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    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ЛитовкинД.В.

должность подпись

Студент гр.      ПрИн-267       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Е. С.

подпись

Студент гр.      ПрИн-267       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сивашова Е. С.

подпись

Студент гр.      ПрИн-267       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Стрекалова А. С.

подпись

Отчет защищен с оценкой:

Волкова Е. С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Сивашова Е. С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Стрекалова А. С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Волгоград 2016 г

**Содержание**

[Введение 3](#_Toc457564181)

[Формулировка задания 4](#_Toc457564182)

[Содержание практики 5](#_Toc457564183)

[Заключение 6](#_Toc457564184)

[Приложение А. Способ упаковки-распаковки данных 7](#_Toc457564185)

[Приложение Б. Внешняя спецификация программы 8](#_Toc457564186)

[Приложение В. Проект программы 19](#_Toc457564187)

[Приложение Г. Модульные тесты 27](#_Toc457564188)

[Приложение Д. Разделение обязанностей между членами команды 81](#_Toc457564189)

[Приложение Е. Порядок конструирования программы 84](#_Toc457564190)

[Приложение Ж. Код программы 87](#_Toc457564191)

[Приложение З. Оценка полученных результатов 112](#_Toc457564192)

# 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 практикадает навыки разработки программы в команде.

# Формулировка задания

Дан файл, содержащий исходный код программы. Сначала необходимо сжать его, получив упакованный (новый) файл. А затем распаковать упакованный файл, получив (новый) исходный файл.

Архиватор должен работать в консольном режиме, получая параметры через командную строку. Помимо упакованных и распакованных файлов архиватор должен создавать файл с логом, содержащим отчет о процессе упаковки и распаковки. Лог должен завершаться коэффициентом упаковки или распаковки данных.

Минимальный формат командной строки в режиме упаковки:

<имя архиватора>.exe -p <путь к файлу для упаковки><путь к новому упакованному файлу>

Минимальный формат командной строки в режиме распаковки:

<имя архиватора>.exe -u <путь к упакованному файлу><путь к новому распакованному файлу>

Командная строка может содержать дополнительные параметры, в зависимости от способа архивирования.

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;
* программадекомпозирована на подпрограммы – см. ПриложениеВ;
* созданы собственные типы данных– см. Приложение В;
* созданы модульные тесты к подпрограммам – см.Приложение Г;
* написан псевдокод для подпрограмм – см. ПриложениеЖ;
* закодирован псевдокод - см. Приложение Ж;
* код отформатирован в соответствии с единым стилем кодирования - см. Приложение Ж.

# Заключение

В ходе прохождения учебной практикизакреплены 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

Упаковать (распаковать) текст по алгоритму №1

Модель алфавита необходимо строить на основе исходного текста.

Так же необходимо реализовать «длинную арифметику», т.е. без

потери точности.

Режим упаковки:

Входные данные: исходный текст

Выходные данные: упакованный текст, модель алфавита

Режим распаковки:

Входные данные: упакованный текст, модель алфавита

Выходные данные: исходный текст

# Приложение Б. Внешняя спецификация программы

1. **Наимен**о**вание программы**

NumeralPacking – программа, предназначенная для упаковки и распаковки текста, используя представление этого текста в виде вещественного числа, записанного в виде двух целых чисел.

1. **Назначение программы**

Программа предназначена для упаковки/распаковки текста и может использоваться для архивации файлов.

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

Для режима упаковки входными данными является исходный текст.

Входные данные передаются через аргументы командной строки. Сначала записывается название программы, а затем через пробел указывается, какое действие должна выполнить программа (при упаковке – -p) и далее указывается путь к файлу для упаковки, а также путь к новому упакованному файлу.

Пример командной строки для упаковки:

NumeralPacking.exe -p C:\Users\User\Desktop\fileForPack.txt C:\Users\User\Desktop\resultFile.txt

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

Результатом работы программы являются текстовой файл. В нем содержатся два целых числа, которое представляет собой текст и длину получившегося числа, а также модель алфавита вида:

<число, равное левой границе интервала символа><символ><длина числа левой границы>

Строка, описывающая вероятность EOF-символа, выглядит следующим образом:

<число левой границы><e><длина числа левой границы><o><число, равное вероятности EOF-символа><f><длина числа вероятности>

Строка, описывающая специальные символы, такие как пробел, перевод строки, символ табуляции, перевод каретки, выглядит следующим образом:

<символ, заменяющий спец. символ><число, равное левой границе интервала символа><заменяющий символ><длина числа левой границы>,

где заменяющий спец. символ:

p – пробел;

n – перевод строки;

t – табуляция;

r – перевод каретки.

Строка с описанием цифры выглядит так:

c<число, равное левой границе интервала символа>c<цифра>c<длина числа левой границы>.

* 1. **Ограничения на выполнение**

Если файл с логом не может быть создан или не будет возможна запись в него, то будет выведено сообщение об ошибке и прекратится выполнение программы.

Если файл с входным текстом не существует или не может быть прочитан, то будет выведено сообщение об ошибке в лог и прекратится выполнение программы.

Длина входного текста ограничена:приблизительно не более 100 Мб.

Если выходной файл не может быть записан, то будет выведено сообщение об ошибке в лог и прекратится выполнение программы.

Подробная информация о сообщениях об ошибках, записях лога и ситуациях, к ним приводящим, содержится в приложении Б.Г.

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

Для режима распаковки на вход программы подается файл, в котором содержится упакованный текст и модель алфавита.

Входные данные передаются через аргументы командной строки. Сначала записывается название программы, а затем через пробел указывается, какое действие должна выполнить программа (при распаковке – -u) и далее указывается путь к упакованному файлу и путь к новому распакованному файлу.

Пример командной строки для распаковки:

NumeralPacking.exe –u C:\Users\User\Desktop\packedFile.txt C:\Users\User\Desktop\unpackedFile.txt

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

Результатом работы программы является текстовый файл с исходными символами.

* 1. **Ограничения на выполнение**

Если файл с логом не может быть создан или не будет возможна запись в него, то будет выведено сообщение об ошибке и прекратится выполнение программы.

Если файл с входным текстом и с моделью алфавита не существует или не может быть прочитан, то будет выведено сообщение об ошибке в лог и прекратится выполнение программы.

Модель алфавита должна быть записана следующим образом:

<число, равное левой границе интервала символа><символ><длина числа левой границы>

Строка, описывающая вероятность EOF-символа, выглядит следующим образом:

<число левой границы><e><длина числа левой границы><o><число, равное вероятности EOF-символа><f><длина числа вероятности>

Строка, описывающая специальные символы, такие как пробел, перевод строки, символ табуляции, перевод каретки, выглядит следующим образом:

<символ, заменяющий спец. символ><число, равное левой границе интервала символа><заменяющий символ><длина числа левой границы>,

где заменяющий спец. символ:

p – пробел;

n – перевод строки;

t – табуляция;

r – перевод каретки.

Строка с описанием цифры выглядит так:

c<число, равное левой границе интервала символа>c<цифра>c<длина числа левой границы>.

Если модель задана некорректно, то будет выведено сообщение об ошибке в лог и выполнение программы будет остановлено. Подробное описание ситуаций с корректным и некорректным оформлением моделей приведено в приложении Б.В.

Если выходной файл не может быть записан, то будет выведено сообщение об ошибке в лог и прекратится выполнение программы.

Подробная информация о сообщениях об ошибках, записях лога и ситуациях, к ним приводящим, содержится в приложении Б.Г.

1. **Ограничения на выполнение**

Если аргументы командной строки заданы некорректно, т.е.:

1. режим упаковки/распаковки не указан или указан неверно;
2. если количество аргументов менее или более 4;

то будет выведено сообщение об ошибке в лог и прекратится выполнение программы.

Размер входного файла с текстом для упаковки не должен превышать 100 Мб памяти на диске.

Размеры входного файла с упакованным текстом и выходного файла определяются операционной системой.

Подробная информация о сообщениях об ошибках, записях лога и ситуациях, к ним приводящим, содержится в приложении Б.Г.

**Приложение Б.А. Алгоритм упаковки и распаковки данных**

1 Открыть лог-файл для записи и сделать запись о начале программы

2 Если открыть файл не удалось

2.1 Вывести сообщение об ошибке и завершить работу программы

3 Проверить корректность переданных аргументов командной строки и разобрать их (inspectAndParseArgumentsOfCmd)

4 Если аргументы указаны неверно

4.1 Вывести сообщение об ошибке в лог и завершить работу программы

4.2 Закрыть лог-файл

5 Открыть входной файл для чтения

6 Если файл не может быть прочитан

6.1 Вывести сообщение об ошибке в лог и завершить работу программы

6.2 Закрыть лог-файл

7 Открыть выходной файл для записи

8 Если невозможна запись в файл

8.1 Вывести сообщение об ошибке в лог и завершить работу программы

8.2 Закрыть лог-файл

8.3 Закрыть входной файл

9 Определить режим работы программы

10 Если выбран режим упаковки (функция packFile)

10.1 Составить модель алфавита (readDataForAlphabet)

10.1.1 Считать входной текст посимвольно

10.1.1.1 Считывая посимвольно

10.1.1.2 Если символ есть в алфавите – инкрементировать счетчик данного символа (buildAlphabetModel)

10.1.1.3 Иначе – добавить символ в алфавит и присвоить счетчику 1

10.1.1.4 Инкрементировать счетчик числа символов в тексте

10.1.2 Добавить информацию о EOF-символе (счетчик – 1)

10.1.3 Рассчитать вероятности и левые границы всех символов алфавита на основе данных счетчика и длины входного текста (convertAlphabetModel)

10.2 Считать входной текст порциями по 10000 символов(readDataForPack)

10.3 Упаковать текст (packText)

10.3.1 Для каждого символа текста:

10.3.1.1Сузить интервал для текущего символа (constrictInterval)

10.3.1.1.1Сохранить полученный интервал

10.3.2 Рассчитать среднее арифметического границ полученного интервала (findArithmeticalMean)

10.4 Записать полученный интервал и модель алфавита в выходной файл (writeDataForPack)

11 Если выбран режим распаковки (unpackFile)

11.1 Считать строку с интервалом из входного файла и записать его в число типа number(readDataForUnpack)

11.2Считать модель алфавита и записать ее в массив строк

11.3 Разобрать считанную модель (buildModelFromText)

11.4 Если модель некорректна

11.4.1 Вывести сообщение об ошибке в лог и прекратить работу программы

11.5 Произвести распаковку текста (unpackText)

11.5.1Пока не встретился символ конца файла

11.5.1.1 Найти для данного числа соответствующий интервал с помощью модели алфавита (findSymbolAccordingToInterval)

11.5.1.1.1Получить букву из алфавита исходя из полученного интервала

11.5.1.2Сохранить в строку с выходным текстом полученную букву

11.6 Записать полученный текст в выходной файл (writeDataForUnpack)

12 Записать в лог сообщение об окончании работы программы

Примечание: числа в памяти компьютера представлены с помощью библиотеки GMP, в которой реализована «длинная» арифметика в виде классов целых чисел.

Подробная информация о сообщениях об ошибках, записях лога и ситуациях, к ним приводящим, содержится в приложении Б.Г.

Подробное описание функций приведено в приложении В.

**Приложение Б.Б. Пример входных и выходных данных**

Режим упаковки:

Входные данные:

Исходный текст:

trollollo

Выходные данные:

Среднее значение полученного диапазона:

143079744

Длина числа: 10

Таблица 1. Модель алфавита для упаковки

|  |  |  |
| --- | --- | --- |
| Символ | Вероятность | Интервал |
| t | 0.1 | [0.0 ; 0.1] |
| r | 0.1 | [0.1 ; 0.2] |
| o | 0.3 | [0.2 ; 0.5] |
| l | 0.4 | [0.5 ; 0.9] |
| EOF | 0.1 | [0.9 ; 1.0] |

Режим распаковки:

Входные данные:

Среднее значение полученного диапазона:

9406833165

Длина числа: 11

Таблица 2. Модель алфавита: для распаковки

|  |  |  |
| --- | --- | --- |
| Буква | Вероятность | Интервал |
| а | 0.3 | [0.0 ; 0.3] |
| д | 0.1 | [0.3 ; 0.4] |
| п | 0.1 | [0.4 ; 0.5] |
| т | 0.1 | [0.5 ; 0.6] |
| ц | 0.1 | [0.6 ; 0.7] |
| и | 0.1 | [0.7 ; 0.8] |
| я | 0.1 | [0.8 ; 0.9] |
| EOF | 0.1 | [0.9 ; 1.0] |

Выходные данные:

Исходный текст:

адаптация

**Приложение Б.В. Описание ситуаций с корректным и некорректным оформлением модели алфавита**

Таблица3.Ситуации с оформлением модели

|  |  |  |  |
| --- | --- | --- | --- |
| Тип ситуации | Описание ситуации | Запись | Правильная запись |
| Кор-ректная запись | - | 456g3 | - |
| Некор-ректная запись | Отсутствует символ | 4563 | 456g3 |
| Более 1 символа | 456gw3 | 456g3 |
| Неверный порядок символа и чисел | g4563 | 456g3 |
| В одной строке более одного описания | 456g3 4t1 | 456g3  4t1 |

**Приложение Б.Г. Сообщения об ошибках и сообщения лога**

Таблица 4. Коды возможных ошибок с расшифровкой

| Код ошибки | Описание ошибки | Пример, приводящий к ошибке | Текст ошибки,записываемый в лог |
| --- | --- | --- | --- |
| 001 | Программе передано недостаточное количество аргументов. | NumeralPacking.exe–uC:\Users\User\Desktop\packedFile.txt | Error: There aren’t enough files. |
| 002 | Программе передано слишком много аргументов. | NumeralPacking.exe -u C:\Users\User\Desktop\packedFile.txt C:\Users\User\Desktop\unpackedFile.txt  C:\Users\User\Desktop\unpackedFile.txt | Error: There are too much files. |
| 003 | Выбран неверный режим. | NumeralPacking.exe -w C:\Users\User\Desktop\packedFile.txt C:\Users\User\Desktop\unpackedFile.txt | Error: The mode is incorrect. |
| 004 | Не удалось прочитать входной файл. | NumeralPacking.exe -u C:\Users\User\Desktop\packedFile.txt C:\Users\User\Desktop\unpackedFile.txt | Error: Input filecan’t be read. |
| 005 | Не удалось записать выходной файл. | NumeralPacking.exe -u C:\Users\User\Desktop\packedFile.txt C:\Users\User\Desktop\unpackedFile.txt | Error: Output file can’t be written. |
| 006 | Упакованный текст записан некорректно. | 123433$78778 | Error: Interval is incorrect. |
| 007 | Модель алфавита некорректна. | 456gw3 | Error: Model is incorrect. |

Таблица 5. Все возможные сообщения лога

|  |  |  |
| --- | --- | --- |
| **№** | **Текст сообщения** | **Описание ситуации, при которой происходит запись данного сообщения** |
| 1 | Program started its execution. | Программа начала свое выполнение. |
| 2 | Pack mode is selected. | Выбран режим упаковки. |
| 3 | Unpack mode is selected. | Выбран режим распаковки. |
| 4 | Inspection of command line arguments was successful. Command line arguments are correct and were parsed successfully. | Проверка командной строки была успешной. Аргументы командной строки верны и были успешно разобраны. |
| 5 | The first byte was read for the alphabet building. The alphabet building was begun. | Первый байт был прочитан для построения алфавита. Построение алфавита началось. |
| 6 | The alphabet building was completed. | Алфавит построен. |
| 7 | Data reading was started. | Началось считывание данных. |
| 8 | The text was read. Its amount of symbols is: <количество прочитанных символов> | Текст считан. Количество прочитанных символов: |
| 9 | Packing was started: file <названиефайла> | Началасьупаковкафайла |
| 10 | Packing was finished: file <название файла> | Завершена упаковка файла |
| 11 | The packed text and alphabet were written in file <название файла> | Запакованный текст и алфавит были записаны в файл |
| 12 | The alphabet was read successfully. | Алфавит успешно считан из файла. |
| 13 | The alphabet was parsed successfully. | Алфавит успешно разобран (при считывании в готовом виде из файла). |
| 14 | Unpacking was started: file <название файла> | Началась распаковка файла |

|  |  |  |
| --- | --- | --- |
| 15 | Unpacking was finished: file <название файла> | Завершена распаковка файла |
| 16 | The unpacked text was written in file <название файла> | Распакованный текст был записан в файл |
| 17 | Program finished its execution. | Программа завершила свое выполнение. |

Продолжение таблицы 5.

# Приложение В. Проект программы

1. **Декомпозиция программы на подпрограммы**

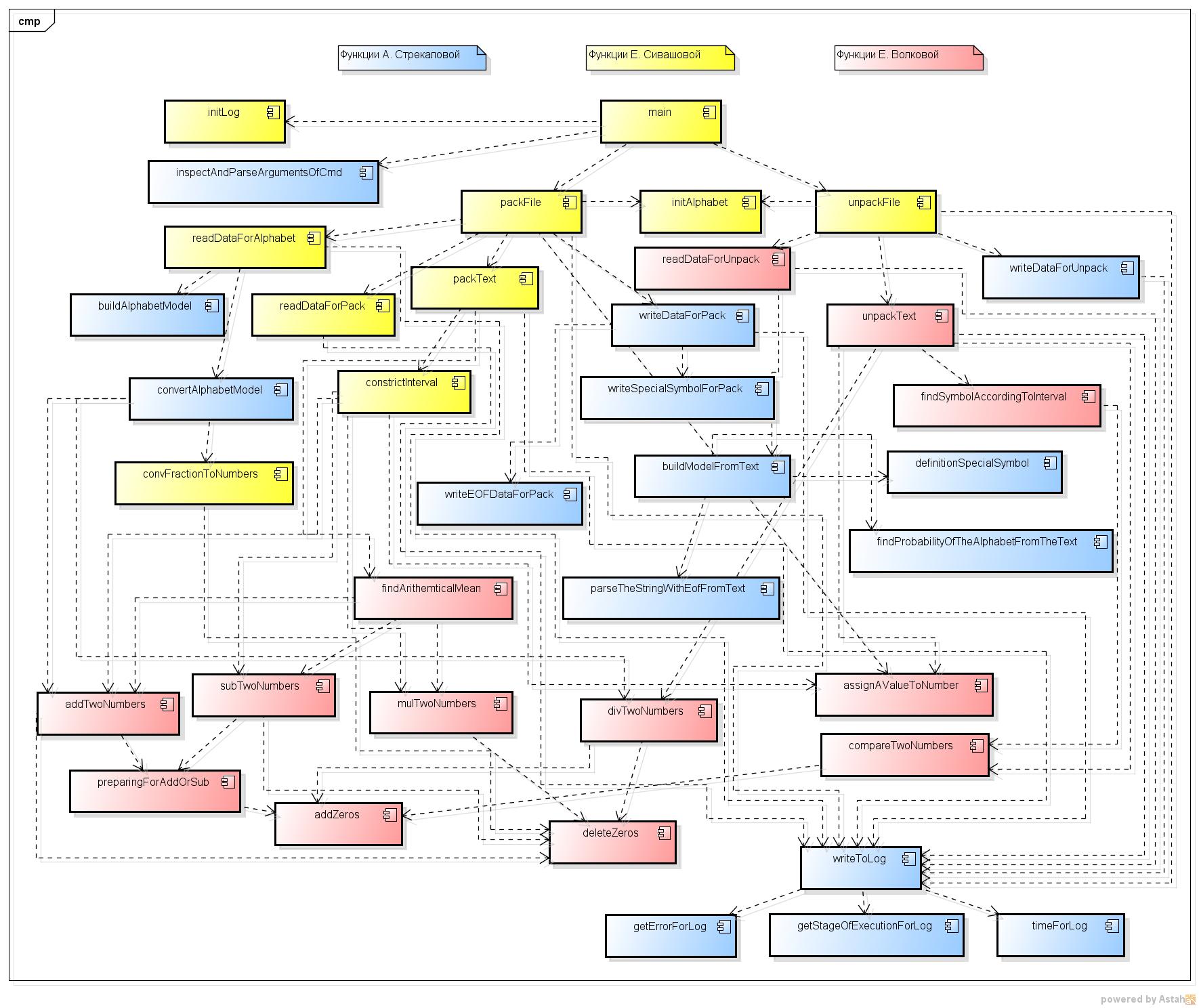


Рисунок В.1 – Иерархия вызова функций

1. **Собственные структуры данных**

/\*!

\*\struct number

\*\brief Структура, содержащая информацию о дробном числе

\*/

struct number

{

unsigned int size; //количество знаков после запятой

mpz\_t num; //значимая численная часть

};

/\*!

\*\struct elementOfAlphabet

\*\brief Структура, содержащая информацию о элементе алфавита

\*/

struct elementOfAlphabet

{

char symbol; //элемент

number probability; //вероятность встречаемости в тексте

number value; //интервал

};

/\*!

\*\enum resultComparing

\*\brief Перечисление, задающее результат сравнения двух чисел

\*/

enum resultComparing

{

equivalent, //равны

less, //меньше

more //больше

};

/\*!

\*\enum positionInText

\*\brief Перечисление, задающее позицию символа в тексте

\*/

enum positionInText

{

isFirst, //первый

isWithin, //между первым и последним

isLast, //последний

isOne //единственный в тексте

};

/\*!

\*\enum typeError

\*\brief Перечисление, задающее тип ошибки

\*/

enum typeError

{

allRight, //нет ошибок

tooLittleArgs, //не хватает аргументов

tooManyArgs, //слишко много аргументов

incorrectMode, //некорректный режим

unluckyReading, //неудачное считывание файла

unluckyWriting, //неудачная запись файла

incorrectInterval, //некорректно задан интервал

incorrectAlphabetModel, //некорректно задана модель алфавита

};

/\*!

\*\enum stageOfExecution

\*\brief Перечисление, задающее этап выполнения программы

\*/

enum stageOfExecution

{

startProgram, //программа начала выполняться

selectedP, //выбран режим упаковки

selectedU, //выбран режим распаковки

inspectArgumCmd, //аргументы корректны и разобраны

firstByteForAlphabetRead, //считан первый байт для алфавита

alphabetWasBuilt, //алфавит построен

firstByteForActionRead, //началось считывание данных

textRead, //текст(число)считан

startPack, //началась упаковка файла <имя файла>

finishPack, //закончена упаковка файла <имя файла>

writePackedTextAndAlphabet, //записаны запакованный текст и алфавит в файл <имя файла>

alphabetRead, //считан алфавит

alphabetWasParsed, //разобран алфавит

startUnpack, //началась распаковка файла <имя файла>

finishUnpack, //закончислась распаковка файла <имя файла>

writeUnpackedText, //записан распакованный текст в файл <имя файла>

finishProgram, //программа завершилась

};

/\*!

