АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1

Автор: API Антиплагиат МУ им. Витте apmuiv@muiv.ru / ID: 18
Проверяющий: API Антиплагиат МУ им. Витте (apmuiv@muiv.ru / ID: 18)
Организация: Московский Университет им. С.Ю. Витте

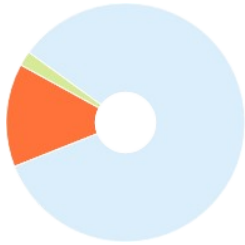
Отчет предоставлен сервисом «Антиплагиат» - http://muiv-vuz.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 12081
Начало загрузки: 14.02.2022 10:21:37
Длительность загрузки: 00:00:20
Имя исходного файла: Князев А.В.docx
Название документа: Князев А.В.docx
Размер текста: 1 кБ
Символов в тексте: 91416
Слов в тексте: 11271
Число предложений: 1636

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 14.02.2022 10:21:58
Длительность проверки: 00:03:15
Комментарии: не указано
Поиск с учетом редактирования: да
Модули поиска: ИПС Адилет, Библиография, Сводная коллекция ЭБС, Интернет Плюс, Сводная коллекция РГБ, Цитирование, Переводные заимствования (RuEn), Переводные заимствования по eLIBRARY.RU (EnRu), Переводные заимствования по Интернету (EnRu), Переводные заимствования издательства Wiley (RuEn), eLIBRARY.RU, СПС ГАРАНТ, Медицина, muiv-vuz, Диссертации НББ, Перефразирования по eLIBRARY.RU, Перефразирования по Интернету, Перефразирования по коллекции издательства Wiley, Патенты СССР, РФ, СНГ, СМИ России и СНГ, Шаблонные фразы, Кольцо вузов, Издательство Wiley, Переводные заимствования



ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
14,02%	0%	1,62%	84,36%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска	Комментарии
[01]	4,07%	Разработка программного продукта для решения прикладных задач | 189490 https://work5.ru	31 Янв 2022	Интернет Плюс	
[02]	2,37%	Silaenkov_A_K.docx	25 Ноя 2021	muiv-vuz	
[03]	2,75%	Ерохин В.В..doc	08 Фев 2022	muiv-vuz	
[04]	0,61%	Высокоуровневые методы программирования КР (Волков Егор).docx	14 Фев 2022	muiv-vuz	
[05]	0,13%	Романов ЮС.docx	10 Фев 2022	muiv-vuz	
[06]	0,1%	Патрашин А.В..docx	27 Янв 2022	muiv-vuz	
[07]	0,1%	Курсовая работа.doc	06 Дек 2021	muiv-vuz	
[08]	0,48%	Бебешин Н.С.docx	21 Янв 2022	muiv-vuz	
[09]	0,08%	Рыбъянов И.С.docx	25 Ноя 2021	muiv-vuz	
[10]	1,5%	Гаев ДН.docx	10 Фев 2022	muiv-vuz	
[11]	0,42%	Анисимов А.С..docx	27 Янв 2022	muiv-vuz	
[12]	0,12%	Пояснительная записка.docx	03 Дек 2021	muiv-vuz	
[13]	0%	Бощаев Н.Е. .docx	27 Янв 2022	muiv-vuz	
[14]	0%	Картинцев И.А.docx	18 Окт 2021	muiv-vuz	
[15]	0%	48565_Курсовая работа Скосарев.docx	17 Дек 2021	muiv-vuz	

