ли словарь

int isSepFound=0; // Найден ли разграничитель

(\*dictCount)=0;

for(int i=0; i<stringsCount && isSepFound==0; i++) // Для каждой строки текста, пока не появится разграничитель...

{

if(text[i][0]=='.' && text[i][1]=='.' && text[i][2]=='.' ) // Если текущая строка-разграничитель

{

isSepFound=1; // Помечаем, что разграничитель найден

// Сохраняем все, что находится до текущей строки в словарь...

for(int n=0; n<i && isValidDict==1; n++) // Для каждой строки словаря пока не найдена ошибка в составлении словаря

{

int a=strlen(text[n]);

if((strlen(text[n])==4 && text[n][3]=='\n')||strlen(text[n])==3)//если в строке три символа и символ перевода строки

{

strncpy(dictionary[\*dictCount].oftenRepeatedCouple,text[n],2); // Копируем пару в словарь

dictionary[\*dictCount].oftenRepeatedCouple[2]='\0';

strncpy(dictionary[\*dictCount].coupleCode,(text[n]+2),1); // Копируем код пары в словарь

dictionary[\*dictCount].coupleCode[1]='\0';

(\*dictCount)++; // Увеличиваем количество пар в словаре

}

else

{

isValidDict=0; // Найдена ошибка составления словаря

}

}

}

}

if(\*dictCount==0 || isSepFound==0 ||isValidDict==0 ) // Если словаря нет или если нет разделителя

{

isValidDict=0;

printf("Переданный на распаковку файл невозможно распаковать из-за ошибок в его структуре\n");

}

return isValidDict;

}

int findText(char\*\* text,int \*stringsCount,int charCount,int dictCount)

{

int i, k=0;

int isOk=1; // Флаг, есть ли ошибка в данных

// Сохранить все, что находится после словаря в текст

for(i=dictCount+1; i<\*stringsCount; i++)

{

strcpy(text[k],text[i]);

k++;

}

\*stringsCount=k; // Сохраняем количество строк

if(\*stringsCount==0) // Если текст пуст, вывести ошибку

{

isOk=0;

printf("Переданный на распаковку файл невозможно распаковать из-за ошибок в его структуре\n");

}

return isOk;

}

int writeToFile(char\* filename,char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int dictSize,int mode)

{

int i;

int success=0;

char\*\* massToFile=(char\*\*)malloc((dictSize+1+stringsCount)\*sizeof(char\*)); // Текст для записи в файл

int massToFileSize=stringsCount; // Количество строк в массиве с текстом для записи в файл

char\*\* dictMass=NULL; // Массив-словарь

if(mode==0) // Если режим записи упакованного файла и словаря

{

dictMass=transDictStructToMass(dictionary,dictSize); // Переводим словарь-структуру в многострочный текст

if(dictMass!=NULL) // Если перевод структуры-словаря в словарь-массив произошел успешно

{

massToFileSize=dictSize+1+stringsCount;

for(i=0; i<dictSize; i++) // Для каждой строки словаря...

{

massToFile[i]=(char\*)malloc(strlen(dictMass[i])+2); // Выделяем память под строку массива massToFile

strcpy(massToFile[i],dictMass[i]); // Копируем текущую строку словаря в массив

free(dictMass[i]); // Освобождаем память под текущую строку словаря

}

massToFile[dictSize]=(char\*)malloc(5); // Выделяем память под троеточие - разделитель между словарем и текстом

strcpy(massToFile[dictSize],"..."); // Копируем троеточие

free(dictMass); // Освобождаем память под словарь

}

}

if(dictMass==NULL)

dictSize=-1;

for(i=1; i<=stringsCount; i++) // Для каждой строки текста...

{

massToFile[dictSize+i]=(char\*)malloc(charCount\*2+1); // Выделяем память под текст

strcpy(massToFile[dictSize+i],text[i-1]); // Копируем строку текста в массив

}

success=writeMassToFile(filename,massToFile,massToFileSize,charCount); // Записываем в файл текст

// Освобождаем память под массив с текстом для записи в словарь

for(i=0; i<massToFileSize; i++)

free(massToFile[i]);

free(massToFile);

return success;

}

int writeMassToFile(char\* filename,char\*\* text,int stringsCount,int charCount)

{

int i;

int success=1; // Успешность записи строки в файл

FILE \*f; // Дескриптор файла

f=fopen(filename, "a"); // Открываем файл на запись

if(f!=NULL) // Если файл был успешно открыт

{

for(i=0; i<stringsCount && success>=0; i++) // Для каждой строки текста пока fputs завершается успешно

{

if(text[i][strlen(text[i])-1]!='\n')

strcat(text[i],"\n");

success=fprintf(f,"%s",text[i]); // Записываем текущую строку текста в файл

}

fclose(f);

if(success>=0) // Если запись строки в файл произлошла успешно

return 1; // Возвращаем 1

}

return 0; // Возвращаем 0

}

void addStringToLogMass(char\*\* logMass,int stringsCount,int isSucceeded,char\* funcDescription)