\*\struct log

\*\brief Структура, содержащая информацию для лог-файла

\*/

struct log

{

char logFilename[1000]; //поток для чтения и записи лог-информации

char inputFileName[1000]; //имя входного файла

char outputFileName[1000]; //имя выходного файла

enum typeError error; //перечисление ошибок

enum stageOfExecution stageInfo; //перечисление этапа выполнения программы

mpz\_t lengthOfText; //длина текста

};

1. **Перечень подпрограмм**

/\*!

\*\brief Cтроит модель алфавита на основе исходного текста

\*\param[in] symbol указатель на символ входного текста

\*\param[in|out] alphabetSize указатель на размер алфавита

\*\param[in|out] alphabet указатель на модель алфавита

\*/

void buildAlphabetModel(char symbol, unsigned int \*alphabetSize, elementOfAlphabet \*alphabet);

/\*!

\*\brief Преобразует модель алфавита к нужному виду

\*\param[in] lengthText длина исходного текста

\*\param[in] alphabetSize размер алфавита

\*\param[in|out] alphabet указатель на модель алфавита

\*/

void convertAlphabetModel(mpz\_t lengthText, unsigned int alphabetSize, elementOfAlphabet \*alphabet);

/\*!

\*\brief Проверяет и строит модель алфавита на основе прочитанного файла

\*\param[in] textWithModel указатель на текст с алфавитом

\*\param[in] alphabetSize размер алфавита

\*\param[out] alphabet указатель на модель алфавита

\*\return типошибки

\*/

typeError buildModelFromText(char\*\* textWithModel, elementOfAlphabet \*alphabet, unsigned int\* alphabetSize);

/\*!

\*\brief Разбирает строку с текста с eof и записывает в алфавит

\*\param[in] eofStr указатель на строку с eof

\*\param[in] alphabetSize размер алфавита

\*\param[in|out] alphabet указатель на модель алфавита

\*/

void parseTheStringWithEofFromText(char \*eofStr, unsigned int alphabetSize, elementOfAlphabet \*alphabet);

/\*!

\*\brief Находит и записывает вероятности символов с текста в алфавит

\*\param[in] alphabetSize размер алфавита

\*\param[in|out] alphabet указатель на модель алфавита

\*/

void findProbabilityOfTheAlphabetFromTheText(unsigned int alphabetSize, elementOfAlphabet \*alphabet);

/\*!

\*\brief Присваивает соответствующий спец. символ буферу исходя из входящего символа

\*\param[in] symbolFromString символ строки

\*\param[in|out] isNumb признак того, что символ является цифрой

\*\param[out] bufSymbol спец. символ соответствующий букве

\*/

char definitionSpecialSymbol(char symbolFromString, bool\* isNumb);

/\*!

\*\brief Создает массив структур с алфавитом и инициализирует нулями все числа

\*\param[in] amount размер создаваемого массива

\*\return созданныймассив

\*/

elementOfAlphabet \* initAlphabet(unsigned int amount);

/\*!

\*\brief Функция добавления нулей в конец числа

\*\param[in] zerosCount Количество нулей

\*\param[in|out] numForAddZeros Число

\*/

void addZeros(unsigned int zerosCount, mpz\_t \*numForAddZeros);

/\*!

\*\brief Функция удаления нулей из конца числа

\*\param[in|out] numForDeleteZeros Число из которого удаляются нули

\*/

void deleteZeros(number \*numForDeleteZeros);

/\*!

\*\brief Функция подготовки числа к сложению или вычитанию

\*\param[in] left Левый операнд

\*\param[in] right Правый операнд

\*\param[out] newNum Численная часть операнда с добавленными нулями

\*\param[out] resultSize Размер результата сложения или вычитания

\*\return Значение, указывающее, какой операнд был изменен: 0 - ни один, 1 - левый, 2 - правый

\*/

int preparingForAddOrSub (number left, number right, mpz\_t\* newNum, unsigned int \* resultSize);

/\*!

\*\brief Функция сложения

\*\param[in] left Левый операнд

\*\param[in] right Правый операнд

\*\return Результат сложения

\*/

number addTwoNumbers(number left, number right);

/\*!

\*\brief Функция вычитания

\*\param[in] left Левый операнд

\*\param[in] right Правый операнд

\*\return Результат вычитания

\*/

number subTwoNumbers(number left, number right);

/\*!

\*\brief Функция умножения

\*\param[in] left Левый операнд

\*\param[in] right Правый операнд

\*\return Результат умножения

\*/

number mulTwoNumbers(number left, number right);

/\*!

\*\brief Функция сравнения

\*\param[in] left Левый операнд

\*\param[in] right Правый операнд

\*\return Результат сравнения, equivalent - операнды равны, less - левый операнд меньше, more - правый операнд больше

\*/

resultComparing compareTwoNumbers(const number left, const number right);

/\*!

\*\brief Функция присваивания значения

\*\param[in] source Число, которое надо присвоить

\*\param[out] newNum Число, в которое присваивается значение

\*/

void assignAValueToNumber(const number source, number \* newNum);

/\*!

\*\brief Функция деления

\*\param[in] dividend Делимое

\*\param[in] divider Делитель

\*\return Результат деления

\*/

number divTwoNumbers(number dividend, number right);

/\*!

\*\brief Функция, преобразующая дробь, записанную 2 числами, в числа типа number

\*\param[in] numerator числитель исходной дроби

\*\param[in] denominator знаменатель исходной дроби

\*\param[out] newNumerator числитель, записанный числом типа number

\*\param[out] newDenominator знаменатель, записанный числом типа number

\*/

void convFractionToNumbers(mpz\_t numerator, mpz\_t denominator, number\* newNumerator, number\* newDenominator);

/\*!

\*\brief Сужает интервал для текущего символа

\*\param[in] symbol текущий символ

\*\param[in] alphabet указатель на модель алфавита

\*\param[in] alphabetSize размер алфавита

\*\param[in] isLast тип позиции символа в тексте

\*\param[in|out] leftBorder левая граница интервала

\*\param[in|out] rightBorder правая граница интервала

\*/

void constrictInterval(char symbol, positionInText posInText, elementOfAlphabet \*alphabet, unsigned int alphabetSize, number& leftBorder, number& rightBorder);

/\*!

\*\brief Функция вычисления середины интервала

\*\param[in] leftBorder указатель на левую границу интервала

\*\param[in] rightBorder указатель на правую границу интервала

\*\param[out] middleValue указатель на среднее значение интервала

\*/

void findArithmeticalMean(number leftBorder, number rightBorder, number\* middleValue);

/\*!

\*\brief Функция нахождения символа по интервалу

\*\param[in] currentNumber указатель на текущее число

\*\param[in] alphabet указатель на алфавит

\*\return искомый символ

\*/

elementOfAlphabet findSymbolAccordingToInterval(number currentNumber, elementOfAlphabet \* alphabet, unsigned int alphabetSize);

/\*!

\*\brief Проверяет корректность аргументов командной строки и разбирает их

\*\param[in] argc количество аргументов командной строки

\*\param[in] argv аргументы командной строки

\*\param[in] logInfo структура с логом

\*\param[out] args массив аргументов командной строки

\*\return тип ошибки

\*/

typeError inspectAndParseArgumentsOfCmd(int argc, char argv[4][1000], log\* logInfo);

/\*!

\*\brief Возвращает инициализированную начальными значениями (нулями) структуру с логом

\*\return инициализированная нулями "чистая" структура

\*/

log initLog();

/\*!

\*\brief Записывает сообщение в лог-файл

\*\param[in] tmp структура, содержащая информацию для лог-файла

\*/

void writeToLog(struct log tmp);

/\*!

\*\brief Получает строку с сообщением о возникшей ошибке в программе

\*\param[in] tmpEr перечисление, задающее тип ошибки

\*\return строка с сообщением об ошибке

\*/

char\* getErrorForLog(typeError tmpEr);

/\*!

\*\brief Получает сообщение с этапом выполнения программы

\*\param[in] tmp структура, содержащая информацию для лог-файла

\*\return строка с этапом выполнения программы

\*/

char\* getStageOfExecutionForLog(struct log tmp);

/\*!

\*\brief Определяет текущее время

\*\param[in|out] clock строка, содержащая текущее время

\*/

void timeForLog(char\* clock);

/\*!

\*\brief Производит упаковку файла

\*\param[in] inputStream входной поток

\*\param[in] logInfo структура лога

\*\param[in|out] outputStream выходной поток

\*\return тип ошибки

\*/

typeError packFile(FILE\* inputStream, log logInfo, FILE\* outputStream);

/\*!

\*\brief Производит упаковку текста

\*\param[in] text исходный текст

\*\param[in] alphabet указатель на модель алфавита

\*\param[in] sizeOfAlphabet размер алфавита

\*\param[in] markOfTest признак того, откуда вызывается функция (из тестов или из основной программы)

\*\param[in] logInfo структура с логом

\*\return запакованный текст в виде числа

\*/

number packText(char\* text, elementOfAlphabet\* alphabet, unsigned int sizeOfAlphabet, bool markOfTest,log logInfo);

/\*!

\*\brief Производит чтение текста для построения алфавита при упаковке

\*\param[in] inputStream входной поток

\*\param[in] logInfo структура лога

\*\param[out] alphabet указатель на алфавит

\*\param[out] sizeOfAlphabet указатель на размер алфавита

\*\return тип ошибки

\*/

typeError readDataForAlphabet(FILE\* inputStream, log logInfo, elementOfAlphabet\* alphabet, unsigned int\* sizeOfAlphabet);

/\*!

\*\brief Производит чтение текста при упаковке

\*\param[in] inputStream входной поток

\*\param[in] logInfo структура лога

\*\param[out] text указатель на считанный текст

\*\return типошибки

\*/

typeError readDataForPack(FILE\* inputStream, log logInfo, char\* text);

/\*!

\*\brief Производит запись данных при упаковке

\*\param[in] logInfo структура лога

\*\param[in] packedText число, представляющее собой запакованный текст

\*\param[in] alphabet уазатель на алфавит

\*\param[in] sizeOfAlphabet указатель на размер алфавита

\*\param[in|out] outputStream выходной поток

\*\return типошибки

\*/

typeError writeDataForPack(log logInfo, number packedText, elementOfAlphabet \* alphabet, unsigned int sizeOfAlphabet, FILE\* outputStream);

/\*!

\*\brief Производит запись данных EOF при упаковке

\*\param[in] alphabet уазатель на алфавит

\*\param[in] sizeOfAlphabet указатель на размер алфавита

\*\param[in|out] outputStream выходной поток

\*\return errorWriting признак ошибки записи

\*/

bool writeEOFDataForPack(elementOfAlphabet \* alphabet, unsigned int sizeOfAlphabet, FILE\* outputStream);

/\*!

\*\brief Производит запись спец. символов('\t', '\n', ' ', цифра) при упаковке

\*\param[in] alphabet структура алфавита

\*\param[in|out] outputStream выходной поток

\*\return errorWriting признак ошибки записи

\*/

bool writeSpecialSymbolForPack(elementOfAlphabet alphabet, FILE\* outputStream);

/\*!

\*\brief Производит чтение данных при распаковке

\*\param[in] inputStream входной поток

\*\param[in] logInfo структура лога

\*\param[out] alphabet указатель на модель алфавита

\*\param[out] sizeOfAlphabet размер алфавита

\*\param[out] NumberForUnpack число, в котором запакован текст

\*\return тип ошибки

\*/

typeError readDataForUnpack(FILE\* inputStream, log logInfo, elementOfAlphabet \*\* alphabet, unsigned int\* sizeOfAlphabet, number \* NumberForUnpack);

/\*!

\*\brief Производит запись данных при распаковке

\*\param[in] text указатель на распакованный текст

\*\param[in] logInfo структура с логом

\*\param[out] outputStream выходной поток

\*\return типошибки

\*/

typeError writeDataForUnpack(char\* unpackText, log logInfo, FILE\* outputStream);

/\*!

\*\brief Функция распаковки текста

\*\param[in] middleValue указатель на среднее значение интервала

\*\param[in] alphabet указатель на модель алфавита

\*\param[in] alphabetSize размер алфавита

\*\param[in] logInfo структура с логом

\*\param[in] markOfTest признак того, откуда вызывается функция (из тестов или из основной программы)

\*\param[out] resultText текст, преобразованный из числа

\*/

void unpackText(number middleValue, elementOfAlphabet \* alphabet, unsigned int alphabetSize, log logInfo, bool markOfTest, char\* resultText);

/\*!

\*\brief Производит распаковку файла

\*\param[in] inputStream входной поток

\*\param[in] logInfo структура лога

\*\param[in|out] outputStream выходной поток

\*\return тип ошибки

\*/

typeError unpackFile(FILE\* inputStream, log logInfo, FILE\* outputStream);

# Приложение Г. Модульные тесты

**Модульные тесты для подпрограммы addTwoNumbers**

//Слагаемые имеют одинаковый размер и число нулей

voidequalSizesAndCountOfZeros()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 31);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 43);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n01. Test: equalSizesAndCountOfZeros fail! The result is not the same!\n");

}

//Слагаемые имеют одинаковый размер

voidequalSizes()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 9);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 21);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n02. Test: equalSizes fail! The result is not the same!\n");

}

//Слагаемые имеют одинаковый число нулей

voidequalCountOfZeros()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=3;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 3);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 42);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n03. Test: equalCountOfZeros fail! The result is not the same!\n");

}

//Слагаемые имеют разный размер и в результате число разрядов увеличивается

void increasingDischargeOfResultWithDifferentSizes()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=3;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 9);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 102);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n04. Test: increasingDischargeOfResultWithDifferentSizes fail! The result is not the same!\n");

}

//В результате сложения число разрядов увеличивается

void increasingDischargeOfResult()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 6);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 6);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 12);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n05. Test: increasingDischargeOfResult fail! The result is not the same!\n");

}

//Левый операнд больше правого

voidleftOperandBiggerThanRight()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=2;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 2);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 31);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 231);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n06. Test: leftOperandBiggerThanRight fail! The result is not the same!\n");

}

//Правый операнд больше левого

void rightOperandBiggerThanLeft()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=3;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 45);

rightOperand.size=2;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 33);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=3;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 375);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n07. Test: rightOperandBiggerThanLeft fail! The result is not the same!\n");

}

//У результата сложения меняется размер

voidresultChangeSize()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 4);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 6);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=3;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 1);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n08. Test: resultChangeSize fail! The result is not the same!");

}

//Прибавляется нуль

voidaddingZero()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 4);

rightOperand.size=0;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 0);

mpz\_init (result.num);

result=addTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 4);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function addTwoNumbers\n08. Test: resultChangeSize fail! The result is not the same!");

}

**Модульные тесты для подпрограммы addZeros**

//Прибавляется 1 нуль

void addZero()

{

mpz\_t result, expectedResult;

mpz\_init (result);

mpz\_set\_ui (result, 1);

mpz\_init (expectedResult);

mpz\_set\_ui (expectedResult, 10);

unsigned int zerosCount=1;

addZeros(zerosCount, &result);

bool testPassed=false;

if (!mpz\_cmp(result, expectedResult))

testPassed=true;

if(testPassed==false)

printf("Testing function addZeros\n01. Test: addZero fail! The result is not the same!\n");

}

//Прибавляется несколько нулей

voidaddTenZeros()

{

mpz\_t result, expectedResult;

mpz\_init (result);

mpz\_set\_ui (result, 1);

mpz\_init (expectedResult);

mpz\_set\_str (expectedResult, "10000000000", 10);

unsigned int zerosCount=10;

addZeros(zerosCount, &result);

bool testPassed=false;

if (!mpz\_cmp(result, expectedResult))

testPassed=true;

if(testPassed==false)

printf("Testing function addZeros\n02. Test: addTenZeros fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы assignAValueToNumber**

//Присваивается нуль

void assignZeroValue()

{

number source, newNum;

source.size=0;

mpz\_init (source.num);

mpz\_set\_ui (source.num, 0);

mpz\_init (newNum.num);

assignAValueToNumber(source, &newNum);

bool testPassed=false;

if (compareTwoNumbers(source, newNum)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function assignAValueToNumber\n01. Test: assignZeroValue fail! The result is not the same!\n");

}

//Присваивается значение

voidassignValue()

{

number source, newNum;

source.size=2;

mpz\_init (source.num);

mpz\_set\_ui (source.num, 4);

mpz\_init (newNum.num);

assignAValueToNumber(source, &newNum);

bool testPassed=false;

if (compareTwoNumbers(source, newNum)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function assignAValueToNumber\n02. Test: assignValue fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы buildAlphabetModel**

//Добавление первого символа в пустую модель алфавита

void theAdditionOfTheFirstSymbolInTheAlphabetEmpty()

{

//arrange

unsigned int number=0;

unsigned int\* size=&number;

char symbol='o';

elementOfAlphabet \*alphabet=initAlphabet(1);

unsigned int testNumber=1;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[1]={NULL};

testAlphabet[0].symbol='o';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 1);

testAlphabet[0].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: theAdditionOfTheFirstSymbolInTheAlphabetEmpty fail! The result is not the same!\n");

delete[]alphabet;

}

//В моделе алфавита нет текущего символа(far)

void inTheAlphabetIsNotTheCurrentSymbol()

{

//arrange

unsigned int number=1;

unsigned int\* size=&number;

char symbol='a';

elementOfAlphabet \*alphabet=initAlphabet(2);

alphabet[0].symbol='f';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui(alphabet[0].probability.num, 1);

alphabet[0].probability.size=0;

unsigned int testNumber=2;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[2]={NULL};

testAlphabet[0].symbol='f';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 1);

testAlphabet[0].probability.size=0;

testAlphabet[1].symbol='a';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 1);

testAlphabet[1].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: inTheAlphabetIsNotTheCurrentSymbol fail! The result is not the same!\n");

delete[]alphabet;

}

//В моделе алфавита есть текущий символ текста

void inTheAlphabeHasACurrentTextSymbol()

{

//arrange

unsigned int number=3;

unsigned int\* size=&number;

char symbol='l';

elementOfAlphabet \*alphabet=initAlphabet(3);

alphabet[0].symbol='d';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=0;

alphabet[2].symbol='l';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 1);

alphabet[2].probability.size=0;

unsigned int testNumber=3;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[3]={NULL};

testAlphabet[0].symbol='d';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 1);

testAlphabet[0].probability.size=0;

testAlphabet[1].symbol='o';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 1);

testAlphabet[1].probability.size=0;

testAlphabet[2].symbol='l';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 2);

testAlphabet[2].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: inTheAlphabeHasACurrentTextSymbol fail! The result is not the same!\n");

delete[]alphabet;

}

//Текст содержащий пробелы( a b)

void theTextContainingSpaces()

{

//arrange

unsigned int number=3;

unsigned int\* size=&number;

char symbol=' ';

elementOfAlphabet \*alphabet=initAlphabet(3);

alphabet[0].symbol=' ';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=0;

alphabet[1].symbol='a';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=0;

alphabet[2].symbol='b';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 1);

alphabet[2].probability.size=0;

unsigned int testNumber=3;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[3]={NULL};

testAlphabet[0].symbol=' ';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 2);

testAlphabet[0].probability.size=0;

testAlphabet[1].symbol='a';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 1);

testAlphabet[1].probability.size=0;

testAlphabet[2].symbol='b';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 1);

testAlphabet[2].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: theTextContainingSpaces fail! The result is not the same!\n");

delete[]alphabet;

}

//Многострочныйтекст(do\re\mi)

void multipleLinesOfText()

{

//arrange

unsigned int number=7;

unsigned int\* size=&number;

char symbol='i';

elementOfAlphabet \*alphabet=initAlphabet(8);

alphabet[0].symbol='d';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=0;

alphabet[2].symbol='\r';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=0;

alphabet[3].symbol='\n';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 2);

alphabet[3].probability.size=0;

alphabet[4].symbol='r';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=0;

alphabet[5].symbol='e';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 1);

alphabet[5].probability.size=0;

alphabet[6].symbol='m';

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=0;

unsigned int testNumber=8;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[8]={NULL};

testAlphabet[0].symbol='d';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 1);

testAlphabet[0].probability.size=0;

testAlphabet[1].symbol='o';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 1);

testAlphabet[1].probability.size=0;

testAlphabet[2].symbol='\r';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 2);

testAlphabet[2].probability.size=0;

testAlphabet[3].symbol='\n';

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 2);

testAlphabet[3].probability.size=0;

testAlphabet[4].symbol='r';

mpz\_init(testAlphabet[4].probability.num);

mpz\_set\_ui (testAlphabet[4].probability.num, 1);

testAlphabet[4].probability.size=0;

testAlphabet[5].symbol='e';

mpz\_init(testAlphabet[5].probability.num);

mpz\_set\_ui (testAlphabet[5].probability.num, 1);

testAlphabet[5].probability.size=0;

testAlphabet[6].symbol='m';

mpz\_init(testAlphabet[6].probability.num);

mpz\_set\_ui (testAlphabet[6].probability.num, 1);

testAlphabet[6].probability.size=0;

testAlphabet[7].symbol='i';

mpz\_init(testAlphabet[7].probability.num);

mpz\_set\_ui (testAlphabet[7].probability.num, 1);

testAlphabet[7].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: multipleLinesOfText fail! The result is not the same!\n");

delete[]alphabet;

}

//Текст содержащий цифры(220)

void theTextContainingNumbers()

{

//arrange

unsigned int number=1;

unsigned int\* size=&number;

char symbol='0';

elementOfAlphabet \*alphabet=initAlphabet(2);

alphabet[0].symbol='2';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 2);

alphabet[0].probability.size=0;

unsigned int testNumber=2;

unsigned int\* testSize=&testNumber;

elementOfAlphabet testAlphabet[2]={NULL};

testAlphabet[0].symbol='2';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 2);

testAlphabet[0].probability.size=0;

testAlphabet[1].symbol='0';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 1);

testAlphabet[1].probability.size=0;

//act

buildAlphabetModel(symbol, size, alphabet);