[16]	0%	Севодин О.М..docx	27 Янв 2022	muiv-vuz	
[17]	0%	Русаков В.В.docx	10 Фев 2022	muiv-vuz	
[18]	0%	Лавров.docx	10 Фев 2022	muiv-vuz	
[19]	0%	Курсовая Кондратьев.docx	27 Янв 2022	muiv-vuz	
[20]	0,16%	Курсовая Тархов П.А..docx	17 Дек 2021	muiv-vuz	
[21]	0%	Курсовая Тумин v11.docx	03 Дек 2021	muiv-vuz	
[22]	1,62%	не указано	13 Янв 2022	Библиография	
[23]	0%	курсовая 1 - Методические указания по выполнению курсовой работы Язык программирования Python Содержание 1Цель и задачи дисциплины 3 https://topuch.ru	18 Окт 2021	Интернет Плюс	
[24]	0,17%	2021ВКР536320КУЩ	14 Фев 2022	Кольцо вузов	
[25]	0,62%	Python 3 и PyQt 5. Разработка приложений http://ibooks.ru	21 Янв 2020	Сводная коллекция ЭБС	
[26]	0%	Горожанов, Алексей Иванович Формирование обучающей виртуальной среды в контексте новых информационных технологий : диссертация ... доктора филологических наук : 10.02.21 Москва 2018 http://dlib.rsl.ru	15 Окт 2019	Сводная коллекция РГБ	
[27]	0%	Алгоритмы двумерной задачи раскроя - упаковки	08 Июн 2020	Кольцо вузов	
[28]	0%	pgm17 — Множества и словари https://server.179.ru	25 Мая 2020	Интернет Плюс	
[29]	0%	Множества и словари http://server.179.ru	27 Мая 2021	Интернет Плюс	
[30]	0%	Множества и словари http://server.179.ru	18 Окт 2021	Интернет Плюс	
[31]	0%	ВКР	17 Июн 2021	Кольцо вузов	
[32]	0%	ПЯВУ_Максудов Темур Бахтиярович	14 Июн 2017	Кольцо вузов	
[33]	0%	Миронова, Дарья Александровна диссертация ... кандидата филологических наук : 10.02.20 Тюмень 2013 http://dlib.rsl.ru	15 Мая 2014	Сводная коллекция РГБ	
[34]	0%	Повышение качества калибрования картофеля поверхностью с изменяющейся скоростью вращения роликов http://dep.nlb.by	16 Янв 2020	Диссертации НББ	
[35]	0,12%	利用pyqt4编写cs计算工具 - CSDN博客 http://blog.csdn.net	29 Мар 2018	Интернет Плюс	
[36]	0%	2016ВКР001119ЕВТИФИЕВ.docx	30 Мая 2016	Кольцо вузов	
[37]	0,21%	Тебекин, Юрий Борисович диссертация ... кандидата технических наук : 05.13.11 Воронеж 2012 http://dlib.rsl.ru	31 Мар 2015	Сводная коллекция РГБ	
[38]	0%	Палагин, Владимир Владимирович диссертация ... кандидата технических наук : 05.13.11 Москва 2014 http://dlib.rsl.ru	25 Дек 2015	Сводная коллекция РГБ	Источник исключен. Причина: Маленький процент пересечения.
[39]	0%	не указано	13 Янв 2022	Шаблонные фразы	Источник исключен. Причина: Маленький процент пересечения.
[40]	0%	Системное программное обеспечение. Лабораторный практикум http://ibooks.ru	09 Дек 2016	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[41]	0%	http://home.deib.polimi.it/casella/controllieng/ControlloSerbatoioDaTare.onb http://home.deib.polimi.it	14 Фев 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[42]	0%	АВТОМАТИЗАЦИЯ ПРОЦЕССА СОСТАВЛЕНИЯ КОМПЛЕКСОВ УПРАЖНЕНИЙ ПО ГРАММАТИКЕ НЕМЕЦКОГО ЯЗЫКА НА ОСНОВЕ КРУПНОГО ХУДОЖЕСТВЕННОГО ПРОИЗВЕДЕНИЯ. http://elibrary.ru	раньше 2011	eLIBRARY.RU	Источник исключен. Причина: Маленький процент пересечения.
[43]	0%	Синтез речеподобных помех для защиты информации от утечки по акустическим каналам http://dep.nlb.by	11 Ноя 2016	Диссертации НББ	Источник исключен. Причина: Маленький процент пересечения.
[44]	0%	Расчет и проектирование зубчатых передач многопарного зацепления трансмиссий тракторов "Беларус" http://dep.nlb.by	20 Дек 2016	Диссертации НББ	Источник исключен. Причина: Маленький процент пересечения.
[45]	0%	Языки программирования высокого уровня - online presentation https://en.ppt-online.org	14 Фев 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[46]	0%	ТЕХНОЛОГИЯ ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ. ЯЗЫК PGRAPH. http://elibrary.ru	11 Июл 2019	Перефразирования по eLIBRARY.RU	Источник исключен. Причина: Маленький процент пересечения.
[47]	0%	210218220812_Соколова_КР_АП_Герасимова.docx	21 Фев 2018	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[48]	0%	Литвинов, Юрий Викторович Методы и средства разработки графических предметно-ориентированных языков : диссертация ... кандидата технических наук : 05.13.11 Санкт-Петербург 2016	27 Дек 2019	Сводная коллекция РГБ	Источник исключен. Причина: Маленький процент пересечения.