{

logMass[stringsCount]=(char\*)malloc(2\*strlen(funcDescription)); // Выделяем память под строку массива-лога

strcpy(logMass[stringsCount],funcDescription);

strcat(logMass[stringsCount]," была выполнена");

if(isSucceeded) // Если функция была выполнена успешно

strcat(logMass[stringsCount]," успешно");

else // Иначе

strcat(logMass[stringsCount]," неуспешно");

}

char\*\* transDictStructToMass(DictString\* dictionary,int dictSize)

{

int i;

char\*\* massDict;

if(dictSize>0 && dictionary!=NULL) // Если размер словаря больше нуля и словарь не пуст

{

massDict=(char\*\*)malloc(dictSize\*sizeof(char\*)); // Словарь-массив

for(i=0; i<dictSize; i++) // Для каждой единицы словаря

{

massDict[i]=(char\*)malloc(4\*sizeof(char)); // Выделяем память под текущую строку массива

strncpy(massDict[i],dictionary[i].oftenRepeatedCouple,2);// Копируем в текущую строку возвращаемого массива текущую часто повторяющуюся пару

massDict[i][2]='\0';

strncat(massDict[i],dictionary[i].coupleCode,1);// Прибавляем текущий код

}

return massDict; // Возвращаем словарь-массив

}

return NULL; // Возвращаем нул

}

void pack(char\*\* text,int stringsCount,int charCount,DictString\* dictionary,int \*dictSize)

{

int isPacked=0;

int logMassSize=-1; // Размер массива с текстом для лог файла

char\*\* logMass=(char\*\*)malloc(4\*sizeof(char\*)); // Массив с текстом для лог файла

findFreeChars(text,stringsCount,charCount,dictionary,dictSize); // Находим символы, которых нет в тексте

if(stringsCount)

{

addStringToLogMass(logMass,++logMassSize,stringsCount,"Функция, совершающая поиск символов, отсутствующих в тексте"); // Добавляем в массив-лог строку

// Находим все пары рядом стоящих символов в тексте, записывая их в словарь с кодами по мере убывания количества повторений

findPopularCouples(text,stringsCount,charCount,dictionary,dictSize);

if(\*dictSize)

{

addStringToLogMass(logMass,++logMassSize,stringsCount,"Функция, совершающая поиск часто повторяющихся пар в тексте"); // Добавляем в массив-лог строку

int isRep=replaceChars(text,stringsCount,charCount,dictionary,\*dictSize,1); // Заменяем часто встречающиеся символы в тексте на их коды

if(isRep)

{

addStringToLogMass(logMass,++logMassSize,stringsCount,"Функция, совершающая замену пар символов на их коды в тексте"); // Добавляем в массив-лог строку

isPacked=1;

}

}

}

else (\*dictSize)=0;

addStringToLogMass(logMass,++logMassSize,isPacked,"Функция, совершающая упаковку заданного текста"); // Добавляем в массив-лог строку

writeMassToFile("log.txt",logMass,logMassSize+1,111); // Добавляем информацию о выполнении функции в лог файл

// Освобождаем выделеннную память

for(int i=0; i<logMassSize+1; i++)

free(logMass[i]);

free(logMass);

}

int unpack(char\*\* text,int stringsCount,int charCount)

{

int logMassSize=-1; // Размер массива с текстом для лог файла

char\*\* logMass=(char\*\*)malloc(4\*sizeof(char\*)); // Массив с текстом для лог файла

DictString\* dict=(DictString\*)malloc(stringsCount\*charCount\*sizeof(DictString)); // Словарь с часто повторяющимися парами и их кодами

int dictSize; // Размер словаря

int isDictFound=0; // Флаг, равный 0, если функция нахождения словаря завершилась успешно, иначе 1

int isTextFound=0; // Флаг, равный 0, если функция нахождения текста завершилась успешно, иначе 1

int isRep=0; // Флаг, равный 1, если замена символов произошла успешно, иначе 0

// Разделяем входной файл на текст, который надо распаковать и словарь

isDictFound=findDict(text,stringsCount,charCount,dict,&dictSize);

addStringToLogMass(logMass,++logMassSize,isDictFound,"Функция, совершающая поиск словаря в массиве"); // Добавляем в массив-лог строку

isTextFound=findText(text,&stringsCount,charCount,dictSize); // Ищем текст

addStringToLogMass(logMass,++logMassSize,isTextFound,"Функция, совершающая поиск упакованного текста в массиве"); // Добавляем в массив-лог строку

if(isDictFound && isTextFound) // Если найден и текст и словарь

{

isRep=replaceChars(text,stringsCount,charCount,dict,dictSize,0); // Заменяем коды символов на соответствующие им пары символов

addStringToLogMass(logMass,++logMassSize,isTextFound,"Функция, совершающая замену кодов символов на соответствующие пары символов в массиве"); // Добавляем в массив-лог строку