//assert

if(\*size!=\*testSize || !isEqualAlphabet(alphabet, testAlphabet, size))

printf("Testing function buildAlphabetModel\n01. Test: theTextContainingNumbers fail! The result is not the same!\n");

delete[]alphabet;

}

**Модульные тесты для подпрограммы buildModelFromText**

//Корректно разобран небольшой текст

void сorrectlyParseSmallText()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[4];

textWithModel[0]="0H0";

textWithModel[1]="125i3";

textWithModel[2]="25!2";

textWithModel[3]="875e3o125f3";

elementOfAlphabet \*alphabet=initAlphabet(4);

unsigned int count=4;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[4]={NULL};

testAlphabet[0].symbol='H';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 125);

testAlphabet[0].probability.size=3;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='i';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 125);

testAlphabet[1].probability.size=3;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 125);

testAlphabet[1].value.size=3;

testAlphabet[2].symbol='!';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 625);

testAlphabet[2].probability.size=3;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 25);

testAlphabet[2].value.size=2;

//EndOfFile

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 125);

testAlphabet[3].probability.size=3;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 875);

testAlphabet[3].value.size=3;

unsigned int testCount=4;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=allRight;

//assert

if(\*alphabetSize!=\*testAlphabetSize || !is\_EqualAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function buildModelFromText\n01. Test: сorrectlyParseSmallText fail! The result is not the same!\n");

delete[]alphabet;

}

//Корректно разобран большой текст(trooooollooooollooooo->trollollo)

voidсorrectlyParseLargeText()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[7];

textWithModel[0]="0t0";

textWithModel[1]="606r4";

textWithModel[2]="1212o4";

textWithModel[3]="66665l5";

textWithModel[4]="90907-5";

textWithModel[5]="93937>5";

textWithModel[6]="96967e5o303f4";

elementOfAlphabet \*alphabet=initAlphabet(7);

unsigned int count=7;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[7]={NULL};

testAlphabet[0].symbol='t';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 606);

testAlphabet[0].probability.size=4;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='r';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 606);

testAlphabet[1].probability.size=4;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 606);

testAlphabet[1].value.size=4;

testAlphabet[2].symbol='o';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 54545);

testAlphabet[2].probability.size=5;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 1212);

testAlphabet[2].value.size=4;

testAlphabet[3].symbol='l';

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 24242);

testAlphabet[3].probability.size=5;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 66665);

testAlphabet[3].value.size=5;

testAlphabet[4].symbol='-';

mpz\_init(testAlphabet[4].probability.num);

mpz\_set\_ui (testAlphabet[4].probability.num, 303);

testAlphabet[4].probability.size=4;

mpz\_init(testAlphabet[4].value.num);

mpz\_set\_ui (testAlphabet[4].value.num, 90907);

testAlphabet[4].value.size=5;

testAlphabet[5].symbol='>';

mpz\_init(testAlphabet[5].probability.num);

mpz\_set\_ui (testAlphabet[5].probability.num, 303);

testAlphabet[5].probability.size=4;

mpz\_init(testAlphabet[5].value.num);

mpz\_set\_ui (testAlphabet[5].value.num, 93937);

testAlphabet[5].value.size=5;

//EndOfFile

mpz\_init(testAlphabet[6].probability.num);

mpz\_set\_ui (testAlphabet[6].probability.num, 303);

testAlphabet[6].probability.size=4;

mpz\_init(testAlphabet[6].value.num);

mpz\_set\_ui (testAlphabet[6].value.num, 96967);

testAlphabet[6].value.size=5;

unsigned int testCount=7;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=allRight;

//assert

if(\*alphabetSize!=\*testAlphabetSize || !is\_EqualAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function buildModelFromText\n01. Test: сorrectlyParseLargeText fail! The result is not the same!\n");

delete[]alphabet;

}

//Корректно разобран текст c повторяющимися символами(ssss)

voidсorrectlyParseTextFromRecurringSymbol()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[2];

textWithModel[0]="0s0";

textWithModel[1]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(2);

unsigned int count=2;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[2]={NULL};

testAlphabet[0].symbol='s';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 8);

testAlphabet[0].probability.size=1;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

//EndOfFile

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 2);

testAlphabet[1].probability.size=1;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 8);

testAlphabet[1].value.size=1;

unsigned int testCount=2;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=allRight;

//assert

if(\*alphabetSize!=\*testAlphabetSize || !is\_EqualAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function buildModelFromText\n01. Test: сorrectlyParseTextFromRecurringSymbol fail! The result is not the same!\n");

delete[]alphabet;

}

//Корректно разобран текст содержащий спец. символ \n(a\nb)

voidсorrectlyParseTextContainingSpecialSymbolSlashN()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[5];

textWithModel[0]="0a0";

textWithModel[1]="r2r1";

textWithModel[2]="n4n1";

textWithModel[3]="6b1";

textWithModel[4]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(5);

unsigned int count=5;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[5]={NULL};

testAlphabet[0].symbol='a';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 2);

testAlphabet[0].probability.size=1;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='\r';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 2);

testAlphabet[1].probability.size=1;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 2);

testAlphabet[1].value.size=1;

testAlphabet[2].symbol='\n';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 2);

testAlphabet[2].probability.size=1;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 4);

testAlphabet[2].value.size=1;

testAlphabet[3].symbol='b';

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 2);

testAlphabet[3].probability.size=1;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 6);

testAlphabet[3].value.size=1;

//EndOfFile

mpz\_init(testAlphabet[4].probability.num);

mpz\_set\_ui (testAlphabet[4].probability.num, 2);

testAlphabet[4].probability.size=1;

mpz\_init(testAlphabet[4].value.num);

mpz\_set\_ui (testAlphabet[4].value.num, 8);

testAlphabet[4].value.size=1;

unsigned int testCount=5;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=allRight;

//assert

if(\*alphabetSize!=\*testAlphabetSize || !is\_EqualAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function buildModelFromText\n01. Test: сorrectlyParseTextContainingSpecialSymbol fail! The result is not the same!\n");

delete[]alphabet;

}

//Корректно разобран текст содержащий спец. символ цифру(1996)

void сorrectlyParseTextContainingSpecialSymbolNumber()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[4];

textWithModel[0]="c0c1c0";

textWithModel[1]="c2c9c1";

textWithModel[2]="c6c6c1";

textWithModel[3]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(4);

unsigned int count=4;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[4]={NULL};

testAlphabet[0].symbol='1';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 2);

testAlphabet[0].probability.size=1;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='9';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 4);

testAlphabet[1].probability.size=1;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 2);

testAlphabet[1].value.size=1;

testAlphabet[2].symbol='6';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 2);

testAlphabet[2].probability.size=1;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 6);

testAlphabet[2].value.size=1;

//EndOfFile

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 2);

testAlphabet[3].probability.size=1;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 8);

testAlphabet[3].value.size=1;

unsigned int testCount=4;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=allRight;

//assert

if(\*alphabetSize!=\*testAlphabetSize || !is\_EqualAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function buildModelFromText\n01. Test: сorrectlyParseTextContainingSpecialSymbolNumber fail! The result is not the same!\n");

delete[]alphabet;

}

//Строка текста содержит только цифры(goo)

void theStringOfTextContainsOnlyDigits()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[3];

textWithModel[0]="050";

textWithModel[1]="25o2";

textWithModel[2]="75e2o25f2";

elementOfAlphabet \*alphabet=initAlphabet(3);

unsigned int count=3;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[3]={NULL};

unsigned int testCount=3;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=incorrectAlphabetModel;

//assert

if(curError!=testError)

printf("Testing function buildModelFromText\n01. Test: theStringOfTextDoesntContainDigits fail! The result is not the same!\n");

delete[]alphabet;

}

//Строка текста не содержит цифры(cat)

void theStringOfTextDoesntContainDigits()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[5];

textWithModel[0]="0c0";

textWithModel[1]="a";

textWithModel[2]="5t1";

textWithModel[3]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(5);

unsigned int count=5;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[5]={NULL};

unsigned int testCount=4;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=incorrectAlphabetModel;

//assert

if(curError!=testError)

printf("Testing function buildModelFromText\n01. Test: theStringOfTextDoesntContainDigits fail! The result is not the same!\n");

delete[]alphabet;

}

//Первый символ в строке разбора не цифра и не спец. символ(д уб)

void firstSymbolIsNotNumberAndNotSpecialInTheString()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[5];

textWithModel[0]="0д0";

textWithModel[1]="#2p1";

textWithModel[2]="4у1";

textWithModel[3]="6б1";

textWithModel[4]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(5);

unsigned int count=5;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[5]={NULL};

unsigned int testCount=4;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=incorrectAlphabetModel;

//assert

if(curError!=testError)

printf("Testing function buildModelFromText\n01. Test: firstSymbolIsNotNumberAndNotSpecialInTheString fail! The result is not the same!\n");

delete[]alphabet;

}

//В строке после главного символа присутствуют не только цифры(\*))))

void afterTheMainSymbolArePresentNotOnlyNumbers()

{

//arrange

typeError curError, testError;

char\*\* textWithModel= newchar\*[3];

textWithModel[0]="0\*0\*\*";

textWithModel[1]="2)1";

textWithModel[2]="8e1o2f1";

elementOfAlphabet \*alphabet=initAlphabet(3);

unsigned int count=3;

unsigned int\* alphabetSize=&count;

elementOfAlphabet testAlphabet[3]={NULL};

unsigned int testCount=3;

unsigned int\* testAlphabetSize=&testCount;

//act

curError=buildModelFromText(textWithModel, alphabet, alphabetSize);

testError=incorrectAlphabetModel;

//assert

if(curError!=testError)

printf("Testing function buildModelFromText\n01. Test: afterTheMainSymbolArePresentNotOnlyNumbers fail! The result is not the same!\n");

delete[]alphabet;

}

**Модульные тесты для подпрограммы compareTwoNumbers**

//Числа одинакового размера и правое число больше левого

void equalSizesAndRightNumMoreThanLeft()

{

number left, right;

resultComparing result;

left.size=4;

mpz\_init (left.num);

mpz\_set\_ui (left.num, 12);

right.size=4;

mpz\_set\_ui (right.num, 31);

result=compareTwoNumbers(left, right);

bool testPassed=false;

if (result==less)

testPassed=true;

if(testPassed==false)

printf("Testing function compareTwoNumbers\n01. Test: equalSizesAndRightNumMoreThanLeft fail! The result is not the same!\n");

}

//Числа одинакового размера и левое число больше правого

void equalSizesAndLeftNumMoreThanRight()

{

number left, right;

resultComparing result;

left.size=4;

mpz\_init (left.num);

mpz\_set\_ui (left.num, 24);

right.size=4;

mpz\_init (right.num);

mpz\_set\_ui (right.num, 3);

result=compareTwoNumbers(left, right);

bool testPassed=false;

if (result==more)

testPassed=true;

if(testPassed==false)

printf("Testing function compareTwoNumbers\n02. Test: equalSizesAndLeftNumMoreThanRight fail! The result is not the same!\n");

}

//Одинаковые числа

void equalNumbers()

{

number left, right;

resultComparing result;

left.size=4;

mpz\_init (left.num);

mpz\_set\_ui (left.num, 3);

right.size=4;

mpz\_init (right.num);

mpz\_set\_ui (right.num, 3);

result=compareTwoNumbers(left, right);

bool testPassed=false;

if (result==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function compareTwoNumbers\n03. Test: equalNumbers fail! The result is not the same!\n");

}

//Числа разного размера и левое число больше правого

void differentSizesAndLeftNumMoreThanRight()

{

number left, right;

resultComparing result;

left.size=3;

mpz\_init (left.num);

mpz\_set\_ui (left.num, 24);

right.size=4;

mpz\_init (right.num);

mpz\_set\_ui (right.num, 3);

result=compareTwoNumbers(left, right);

bool testPassed=false;

if (result==more)

testPassed=true;

if(testPassed==false)

printf("Testing function compareTwoNumbers\n04. Test: differentSizesAndLeftNumMoreThanRight fail! The result is not the same!\n");

}

//Числа разного размера и правое число больше левого

void differentSizesAndRightNumMoreThanLeft()

{

number left, right;

resultComparing result;

left.size=4;

mpz\_init (left.num);

mpz\_set\_ui (left.num, 24);

right.size=3;

mpz\_init (right.num);

mpz\_set\_ui (right.num, 3);

result=compareTwoNumbers(left, right);

bool testPassed=false;

if (result==more)

testPassed=true;

if(testPassed==false)

printf("Testing function compareTwoNumbers\n05. Test: differentSizesAndLeftNumMoreThanRight fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы constrictInterval**

//Проверяемый символ является первым символом текста

void firstSymbolInText()

{

//arrange

char symbol='а';

positionInText posInText=isFirst;

unsigned int alphabetSize=8;

elementOfAlphabet alphabet[8];

alphabet[0].symbol='а';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='д';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='п';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 1);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 4);

alphabet[2].value.size=1;

alphabet[3].symbol='т';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 5);

alphabet[3].value.size=1;

alphabet[4].symbol='ц';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 6);

alphabet[4].value.size=1;

alphabet[5].symbol='и';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 1);

alphabet[5].probability.size=1;

mpz\_init(alphabet[5].value.num);

mpz\_set\_ui (alphabet[5].value.num, 7);

alphabet[5].value.size=1;

alphabet[6].symbol='я';

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=1;

mpz\_init(alphabet[6].value.num);

mpz\_set\_ui (alphabet[6].value.num, 8);

alphabet[6].value.size=1;

//EndOfFile

mpz\_init(alphabet[7].probability.num);

mpz\_set\_ui (alphabet[7].probability.num, 1);

alphabet[7].probability.size=1;

mpz\_init(alphabet[7].value.num);

mpz\_set\_ui (alphabet[7].value.num, 9);

alphabet[7].value.size=1;

number leftBorder;

mpz\_init(leftBorder.num);

leftBorder.size=0;

number rightBorder;

mpz\_init(rightBorder.num);

rightBorder.size=0;

number leftExpect;

mpz\_init(leftExpect.num);

leftExpect.size=0;

number rightExpect;

mpz\_init\_set\_ui(rightExpect.num, 3);

rightExpect.size=1;

//act

constrictInterval(symbol, posInText, alphabet, alphabetSize, leftBorder, rightBorder);

//assert

bool isEqual=isEqualBorders(leftBorder, rightBorder, leftExpect, rightExpect);

if (!isEqual)

printf("Testing function constrictInterval\n01 Test: firstSymbolInText fail! The result is not the same!");

}

//Проверяемый символ находится в середине текста

void symbolWithinText()

{

//arrange

char symbol='e';

positionInText posInText=isWithin;

unsigned int alphabetSize=4;

elementOfAlphabet alphabet[4];

alphabet[0].symbol='y';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 25);

alphabet[0].probability.size=2;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='e';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 25);

alphabet[1].value.size=2;

alphabet[2].symbol='s';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 5);

alphabet[2].value.size=1;

//EndOfFile

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 25);

alphabet[3].probability.size=2;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 75);

alphabet[3].value.size=2;

number leftBorder;

mpz\_init(leftBorder.num);

leftBorder.size=0;

number rightBorder;

mpz\_init\_set\_ui(rightBorder.num, 25);

rightBorder.size=2;

number leftExpect;

mpz\_init\_set\_ui(leftExpect.num, 625);

leftExpect.size=4;

number rightExpect;

mpz\_init\_set\_ui(rightExpect.num, 125);

rightExpect.size=3;

//act

constrictInterval(symbol, posInText, alphabet, alphabetSize, leftBorder, rightBorder);

//assert

bool isEqual=isEqualBorders(leftBorder, rightBorder, leftExpect, rightExpect);

if (!isEqual)

printf("Testing function constrictInterval\n02 Test: symbolWithinText fail! The result is not the same!");

}

//Проверяемый символ является последним символом текста

void lastSymbolInText()

{

//arrange

char symbol='t';

positionInText posInText=isLast;

unsigned int alphabetSize=4;

elementOfAlphabet alphabet[4];

alphabet[0].symbol='c';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 25);

alphabet[0].probability.size=2;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='a';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 25);

alphabet[1].value.size=2;

alphabet[2].symbol='t';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 5);

alphabet[2].value.size=1;

//EndOfFile

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 25);

alphabet[3].probability.size=2;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 75);

alphabet[3].value.size=2;

number leftBorder;

mpz\_init\_set\_ui(leftBorder.num, 625);

leftBorder.size=4;

number rightBorder;

mpz\_init\_set\_ui(rightBorder.num, 125);

rightBorder.size=3;

number leftExpect;

mpz\_init\_set\_ui(leftExpect.num, 10546875);

leftExpect.size=8;

number rightExpect;

mpz\_init\_set\_ui(rightExpect.num, 109375);

rightExpect.size=6;

//act

constrictInterval(symbol, posInText, alphabet, alphabetSize, leftBorder, rightBorder);

//assert

bool isEqual=isEqualBorders(leftBorder, rightBorder, leftExpect, rightExpect);

if (!isEqual)

printf("Testing function constrictInterval\n03 Test: lastSymbolInText fail! The result is not the same!");

}

**Модульные тесты для подпрограммы convertAlphabetModel**

//Правильно преобразован в алфавит небольшой текст(too)

voidсorrectlyTransformedInAlphabetASmallText()

{

// arrange

mpz\_t length;

mpz\_init(length);

mpz\_set\_ui (length, 4);

unsigned int alphabetSize=3;

elementOfAlphabet \*alphabet=initAlphabet(3);

alphabet[0].symbol='t';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 2);

alphabet[1].probability.size=0;

elementOfAlphabet testAlphabet[3]={NULL};

testAlphabet[0].symbol='t';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 25);

testAlphabet[0].probability.size=2;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='o';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 5);

testAlphabet[1].probability.size=1;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 25);

testAlphabet[1].value.size=2;

//EndOfFile

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 25);

testAlphabet[2].probability.size=2;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 75);

testAlphabet[2].value.size=2;

//act

convertAlphabetModel(length, alphabetSize, alphabet);

//assert

if(!isEquaProbabilityAndValueOfTheAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function convertAlphabetModel\n01. Test: ссorrectlyTransformedInAlphabetASmallText! The result is not the same!\n");

delete[]alphabet;

}

//Правильно преобразован в алфавит большой текст(you funny\r\nahahahahahahahahaha)

voidсorrectlyTransformedInAlphabetALargeText()

{

// arrange

mpz\_t length;

mpz\_init(length);

mpz\_set\_ui (length, 31);

unsigned int alphabetSize=11;

elementOfAlphabet \*alphabet=initAlphabet(11);

alphabet[0].symbol='y';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 2);

alphabet[0].probability.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=0;

alphabet[2].symbol='u';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=0;

alphabet[3].symbol=' ';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=0;

alphabet[4].symbol='f';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=0;

alphabet[5].symbol='n';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 2);

alphabet[5].probability.size=0;

alphabet[6].symbol='\r';

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=0;

alphabet[7].symbol='\n';

mpz\_init(alphabet[7].probability.num);

mpz\_set\_ui (alphabet[7].probability.num, 1);

alphabet[7].probability.size=0;

alphabet[8].symbol='a';

mpz\_init(alphabet[8].probability.num);

mpz\_set\_ui (alphabet[8].probability.num, 10);

alphabet[8].probability.size=0;

alphabet[9].symbol='h';

mpz\_init(alphabet[9].probability.num);

mpz\_set\_ui (alphabet[9].probability.num, 9);

alphabet[9].probability.size=0;

elementOfAlphabet testAlphabet[11]={NULL};

testAlphabet[0].symbol='y';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 6451);

testAlphabet[0].probability.size=5;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='o';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 3225);

testAlphabet[1].probability.size=5;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 6451);

testAlphabet[1].value.size=5;

testAlphabet[2].symbol='u';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 6451);

testAlphabet[2].probability.size=5;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 9676);

testAlphabet[2].value.size=5;

testAlphabet[3].symbol=' ';

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 3225);

testAlphabet[3].probability.size=5;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 16127);

testAlphabet[3].value.size=5;

testAlphabet[4].symbol='f';

mpz\_init(testAlphabet[4].probability.num);

mpz\_set\_ui (testAlphabet[4].probability.num, 3225);

testAlphabet[4].probability.size=5;

mpz\_init(testAlphabet[4].value.num);

mpz\_set\_ui (testAlphabet[4].value.num, 19352);

testAlphabet[4].value.size=5;

testAlphabet[5].symbol='n';

mpz\_init(testAlphabet[5].probability.num);

mpz\_set\_ui (testAlphabet[5].probability.num, 6451);

testAlphabet[5].probability.size=5;

mpz\_init(testAlphabet[5].value.num);

mpz\_set\_ui (testAlphabet[5].value.num, 22577);

testAlphabet[5].value.size=5;

testAlphabet[6].symbol='\r';

mpz\_init(testAlphabet[6].probability.num);

mpz\_set\_ui (testAlphabet[6].probability.num, 3225);

testAlphabet[6].probability.size=5;

mpz\_init(testAlphabet[6].value.num);

mpz\_set\_ui (testAlphabet[6].value.num, 29028);

testAlphabet[6].value.size=5;

testAlphabet[7].symbol='\n';

mpz\_init(testAlphabet[7].probability.num);

mpz\_set\_ui (testAlphabet[7].probability.num, 3225);

testAlphabet[7].probability.size=5;

mpz\_init(testAlphabet[7].value.num);

mpz\_set\_ui (testAlphabet[7].value.num, 32253);

testAlphabet[7].value.size=5;

testAlphabet[8].symbol='a';

mpz\_init(testAlphabet[8].probability.num);

mpz\_set\_ui (testAlphabet[8].probability.num, 3225);

testAlphabet[8].probability.size=4;

mpz\_init(testAlphabet[8].value.num);

mpz\_set\_ui (testAlphabet[8].value.num, 35478);

testAlphabet[8].value.size=5;

testAlphabet[9].symbol='h';

mpz\_init(testAlphabet[9].probability.num);

mpz\_set\_ui (testAlphabet[9].probability.num, 29032);

testAlphabet[9].probability.size=5;

mpz\_init(testAlphabet[9].value.num);

mpz\_set\_ui (testAlphabet[9].value.num, 67728);

testAlphabet[9].value.size=5;