		http://dlib.rsl.ru			
[49]	0%	"ОБЩЕРОССИЙСКИЙ КЛАССИФИКАТОР ВИДОВ ЭКОНОМИЧЕСКОЙ ДЕЯТЕЛЬНОСТИ, ПРОДУКЦИИ И УСЛУГ" ОК 004-93 (ред. от 01.02.2002) (утв. Постановлением Госстандарта РФ от 06.08.1993 № 17) (Часть III разделы D (коды 3510000 - 3720000), E - Q, часть IV) http://durex-promo.ru	14 Фев 2019	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[50]	0%	МЕЖБАНКОВСКИЙ КРЕДИТ И ЛИКВИДНОСТЬ БАНКА.	14 Фев 2022	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[51]	0%	Руководство по PyQt5 для начинающих - GUI Python https://python-scripts.com	23 Мая 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[52]	0%	Руководство по PyQt5 для начинающих - GUI Python https://python-scripts.com	16 Окт 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[53]	0%	Руководство по PyQt5 для начинающих - GUI Python https://python-scripts.com	16 Окт 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[54]	0%	Руководство по PyQt5 для начинающих - GUI Python https://python-scripts.com	21 Янв 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[55]	0%	Мир науки, культуры, образования. № 4 (41), август 2013 http://bibliorossica.com	26 Мая 2016	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[56]	0%	10.1. Теория — Курс Python (2021) https://yuripetrov.ru	08 Фев 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[57]	0%	Виртуальные валюты и виртуальные биржи в мире	14 Фев 2022	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[58]	0%	Постановление администрации сельского поселения Мансуровский сельсовет от 29 июля 2016 г. N 34 "Об утверждении Правил определения требований к закупаемым Администрацией сельского поселения Мансуровский сельсовет МР Учалинский район РБ отдельным видам то... http://municipal.garant.ru	21 Дек 2016	СПС ГАРАНТ	Источник исключен. Причина: Маленький процент пересечения.
[59]	0%	Создание Web-сайта. Недостающее руководство. 3-е изд. [Creating a Website: The Missing Manual by Matthew MacDonald] http://ibooks.ru	09 Дек 2016	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[60]	0%	python - AttributeError: 'MyMainWindow' object has no attribute 'pushButton' - Stack Overflow https://stackoverflow.com	07 Июн 2021	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[61]	0%	Рисование в PyQt5 | Python 3 для начинающих и чайников https://pythonworld.ru	14 Фев 2022	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[62]	0%	PyQT5 за 5 минут - YouTube https://youtube.com	11 Ноя 2021	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[63]	0%	1.2 Характеристика безналичного денежного оборота: принципы и формы. Система наличного и безналичного денежного оборота - курсовая работа http://banki.bobrodobro.ru	20 Мар 2019	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.

титульный лист

Аннотация

Курсовая работа по предмету «Высокоуровневые методы программирования» содержит описание разработки четырех программ. Первая программа предназначена для подсчета числа использования различных слов в текстовом файле, не содержит пользовательского интерфейса. Вторая программа представляет из себя диалоговую банковскую систему, позволяющую создавать вклады, переводить средства между различными счетами, начислять проценты по вкладам и т.д. Третья программа представляет из себя программу-калькулятор, реализующую основные математические операции и дополнительные операции согласно заданию. Четвертая программа представляет из себя решение головоломки «Ханойские башни» с восемью шпинделями. Для разработки программного обеспечения используется язык программирования Python. В качестве библиотеки для разработки пользовательского интерфейса используется библиотека PyQt5.