}

addStringToLogMass(logMass,++logMassSize,isRep,"Функция, совершающая распаковку заданного текста"); // Добавляем в массив-лог строку

writeMassToFile("log.txt",logMass,logMassSize+1,111); // Добавляем информацию о выполнении функции в лог файл

// Освобождаем выделеннную память

free(dict);

for(int i=0; i<logMassSize+1; i++)

free(logMass[i]);

free(logMass);

return (isRep\*stringsCount);}// Возвращаем успешность выполнения распаковки текста

void Check::check\_pack(char\*\* text,int m,int count, int expectCount, DictString\* dict, DictString\* expectDict,char\*\* errors,int \*errorsCount)

{

\*errorsCount=0;

if(count>expectCount)// Проверяем, что пар найдено не больше, чем нужно

{

strcpy(errors[\*errorsCount],"Заменено слишком много пар");

(\*errorsCount)++;

}

for(int i=0;i<count;i++)// Проверяем, что каждая найденная пара есть в списке ожидаемых

{

int find=0;

for (int j=0;j<expectCount && find==0;j++)

if (strncmp(dict[i].oftenRepeatedCouple,expectDict[j].oftenRepeatedCouple,2)==0)

find=1;

if(find==0)

{

char message[56]={"В словаре найдена лишняя пара "};

strncat(message,dict[i].oftenRepeatedCouple,2);

strcpy(errors[\*errorsCount],message);

(\*errorsCount)++;

}

}

for(int i=0;i<count;i++)// Проверяем, что ключей, использованных в словаре нет в тексте

{//для каждого ключа

int error=0;

for(int j=0;j<m && error==0;j++)//для каждой строки текста

if(strchr(text[j],\*dict[i].coupleCode)!=0)//если ключ и символ текста равны

{

error=1;

char message[95]={"Как ключ использован символ, который есть в тексте: "};

strncat(message,dict[i].coupleCode,1);

strcpy(errors[(\*errorsCount)],message);

(\*errorsCount)++;

}

}

for(int i=0;i<count;i++)// Проверяем, что ключи в словаре не повторяются

{

int error=0;

for(int j=0;j<count && error==0;j++)

{

if(strncmp(dict[i].coupleCode,dict[j].coupleCode,1)==0 && i!=j)

{

error=1;

char message[50]={"Ключ "};

strncat(message,dict[i].coupleCode,1);

strcat(message," используется в словаре несколько раз");

strcpy(errors[(\*errorsCount)],message);

(\*errorsCount)++;

}

}

}

}

void Check::check\_findPopularCouples(DictString\* dictionary,DictString\* expectDictionary,int dictCount,int expectDictCount,char\*\* errors,int \*errorsCount)

{

\*errorsCount=0;

if(dictCount>expectDictCount)// Проверяем, что словарь не больше, чем должен быть

{

strcpy(errors[\*errorsCount],"Словарь слишком большой");

(\*errorsCount)++;

}

if(dictCount<expectDictCount)// Проверяем, что словарь не меньше, чем должен быть

{

strcpy(errors[\*errorsCount],"Словарь слишком маленький");

(\*errorsCount)++;

}

for(int i=0;i<expectDictCount;i++)// Проверяем, что все нужные строки словаря функция заполнила

{

int find=0;

for (int j=0;j<dictCount && find==0;j++)

if(strncmp(dictionary[i].oftenRepeatedCouple,expectDictionary[j].oftenRepeatedCouple,2)==0)

find=1;

if (find==0)// Если какой-то пары в словаре нет

{

char message[30]={"Не найдена пара "};

strncat(message,expectDictionary[i].oftenRepeatedCouple,2);

strcat(message,"\0");

strcpy(errors[\*errorsCount],message);

(\*errorsCount)++;

}

}

}

void Check::check\_findDict(DictString\* dictionary,DictString\* expectDictionary,int dictCount,int expectDictCount,char\*\* errors,int \*errorsCount)

{

\*errorsCount=0;

//проверяем, что найденное число пар в словаре верное

if (dictCount!=expectDictCount)//если число найденных пар не равно ожидаемому количетсву

{

if(dictCount>expectDictCount)// Проверяем, что пар найдено не больше, чем нужно

{

strcpy(errors[\*errorsCount],"Найдено слишком много пар");

(\*errorsCount)++;

}

if(dictCount<expectDictCount)// Проверяем, что пар найдено не меньше, чем нужно

{

strcpy(errors[\*errorsCount],"Найдены не все пары");

(\*errorsCount)++;