//EndOfFile

mpz\_init(testAlphabet[10].probability.num);

mpz\_set\_ui (testAlphabet[10].probability.num, 3225);

testAlphabet[10].probability.size=5;

mpz\_init(testAlphabet[10].value.num);

mpz\_set\_ui (testAlphabet[10].value.num, 9676);

testAlphabet[10].value.size=4;

//act

convertAlphabetModel(length, alphabetSize, alphabet);

//assert

if(!isEquaProbabilityAndValueOfTheAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function convertAlphabetModel\n01. Test: сorrectlyTransformedInAlphabetALargeText! The result is not the same!\n");

delete[]alphabet;

}

**Модульные тесты для подпрограммы convFractionToNumbers**

//Числитель и знаменатель одинаковой длины

void equalSize()

{

//arrange

mpz\_t numerator, denominator;

number newNumerator, newDenominator, expectNewNumerator, expectNewDenominator;

mpz\_init (numerator);

mpz\_set\_ui (numerator, 1);

mpz\_init (denominator);

mpz\_set\_ui (denominator, 2);

expectNewNumerator.size=1;

mpz\_init (expectNewNumerator.num);

mpz\_set\_ui (expectNewNumerator.num, 1);

expectNewDenominator.size=1;

mpz\_init (expectNewDenominator.num);

mpz\_set\_ui (expectNewDenominator.num, 2);

mpz\_init (newNumerator.num);

mpz\_init (newDenominator.num);

//act

convFractionToNumbers(numerator, denominator, &newNumerator, &newDenominator);

//assert

bool testPassed=false;

if (!mpz\_cmp(newNumerator.num, expectNewNumerator.num)

&& !mpz\_cmp(newDenominator.num, expectNewDenominator.num)

&& newNumerator.size==expectNewNumerator.size

&& newDenominator.size==expectNewDenominator.size)

testPassed=true;

if(testPassed==false)

printf("Testing function convFractionToNumbers\n01. Test: equalSize fail! The result is not the same!\n");

}

//Числитель и знаменатель одинаковой длины и число нулей в них одинаково

void equalSizeAndAmountOfZeros()

{

//arrange

mpz\_t numerator, denominator;

number newNumerator, newDenominator, expectNewNumerator, expectNewDenominator;

mpz\_init (numerator);

mpz\_set\_ui (numerator, 325850);

mpz\_init (denominator);

mpz\_set\_ui (denominator, 335790);

expectNewNumerator.size=5;

mpz\_init (expectNewNumerator.num);

mpz\_set\_ui (expectNewNumerator.num, 32585);

expectNewDenominator.size=5;

mpz\_init (expectNewDenominator.num);

mpz\_set\_ui (expectNewDenominator.num, 33579);

mpz\_init (newNumerator.num);

mpz\_init (newDenominator.num);

//act

convFractionToNumbers(numerator, denominator, &newNumerator, &newDenominator);

//assert

bool testPassed=false;

if (!mpz\_cmp(newNumerator.num, expectNewNumerator.num)

&& !mpz\_cmp(newDenominator.num, expectNewDenominator.num)

&& newNumerator.size==expectNewNumerator.size

&& newDenominator.size==expectNewDenominator.size)

testPassed=true;

if(testPassed==false)

printf("Testing function convFractionToNumbers\n02. Test: equalSizeAndAmountOfZeros fail! The result is not the same!\n");

}

//Учитывая нули, длина знаменателя больше, чем длина числителя

void denominatorWithZerosIsLongerThanNumerator()

{

//arrange

mpz\_t numerator, denominator;

number newNumerator, newDenominator, expectNewNumerator, expectNewDenominator;

mpz\_init (numerator);

mpz\_set\_ui (numerator, 1);

mpz\_init (denominator);

mpz\_set\_ui (denominator, 1000000000);

expectNewNumerator.size=10;

mpz\_init (expectNewNumerator.num);

mpz\_set\_ui (expectNewNumerator.num, 1);

expectNewDenominator.size=1;

mpz\_init (expectNewDenominator.num);

mpz\_set\_ui (expectNewDenominator.num, 1);

mpz\_init (newNumerator.num);

mpz\_init (newDenominator.num);

//act

convFractionToNumbers(numerator, denominator, &newNumerator, &newDenominator);

//assert

bool testPassed=false;

if (!mpz\_cmp(newNumerator.num, expectNewNumerator.num)

&& !mpz\_cmp(newDenominator.num, expectNewDenominator.num)

&& newNumerator.size==expectNewNumerator.size

&& newDenominator.size==expectNewDenominator.size)

testPassed=true;

if(testPassed==false)

printf("Testing function convFractionToNumbers\n03. Test: denominatorWithZerosIsLongerThanNumerator fail! The result is not the same!\n");

}

//Длина знаменателя больше, чем длина числителя

void denominatorIsLongerThanNumerator()

{

//arrange

mpz\_t numerator, denominator;

number newNumerator, newDenominator, expectNewNumerator, expectNewDenominator;

mpz\_init (numerator);

mpz\_set\_ui (numerator, 100);

mpz\_init (denominator);

mpz\_set\_ui (denominator, 5894);

expectNewNumerator.size=2;

mpz\_init (expectNewNumerator.num);

mpz\_set\_ui (expectNewNumerator.num, 1);

expectNewDenominator.size=4;

mpz\_init (expectNewDenominator.num);

mpz\_set\_ui (expectNewDenominator.num, 5894);

mpz\_init (newNumerator.num);

mpz\_init (newDenominator.num);

//act

convFractionToNumbers(numerator, denominator, &newNumerator, &newDenominator);

//assert

bool testPassed=false;

if (!mpz\_cmp(newNumerator.num, expectNewNumerator.num)

&& !mpz\_cmp(newDenominator.num, expectNewDenominator.num)

&& newNumerator.size==expectNewNumerator.size

&& newDenominator.size==expectNewDenominator.size)

testPassed=true;

if(testPassed==false)

printf("Testing function convFractionToNumbers\n04. Test: denominatorIsLongerThanNumerator fail! The result is not the same!\n");

}

//Длина численной части знаменателя меньше, чем длина числителя

void numberPartOfDenominatorIsShorterThanNumerator()

{

//arrange

mpz\_t numerator, denominator;

number newNumerator, newDenominator, expectNewNumerator, expectNewDenominator;

mpz\_init (numerator);

mpz\_set\_ui (numerator, 91);

mpz\_init (denominator);

mpz\_set\_ui (denominator, 200);

expectNewNumerator.size=3;

mpz\_init (expectNewNumerator.num);

mpz\_set\_ui (expectNewNumerator.num, 91);

expectNewDenominator.size=1;

mpz\_init (expectNewDenominator.num);

mpz\_set\_ui (expectNewDenominator.num, 2);

mpz\_init (newNumerator.num);

mpz\_init (newDenominator.num);

//act

convFractionToNumbers(numerator, denominator, &newNumerator, &newDenominator);

//assert

bool testPassed=false;

if (!mpz\_cmp(newNumerator.num, expectNewNumerator.num)

&& !mpz\_cmp(newDenominator.num, expectNewDenominator.num)

&& newNumerator.size==expectNewNumerator.size

&& newDenominator.size==expectNewDenominator.size)

testPassed=true;

if(testPassed==false)

printf("Testing function convFractionToNumbers\n05. Test: numberPartOfDenominatorIsShorterThanNumerator fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы deleteZeros**

//Удаляется 1 нуль

void deleteOneZero()

{

number result, expectedResult;

result.size=4;

mpz\_init (result.num);

mpz\_set\_ui (result.num, 160);

deleteZeros(&result);

expectedResult.size=3;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 16);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function deleteZeros\n01. Test: deleteOneZero fail! The result is not the same!\n");

}

//Удаляются 6 нулей

void deleteSixZeros()

{

number result, expectedResult;

result.size=7;

mpz\_init (result.num);

mpz\_set\_ui (result.num, 1000000);

deleteZeros(&result);

expectedResult.size=1;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 1);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function deleteZeros\n02. Test: deleteSixZeros fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы divTwoNumbers**

//Делятся числа одинакового размера

void divWithEqualSizeOfNumbers()

{

number dividendOperand, dividerOperand, result, expectedResult;

mpz\_init (dividendOperand.num);

mpz\_init (dividerOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

dividendOperand.size=4;

mpz\_set\_ui (dividendOperand.num, 12);

dividerOperand.size=4;

mpz\_set\_ui (dividerOperand.num, 222);

result=divTwoNumbers( dividendOperand, dividerOperand);

expectedResult.size=5;

mpz\_set\_ui (expectedResult.num, 5405);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function divWithEqualSizeOfNumbers\n01. Test: divWithEqualSizeNumbers fail! The result is not the same!\n");

}

//Делятся числа разного размера

voiddivWithDifferentSizeNumbers()

{

number dividendOperand, dividerOperand, result, expectedResult;

mpz\_init (dividendOperand.num);

mpz\_init (dividerOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

dividendOperand.size=4;

mpz\_set\_ui (dividendOperand.num, 3);

dividerOperand.size=2;

mpz\_set\_ui (dividerOperand.num, 6);

result=divTwoNumbers( dividendOperand, dividerOperand);

expectedResult.size=3;

mpz\_set\_ui (expectedResult.num, 5);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function divTwoNumbers\n02. Test: divWithDifferentSizeNumbers fail! The result is not the same!\n");

}

//Делятся числа абсолютно разного размера

void divWithVeryDifferentSizeNumbers()

{

number dividendOperand, dividerOperand, result, expectedResult;

mpz\_init (dividendOperand.num);

mpz\_init (dividerOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

dividendOperand.size=10;

mpz\_set\_ui (dividendOperand.num, 1);

dividerOperand.size=1;

mpz\_set\_ui (dividerOperand.num, 1);

result=divTwoNumbers( dividendOperand, dividerOperand);

expectedResult.size=9;

mpz\_set\_ui (expectedResult.num, 1);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function divTwoNumbers\n03. Test: divWithVeryDifferentSizeNumbers fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы getErrorForLog**

//Сообщение об ошибке, что слишком мало аргументов

void errorTooLittleArgs()

{

//arrange

typeError tmp=tooLittleArgs;

char\* strError;

char\* test\_strError="Error: There aren’t enough files. ";

//act

strError=getErrorForLog(tmp);

//assert

if(strcmp(strError,test\_strError)!=0)

printf("Testing function getErrorForLog\n01. Test: errorTooLittleArgs fail! The result is not the same!\n");

}

//Сообщение об ошибке неудачного считывания

void errorUnluckyReading()

{

//arrange

typeError tmp=unluckyReading;

char\* strError;

char\* test\_strError="Error: Input file can’t be read. ";

//act

strError=getErrorForLog(tmp);

//assert

if(strcmp(strError,test\_strError)!=0)

printf("Testing function getErrorForLog\n01. Test: errorUnluckyReading fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы findArithmeticalMean**

//Два небольших числа

voidtwoLittleNumbers()

{

number leftBorder, rightBorder, result, expectedResult;

mpz\_init (leftBorder.num);

mpz\_init (rightBorder.num);

mpz\_init (expectedResult.num);

mpz\_init (result.num);

leftBorder.size=1;

mpz\_set\_ui (leftBorder.num, 1);

rightBorder.size=1;

mpz\_set\_ui (rightBorder.num, 2);

expectedResult.size=2;

mpz\_set\_ui (expectedResult.num, 15);

findArithmeticalMean(leftBorder, rightBorder, &result);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function findArithmeticalMean\n01. Test: twoLittleNumbers fail! The result is not the same!\n");

}

//Одним из чисел является нуль

voidoneOfNumbersIsZero()

{

number leftBorder, rightBorder, result, expectedResult;

mpz\_init (leftBorder.num);

mpz\_init (rightBorder.num);

mpz\_init (expectedResult.num);

mpz\_init (result.num);

leftBorder.size=0;

mpz\_set\_ui (leftBorder.num, 0);

rightBorder.size=1;

mpz\_set\_ui (rightBorder.num, 2);

expectedResult.size=1;

mpz\_set\_ui (expectedResult.num, 1);

findArithmeticalMean(leftBorder, rightBorder, &result);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function findArithmeticalMean\n02. Test: oneOfNumbersIsZero fail! The result is not the same!\n");

}

//Два обычных числа

voidusualNumbers()

{

number leftBorder, rightBorder, result, expectedResult;

mpz\_init (leftBorder.num);

mpz\_init (rightBorder.num);

mpz\_init (expectedResult.num);

leftBorder.size=8;

mpz\_set\_ui (leftBorder.num, 12548678);

rightBorder.size=7;

mpz\_set\_ui (rightBorder.num, 3564251);

expectedResult.size=8;

mpz\_set\_ui (expectedResult.num, 24095594);

findArithmeticalMean(leftBorder, rightBorder, &result);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function findArithmeticalMean\n03. Test: usualNumbers fail! The result is not the same!\n");

}

//Два больших числа

void twoBigNumbers()

{

number leftBorder, rightBorder, result, expectedResult;

mpz\_init (leftBorder.num);

mpz\_init (rightBorder.num);

mpz\_init (expectedResult.num);

leftBorder.size=31;

mpz\_set\_ui (leftBorder.num, 1);

rightBorder.size=31;

mpz\_set\_ui (rightBorder.num, 2);

expectedResult.size=32;

mpz\_set\_ui (expectedResult.num, 15);

findArithmeticalMean(leftBorder, rightBorder, &result);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function findArithmeticalMean\n04. Test: twoBigNumbers fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы findProbabilityOfTheAlphabetFromTheText**

//Корректно разобраны вероятности(Hi!!!!!)

void probabilityIsCorrectlyParsed()

{

unsigned int alphabetSize=4;

elementOfAlphabet \*alphabet=initAlphabet(4);

alphabet[0].symbol='H';

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='i';

mpz\_set\_ui (alphabet[1].value.num, 125);

alphabet[1].value.size=3;

alphabet[2].symbol='!';

mpz\_set\_ui (alphabet[2].value.num, 25);

alphabet[2].value.size=2;

//eof

mpz\_set\_ui (alphabet[3].probability.num, 125);

alphabet[3].probability.size=3;

mpz\_set\_ui (alphabet[3].value.num, 875);

alphabet[3].value.size=3;

elementOfAlphabet testAlphabet[4]={NULL};

testAlphabet[0].symbol='H';

mpz\_init(testAlphabet[0].probability.num);

mpz\_set\_ui (testAlphabet[0].probability.num, 125);

testAlphabet[0].probability.size=3;

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='i';

mpz\_init(testAlphabet[1].probability.num);

mpz\_set\_ui (testAlphabet[1].probability.num, 125);

testAlphabet[1].probability.size=3;

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 125);

testAlphabet[1].value.size=3;

testAlphabet[2].symbol='!';

mpz\_init(testAlphabet[2].probability.num);

mpz\_set\_ui (testAlphabet[2].probability.num, 625);

testAlphabet[2].probability.size=3;

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 25);

testAlphabet[2].value.size=2;

//eof

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 125);

testAlphabet[3].probability.size=3;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 875);

testAlphabet[3].value.size=3;

//act

findProbabilityOfTheAlphabetFromTheText(alphabetSize, alphabet);

//assert

if(!isEqualProbability(alphabet, testAlphabet, alphabetSize))

printf("Testing function findProbabilityOfTheAlphabetFromTheText\n01. Test: probabilityIsCorrectlyParsed fail! The result is not the same!\n");

delete[]alphabet;

}

**Модульные тесты для подпрограммы findSymbolAccordingToInterval**

//Один символ в алфавите

void oneSymbolInAlphabet()

{

number currentNumber;

mpz\_init(currentNumber.num);

mpz\_set\_ui (currentNumber.num, 43);

currentNumber.size=2;

unsigned int alphabetSize=2;

elementOfAlphabet alphabet[2];

alphabet[0].symbol='a';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 5);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

//EndOfFile

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 5);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 5);

alphabet[1].value.size=1;

elementOfAlphabet resultSymbol;

mpz\_init(resultSymbol.probability.num);

mpz\_init(resultSymbol.value.num);

elementOfAlphabet expectedSymbol;

expectedSymbol.symbol='a';

mpz\_init(expectedSymbol.probability.num);

mpz\_set\_ui (expectedSymbol.probability.num, 5);

expectedSymbol.probability.size=1;

mpz\_init(expectedSymbol.value.num);

mpz\_set\_ui (expectedSymbol.value.num, 0);

expectedSymbol.value.size=0;

resultSymbol=findSymbolAccordingToInterval(currentNumber, alphabet, alphabetSize);

bool isEqual=false;

if( expectedSymbol.symbol==resultSymbol.symbol

&& compareTwoNumbers(expectedSymbol.probability, resultSymbol.probability)==equivalent

&&compareTwoNumbers(expectedSymbol.value, resultSymbol.value)==equivalent)

{

isEqual=true;

}

if (!isEqual)

printf("Testing function findSymbolAccordingToInterval\n01. Fail! The result is not the same!");

}

//Обычный символ в алфавите

void usualSymbolInAlphabet()

{

number currentNumber;

mpz\_init(currentNumber.num);

mpz\_set\_ui (currentNumber.num, 72);

currentNumber.size=2;

unsigned int alphabetSize=5;

elementOfAlphabet alphabet[5];

alphabet[0].symbol='l';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 3);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='r';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 6);

alphabet[2].value.size=1;

alphabet[3].symbol='t';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 8);

alphabet[3].value.size=1;

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 9);

alphabet[4].value.size=1;

elementOfAlphabet resultSymbol;

mpz\_init(resultSymbol.probability.num);

mpz\_init(resultSymbol.value.num);

elementOfAlphabet expectedSymbol;

expectedSymbol.symbol='r';

mpz\_init(expectedSymbol.probability.num);

mpz\_set\_ui (expectedSymbol.probability.num, 2);

expectedSymbol.probability.size=1;

mpz\_init(expectedSymbol.value.num);

mpz\_set\_ui (expectedSymbol.value.num, 6);

expectedSymbol.value.size=1;

resultSymbol=findSymbolAccordingToInterval(currentNumber, alphabet, alphabetSize);

bool isEqual=false;

if( expectedSymbol.symbol==resultSymbol.symbol

&& compareTwoNumbers(expectedSymbol.probability, resultSymbol.probability)==equivalent

&&compareTwoNumbers(expectedSymbol.value, resultSymbol.value)==equivalent)

{

isEqual=true;

}

if (!isEqual)

printf("Testing function findSymbolAccordingToInterval\n02. Fail! The result is not the same!");

}

//Первый символ в алфавите

voidfirstSymbolInAlphabet()

{

number currentNumber;

mpz\_init(currentNumber.num);

mpz\_set\_ui (currentNumber.num, 1);

currentNumber.size=3;

unsigned int alphabetSize=5;

elementOfAlphabet alphabet[5];

alphabet[0].symbol='l';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 3);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='r';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 6);

alphabet[2].value.size=1;

alphabet[3].symbol='t';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 8);

alphabet[3].value.size=1;

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 9);

alphabet[4].value.size=1;

elementOfAlphabet resultSymbol;

mpz\_init(resultSymbol.probability.num);

mpz\_init(resultSymbol.value.num);

elementOfAlphabet expectedSymbol;

expectedSymbol.symbol='l';

mpz\_init(expectedSymbol.probability.num);

mpz\_set\_ui (expectedSymbol.probability.num, 3);

expectedSymbol.probability.size=1;

mpz\_init(expectedSymbol.value.num);

mpz\_set\_ui (expectedSymbol.value.num, 0);

expectedSymbol.value.size=0;

resultSymbol=findSymbolAccordingToInterval(currentNumber, alphabet, alphabetSize);

bool isEqual=false;

if( expectedSymbol.symbol==resultSymbol.symbol

&& compareTwoNumbers(expectedSymbol.probability, resultSymbol.probability)==equivalent

&&compareTwoNumbers(expectedSymbol.value, resultSymbol.value)==equivalent)

{

isEqual=true;

}

if (!isEqual)

printf("Testing function findSymbolAccordingToInterval\n03. Fail! The result is not the same!");

}

//Последний символ в алфавите

voidlastSymbolInAlphabet()

{

number currentNumber;

mpz\_init(currentNumber.num);

mpz\_set\_ui (currentNumber.num, 89);

currentNumber.size=2;

unsigned int alphabetSize=5;

elementOfAlphabet alphabet[5];

alphabet[0].symbol='l';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 3);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='r';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 6);

alphabet[2].value.size=1;

alphabet[3].symbol='t';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 8);

alphabet[3].value.size=1;

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 9);

alphabet[4].value.size=1;

elementOfAlphabet resultSymbol;

mpz\_init(resultSymbol.probability.num);

mpz\_init(resultSymbol.value.num);

elementOfAlphabet expectedSymbol;

expectedSymbol.symbol='t';

mpz\_init(expectedSymbol.probability.num);

mpz\_set\_ui (expectedSymbol.probability.num, 1);

expectedSymbol.probability.size=1;

mpz\_init(expectedSymbol.value.num);

mpz\_set\_ui (expectedSymbol.value.num, 8);

expectedSymbol.value.size=1;

resultSymbol=findSymbolAccordingToInterval(currentNumber, alphabet, alphabetSize);

bool isEqual=false;

if( expectedSymbol.symbol==resultSymbol.symbol

&& compareTwoNumbers(expectedSymbol.probability, resultSymbol.probability)==equivalent

&&compareTwoNumbers(expectedSymbol.value, resultSymbol.value)==equivalent)

{

isEqual=true;

}

if (!isEqual)

printf("Testing function findSymbolAccordingToInterval\n04. Fail! The result is not the same!");

}

**Модульные тесты для подпрограммы inspectAndParseArgumentsOfCmd**

//Корректные аргументы

void correctArguments()

{

//arrange

int argc = 4;

char argv[4][1000] = {"NumeralPack.exe","-p", "C:\\Users\\User\\Desktop\\fileForPack.txt","C:\\Users\\User\\Desktop\\resultFile.txt"};

log logInfo=initLog();

typeError resultReal = allRight;

//act

typeError error=inspectAndParseArgumentsOfCmd(argc, argv, &logInfo);

//assert

if(error!=resultReal)

printf("Testing function inspectAndParseArgumentsOfCmd\n01. Test correctArguments fail! The result is not the same!");

}

//Много аргументов

void manyArguments()

{

//arrange

int argc = 5;

char argv[4][1000] = {"NumeralPack.exe","-p", "C:\\Users\\User\\Desktop\\fileForPack.txt","C:\\Users\\User\\Desktop\\resultFile.txt"};

log logInfo=initLog();

typeError resultReal = tooManyArgs;