Оглавление

Оглавление..... 3
Введение..... 5
1. Анализ заданий курсовой работы..... 6
1.1. Исходные данные к заданиям курсовой работы..... 6
1.2. Выбор и обоснование необходимых библиотек..... 10
1.3. Выводы по первой главе..... 12
2. Разработка программного продукта для решения прикладных задач.... 13
2.1. Работа с наборами данных..... 13
2.1.1. Построение алгоритма решения задания без графического интерфейса..... 13
2.1.2. Разработка программной реализации на языке программирования. 14
2.1.3. Тестирование и отладка..... 15
2.1.4. Формирование выходных файлов..... 19
2.2. Разработка экспертной системы..... 20
2.2.1. Построение алгоритма решения задания с пользовательским интерфейсом..... 20
2.2.2. Разработка программной реализации на языке программирования и с использованием дополнительных библиотек..... 23
2.2.3. Тестирование и отладка..... 24
2.2.4. Формирование выходных файлов..... 26
2.3. Разработка аналитической системы..... 27
2.3.1. Построение алгоритма решения задания с пользовательским интерфейсом..... 27
2.3.2. Разработка программной реализации на языке программирования. 31
2.3.3. Тестирование и отладка..... 31
2.3.4. Формирование выходных файлов..... 32
2.4. Разработка логико-аналитической системы..... 33
2.4.1. Построение алгоритма решения задания «Ханойские башни»..... 33
2.4.2. Разработка программной реализации..... 36

2.4.3. Тестирование и отладка..... 36
2.4.4. Формирование выходных файлов..... 36
2.5. Выводы по второй главе.....38
3. Разработка требований к техническим средствам реализации программного обеспечения для решения прикладных задач..... 41
Выводы..... 42
Список литературы.....43
Приложение А. Листинг текстов заданий..... 44
А.1. Текст программы exercise_1.py..... 44
А.2. Текст программы exercise_2.py..... 45
А.3. Текст программы exercise_3.py..... 50
А.4. Текст программы exercise_4.py..... 69
Приложение Б. Образцы GUI заданий..... 74
Б.1. Образец GUI по второму заданию..... 74
Б.2. Образец GUI по третьему заданию..... 74
Б.3. Образец GUI по четвертому заданию.....74

Введение

Целью данной курсовой работы является получение навыков разработки, отладки и тестирования программных систем с использованием современных языков программирования высокого уровня. Современные языки программирования высокого уровня предоставляют программисту широкие возможности для быстрого создания программ, отвечающих поставленным требованиям. Для этого используются богатые возможности стандартных и нестандартных библиотек, декомпозиция решаемой задачи и моделируемой предметной области на систему взаимосвязанных объектов в рамках парадигмы объектно-ориентированного программирования, разделение программного кода на отдельные модули и функции в рамках концепции модульности.

В рамках курсовой работы разрабатываются программная система для обработки текстовых данных, экспертная банковская система, аналитическая система для математических вычислений (калькулятор) и программа, визуализирующая решение одной из популярных головоломок прошлого (Ханойская башня). Часть разработанных программ работает в консольном режиме, другая часть снабжена пользовательским интерфейсом. Для разработки используется язык программирования Python [1,2,3,4,5,6], который широко применяется в настоящее время как в коммерческих проектах, так и при обучении программированию. Язык Python обладает элегантным синтаксисом, доступен для использования на большинстве современных платформ, имеет активное сообщество. Используемые в языке Python библиотеки, а также сам интерпретатор и поставляемые вместе с ним инструменты распространяются с открытым исходным кодом, так что любой желающий может разобраться, как все устроено, и при необходимости внести изменения с целью расширения функциональности или исправления дефектов.

1. Анализ заданий курсовой работы

Целью данной главы является анализ исходных заданий на курсовую работу, предоставленных в методических указаниях. В данной главе подробно расписана постановка задачи на курсовую с уточнением параметров, которые зависят от ID студента и его ФИО. Завершает данную главу анализ инструментов и библиотек, которые могут быть использованы для выполнения задания курсовой работы. Выполняется обоснование выбора подходящих инструментов и библиотек.