}

}

for(int i=0;i<expectDictCount;i++)//проверяем, все ли строки словаря нашла функция

{

int find=0;

for(int j=0;j<dictCount && find==0;j++)

if(strncmp(expectDictionary[i].oftenRepeatedCouple,dictionary[j].oftenRepeatedCouple,2)==0)

{

find=1;

if(strncmp(expectDictionary[i].coupleCode,dictionary[j].coupleCode,1)!=0)

{

char message[35]={"Неверный код у пары "};

strncat(message,expectDictionary[i].oftenRepeatedCouple,2);

strcpy(errors[\*errorsCount],message);

(\*errorsCount)++;

}

}

if(find==0)

{

char message[35]={"Не найдена пара "};

strncat(message,expectDictionary[i].oftenRepeatedCouple,2);

strcpy(errors[\*errorsCount],message);

(\*errorsCount)++;

}

}

}

void Check::check\_findText(char\*\* text,int stringsCount,int charCount,char\*\* expectText,int expectStringsCount,int expectCharCount,char\*\* errors,int \*errorsCount)

{

\*errorsCount=0;

//проверяем, что в результируемем тексте столько же строк, сколько и в ожидаемом

if (stringsCount!=expectStringsCount)//если число строк в ожидаемом тексте не равно числу строк в результрующем тексте

{

if(stringsCount>expectStringsCount)// Проверяем, что строк не больше чем нужно

{

strcpy(errors[\*errorsCount],"Строк слишком много");

(\*errorsCount)++;

}

if(stringsCount<expectStringsCount)// Проверяем, что строк не меньше чем нужно

{

strcpy(errors[\*errorsCount],"Строк слишком мало");

(\*errorsCount)++;

}

}

for(int i=0;i<expectStringsCount;i++)//сравниваем ожидаемый текст с полученным

{

int find=0;

for(int j=0;j<stringsCount;j++)

if(strcmp(expectText[i],text[j])==0)

{

find=1;

}

if(find==0)

{

char message[50]={"Отсутствует строка под номером "};

char str[100];

itoa(i,str,10);

strcat(message,str);

strcpy(errors[\*errorsCount],message);

(\*errorsCount)++;

}

}

}

# Приложение З. Оценка полученных результатов

Таблица З.1. Оценка Шалиевской А. полученных результатов

|  |  |  |
| --- | --- | --- |
| Критерий оценки | Достоинства | Недостатки |
| Работоспособность + функциональность программы | Программа работает на всех функциональных тестах | Программа не анализирует пересечения пар, возможен случай, что свободный символ будет занят, но не использован |
| Внешняя спецификация программы | Наличие | Слишком большая |
| Декомпозиция программы на подпрограммы | Все замечания учтены и исправлены | Нет |
| Декомпозиция программы на модули | Все отлично | Без нее было лучше |
| Псевдокод | Появился раньше кода | Исправлялся и дополнялся в процессе |
| Код | Использованы две структуры данных: DictString и Couple | Возможно добавление еще структуры, содержащей текст, количество строк в нем и максимальную длину строки |
| Форматирование кода | Все отлично | Нет |
| Модульные тесты | Для проверки выходных параметров со сложной структурой использовался класс Check | На нескольких простых функциях отсутствуют |
| Ведение репозитория кода | Заполнялся регулярно | Нет |
| Порядок конструирования программы + распределение подпрограмм между членами команды | Все отлично | Функция считывания и записи в файл выполнялась разными участниками |

Таблица З.2. Оценка Тажибовой А. полученных результатов

|  |  |  |
| --- | --- | --- |
| Критерий оценки | Достоинства | Недостатки |
| Работоспособность + функциональность программы | Программа работает на всех функциональных тестах | Программа не анализирует пересечения пар, поэтому возможен случай, что свободный символ будет занят, но не использован |
| Внешняя спецификация программы | Наличие | Нет |
| Декомпозиция программы на подпрограммы | Все замечания учтены и исправлены | Нет |
| Декомпозиция программы на модули | Все отлично | Нет |
| Псевдокод | Появился раньше кода | Исправлялся и дополнялся в процессе. В одной функции появился только в виде комментариев к коду. |
| Код (эффективность структур данных, реализация типовых алгоритмических структур, их комбинация) | Использованы две структуры данных: DictString и Couple | Возможно добавление еще структуры, содержащей текст, количество строк в нем и максимальную длину строки |
| Форматирование кода | Все отлично | Нет |
| Модульные тесты | Для проверки выходных параметров со сложной структурой использовался класс Check | Отсутсвуют на функцию addStringToLogMass так как она только добавляет в массив строки и ничего более. |
| Ведение репозитория кода | Заполнялся регулярно | Присутствуют неподробные коммиты и комплексные коммиты, которые можно было разбить на несколько |
| Порядок конструирования программы + распределение подпрограмм между членами команд | Все отлично | Нет |