//act

typeError error=inspectAndParseArgumentsOfCmd(argc,argv,&logInfo);

//assert

if(error!=resultReal)

printf("Testing function inspectAndParseArgumentsOfCmd\n03. Test manyArguments fail! The result is not the same!");

}

//Второй аргумент некорректный

void secondArgumentIsIncorrect()

{

//arrange

int argc = 4;

char argv[4][1000] = {"ArchiveProgram.exe", "-g","C:\\Users\\User\\Desktop\\fileForPack.txt","C:\\Users\\User\\Desktop\\resultFile.txt"};

log logInfo=initLog();

typeError resultReal = incorrectMode;

//act

typeError error=inspectAndParseArgumentsOfCmd(argc,argv,&logInfo);

//assert

if(error!=resultReal)

printf("Testing function inspectAndParseArgumentsOfCmd\n04. Test secondArgumentIsIncorrect fail! The result is not the same!");

}

**Модульные тесты для подпрограммы mulTwoNumbers**

//Число нулей в множителях одинаково

voidsameCountOfZeros()

{

number leftOperand, rightOperand, result, expectedResult;

mpz\_init (leftOperand.num);

mpz\_init (rightOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

leftOperand.size=4;

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=4;

mpz\_set\_ui (rightOperand.num, 14);

result=mulTwoNumbers(leftOperand, rightOperand);

expectedResult.size=8;

mpz\_set\_ui (expectedResult.num, 168);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function mulTwoNumbers\n01. Test: sameCountOfZeros fail! The result is not the same!\n");

}

//При умножении число нулей сокращается

voidmulWithDeletingZero()

{

number leftOperand, rightOperand, result, expectedResult;

mpz\_init (leftOperand.num);

mpz\_init (rightOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

leftOperand.size=2;

mpz\_set\_ui (leftOperand.num, 5);

rightOperand.size=2;

mpz\_set\_ui (rightOperand.num, 2);

result=mulTwoNumbers(leftOperand, rightOperand);

expectedResult.size=3;

mpz\_set\_ui (expectedResult.num, 1);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function mulTwoNumbers\n02. Test: mulWithDeletingZero fail! The result is not the same!\n");

}

//Размеры множителей различны и число нулей сокращается

void differentSizesOfOperandsWithDeletingZero()

{

number leftOperand, rightOperand, result, expectedResult;

mpz\_init (leftOperand.num);

mpz\_init (rightOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

leftOperand.size=2;

mpz\_set\_ui (leftOperand.num, 2);

rightOperand.size=4;

mpz\_set\_ui (rightOperand.num, 65);

result=mulTwoNumbers(leftOperand, rightOperand);

expectedResult.size=5;

mpz\_set\_ui (expectedResult.num, 13);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function mulTwoNumbers\n03. Test: differentSizesOfOperandsWithDeletingZero fail! The result is not the same!\n");

}

//Размеры множителей различны

voiddifferentSizesOfOperands()

{

number leftOperand, rightOperand, result, expectedResult;

mpz\_init (leftOperand.num);

mpz\_init (rightOperand.num);

mpz\_init (result.num);

mpz\_init (expectedResult.num);

leftOperand.size=1;

mpz\_set\_ui (leftOperand.num, 6);

rightOperand.size=2;

mpz\_set\_ui (rightOperand.num, 66);

result=mulTwoNumbers(leftOperand, rightOperand);

expectedResult.size=3;

mpz\_set\_ui (expectedResult.num, 396);

bool testPassed=false;

if (compareTwoNumbers(result, expectedResult)==equivalent)

testPassed=true;

if(testPassed==false)

printf("Testing function mulTwoNumbers\n04. Test: differentSizesOfOperands fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы packText**

//Текст состоит из нескольких повторяющихся символов

void textConsistsOfDuplicateSymbols()

{

//arrange

char text[5]={"ssss"};

elementOfAlphabet alphabet[2]={NULL};

unsigned int sizeOfAlphabet=2;

log logInfo=initLog();

alphabet[0].symbol='s';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 8);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

//EndOfFile

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 2);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 8);

alphabet[1].value.size=1;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_ui(expectValue.num, 36864);

expectValue.size=5;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n01 Test: textConsistsOfDuplicateSymbols fail! The result is not the same!");

}

//Текст включает в себя повторяющиеся символы

void usualTextWithDuplicateSymbols()

{

//arrange

char text[10]={"trollollo"};

elementOfAlphabet alphabet[5]={NULL};

unsigned int sizeOfAlphabet=5;

log logInfo=initLog();

alphabet[0].symbol='t';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='r';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 1);

alphabet[1].value.size=1;

alphabet[2].symbol='o';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 3);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 2);

alphabet[2].value.size=1;

alphabet[3].symbol='l';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 4);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 5);

alphabet[3].value.size=1;

//EndOfFile

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 9);

alphabet[4].value.size=1;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_ui(expectValue.num, 143079744);

expectValue.size=10;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n02 Test: usualTextWithDuplicateSymbols fail! The result is not the same!");

}

//Типичныйтекст

void usualText()

{

//arrange

char text[10]={"адаптация"};

unsigned int sizeOfAlphabet=8;

elementOfAlphabet alphabet[8];

log logInfo=initLog();

alphabet[0].symbol='а';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='д';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='п';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 1);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 4);

alphabet[2].value.size=1;

alphabet[3].symbol='т';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 5);

alphabet[3].value.size=1;

alphabet[4].symbol='ц';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 6);

alphabet[4].value.size=1;

alphabet[5].symbol='и';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 1);

alphabet[5].probability.size=1;

mpz\_init(alphabet[5].value.num);

mpz\_set\_ui (alphabet[5].value.num, 7);

alphabet[5].value.size=1;

alphabet[6].symbol='я';

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=1;

mpz\_init(alphabet[6].value.num);

mpz\_set\_ui (alphabet[6].value.num, 8);

alphabet[6].value.size=1;

//EndOfFile

mpz\_init(alphabet[7].probability.num);

mpz\_set\_ui (alphabet[7].probability.num, 1);

alphabet[7].probability.size=1;

mpz\_init(alphabet[7].value.num);

mpz\_set\_ui (alphabet[7].value.num, 9);

alphabet[7].value.size=1;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_str(expectValue.num, "9406833165", 10);

expectValue.size=11;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n03 Test: usualText fail! The result is not the same!");

}

//Текст состоит из одной буквы

void oneLetter()

{

//arrange

char text[2]={"f"};

unsigned int sizeOfAlphabet=2;

elementOfAlphabet alphabet[2]={NULL};

log logInfo=initLog();

alphabet[0].symbol='f';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 5);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

//EndOfFile

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 5);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 5);

alphabet[1].value.size=1;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_ui(expectValue.num, 375);

expectValue.size=3;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n04 Test: oneLetter fail! The result is not the same!");

}

//Короткийтекст

void simpleText()

{

//arrange

char text[4]={"yes"};

unsigned int sizeOfAlphabet=4;

elementOfAlphabet alphabet[4]={NULL};

log logInfo=initLog();

alphabet[0].symbol='y';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 25);

alphabet[0].probability.size=2;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='e';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 25);

alphabet[1].value.size=2;

alphabet[2].symbol='s';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 5);

alphabet[2].value.size=1;

//EndOfFile

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 25);

alphabet[3].probability.size=2;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 75);

alphabet[3].value.size=2;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_ui(expectValue.num, 107421875);

expectValue.size=9;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n05 Test: simpleText fail! The result is not the same!");

}

//Текст включает в себя пробелы

void textWithSpaces()

{

//arrange

char text[10]={"yes it is"};

unsigned int sizeOfAlphabet=7;

elementOfAlphabet alphabet[7]={NULL};

log logInfo=initLog();

alphabet[0].symbol='y';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='e';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 1);

alphabet[1].value.size=1;

alphabet[2].symbol='s';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 2);

alphabet[2].value.size=1;

alphabet[3].symbol=' ';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 2);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 4);

alphabet[3].value.size=1;

alphabet[4].symbol='i';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 6);

alphabet[4].value.size=1;

alphabet[5].symbol='t';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 2);

alphabet[5].probability.size=1;

mpz\_init(alphabet[5].value.num);

mpz\_set\_ui (alphabet[5].value.num, 7);

alphabet[5].value.size=1;

//EndOfFile

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=1;

mpz\_init(alphabet[6].value.num);

mpz\_set\_ui (alphabet[6].value.num, 9);

alphabet[6].value.size=1;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_ui(expectValue.num, 130722224);

expectValue.size=10;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n06 Test: textWithSpaces fail! The result is not the same!");

}

//Многострочныйтекст

void multiplineText()

{

//arrange

char text[8]={"lol\nlo\n"};

unsigned int sizeOfAlphabet=4;

elementOfAlphabet alphabet[4]={NULL};

log logInfo=initLog();

alphabet[0].symbol='l';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 375);

alphabet[0].probability.size=3;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 375);

alphabet[1].value.size=3;

alphabet[2].symbol='\n';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 625);

alphabet[2].value.size=3;

//EndOfFile

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 125);

alphabet[3].probability.size=3;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 875);

alphabet[3].value.size=3;

number packedText;

mpz\_init(packedText.num);

packedText.size=0;

number expectValue;

mpz\_init(expectValue.num);

mpz\_set\_str(expectValue.num, "164541721343994140625", 10);

expectValue.size=21;

//act

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, true, logInfo), &packedText);

//assert

bool isEqual=isEqualBorders(packedText, expectValue);

if (!isEqual)

printf("Testing function packText\n07 Test: multiplineText fail! The result is not the same!");

}

**Модульные тесты для подпрограммы parseTheStringWithEofFromText**

//eof строка корректно разобрана(cut)

void eofStringCorrectlyParsed()

{

//arrange

char eofStr[]="75e2o25f2";

unsigned int alphabetSize=4;

elementOfAlphabet \*alphabet=initAlphabet(4);

alphabet[0].symbol='c';

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='u';

mpz\_set\_ui (alphabet[1].value.num, 25);

alphabet[1].value.size=2;

alphabet[2].symbol='t';

mpz\_set\_ui (alphabet[2].value.num, 5);

alphabet[2].value.size=1;

elementOfAlphabet testAlphabet[4]={NULL};

testAlphabet[0].symbol='c';

mpz\_init(testAlphabet[0].value.num);

mpz\_set\_ui (testAlphabet[0].value.num, 0);

testAlphabet[0].value.size=0;

testAlphabet[1].symbol='u';

mpz\_init(testAlphabet[1].value.num);

mpz\_set\_ui (testAlphabet[1].value.num, 25);

testAlphabet[1].value.size=2;

testAlphabet[2].symbol='t';

mpz\_init(testAlphabet[2].value.num);

mpz\_set\_ui (testAlphabet[2].value.num, 5);

testAlphabet[2].value.size=1;

//eof

mpz\_init(testAlphabet[3].probability.num);

mpz\_set\_ui (testAlphabet[3].probability.num, 25);

testAlphabet[3].probability.size=2;

mpz\_init(testAlphabet[3].value.num);

mpz\_set\_ui (testAlphabet[3].value.num, 75);

testAlphabet[3].value.size=2;

//act

parseTheStringWithEofFromText(eofStr, alphabetSize, alphabet);

//assert

if(!isEqualEofInAlphabet(alphabet, testAlphabet, alphabetSize))

printf("Testing function parseEofFromText\n01. Test: eofStringCorrectlyParsed fail! The result is not the same!\n");

delete[]alphabet;

}

**Модульные тесты для подпрограммы getStageOfExecutionForLog**

//Сообщение о том, что программа начала выполняться

void stageStartProgram()

{

//arrange

log tmp;

tmp.stageInfo=startProgram;

char\* strError;

char\* test\_strError="Program started its execution. ";

//act

strError=getStageOfExecutionForLog(tmp);

//assert

if(strcmp(strError,test\_strError)!=0)

printf("Testing function getStageOfExecutionForLog\n01. Test: stageStartProgram fail! The result is not the same!\n");

}

//Сообщение о том, что алфавит разобран

void packingWasStart()

{

//arrange

log tmp;

tmp.stageInfo=startPack;

tmp.inputFileName="C:\Users\User\Desktop\fileForPack.txt";

char\* strError;

char\* test\_strError="Packing was started: file C:\Users\User\Desktop\fileForPack.txt";

//act

strError=getStageOfExecutionForLog(tmp);

//assert

if(strcmp(strError,test\_strError)!=0)

printf("Testing function getStageOfExecutionForLog\n02. Test: packingWasStart fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы subTwoNumbers**

//Результат вычитания равен нулю

void subWithZeroResult()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 12);

mpz\_init (result.num);

result=subTwoNumbers(leftOperand, rightOperand);

expectedResult.size=0;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 0);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function subTwoNumbers\n01. Test: subWithZeroResult fail! The result is not the same!\n");

}

//Операнды одинакового размера

voidtwoSameSizeNumbers()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 12);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 2);

mpz\_init (result.num);

result=subTwoNumbers(leftOperand, rightOperand);

expectedResult.size=3;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 1);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function subTwoNumbers\n02. Test: twoSameSizeNumbers fail! The result is not the same!\n");

}

//Левый операнд имеет больший размер, чем правый

void leftHasBiggerSize()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=4;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 66);

rightOperand.size=3;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 3);

mpz\_init (result.num);

result=subTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 36);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function subTwoNumbers\n03. Test: leftHasBiggerSize fail! The result is not the same!\n");

}

//Правый операнд имеет больший размер, чем левый

void rightHasBiggerSize()

{

number leftOperand, rightOperand, result, expectedResult;

leftOperand.size=2;

mpz\_init (leftOperand.num);

mpz\_set\_ui (leftOperand.num, 2);

rightOperand.size=4;

mpz\_init (rightOperand.num);

mpz\_set\_ui (rightOperand.num, 65);

mpz\_init (result.num);

result=subTwoNumbers(leftOperand, rightOperand);

expectedResult.size=4;

mpz\_init (expectedResult.num);

mpz\_set\_ui (expectedResult.num, 135);

bool testPassed=false;

if (!mpz\_cmp(result.num, expectedResult.num) && result.size==expectedResult.size)

testPassed=true;

if(testPassed==false)

printf("Testing function subTwoNumbers\n04. Test: rightHasBiggerSize fail! The result is not the same!\n");

}

**Модульные тесты для подпрограммы unpackText**

//Запакован один символ

void oneCharacterIsPacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_ui (middleValue.num, 375);

middleValue.size=3;

log logInfo=initLog();

unsigned int alphabetSize=2;

elementOfAlphabet alphabet[2];

alphabet[0].symbol='a';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 5);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

//EndOfFile

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 5);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 5);

alphabet[1].value.size=1;

char expectedResult[20]="a";

char resultText[20]="";

unpackText(middleValue, alphabet, alphabetSize, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n01. Test oneCharacterIsPacked fail! The result is not the same!");

}

//Запакованы повторяющиеся символы

void duplicateSymbolsArePacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_ui (middleValue.num, 46875);

middleValue.size=6;

log logInfo=initLog();

unsigned int alphabetSize=2;

elementOfAlphabet alphabet[2];

alphabet[0].symbol='s';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 5);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

//EndOfFile

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 5);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 5);

alphabet[1].value.size=1;

char expectedResult[20]="ssss";

char resultText[20]="";

unpackText(middleValue, alphabet, alphabetSize, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n02. Test duplicateSymbolsArePacked fail! The result is not the same!");

}

//Запакован простой текст

void simpleTextIsPacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_ui (middleValue.num, 52929687);

middleValue.size=8;

log logInfo=initLog();

unsigned int alphabetSize=4;

elementOfAlphabet alphabet[4];

alphabet[0].symbol='e';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 25);

alphabet[0].probability.size=2;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='s';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 25);

alphabet[1].value.size=2;

alphabet[2].symbol='y';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 5);

alphabet[2].value.size=1;

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 25);

alphabet[3].probability.size=2;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 75);

alphabet[3].value.size=2;

char expectedResult[20]="yes";

char resultText[20]="";

unpackText(middleValue, alphabet, alphabetSize, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n03. Test simpleTextIsPacked fail! The result is not the same!");

}

//Запакован текст из английских символов

void englishCharactersArePacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_ui (middleValue.num, 143079744);

middleValue.size=10;

elementOfAlphabet alphabet[5]={NULL};

unsigned int sizeOfAlphabet=5;

log logInfo=initLog();

alphabet[0].symbol='t';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='r';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 1);

alphabet[1].value.size=1;

alphabet[2].symbol='o';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 3);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 2);

alphabet[2].value.size=1;

alphabet[3].symbol='l';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 4);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 5);

alphabet[3].value.size=1;

//EndOfFile

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 9);

alphabet[4].value.size=1;

char expectedResult[20]="trollollo";

char resultText[20]="";

unpackText(middleValue, alphabet, sizeOfAlphabet, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n. Test englishCharactersArePacked fail! The result is not the same!");

}

//Запакован текст из русских символов

void russianCaractersArePacked()

{

number middleValue;

mpz\_init(middleValue.num);

//816898573

mpz\_set\_str (middleValue.num, "9406833165", 10);

middleValue.size=11;

unsigned int sizeOfAlphabet=8;

elementOfAlphabet alphabet[8];

log logInfo=initLog();

alphabet[0].symbol='а';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 3);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='д';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 3);

alphabet[1].value.size=1;

alphabet[2].symbol='п';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 1);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 4);

alphabet[2].value.size=1;

alphabet[3].symbol='т';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 1);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 5);

alphabet[3].value.size=1;

alphabet[4].symbol='ц';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 6);

alphabet[4].value.size=1;

alphabet[5].symbol='и';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 1);

alphabet[5].probability.size=1;

mpz\_init(alphabet[5].value.num);

mpz\_set\_ui (alphabet[5].value.num, 7);

alphabet[5].value.size=1;

alphabet[6].symbol='я';

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=1;

mpz\_init(alphabet[6].value.num);

mpz\_set\_ui (alphabet[6].value.num, 8);

alphabet[6].value.size=1;

//EndOfFile

mpz\_init(alphabet[7].probability.num);

mpz\_set\_ui (alphabet[7].probability.num, 1);

alphabet[7].probability.size=1;

mpz\_init(alphabet[7].value.num);

mpz\_set\_ui (alphabet[7].value.num, 9);

alphabet[7].value.size=1;

char expectedResult[20]="адаптация";

char resultText[20]="";

unpackText(middleValue, alphabet, sizeOfAlphabet, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n. Test russianCaractersArePacked fail! The result is not the same!");

}

//Запакован текст с пробелами

void textWithSpacesIsPacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_ui (middleValue.num, 130722224);

middleValue.size=10;

unsigned int sizeOfAlphabet=7;

elementOfAlphabet alphabet[7]={NULL};

log logInfo=initLog();

alphabet[0].symbol='y';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 1);

alphabet[0].probability.size=1;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='e';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 1);

alphabet[1].probability.size=1;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 1);

alphabet[1].value.size=1;

alphabet[2].symbol='s';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 2);

alphabet[2].probability.size=1;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 2);

alphabet[2].value.size=1;

alphabet[3].symbol=' ';

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 2);

alphabet[3].probability.size=1;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 4);

alphabet[3].value.size=1;

alphabet[4].symbol='i';

mpz\_init(alphabet[4].probability.num);

mpz\_set\_ui (alphabet[4].probability.num, 1);

alphabet[4].probability.size=1;

mpz\_init(alphabet[4].value.num);

mpz\_set\_ui (alphabet[4].value.num, 6);

alphabet[4].value.size=1;

alphabet[5].symbol='t';

mpz\_init(alphabet[5].probability.num);

mpz\_set\_ui (alphabet[5].probability.num, 2);

alphabet[5].probability.size=1;

mpz\_init(alphabet[5].value.num);

mpz\_set\_ui (alphabet[5].value.num, 7);

alphabet[5].value.size=1;

//EndOfFile

mpz\_init(alphabet[6].probability.num);

mpz\_set\_ui (alphabet[6].probability.num, 1);

alphabet[6].probability.size=1;

mpz\_init(alphabet[6].value.num);

mpz\_set\_ui (alphabet[6].value.num, 9);

alphabet[6].value.size=1;

char expectedResult[20]="yes it is";

char resultText[20]="";

unpackText(middleValue, alphabet, sizeOfAlphabet, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n. Test textWithSpaces fail! The result is not the same!");

}

//Запакован текст из 2 строк

void textIncludeTwoStringsIsPacked()

{

number middleValue;

mpz\_init(middleValue.num);

mpz\_set\_str (middleValue.num, "164541721343994140625", 10);

middleValue.size=21;

unsigned int sizeOfAlphabet=4;

elementOfAlphabet alphabet[4]={NULL};

log logInfo=initLog();

alphabet[0].symbol='l';

mpz\_init(alphabet[0].probability.num);

mpz\_set\_ui (alphabet[0].probability.num, 375);

alphabet[0].probability.size=3;

mpz\_init(alphabet[0].value.num);

mpz\_set\_ui (alphabet[0].value.num, 0);

alphabet[0].value.size=0;

alphabet[1].symbol='o';

mpz\_init(alphabet[1].probability.num);

mpz\_set\_ui (alphabet[1].probability.num, 25);

alphabet[1].probability.size=2;

mpz\_init(alphabet[1].value.num);

mpz\_set\_ui (alphabet[1].value.num, 375);

alphabet[1].value.size=3;

alphabet[2].symbol='\n';

mpz\_init(alphabet[2].probability.num);

mpz\_set\_ui (alphabet[2].probability.num, 25);

alphabet[2].probability.size=2;

mpz\_init(alphabet[2].value.num);

mpz\_set\_ui (alphabet[2].value.num, 625);

alphabet[2].value.size=3;

//EndOfFile

mpz\_init(alphabet[3].probability.num);

mpz\_set\_ui (alphabet[3].probability.num, 125);

alphabet[3].probability.size=3;

mpz\_init(alphabet[3].value.num);

mpz\_set\_ui (alphabet[3].value.num, 875);

alphabet[3].value.size=3;

char expectedResult[20]="lol\nlo\n";

char resultText[20]="";

unpackText(middleValue, alphabet, sizeOfAlphabet, logInfo, true, resultText);

bool result=!strcmp(resultText,expectedResult);

if (!result)

printf("Testing function unpackText\n. Test textIncludeTwoStrings fail! The result is not the same!");

}

# Приложение Д. Разделение обязанностей между членами команды

Таблица Д.1. Реализация подпрограмм членами команд

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Подпрограмма | Заголовок | Модульные тесты | Псевдокод | Код | Тестирование и отладка |
| main |  | \*Функциональные тесты – Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| buildAlphabetModel | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| convertAlphabetModel | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| buildModelFromText | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| parseTheStringWithEofFromText | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| findProbabilityOfTheAlphabetFromTheText | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| definitionSpecialSymbol | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| initAlphabet | Сивашова Е. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| addZeros | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| deleteZeros | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| preparingForAddOrSub | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| addTwoNumbers | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| subTwoNumbers | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| mulTwoNumbers | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| compareTwoNumbers | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| assignAValueToNumber | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| divTwoNumbers | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| convFractionToNumbers | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| constrictInterval | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| findArithmeticalMean | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| findSymbolAccordingToInterval | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. |
| inspectAndParseArgumentsOfCmd | Сивашова Е. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| initLog | Сивашова Е. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| writeToLog | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| getErrorForLog | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| getStageOfExecutionForLog | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| timeForLog | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| packFile | Сивашова Е. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| packText | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| readDataForAlphabet | Стрекалова А. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| readDataForPack | Стрекалова А. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |
| writeDataForPack | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| writeEOFDataForPack | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| writeSpecialSymbolForPack | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| readDataForUnpack | Стрекалова А. |  | Сивашова Е.-15%  Волкова Е.-85% | Волкова Е.-85%  Сивашова Е.-15% | Волкова Е. |
| writeDataForUnpack | Стрекалова А. |  | Стрекалова А. | Стрекалова А. | Стрекалова А. |
| unpackText | Волкова Е. | Волкова Е. | Волкова Е. | Волкова Е. – 90%  Сивашова Е. – 10% | Волкова Е. |
| unpackFile | Сивашова Е. |  | Сивашова Е. | Сивашова Е. | Сивашова Е. |

Дополнительные обязанности Волковой Е.:

* разработала принцип хранения вещественного числа в виде пары целых чисел и реализовала «длинную» арифметику.