1.1. Исходные данные к заданиям курсовой работы

Курсовая работа состоит из четырех заданий:

1) В первом задании необходимо разработать программу, которая выполнит анализ входного текстового файла. Необходимо прочитать входной файл, разбить его на отдельные слова и по каждому слову посчитать, сколько раз оно встречается в тексте. В выходной файл построчно записываются слова с указанием количества раз, которое соответствующее слово встречается в тексте. Список должен быть отсортирован по убыванию количества появлений в тексте, а при одинаковой частоте появления – в лексикографическом порядке по возрастанию.

2) Во втором задании необходимо разработать диалоговую банковскую систему. Система хранит в памяти список клиентов банка, для каждого клиента известен текущий остаток средств на его счету. Первоначально в системе должен быть только один счет, открытый на фамилию студента, а остаток на этом счету должен быть равен ID. Это значит, что в системе должен быть открыт счет на фамилию Knyazev с остатком 70143770. Программа должна предоставлять пользовательский интерфейс, включающий два поля ввода (одно

для ввода команд и одно – для выходных данных), и две кнопки («Calculate» – для запуска обработки введенных команд и «Clear» – для очистки обоих полей ввода). Допускается ввод до 20 команд в левое поле ввода (то есть максимальное количество вводимых строк должно быть ограничено). Допускается ввод пустых строк в левое поле ввода: это означает, что никакая команда при обработке данной строки не должна выполняться. Предполагается, что пользователь не совершает ошибок при вводе команд как при вводе самих команд, так и при вводе имен пользователей и чисел. Перечень команд, которые должны обрабатываться системой и их подробное описание, приведены в таблице 1.

3) В третьем задании необходимо разработать программу-калькулятор. Калькулятор должен позволять вводить цифры, выполнять сложение, вычитание, умножение, деление, возведение в степень, извлечение квадратного корня, изменение знака. Должна быть предусмотрена кнопка сброса текущего значения в ноль, кнопка «равно» и возможность работы с памятью с помощью следующих кнопок:

- MS – «сохранить в памяти», записывает текущее значение на дисплее в память.
- MC – «очистить память», обнуляет все данные в памяти.
- MR – «прочитать память», отображает текущее значение в памяти на экране.
- M- – вычитает текущее значение в памяти из текущего значения на экране и результат записывает в память.
- M+ – прибавляет текущее значение в памяти к текущему значению на экране и результат записывает в память.

Должна быть предусмотрена возможность перехода калькулятора в расширенный режим, в котором доступны дополнительные функции.

В расширенном режиме должно быть доступно несколько ячеек памяти. Определим их количество согласно методическим рекомендациям:

2+9+8=19

1+9=10

1+0=1

Поскольку мы получили в ответе число 1, согласно методическим рекомендациям, количество ячеек памяти должно быть равно 2.

В расширенном режиме должен быть доступен цифровой дисплей, отображающий историю выполнения команд. Цифровой дисплей в одной строке отображает числа и выполняемые над ними операции, отдельной строкой отображается символ «=» и после него отдельной строкой идет результат вычисления выражения. Также отдельной строкой отображается символ «C», означающий сброс текущего выражения. Определим число строк цифрового дисплея согласно методическим рекомендациям:

7+0+1+5+0+2+9+8=32

3+2=5

Таблица 1. Команды банковской системы

Команда	**Описание**
DEPOSIT name sum	Зачислить сумму sum на счет клиента name. Если клиента с таким именем нет в системе, то он создается и на его счет заносится указанная сумма.
WITHDRAW name sum	Снять сумму sum со счета клиента name. Если клиента с таким именем нет, то создается новый счет с соответствующим отрицательным балансом.
BALANCE name	Вывести остаток средств на счету клиента name. Если клиента с именем name нет в системе, выводится сообщение «NO CLIENT». Если имя клиента не указано, то выводится баланс всех существующих клиентов.
TRANSFER name1 name2 sum	Перевести сумму sum со счета клиента name1 на счет клиента name2. Если какого-либо из клиентов (name1 или name2) нет в системе, то они создаются.
INCOME p	Начислить всем клиентам p% от суммы счета. Проценты начисляются только тем клиентам, остаток на счету которых положителен.

Поскольку мы получили в ответе число 5, число строк цифрового дисплея в расширенном режиме калькулятора должно быть равно 5.

Также в расширенном режиме калькулятора должны быть реализованы следующие функции: арксинус, арккосинус, арктангенс, логарифм по основанию, факториал.

Рекомендуемый вид окна калькулятора в стандартном режиме приведен на рисунке 1. Рекомендуемый вид окна калькулятора в расширенном режиме приведен на рисунке 2.

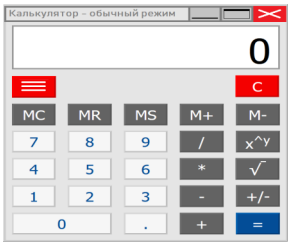


Рисунок 1 – Рекомендуемый внешний вид окна калькулятора в стандартном режиме

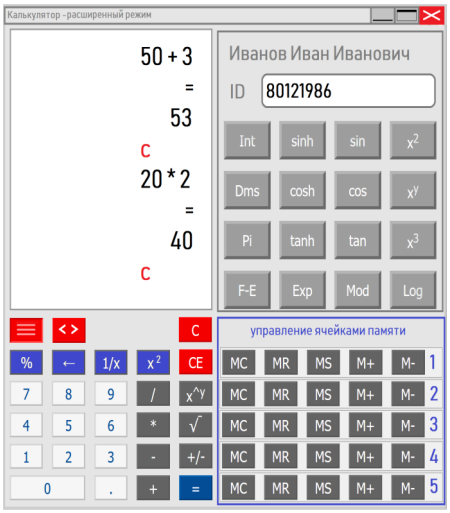


Рисунок 2 – Рекомендуемый внешний вид окна калькулятора в расширенном режиме

4) В четвертом задании необходимо разработать программу, которая будет демонстрировать решение популярной головоломки прошлого под названием Ханойская башня (в модифицированном виде). Имеется восемь шпинделей, пронумерованных от 8 до 1 слева направо. На шпинделях находятся диски, число дисков на шпинделе равно соответствующей по порядку цифре из ID студента. Все диски имеют разные диаметры: диаметр диска равен 10 * M + N, где M – номер шпинделя, N – номер диска на шпинделе. Программа должна визуализировать данную задачу, то есть изображать шпиндели и диски на них. Цвет дисков определяется случайным образом. При изображении дисков программа должна показывать диаметр диска в виде числа, изображенного поверх самого диска. Необходимо решить задачу о переносе всех дисков на шпиндель номер один. При решении задачи необходимо учитывать следующие правила:

- за одну итерацию можно переместить не более одного диска.
- разрешается класть только диск меньшего диаметра на диск большего диаметра.
- со шпинделя номер 8 можно перекладывать диски только на шпиндели номер 7 и 6.
- со шпинделя номер 1 можно перекладывать диски только на шпиндели номер 2 и 3.
- со шпинделей с 2 по 7 можно перекладывать диски только на два соседних шпинделя.

Необходимо отобразить начальное и конечное расположение дисков на шпинделях, для этого надо предусмотреть две кнопки «Начало» и «Конец». Под схемой расположения дисков на шпинделях должен выводиться номер итерации. Также необходимо предусмотреть четыре кнопки, переводящие схему расположения дисков на шпинделях в промежуточные итерации. Над этими кнопками располагаются поля ввода, в которых задается процент завершения исходной задачи на целевой итерации. Первоначально поля ввода заполняются двузначными числами, взятыми из ID студента, разбитого на четыре числа по

две цифры в каждом. При щелчке по соответствующей кнопке вычисляется итерация, на которой процент решения задачи соответствует введенному значению, полученная итерация отображается на схеме расположения дисков на шпинделях и в подписи под схемой. Если по какому-то проценту решения задачи получается дробная итерация, то необходимо отобразить номер итерации с округлением до трех цифр после запятой и переносимый на итерации диск изобразить в воздухе между соответствующими шпинделями.

1.2. Выбор и обоснование необходимых библиотек

Согласно требованиям методических рекомендаций, для разработки необходимо использовать язык Python и среду разработки PyCharm, однако в части библиотеки для отображения интерфейса пользователя допускается свободный выбор подходящего варианта. Рассмотрим существующие в настоящее время библиотеки для создания пользовательского интерфейса с использованием языка Python.