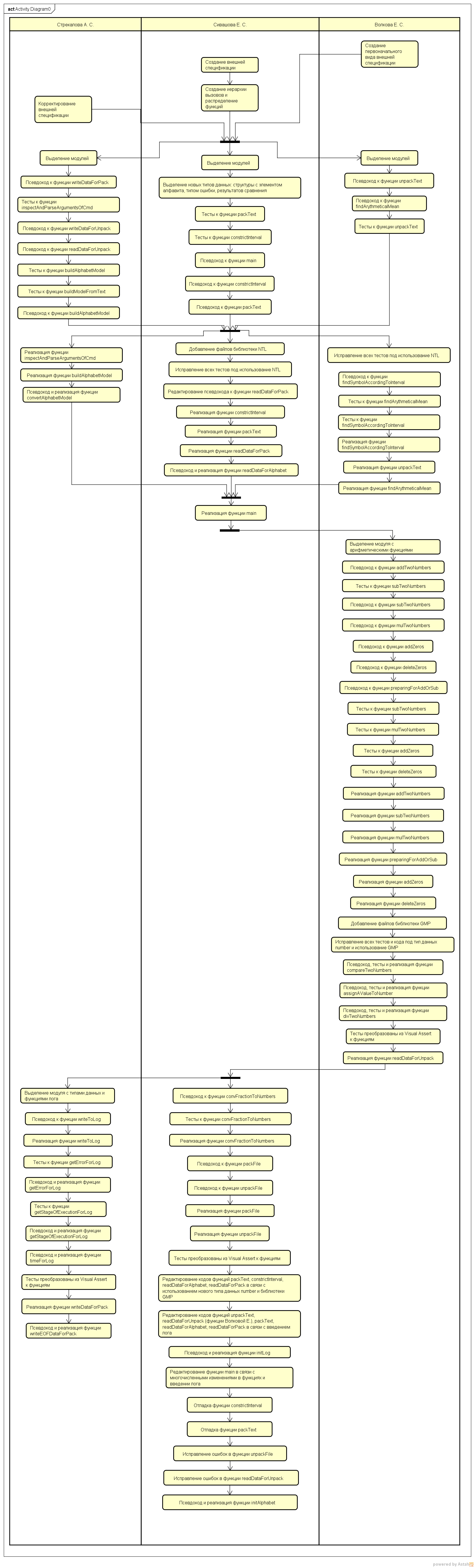
Дополнительные обязанности Стрекаловой А.:

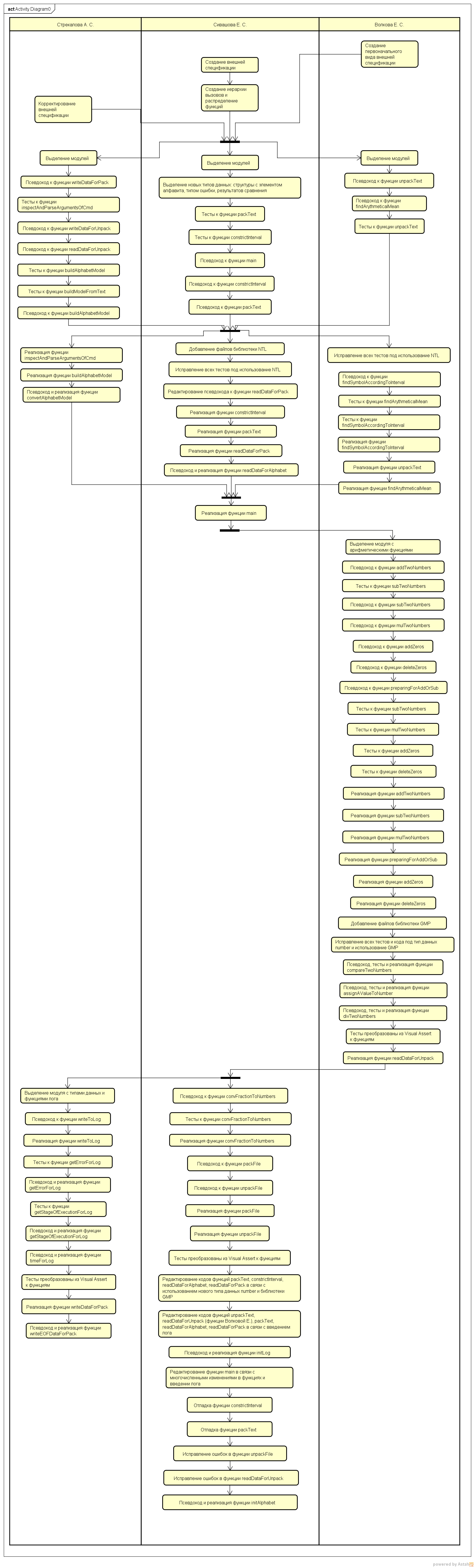
* определяла способ взаимодействия проблемно-ориентированных функций с логом.

Дополнительные обязанности Сивашовой Е.:

* внедряла использование функций с логом в остальные подпрограммы;
* отладка всей программы в целом.

# Приложение Е. Порядок конструирования программы





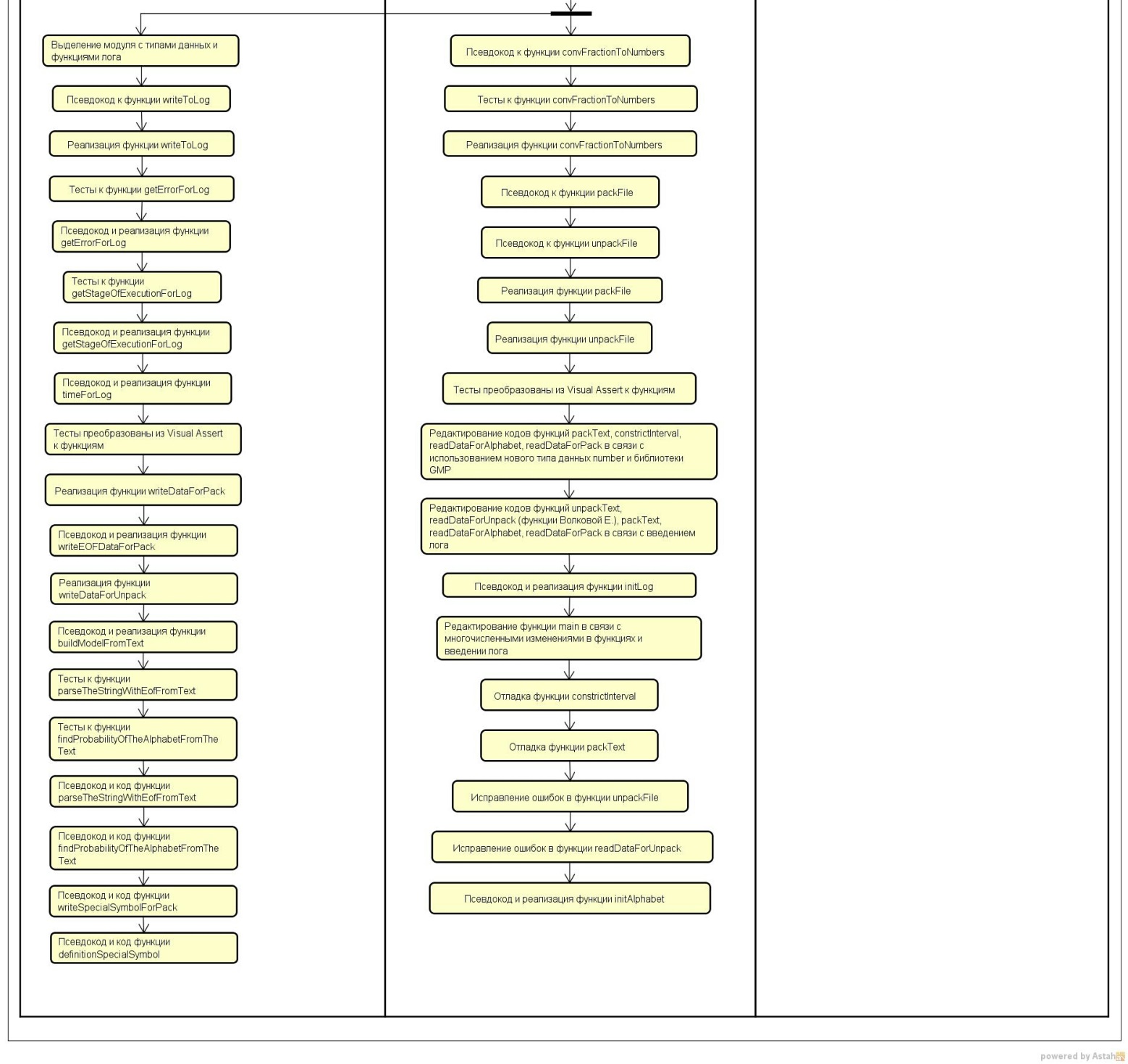


Рисунок Е.1 – Порядок реализации подпрограмм членами команд

# Приложение Ж. Кодпрограммы

void buildAlphabetModel(char symbol, unsigned int \*alphabetSize, elementOfAlphabet \*alphabet)

{

bool inAlphabet=false;//признак того, что текущий символ находится в алфавите

mpz\_t one;

mpz\_init\_set\_ui (one, 1);//инициализировать и присвоить значение переменной

//Для каждого символа в алфавите

for(int i=0; i<\*alphabetSize; i++)

{

//Если символ есть в моделе алфавита

if(alphabet[i].symbol==symbol)

{

//Увеличитьеговероятность

mpz\_add(alphabet[i].probability.num, alphabet[i].probability.num, one);

//Поставить флаг, означающий что текущий символ находится в алфавите

inAlphabet=true;

//Закончить перебор алфавита

i=\*alphabetSize;

}

}

//Если символа нет в алфавите

if(inAlphabet==false)

{

//Добавить символ в модель алфавита

alphabet[\*alphabetSize].symbol=symbol;

//Добавить 1 в его вероятность

mpz\_init\_set\_ui(alphabet[\*alphabetSize].probability.num, 1);

alphabet[\*alphabetSize].probability.size=0;

//Увеличить счетчик размера алфавита

(\*alphabetSize)++;

}

}

void convertAlphabetModel(mpz\_t lengthText, unsigned int alphabetSize, elementOfAlphabet \*alphabet)

{

number numLength, numProbability;//переменные, для преобразования в тип number

mpz\_init(numLength.num);//инициализация numLength

mpz\_init(numProbability.num);//инициализация numProbability

//Занести левую границу для первого символа в модель алфавита

alphabet[0].value.size=0;

mpz\_init\_set\_ui(alphabet[0].value.num, 0);

//Для каждого символа алфавита

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

{

//Если символ eof

if(i==alphabetSize-1)

{

//Добавить 1 в вероятность eof

mpz\_init\_set\_ui(alphabet[i].probability.num, 1);

alphabet[i].probability.size=0;

}

//Расчитать вероятность символа

//Преобразовать дробь(вероятность/длина) в тип number

convFractionToNumbers(alphabet[i].probability.num, lengthText, &numProbability, &numLength);

//Значение лежащее в вероятности текущего символа поделить на длину текста и обновить вероятность

alphabet[i].probability=divTwoNumbers(numProbability, numLength);

//Посчитать левую границу следующуго символа

//К левой границе текущего симыола прибавить вероятность

alphabet[i+1].value=addTwoNumbers(alphabet[i].value, alphabet[i].probability);

}

}

typeError buildModelFromText(char\*\* textWithModel, elementOfAlphabet \*alphabet, unsigned int\* alphabetSize)

{

unsigned int index=0;

int position=0;//позицияцифрывстроке

char allNumber[]={"1234567890"};//строкацифр

char specSymbol[]={"rptnc"};//строкаобозначенийспец. символов

char bufSpecSymbol;//буферспец. символа

char \*bufStr=newchar [10000];//строка-буфердляхранениячастейинтерваласимвола

char \*existSpecSymb="";//указатель на существующий спец. символ

bool spec=false;//признак существования спец. символа в строке

bool isNumber=false;//признак, что спец. символ является цифрой

//Для каждой строки текста кроме последней с eof

for(int i=0; i<\*alphabetSize-1; i++)

{

//Разобрать строки и занести их в алфавит

//Если первый символ цифра или спец. символ

if(isdigit(textWithModel[i][0]) || (existSpecSymb=strchr(specSymbol, (int)textWithModel[i][0])))

{

//Если первый символ спец. символ

if(strlen(existSpecSymb))

{

position=1;//Присвоить единицу позиции цифры в строке

spec=true; //Поставить признак что существует спец. символ в строке

//Присвоить соответствующий спец. символ буферу

bufSpecSymbol=definitionSpecialSymbol(textWithModel[i][0], &isNumber);

}

//С нужной позиции найти длину первых цифр в строке начиная

index=strspn(textWithModel[i]+position, allNumber);

//Если кол-во цифр не равняется длине всей строки и кол-во цифр не равняется нулю

if(index!=strlen(textWithModel[i]+position) && index!=0)

{

//Считать в строку-буфер цифры которые идут до первого символа

strncpy(bufStr, textWithModel[i]+position, index);

bufStr[index]='\0';

//Записать цифры в значимую численную часть диапазона алфавита

mpz\_set\_str(alphabet[i].value.num, bufStr, 10);

if(spec==true)//Если в строке есть спец. символ

{

if(isNumber==true)//Если спец.символ является цифрой

{

bufSpecSymbol=textWithModel[i][index+2];//С нужной позиции добавить в буфер символ-цифру

position=3;//Изменить позицию цифры в строке

isNumber=false;

}

alphabet[i].symbol=bufSpecSymbol;//Записать спец. символ в алфавит

spec=false;

}

else//Иначе

{

//С нужной позиции занести следущий символ, который идет после начальных цифр, в алфавит

alphabet[i].symbol=textWithModel[i][index];

}

//Считать в строку-буфер оставшуюсь строку

strcpy(bufStr, &textWithModel[i][index+position+1]);

position=0;//Обнулить позицию цифры в строке

//Если оставшаяся строка, которая идет после главного символа, состоит только из цифр

if(strspn(bufStr, allNumber)==strlen(bufStr))

{

//Занести кол-во значимой числен. части диапазона в алфавит

alphabet[i].value.size=atol(bufStr);

existSpecSymb="";

}

//Иначе, вернутьошибку

else

return incorrectAlphabetModel;

}

//Иначе, вернутьошибку

else

return incorrectAlphabetModel;

}

//Иначе, вернутьошибку

else

return incorrectAlphabetModel;

}

//Разобратьстрокус eof

parseTheStringWithEofFromText(textWithModel[\*alphabetSize-1], \*alphabetSize, alphabet);

//Посчитать вероятность каждого символа кроме eof

findProbabilityOfTheAlphabetFromTheText(\*alphabetSize, alphabet);

//Освободить выделенную память

return allRight;

}

void parseTheStringWithEofFromText(char \*eofStr, unsigned int alphabetSize, elementOfAlphabet \*alphabet)

{

char \*bufEofStr=newchar [10000];//строка-буфердлястроки eof

int count=0;//кол-во слов в строке

char seps[] = "eof"; //строка разделителей(eof)

//Выделить память для указателей содержащих строки предназначенные для занесения в алфавит

char \*\*strings=newchar\*[4];

//Выделить память под массив указателей

strcpy(bufEofStr,eofStr);

//Разбить строку cодержащую информацию eof на подстроки

strings[count] = strtok(bufEofStr, seps);//Выделениепервогослова

while(strings[count] != NULL)

{

count++;

strings[count] = strtok(NULL, seps); //Выделениеслед. слова

}

//Занести первую цифру в значимую численную часть диапазона алфавита

mpz\_set\_str(alphabet[alphabetSize-1].value.num, strings[0], 10);

//Занести вторую цифру в кол-во значимой числен. части диапазона алфавита

alphabet[alphabetSize-1].value.size=atol(strings[1]);

//Занести третюю цифру в значимую численную часть вероятности алфавита

mpz\_set\_str(alphabet[alphabetSize-1].probability.num, strings[2], 10);

//Занести четвертую цифру в кол-во значимой числен. части вероятности алфавита

alphabet[alphabetSize-1].probability.size=atol(strings[3]);

//Освободить выделенную память

delete(bufEofStr);

}

void findProbabilityOfTheAlphabetFromTheText(unsigned int alphabetSize, elementOfAlphabet \*alphabet)

{

//Для каждого символа в алфавите кроме eof

for(int i=0; i<alphabetSize-1; i++)

{

//Вычесть из следующего диапазона алфавита текущий диапазон и записать в текущую вероятность символа

alphabet[i].probability=subTwoNumbers(alphabet[i+1].value, alphabet[i].value);

}

}

char definitionSpecialSymbol(char symbolFromString, bool\* isNumb)

{

char bufSymbol={'\0'};//буфердлясимвола

//Еслисимвол 'p'

if(symbolFromString=='p')

//Записать пробел в буфер

bufSymbol=' ';

//Еслисимвол 't'

if(symbolFromString=='t')

//Записать символ табуляции в буфер

bufSymbol='\t';

//Еслисимвол 'n'

if(symbolFromString=='n')

//Записать символ перевода строки в буфер

bufSymbol='\n';

//Если символ 'r'

if(symbolFromString=='r')

//Записать символ перевода каретки в буфер

bufSymbol='\r';

//Если символ 'c'

if(symbolFromString=='c')

//Поставить признак того, что символ является цифрой

\*isNumb=true;

return bufSymbol;

}

elementOfAlphabet \* initAlphabet(unsigned int amount)

{

//Выделить динамическую память под указанное число структур с алфавитом

elementOfAlphabet \*alphabet=new elementOfAlphabet[amount]; //Массивсалфавитом

//Длякаждойструктуры

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

{

//Инициализировать нулем вероятность и левую границу символа

mpz\_init(alphabet[i].probability.num);

alphabet[i].probability.size=0;

mpz\_init(alphabet[i].value.num);

alphabet[i].value.size=0;

}

//Вернуть массив с алфавитом

return alphabet;

}

void addZeros(unsigned int zerosCount, mpz\_t \*numForAddZeros)

{

//Добавить указанное количество нулей к числу

for(unsigned int i=0; i<zerosCount; i++)

{

mpz\_mul\_ui(\*numForAddZeros, \*numForAddZeros, 10);

}

}

void deleteZeros(number \*numForDeleteZeros)

{

//Покачисленнаячастькратна 10

while(mpz\_divisible\_ui\_p(numForDeleteZeros->num,10))

{

//Делимна 10

mpz\_cdiv\_q\_ui(numForDeleteZeros->num, numForDeleteZeros->num,10);

//Уменьшаем размер числа на 1

numForDeleteZeros->size--;

}

}

int preparingForAddOrSub (number left, number right, mpz\_t\* newNum, unsigned int \* resultSize)

{

int leftOrRightOperandWasChanged; //Переменная, указывающая был ли изменен один из операндов

//Определяем размер результата сложения двух операндов...

//Если размеры операндов совпадают

if(left.size==right.size)

{

//Сохраняем в результат размер левого операнда

\*resultSize=left.size;

leftOrRightOperandWasChanged=0;

}

//Иначе если количество нулей в левом операнде меньше, чем в правом

elseif(left.size<right.size)

{

//Добавляем к численной части левого операнда недостающее количество нулей

mpz\_set(\*newNum, left.num);

addZeros(right.size-left.size, newNum);

//Сохраняем в результат размер правого операнда

\*resultSize=right.size;

leftOrRightOperandWasChanged=1;

}

//Иначе

else

{

//Добавляем к численной части правого операнда недостающее количество нулей

mpz\_set(\*newNum, right.num);

addZeros(left.size-right.size, newNum);

//Сохраняем в результат размер левого операнда

\*resultSize=left.size;

leftOrRightOperandWasChanged=2;

}

return leftOrRightOperandWasChanged;

}

number addTwoNumbers(number left, number right)

{

number result; //Результатсложения

mpz\_init (result.num);

int leftOrRightOperandWasChanged=0; //Переменная, указывающая был ли изменен один из операндов

mpz\_t newNum; //Один из измененных операндов

mpz\_init (newNum);

unsigned int resultSize=0; //Размеррезультата

//Подготавливаемоперандыксложению

leftOrRightOperandWasChanged=preparingForAddOrSub(left, right, &newNum, & result.size);

//Складываем численные части в результат

if(leftOrRightOperandWasChanged==1)

{

mpz\_add(result.num, newNum, right.num);

}

elseif(leftOrRightOperandWasChanged==2)

{

mpz\_add(result.num, newNum, left.num);

}

else

{

mpz\_add(result.num, left.num, right.num);

}

//Отбрасываем лишние нули численной части

deleteZeros(&result);

//Возвращаемрезультат

return result;

}

number subTwoNumbers(number left, number right)

{

number result; //Результатвычитания

mpz\_init (result.num);

int leftOrRightOperandWasChanged=0; //Переменная, указывающая был ли изменен один из операндов

mpz\_t newNum; //Один из измененных операндов

mpz\_init (newNum);

unsigned int resultSize=0; //Размеррезультата

bool isPossible=true; //Флаг, указывающий возможность вычитания

//Проверка того, что правый операнд меньше левого

if (mpz\_cmp(left.num,right.num)<=0 && left.size==right.size)

{

isPossible=false;

result.size=0;

mpz\_set\_ui(result.num,0);

}

//Если вычитание возможно

if(isPossible)

{

//Подготавливаем операнды к вычитанию

leftOrRightOperandWasChanged=preparingForAddOrSub(left, right, &newNum, & result.size);

//Вычитаем численные части в результат

if(leftOrRightOperandWasChanged==1)

{

mpz\_sub(result.num, newNum, right.num);

}

elseif(leftOrRightOperandWasChanged==2)

{

mpz\_sub(result.num, left.num, newNum);

}

else

{

mpz\_sub(result.num, left.num, right.num);

}

//Отбрасываем лишние нули численной части

deleteZeros(&result);

}

//Возвращаемрезультат

return result;

}

number mulTwoNumbers(number left, number right)

{

number result; //Результатумножения

mpz\_init (result.num);

//Перемножаем численные части в результат

mpz\_mul(result.num,left.num, right.num);

//Сохраняем в результат размер равный сумме размеров двух операндов

result.size=left.size+right.size;

//Отбрасываем лишние нули численной части

deleteZeros(&result);

//Возвращаемрезультат

return result;

}

resultComparing compareTwoNumbers(const number left, const number right)

{

resultComparing result; //Результат сравнения двух чисел

mpz\_t newNum;

mpz\_init (newNum);

int compare; //Результат сравнения численных частей

//Если левое число имеет меньший размер, чем правое

if(left.size<right.size)

{

//Добавляем к численной части левого числа недостающее количество нулей

mpz\_set(newNum, left.num);

addZeros(right.size-left.size, &newNum);

//Сравниваем новое число с правым

compare=mpz\_cmp(newNum, right.num);

}

//Иначе если правое число имеет меньший размер, чем левое

elseif(left.size>right.size)

{

//Добавляем к численной части правого числа недостающее количество нулей

mpz\_set(newNum, right.num);

addZeros(left.size-right.size, &newNum);

//Сравниваем левое число с новым

compare=mpz\_cmp(left.num, newNum);

}

//Иначе

else

{

//Сравниваем левое число с правым

compare=mpz\_cmp(left.num, right.num);

}

//Сохраняем результат сравнения в result

if (compare==0)

{

result=equivalent;

}

elseif (compare>0)

{

result=more;

}

elseif (compare<0)

{

result=less;

}

//Возвращаемрезультат

return result;

}

void assignAValueToNumber(const number source, number \* newNum)

{

//Сохраняем размер из источника в число

newNum->size=source.size;

//Сохраняем численную часть из источника в число

mpz\_set(newNum->num, source.num);

}

number divTwoNumbers(number dividend, number divider)

{

number result;

mpz\_init (result.num);

mpz\_t newNum;

mpz\_init (newNum);

mpz\_set(newNum, dividend.num);

//Добавляем 5 нулей в делимое для увеличения точности результата

addZeros(5,&newNum);

//Выполняем деление численных частей

mpz\_div(result.num,newNum, divider.num);

//Сохраняем размер результата, вычитая из новой длины делимого длину делителя

result.size=dividend.size+5-divider.size;

//Отбрасываем лишние нули численной части

deleteZeros(&result);

//Возвращаемрезультат

return result;

}

void convFractionToNumbers(mpz\_t numerator, mpz\_t denominator, number\* newNumerator, number\* newDenominator)

{

//Определить длины числителя и знаменателя

char\* numInStr=NULL; //Строка, в которую будет переведено число

numInStr=mpz\_get\_str (numInStr, 10, numerator);

size\_t sizeNum=strlen(numInStr); //Длиначислителя

numInStr=NULL;

numInStr=mpz\_get\_str (numInStr, 10, denominator);

size\_t sizeDen=strlen(numInStr); //Длиназнаменателя

//Вычесть из длины знаменателя длину числителя

size\_t difference=sizeDen-sizeNum; //Разность длин знаменателя и числителя

//Добавить это число в количество знаков после запятой нового числителя

newNumerator->size=difference;

//Добавить числитель и знаменатель в численные части новых числителя и знаменателя

mpz\_set(newNumerator->num, numerator);

mpz\_set(newDenominator->num, denominator);

//Добавить длину знаменателя в количество знаков после запятой нового знаменателя

newDenominator->size=sizeDen;

//Прибавить длину числителя к количеству знаков после запятой нового числителя

newNumerator->size+=sizeNum;

//Убрать лишние нули из числителя и знаменателя с помощью функции deleteZeros

deleteZeros(newNumerator);

deleteZeros(newDenominator);

}

void constrictInterval(char symbol, positionInText posInText, elementOfAlphabet \*alphabet, unsigned int alphabetSize, number& leftBorder, number& rightBorder)

{

number buf; //Буфер

mpz\_init(buf.num);

number diff; //Разность между начальными границами интервала

mpz\_init(diff.num);

//Если искомый символ не является первым в тексте

if(posInText!=isFirst && posInText!=isOne)

{

//Найти разность между начальными границами интервала

diff=subTwoNumbers(rightBorder, leftBorder);

}

bool isFoundSymbol=false; //Флаг, показывающий, найден ли искомый символ

int position=0; //Позиция искомого символа в алфавите

//Пока не будет пройден нужный символ в алфавите

while(!isFoundSymbol)

{

//Если искомый символ совпал с текущим

if(alphabet[position].symbol==symbol)

{

//Символ найден

isFoundSymbol=true;

}

else

{

//Увеличить позицию текущего символа

position++;

}

}

//Пока не поиск не дойдет до начала интервала искомого символа

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

{

//Если искомый символ не является первым в тексте

if(posInText!=isFirst && posInText!=isOne)

{

//Умножить разность на вероятность текущего символа алфавита

buf=mulTwoNumbers(diff, alphabet[i].probability);

}

//Иначе

else

{

/\*В этом случае разность между границами интервала равна 1,

а это число нельзя представить с помощью структуры number,

а т.к. x\*1=x, где x - некоторое число, то всего лишь

сохраним вероятность искомого символа\*/

assignAValueToNumber(alphabet[i].probability, &buf);

}

//Прибавить полученное число к левой границе интервала

leftBorder=addTwoNumbers(leftBorder, buf);

}

//Очистить буфер

mpz\_set\_str(buf.num, "0", 10);

buf.size=0;

//Если искомый символ не является первым в тексте

if(posInText!=isFirst && posInText!=isOne)

{

//Умножить разность на вероятность искомого символа

buf=mulTwoNumbers(diff, alphabet[position].probability);

}

//Если символ является первым в тексте

else

{

//Сохранить в буфер вероятность искомого символа

assignAValueToNumber(alphabet[position].probability, &buf);

}

//Прибавить полученное число к левой границе и сохранить это значение для правой границы

rightBorder=addTwoNumbers(leftBorder, buf);

//Очистить разность

mpz\_set\_str(diff.num, "0", 10);

diff.size=0;

//Если символ - последний во всем тексте

if(posInText==isLast || posInText==isOne)

{

//Вычислить текущую разницу между границами

diff=subTwoNumbers(rightBorder, leftBorder);

//Умножить полученную разность на вероятность EOF-символа и сохранить в буфер

buf=mulTwoNumbers(diff, alphabet[alphabetSize-1].probability);

//Вычесть из правой границы значение буфера и сохранить в левую границу

leftBorder=subTwoNumbers(rightBorder, buf);

}

}

void findArithmeticalMean(number leftBorder, number rightBorder, number\* middleValue)

{

number tmpValue;

number dividingNumber;

mpz\_init(tmpValue.num);

mpz\_init(dividingNumber.num);

dividingNumber.size=1;

mpz\_set\_ui (dividingNumber.num, 5);

//Вычестьизлевойграницыправую

tmpValue=subTwoNumbers(rightBorder, leftBorder);

//Умножитьна 0,5

tmpValue=mulTwoNumbers(tmpValue, dividingNumber);

//Прибавитькправойгранице

\*middleValue=addTwoNumbers(leftBorder, tmpValue);

}

elementOfAlphabet findSymbolAccordingToInterval(number currentNumber, elementOfAlphabet \* alphabet, unsigned int alphabetSize)

{

elementOfAlphabet resultSymbol; //Найденныйсимвол

mpz\_init(resultSymbol.probability.num);

mpz\_init(resultSymbol.value.num);

bool isFounded=false; //Флагнайденлисимвол

unsigned int i=0;

//Искать интервал для данного числа

do

{

if(compareTwoNumbers(alphabet[i].value, currentNumber)==less

&& compareTwoNumbers(alphabet[i+1].value, currentNumber)==more)

{

isFounded=true;

//Сохранитьсимвол

resultSymbol.symbol=alphabet[i].symbol;

assignAValueToNumber(alphabet[i].probability, &resultSymbol.probability);

assignAValueToNumber(alphabet[i].value, &resultSymbol.value);

}

i++;

}

//Пока искомый символ не будет найден и не кончится алфавит

while (isFounded==false&& i<alphabetSize-1);

return resultSymbol;

}

typeError inspectAndParseArgumentsOfCmd(int argc, char argv[4][1000], log\* logInfo)

{

//Проверить кол-во аргументов командной строки

//Если кол-во аргументов cmd больше 4

if(argc>4)

{

//Записать в лог сообщение об ошибке и вернуть ошибку

logInfo->error=tooManyArgs;

return tooManyArgs;

}

//Иначе, проверить второй аргумент cmd

else

{

//Если второй аргумент не равен ни "-p", ни "-u"

if(strcmp(argv[1], "-p")!=0 && strcmp(argv[1], "-u")!=0)

{

//Записать в лог сообщение об ошибке и вернуть ошибку

logInfo->error=incorrectMode;

return incorrectMode;

}

}

//Записать в лог имена входного и выходного файлов

strcpy(logInfo->inputFileName, argv[2]);

strcpy(logInfo->outputFileName, argv[3]);

//Записать в лог сообщение о том, что аргументы проверены и разобранны

logInfo->stageInfo=inspectArgumCmd;

writeToLog(\*logInfo);

return allRight;

}

log initLog()

{

//Инициализируем структуру начальными значениями

log logInfo;

strcpy(logInfo.logFilename, "");

strcpy(logInfo.inputFileName, "");

strcpy(logInfo.outputFileName, "");

logInfo.error=allRight;

logInfo.stageInfo=selectedP;

mpz\_init(logInfo.lengthOfText);

//Возвращаем "чистую структуру"

return logInfo;

}

void writeToLog(struct log tmp)

{

char logStr[400]="";//строка с сообщением

char time[500]="";//текущее время

//Открыть лог-файл на запись

FILE\* logStream=fopen(tmp.logFilename, "a");

//Если удалось открыть лог-файл

if(logStream)

{

//Если перечисление содержит ошибку

if(tmp.error!=allRight)

{

//Получить строку с ошибкой для лог-файла

strcpy(logStr,getErrorForLog(tmp.error));

}

//Иначе...

else

{

//Получить строку с инфомацией для лог-файла

strcpy(logStr, getStageOfExecutionForLog(tmp));

}

//Получить время этапа выполнения программы

timeForLog(time);

//Добавить к строке содержащей сообщение, время выполения программы

strcat(time, logStr);

//С новой строки вывести строку в лог-файл

strcat(time,"\n");

fputs(time, logStream)==EOF;

}

fclose(logStream);

}

har\* getErrorForLog(typeError tmpEr)

{

char errorStr[40]="";//Строка для записи сообщения

switch(tmpEr)

{

//Если входной параметр принимает определенное значение

case tooLittleArgs:

//Выполнить последовательность операторов соответствующих этому значению

strcpy(errorStr, "Error: There aren’t enough files.\n ");

break;

case tooManyArgs:

strcpy(errorStr, "Error: There are too much files.\n ");

break;

case incorrectMode:

strcpy(errorStr, "Error: The mode is incorrect.\n ");

break;

case unluckyReading:

strcpy(errorStr, "Error: Input file can’t be read.\n ");

break;

case unluckyWriting:

strcpy(errorStr, "Error: Output file can’t be written.\n ");

break;

case incorrectInterval:

strcpy(errorStr, "Error: Interval is incorrect.\n ");

break;

case incorrectAlphabetModel:

strcpy(errorStr, "Error: Model is incorrect.\n ");

}

return errorStr;

}

char\* getStageOfExecutionForLog(struct log tmp)

{

char\* buf=NULL;

char infoStr[200]="";

switch(tmp.stageInfo)

{

//Если входной параметр принимает определенное значение

case startProgram:

//Выполнить последовательность операторов соответствующих этому значению

strcpy(infoStr, "Program started its execution. ");

break;

case selectedP:

strcpy(infoStr, "Pack mode is selected. ");

break;

case selectedU:

strcpy(infoStr, "Unpack mode is selected. ");

break;

case inspectArgumCmd:

strcpy(infoStr, "Inspection of command line arguments was successful. Command line arguments are correct and were parsed successfully. ");

break;

case firstByteForAlphabetRead:

strcpy(infoStr, "The first byte was read for the alphabet building. The alphabet building was began. ");

break;

case alphabetWasBuilt:

strcpy(infoStr, "The alphabet building was completed. ");

break;

case firstByteForActionRead:

strcpy(infoStr, "Data reading was started. ");

break;

case textRead:

strcat(strcpy(infoStr, "The text or number was read. Its amount of symbols is: "), mpz\_get\_str(buf, 10, tmp.lengthOfText));

break;

case startPack:

strcat(strcpy(infoStr, "Packing was started: file "), tmp.inputFileName);

break;

case finishPack:

strcat(strcpy(infoStr, "Packing was finished: file "), tmp.inputFileName);

break;

case writePackedTextAndAlphabet:

strcat(strcpy(infoStr, "The packed text and alphabet were written in file "), tmp.outputFileName);

break;

case alphabetRead:

strcpy(infoStr, "The alphabet was read successfully. ");

break;

case alphabetWasParsed:

strcpy(infoStr, "The alphabet was parsed successfully. ");

break;

case startUnpack:

strcat(strcpy(infoStr, "Unpacking was started: file "), tmp.inputFileName);;

break;

case finishUnpack:

strcat(strcpy(infoStr, "Unpacking was finished: file "), tmp.inputFileName);;

break;

case writeUnpackedText:

strcat(strcpy(infoStr, "The unpacked text was written in file "), tmp.outputFileName);

break;

case finishProgram:

strcpy(infoStr, "Program finished its execution.\n ");

}

return infoStr;

}

void timeForLog(char\* clock)

{

struct tm\* convTime ;//структура, содержащая компоненты календарного времени

time\_t currentTime;

char strTime[40];

//Получить текущее время в секундах

time(&currentTime);

//Преобразовать время в местное предствление

convTime=localtime(&currentTime);

//Преобразовать представление времени в строку

strftime(strTime, 50, "%d.%m.%Y %H:%M:%S %A ",convTime);

//Скопировать полученное время в выходную строку

strcpy(clock, strTime);

typeError packFile(FILE\* inputStream, log logInfo, FILE\* outputStream)

{

typeError error=allRight; //Тип ошибки

//Выделить память

char \*text=new char[100000000]; //Текст, считанный из файла

elementOfAlphabet \*alphabet=initAlphabet(257); //Массив с алфавитом

unsigned int sizeOfAlphabet=0; //Размер алфавита

number packedText; //Запакованный текст

mpz\_init(packedText.num);

//Записать в лог сообщение о выбранном режиме упаковки

logInfo.stageInfo=selectedP;

writeToLog(logInfo);

//Прочитать входной файл посимвольно и составить алфавит

error=readDataForAlphabet(inputStream, logInfo, alphabet, &sizeOfAlphabet);

//Если чтение невозможно

if(error!=allRight)

{

//Вернуть ошибку и прекратить работу функции

return error;

}

//Прочитать входной файл порциями по 10000 символов

error=readDataForPack(inputStream, logInfo, text);

//Если чтение невозможно

if(error!=allRight)

{

//Вернуть ошибку и прекратить работу функции

return error;

}

//Упаковать входной текст, используя составленный алфавит

assignAValueToNumber(packText(text, alphabet, sizeOfAlphabet, false, logInfo), &packedText);

//Записать запакованный текст в выходной файл

error=writeDataForPack(logInfo, packedText, alphabet, sizeOfAlphabet, outputStream);

//Если запись не удалась

if(error!=allRight)

{

//Вернуть ошибку и прекратить работу функции

return error;

}

//Освободить выделенную память

delete(text);

delete(alphabet);

//Вернуть признак успешности

return allRight;

}

number packText(char\* text, elementOfAlphabet\* alphabet, unsigned int sizeOfAlphabet, bool markOfTest, log logInfo)

{

number leftBorder; //Левая раница интервала

mpz\_init(leftBorder.num);

leftBorder.size=0;

number rightBorder; //Правая граница интервала

mpz\_init(rightBorder.num);

rightBorder.size=0;

number packedText; //Упакованный текст в виде числа

mpz\_init(packedText.num);

packedText.size=0;

unsigned int lengthOfText=strlen(text); //Длина текста

unsigned int counter=0; //Счетчик

positionInText posInText=isWithin; //Позиция символа в тексте

//Пока не будет пройден последний символ текста

while(counter<lengthOfText)

{

//Установить значение "внутри" для типа позиции символа в тексте

posInText=isWithin;

//Если символ является первым

if(counter==0)

{

//Установить значение "первый" для типа позиции символа в тексте

posInText=isFirst;

//Если данная функция вызвана не из тестов

if(!markOfTest)

{

//Записать в лог сообщение о начале упаковки

logInfo.stageInfo=startPack;

writeToLog(logInfo);

}

}

//Если символ является последним

if(counter==(lengthOfText-1))

{

//Установить значение "последний" для типа позиции символа в тексте

posInText=isLast;

}

//Если текст состоит всего из 1 символа

if(lengthOfText==1)

{

//Установить значение "единственный" для типа позиции символа в тексте

posInText=isOne;

}

//Найти интервал для текущего символа (с помощью функции constrictInterval)

constrictInterval(text[counter], posInText, alphabet, sizeOfAlphabet, leftBorder, rightBorder);

//Перейти на следующий символ

counter++;

}

//Найти среднее арифметическое для границ интервала

findArithmeticalMean(leftBorder, rightBorder, &packedText);

//Если данная функция вызвана не из тестов

if(!markOfTest)

{

//Записать в лог сообщение об окончании упаковки

logInfo.stageInfo=finishPack;

writeToLog(logInfo);

}

//Вернуть запакованный текст в виде числа

return packedText;

}

typeError readDataForAlphabet(FILE\* inputStream, log logInfo, elementOfAlphabet\* alphabet, unsigned int\* sizeOfAlphabet)

{

char symbol; //Текущий символ

int symb=0; //Значение текущего считываемого символа

mpz\_t one; //1

mpz\_init\_set\_str (one, "1", 10);

mpz\_t lengthOfText; //Количество символов в тексте

mpz\_init(lengthOfText);

bool firstReading=true; //Флаг, сигналирующий о том, что считан первый символ

//Если считывание невозможно

if(!inputStream)

{

//Закрыть дескриптор файла

fclose(inputStream);

//Записать в лог сообщение об ошибке

logInfo.error=unluckyReading;

writeToLog(logInfo);

//Вернуть ошибку и прекратить работу функции

return unluckyReading;

}

//Считывая каждый символ

while((symb=getc(inputStream))!=EOF)

{

symbol=char(symb);

//Если считался первый символ

if(firstReading)

{

//Записать в лог сообщение о первом прочитанном байте

logInfo.stageInfo=firstByteForAlphabetRead;

writeToLog(logInfo);

firstReading=false;

}

//Инкрементировать счетчик числа символов в тексте

mpz\_add(lengthOfText, lengthOfText, one);

//Проверить наличие текущего символа в алфавите (с помощью функции buildAlphabetModel)

buildAlphabetModel(symbol, sizeOfAlphabet, alphabet);

}

//Добавить к размеру массива 1, учитывая символ eof

(\*sizeOfAlphabet)++;

//Прибавить 1 к счетчику числа символов в тексте (EOF-символ)

mpz\_add(lengthOfText, lengthOfText, one);

//Вычислить вероятности и интервалы каждого символа из алфавита (функция convertAlphabetModel)

convertAlphabetModel(lengthOfText, \*sizeOfAlphabet, alphabet);

//Записать в лог сообщение о том, что алфавит построен

logInfo.stageInfo=alphabetWasBuilt;

writeToLog(logInfo);

//Перемещаем курсор на начало файла

fseek(inputStream, 0, SEEK\_SET);

//Вернуть признак успешности

return allRight;

}

typeError readDataForPack(FILE\* inputStream, log logInfo, char\* text)

{

char buf[10001]=""; //Буфер

unsigned int counter=0; //Счетчик считанных символов

bool firstReading=true; //Флаг, сигналирующий о том, что считан первый символ

//Если считывание невозможно

if(!inputStream)

{

//Закрыть дескриптор файла

fclose(inputStream);

//Записать в лог сообщение об ошибке

logInfo.error=unluckyReading;

writeToLog(logInfo);

//Вернуть ошибку и прекратить работу функции

return unluckyReading;

}

//Пока не конец файла

while (!feof(inputStream))

{

//Считываем из файла в буфер по 10000 символов

fread(buf, 1, 10000, inputStream);

//Если считываем первый раз

if (firstReading)

{

//Записать в лог сообщение о начале считывания

logInfo.stageInfo=firstByteForActionRead;

writeToLog(logInfo);

firstReading=false;

//Копируем в строку прочитанные символы

strcpy(text, buf);

}

//Иначе

else

{

//Добавляем к тексту прочитанные символы

strcat(text, buf);

}

}

//Узнать длину считанного текста

counter=strlen(text)+1;

//Записать полученное число в структуру с логом

mpz\_set\_ui (logInfo.lengthOfText, counter);

//Записать в лог сообщение об окончании считывания

logInfo.stageInfo=textRead;

writeToLog(logInfo);

//Вернуть признак успешности

return allRight;

}

typeError writeDataForPack(log logInfo, number packedText, elementOfAlphabet \* alphabet, unsigned int sizeOfAlphabet, FILE\* outputStream)

{

char allNumbers[]={"1234567890"};

bool errorWriting=false;//признак, ошибки записи

//Записать значимую численную часть запакованного текста в выходной поток

//Если не удалось записать, вернуть ошибку

if(mpz\_out\_str(outputStream, 10, packedText.num)==0)

errorWriting=true;

//Записать количество знаков после запятой запакованного текста в выходной поток

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, " %lu\n" , packedText.size)<0)

errorWriting=true;

//Для каждого символа в алфавите кроме eof

for(int i=0; i<sizeOfAlphabet-1; i++)

{

//Если текущий символ в алфавите является спец. символом

if(alphabet[i].symbol=='\t' || alphabet[i].symbol=='\n' || alphabet[i].symbol==' ' ||

alphabet[i].symbol=='\r' || strspn(&alphabet[i].symbol, allNumbers)!=0)

{

//Записать спец. символ в выходной поток

errorWriting=writeSpecialSymbolForPack(alphabet[i], outputStream);

}

//Иначе

else

{

//Записать значимую численную часть левой границы символа в выходной поток

//Если не удалось записать, вернуть ошибку

if(mpz\_out\_str (outputStream, 10, alphabet[i].value.num)==0)

errorWriting=true;

//Записать символ и количество знаков после запятой в выходной поток

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, "%c%lu\n" , alphabet[i].symbol, alphabet[i].value.size)<0)

errorWriting=true;

}

}

//Записать данные EOF в файл

errorWriting=writeEOFDataForPack(alphabet, sizeOfAlphabet, outputStream);

//Закрыть поток выходного файла

fclose(outputStream);

//Если произошла ошибка записи данных

if(errorWriting)

{

//Записать в лог сообщение об ошибке и вернуть ошибку

logInfo.error=unluckyWriting;

writeToLog(logInfo);

return unluckyWriting;

}

//Записать в лог сообщение о записи запакованного текста и алфавита

logInfo.stageInfo=writePackedTextAndAlphabet;

writeToLog(logInfo);

return allRight;

}

bool writeEOFDataForPack(elementOfAlphabet \* alphabet, unsigned int sizeOfAlphabet, FILE\* outputStream)

{

bool errorWriting=false;//признак, ошибки записи

//Записать значимую численную часть диапазона EOF в выходной поток

//Если не удалось записать, вернуть ошибку

if(mpz\_out\_str (outputStream, 10, alphabet[sizeOfAlphabet-1].value.num)==0)

errorWriting=true;

//Записать символ 'e', количество знаков после запятой и символ 'o'в выходной поток

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, "%c%lu%c", 'e', alphabet[sizeOfAlphabet-1].value.size, 'o')<0)

errorWriting=true;

//Записать значимую численную часть вероятности EOF в выходной поток

//Если не удалось записать, вернуть ошибку

if(mpz\_out\_str (outputStream, 10, alphabet[sizeOfAlphabet-1].probability.num)==0)

errorWriting=true;

//Записать символ 'f' и количество знаков после запятой в выходной поток

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, "%c%lu", 'f', alphabet[sizeOfAlphabet-1].probability.size)<0)

errorWriting=true;

return errorWriting;

}

bool writeSpecialSymbolForPack(elementOfAlphabet alphabet, FILE\* outputStream)

{

char specSymbol;

bool errorWriting=false;//признак, ошибкизаписи

bool isNumber=false;//признак цифры в символе алфавита

if(alphabet.symbol=='\t')//Если в алфавите символ табуляции

specSymbol='t';//Записать соответствующую букву (t) в строку-буфер

else

{

if(alphabet.symbol=='\n')//Если в алфавите символ перевода строки

specSymbol='n';//Записать соответствующую букву (n) в строку-буфер

else

{

if(alphabet.symbol==' ')//Если в алфавите пробел

specSymbol='p';//Записать соответствующую букву (p) в строку-буфер

else

{

if(alphabet.symbol=='\r')//Если в алфавите символ перевода каретки

specSymbol='r';//Записать соответствующую букву (r) в строку-буфер

else

{

//Записать букву (c) в строку-буфер означающую что текущий символ алфавита является цифра

specSymbol='c';

isNumber=true;//Признак цифры в алфавите

}

}

}

}

//Записать в выходной поток букву соответствующую спец. символу

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, "%c", specSymbol)<0)

errorWriting=true;

//Записать значимую численную часть левой границы символа в выходной поток

//Если не удалось записать, вернуть ошибку

if(mpz\_out\_str (outputStream, 10, alphabet.value.num)==0)

errorWriting=true;

//Если символ в алфавите оказался не цифрой

if(!isNumber)

{

//Записать записать букву соответствующую спец. символу

//Если не удалось записать, вернуть ошибку

if(fprintf (outputStream, "%c%lu\n", specSymbol, alphabet.value.size)<0)

errorWriting=true;

}

else

{

//Записать цифру и вероятность в соответствующем формате

if(fprintf (outputStream, "%c%c%c%lu\n", specSymbol, alphabet.symbol, specSymbol, alphabet.value.size)<0)

errorWriting=true;

}

return errorWriting;

}

typeError readDataForUnpack(FILE\* inputStream, log logInfo, elementOfAlphabet \*\* alphabet, unsigned int\* sizeOfAlphabet, number \* NumberForUnpack)

{

typeError error=allRight; //Тип ошибки

char buf[10001]=""; //Буфер

char buffer[10001]=""; //Буфер

char \*lineBreak; //Указатель на конец строки с числом

char \*stringWithNumber=new char[300000000]; //Число, считанное из файла

char \*\*alphabetArray= new char\*[257]; //Массив с алфавитом

bool numberIsRead=false; //Флаг, указывающий считано ли число

bool firstReading=true; //Флаг, указывающий производится ли чтение из буфера в первый раз

bool numberIsIncorrect=false; //Флаг, указывающий корректность считанного числа

unsigned int counter=0; //Счетчик считанных символов

bool sizeIsRead=false; //Флаг, указыващий, считан ли размер числа

char symbol; //Считываемый символ

int symb=0; //Значение считываемого символа

//Выделение памяти для алфавита

for(int i=0; i<257;i++)

{

alphabetArray[i]=new char[10000];

}

//Если файл не может быть прочитан

if (!inputStream)

{

//Закрыть дескриптор файла

fclose(inputStream);

//Записать в лог сообщение об ошибке

logInfo.error=unluckyReading;

writeToLog(logInfo);

//Вернуть ошибку и прекратить работу функции

return unluckyReading;

}

//Иначе

else

{

//Пока не конец файла или число не считано

while (!feof(inputStream) || !numberIsRead)

{

//Считываем из файла в буфер

fread(buf,1,10000,inputStream);

//Если считываем первый раз

if (firstReading)

{

//Записать в лог сообщение о начале считывания

logInfo.stageInfo=firstByteForActionRead;

writeToLog(logInfo);

//Копируем в строку часть числа

strcpy(stringWithNumber,buf);

firstReading=false;

}

//Иначе

else

{

//Добавляем к строке часть числа

strcat(stringWithNumber,buf);

}

//Ищем конец числа

lineBreak = strchr(stringWithNumber,' ');

//Если конец числа найден

if (lineBreak != NULL)

{

//Число считано

\*lineBreak=0;

numberIsRead=true;

//Узнать длину считанного текста

counter=strlen(stringWithNumber)+1;

//Записать полученное число в структуру с логом

mpz\_set\_ui (logInfo.lengthOfText, counter);

//Записать в лог сообщение об окончании считывания

logInfo.stageInfo=textRead;

writeToLog(logInfo);

}

strcpy(buf, " ");

}

//Сохраняем число

numberIsIncorrect=mpz\_set\_str(NumberForUnpack->num, stringWithNumber, 10);

//Если число не удалось сохранить из строки

if(numberIsIncorrect)

{

//Записать в лог сообщение об ошибке

logInfo.error=incorrectInterval;

writeToLog(logInfo);

//Вернуть ошибку

return incorrectInterval;

}

else

{

strcpy(buf, " ");

counter=0;

//Перемещаем курсор на начало размера числа

fseek(inputStream,strlen(stringWithNumber)+1,SEEK\_SET);

//Пока не конец файла или размер числа не считан

while ((symb=getc(inputStream))!=EOF && !sizeIsRead)

{

symbol=char(symb);

//Считываем посимвольно размер числа

//Если текущий символ является переводом строки

if(symbol=='\n')

{

//Прекращаем считывание

sizeIsRead=true;

}

else

{

//Сохраняем цифру

buffer[counter]=symbol;

//Инкрементируем счетчик символов

counter++;

}

}

buffer[counter]='\0'; //ставим символ конца строки

//Сохраняем размер числа

NumberForUnpack->size=atol(buffer);

//Перемещаем курсор на начало текста

fseek(inputStream,strlen(stringWithNumber)+2+strlen(buffer),SEEK\_SET);

//Пока не конец файла

while(!feof(inputStream))

{

//Считать модель алфавита из входного файла и записать ее в массив строк

fscanf(inputStream,"%s\n",alphabetArray[\*sizeOfAlphabet]);

(\*sizeOfAlphabet)++;

}

//Записать в лог сообщение об окончании считывания алфавита

logInfo.stageInfo=alphabetRead;

writeToLog(logInfo);

//Вызов функции buildModelFromText

error=buildModelFromText(alphabetArray, \*alphabet, sizeOfAlphabet);

//Если построить алфавит не удалось

if(error!=allRight)

{

//Записать в лог сообщение об ошибке

logInfo.error=incorrectAlphabetModel;

writeToLog(logInfo);

//Вернуть ошибку и прекратить работу функции

return error;

}

else

{

//Записать в лог сообщение о построении алфавита

logInfo.stageInfo=alphabetWasParsed;

writeToLog(logInfo);

}

}

}

//Освободить выделенную память

delete(stringWithNumber);

//Освобождение памяти для алфавита

for(int i=0; i<257;i++)

{

delete(alphabetArray[i]);

}

delete(alphabetArray);

//Вернуть признак успешности

return allRight;

}

typeError writeDataForUnpack(char\* unpackText, log logInfo, FILE\* outputStream)

{

//Записать в выходной поток распакованный текст

//Если запись не получилась

if(fputs(unpackText, outputStream)<0)

{

//Записать в лог сообщение о неудачной записи запакованного текста

logInfo.error=unluckyWriting;

writeToLog(logInfo);

//Вернуть ошибку

return unluckyWriting;

}

//Закрыть поток выходного файла

fclose(outputStream);

//Записать в лог сообщение о том, что произошла запись запакованного текста

logInfo.stageInfo=writeUnpackedText;

writeToLog(logInfo);

return allRight;

}

typeError unpackFile(FILE\* inputStream, log logInfo, FILE\* outputStream)

{

typeError error=allRight; //Тип ошибки

//Выделить память

char \*text=new char[300000000]; //Текст, считанный из файла

elementOfAlphabet \*alphabet=initAlphabet(257); //Массив с алфавитом

unsigned int sizeOfAlphabet=0; //Размер алфавита

number packedNumber; //Запакованный текст

mpz\_init\_set\_ui(packedNumber.num, 0);

//Записать в лог сообщение о выбранном режиме распаковки

logInfo.stageInfo=selectedU;

writeToLog(logInfo);

//Прочитать число (запакованный текст) и алфавит из входного файла

error=readDataForUnpack(inputStream, logInfo, &alphabet, &sizeOfAlphabet, &packedNumber);

//Если чтение невозможно

if(error!=allRight)

{

//Вернуть ошибку и прекратить работу функции

return error;

}

//Распаковать текст, используя прочитанный алфавит

unpackText(packedNumber, alphabet, sizeOfAlphabet, logInfo, false, text);

//Записать распакованный текст в выходной файл

error=writeDataForUnpack(text, logInfo, outputStream);

//Если запись не удалась

if(error!=allRight)

{

//Вернуть ошибку и прекратить работу функции

return error;

}

//Освободить выделенную память

delete(text);

delete(alphabet);

//Вернуть признак успешности

return allRight;

}

void unpackText(number middleValue, elementOfAlphabet \* alphabet, unsigned int alphabetSize, log logInfo, bool markOfTest, char\* resultText)

{

int i=0; //Счетчик для записи текста

elementOfAlphabet currentSymbol; //Текущий символ в алфавите

mpz\_init(currentSymbol.probability.num);

mpz\_init(currentSymbol.value.num);

number currentValue; //Текущее значение интервала

number endOfFile; //Символ конца файла

mpz\_init(currentValue.num);

mpz\_init(endOfFile.num);

//Если данная функция вызвана не из тестов

if(!markOfTest)

{

//Записать в лог сообщение о начале распаковки

logInfo.stageInfo=startUnpack;

writeToLog(logInfo);

}

//Присваиваем текущему значению среднее арифметическое интервала

assignAValueToNumber(middleValue, &currentValue);

//Сохраняем в переменную endOfFile значение из алфавита

assignAValueToNumber(alphabet[alphabetSize-1].value, &endOfFile);

//Пока не получен EOF

while (compareTwoNumbers(currentValue,endOfFile)==less)

{

//Находим символ

currentSymbol=findSymbolAccordingToInterval(currentValue, alphabet, alphabetSize);

//Сохраняем символ в текст

resultText[i]=currentSymbol.symbol;

//Пересчитываем число для определения нового интервала

currentValue=divTwoNumbers((subTwoNumbers(currentValue,currentSymbol.value)), currentSymbol.probability);

i++;

}

//Присвоить концу текста нуль-символ

resultText[i]='\0';

//Если данная функция вызвана не из тестов

if(!markOfTest)

{

//Записать в лог сообщение об окончании распаковки

logInfo.stageInfo=finishUnpack;

writeToLog(logInfo);

}

}

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

{

enum typeError error=allRight; //Тип ошибки

log logInfo=initLog(); //Структура с информацией о логе

char args[4][1000]={NULL}; //Аргументы командной строки

FILE\* inputStream; //Входной поток

FILE\* outputStream; //Выходной поток

FILE\* logStream; //Поток лога

//Открыть лог-файл

strcpy(logInfo.logFilename, "NumeralPackLog.txt"); //Имя файла с логом

logStream=fopen(logInfo.logFilename, "a");

//Если не удалось открыть лог-файл

if(!logStream)

{

//Вывести сообщение о неудаче открытия лога в консоль

printf("Log file can't be opened.");

//Закрыть лог-файл

fclose(logStream);

//Завершить программу

\_getch();

return 0;

}

//Закрыть лог-файл

fclose(logStream);

//Записать в лог о начале старта программы

logInfo.stageInfo=startProgram;

writeToLog(logInfo);

//Если количество аргументов командной строки не менее необходимого

if(argc>=4)

{

//Скопировать первые 4 аргумента в буфер

strcpy(args[0], argv[0]);

strcpy(args[1], argv[1]);

strcpy(args[2], argv[2]);

strcpy(args[3], argv[3]);

}

else

{

//Записать в лог сообщение об ошибке

logInfo.error=tooLittleArgs;

writeToLog(logInfo);

//Завершить программу

\_getch();

return 0;

}

//Проверить и разобрать аргументы командной строки

error=inspectAndParseArgumentsOfCmd(argc, args, &logInfo);

//Если аргументы командной строки ошибочны

if(error!=allRight)

{

//Завершить программу

\_getch();

return 0;

}

//Открыть входной поток

inputStream=fopen(logInfo.inputFileName, "rb");

//Если поток не удалось открыть

if(!inputStream)

{

//Записать в лог сообщение об ошибке

logInfo.error=unluckyReading;

writeToLog(logInfo);

//Закрыть входной поток

fclose(inputStream);

//Завершить программу

\_getch();

return 0;

}

//Открыть выходной поток

outputStream=fopen(logInfo.outputFileName, "a");

//Если поток не удалось открыть

if(!outputStream)

{

//Записать в лог сообщение об ошибке

logInfo.error=unluckyWriting;

writeToLog(logInfo);

//Закрыть входной и выходной потоки

fclose(inputStream);

fclose(outputStream);

//Завершить программу

\_getch();

return 0;

}

//Если выбран режим упаковки

if(strcmp(args[1], "-p")==0)

{

//Упаковать входной файл

error=packFile(inputStream, logInfo, outputStream);

//Если при упаковке файла произошла ошибка

if(error!=allRight)

{

//Закрыть входной и выходной потоки

fclose(inputStream);

fclose(outputStream);

//Завершить программу

\_getch();

return 0;

}

}

//Иначе если выбран режим распаковки

else if(strcmp(args[1], "-u")==0)

{

//Распаковать входной файл

error=unpackFile(inputStream, logInfo, outputStream);

//Если при распаковке файла произошла ошибка

if(error!=allRight)

{

//Закрыть входной и выходной потоки

fclose(inputStream);

fclose(outputStream);

//Завершить программу

\_getch();

return 0;

}

}

//Закрыть входной поток

fclose(inputStream);

//Записать в лог о завершении работы программы

logInfo.stageInfo=finishProgram;

writeToLog(logInfo);

//Завершить работу программы

\_getch();

return 0;

}

# Приложение З. Оценка полученных результатов

Таблица З.1. Оценка Волковой Е. полученных результатов

|  |  |  |
| --- | --- | --- |
| Критерий оценки | Достоинства | Недостатки |
| Работоспособность + функциональность программы | Программа выполняет все заявленные функции | Очень долго распаковывает файлы т.к. приходится выполнять деление и умножение больших чисел с высокой точностью. Профилировщиком very sleepy было подтверждено, что практически все время затрачивается на функцию умножения библиотеки gmp. |
| Внешняя спецификация программы | Целостно отображает структуру программы | Последняя версия спецификации была сделана после того, как я закончила свою часть работы |
| Декомпозиция программы на подпрограммы | Все подпрограммы выделены в отдельные функции | Не выявлено |
| Декомпозиция программы на модули | В целом каждый из модулей программы реализован определенным участником | На этапе разделения программы на модули преподавателем были внесены изменения, которые, на мой взгляд, противоречили логике разделения (считывание делает один человек, а запись – другой) |
| Псевдокод | Не вызывает нареканий и понятен каждому из участников проекта | Нет |
| Код (эффективность структур данных, реализация типовых алгоритмических структур, их комбинация) | Все отлично | Нет |

|  |  |  |
| --- | --- | --- |
| Форматирование кода | Не вызывает нареканий | Нет |
| Модульные тесты | Все хорошо | Нет |
| Ведение репозитория кода | Полностью отражает ход работы над программой | Существуют коммиты с исправлением опечаток |
| Порядок конструирования программы + распределение подпрограмм между членами команды | Хороший, но не идеальный | Возникли небольшие недочеты из-за разницы во времени реализации подпрограмм |

Продолжение Таблицы З.1.

Таблица З.2. Оценка Сивашовой Е. полученных результатов

|  |  |  |
| --- | --- | --- |
| Критерий оценки | Достоинства | Недостатки |
| Работоспособность + функциональность программы | Программа выполняет все заявленные функции | Программа справляется только с файлами небольшого размера (до 15 Кб) |
| Внешняя спецификация программы | Большая часть была сделана мной | Ее созданием занималась практически одна я |
| Декомпозиция программы на подпрограммы | Все выделенные подпрограммы соответствуют своим целям | Нет |
| Декомпозиция программы на модули | Все отлично | Возможно, можно было бы уменьшить число модулей |
| Псевдокод | Отражает решение задачи | Нет |

|  |  |  |
| --- | --- | --- |
| Код (эффективность структур данных, реализация типовых алгоритмических структур, их комбинация) | Все отлично | Всегда можно написать лучше |
| Форматирование кода | Все прекрасно | Нет |
| Модульные тесты | Все отлично | Можно было бы сделать модульные тесты к функциям чтения данных |
| Ведение репозитория кода | Отражает последовательность разработки программы. Коммиты подробны и многочисленны. | Коммитов, пожалуй, даже слишком много |
| Порядок конструирования программы + распределение подпрограмм между членами команды | Относительно хорошо: каждый получил задачу по своим способностям. Также благодаря своевременному участию всех участников команды появляющиеся проблемы довольно быстро решались | Некоторые подпрограммы были изначально неверно распределены, из-за чего пришлось продолжать разработку чужих подпрограмм, а также разница во времени выполнения различных частей программы привела к тому, что нагрузка на каждого участника команды легла различная |

Продолжение Таблицы З.2.

Таблица З.3. Оценка Стрекаловой А. полученных результатов

|  |  |  |
| --- | --- | --- |
| Критерий оценки | Достоинства | Недостатки |
| Работоспособность + функциональность программы | Программа выполняет все функции | Редко, но в программе возникает исключительная ситуация при упаковке больших файлов |
| Внешняя спецификация программы | Целостно отображает структуру программы | Ее созданием занималcя практически один человек из нашей подгруппы |
| Декомпозиция программы на подпрограммы | Все подпрограммы выделены в отдельные функции | Некоторые функции не являются легко обозримыми |
| Декомпозиция программы на модули | Все отлично | Нет |
| Псевдокод | Все отлично. Отражает идею решения программы | Возможно в одной из реализованных мною функций большой псевдокод, хоть и выделены в ней вспомогательные функции |
| Код (эффективность структур данных, реализация типовых алгоритмических структур, их комбинация) | Все отлично | Некоторые функции могли бы быть лучше закодированными |
| Форматирование кода | Все отлично | Нет |

|  |  |  |
| --- | --- | --- |
| Модульные тесты | Относительно хорошо | Возможно в некоторых функциях недостаточное количество тестов. Так же некоторые тесты могут показаться слишком объемными |
| Ведение репозитория кода | Все отлично | Не всегда может быть понятным назначение коммита |
| Порядок конструирования программы + распределение подпрограмм между членами команды | Благодаря скоординированной работе, мне с помощью Сивашовой Е. удалось относительно быстро отладить свою часть программы | Вследствие того, что разработка программы велась неравномерно, возникали ситуации, когда требовалось изменить чужие функции, а также часто приходилось исправлять чужие недочеты |

Продолжение Таблицы З.3.