Библиотека PyQt5 [7] является оберткой над кроссплатформенной библиотекой Qt, которая используется в первую очередь для разработки пользовательских интерфейсов на различных платформах, включая Windows, Linux, MacOS, Android и iOS. Библиотека распространяется под лицензией GPL, либо под коммерческой лицензией. Лицензия GPL требует, чтобы программное обеспечение, разработанное с помощью этой библиотеки, также распространялось под лицензией GPL, требует распространять программу вместе с исходным кодом, а также запрещает использование проприетарных библиотек в составе разрабатываемой программы. Для визуального редактирования форм и размещения на них элементов управления можно использовать стандартный дизайнер, который поставляется вместе с Qt. Библиотека PyQt5 активно обновляется.

Библиотека Tkinter является стандартной встроенной в Python библиотекой, предназначенной в основном для разработки пользовательского интерфейса. Это значит, что при разработке не потребуется установка дополнительного программного обеспечения для разработки с использованием данной библиотеки, что является несомненным плюсом. К недостаткам библиотеки относятся нестандартный для операционной системы внешний вид приложения (может отличаться от других программ), бедный набор виджетов, компоновка элементов управления при переносе на другую платформу может «поехать». Можно сделать такой вывод, что библиотека Tkinter предназначена для нетяжеловесных учебных проектов.

Библиотека wxPython [8] является оберткой над кроссплатформенной библиотекой wxWidgets, разработанной для C++. В отличие от Tkinter, виджеты, предоставляемые библиотекой wxPython, будут выглядеть «по-родному» для соответствующей операционной системы, на которой выполняется запуск программы. Набор виджетов, предоставляемых библиотекой wxPython, достаточно широк и, в частности, он более многообразен по сравнению с набором виджетов, предоставляемых библиотекой Tkinter. Популярное клиентское приложение «Диск Goggle» разработано с использованием библиотеки wxPython. В отличие от Tkinter, библиотека wxPython требует отдельной установки.

Библиотека PySimpleGUI [9,10] является простой библиотекой для разработки пользовательского интерфейса на языке Python. Внутри для создания интерфейсов могут использоваться Qt, wxWidgets, HTML или даже Tkinter. Библиотека может работать на большом количестве рахных операционных систем, включая Windows, Linux и MacOS. Библиотека требует отдельной установки, аналогично большинству остальных рассматриваемых библиотек. Библиотека активно развивается, последнее обновление вышло 21.03.2021.

Для использования в данной курсовой работе выбрана библиотека PyQt5, потому что в сети интернет доступно большое количество материалов по

данной библиотеке и используемому ей фреймворку Qt. Еще одним плюсом в пользу данной библиотеки является наличие визуального редактора, позволяющего отредактировать форму приложения в визуальном режиме.

1.3. Выводы по первой главе

В рамках данной главы были подробно проанализированы задания на курсовую работу, были уточнены задания, которые зависят от фамилии и ID студента. Также был произведен выбор библиотеки для разработки пользовательского интерфейса.

2. Разработка программного продукта для решения прикладных задач

Целью данной главы курсовой работы является проектирование алгоритмов, структур данных и структур классов для решения прикладных задач, которые были поставлены в предыдущей главе.

2.1. Работа с наборами данных

В данном разделе рассматривается решение первой задачи курсовой работы, связанной с обработкой текстового файла.

2.1.1. Построение алгоритма решения задания без графического интерфейса

Блок-схема алгоритма решения первого задания курсовой работы представлена на рисунке 3.

Текстовое описание алгоритма решения первого задания курсовой работы представлено ниже:

1. Ввод строки line.
2. Разбить строку line на слова words.
3. Инициализировать список word_infos пустым значением.
4. Если еще не все слова words обработаны, то перейти к шагу 5, иначе перейти к шагу 12.
5. Выбрать очередное слово word из списка слов words для обработки.
6. Найти такое описание word_info в списке word_infos, которое соответствует слову word.

7. Если word_info найден, то перейти к шагу 8, иначе перейти к шагу 10.
8. Увеличить число вхождений слова word в текст на 1.
9. Перейти к шагу 11.
10. Добавить в список word_infos новое слово word с числом вхождений 1.
11. Перейти к шагу 4.
12. Отсортировать список word_infos.
13. Вывести список word_infos.

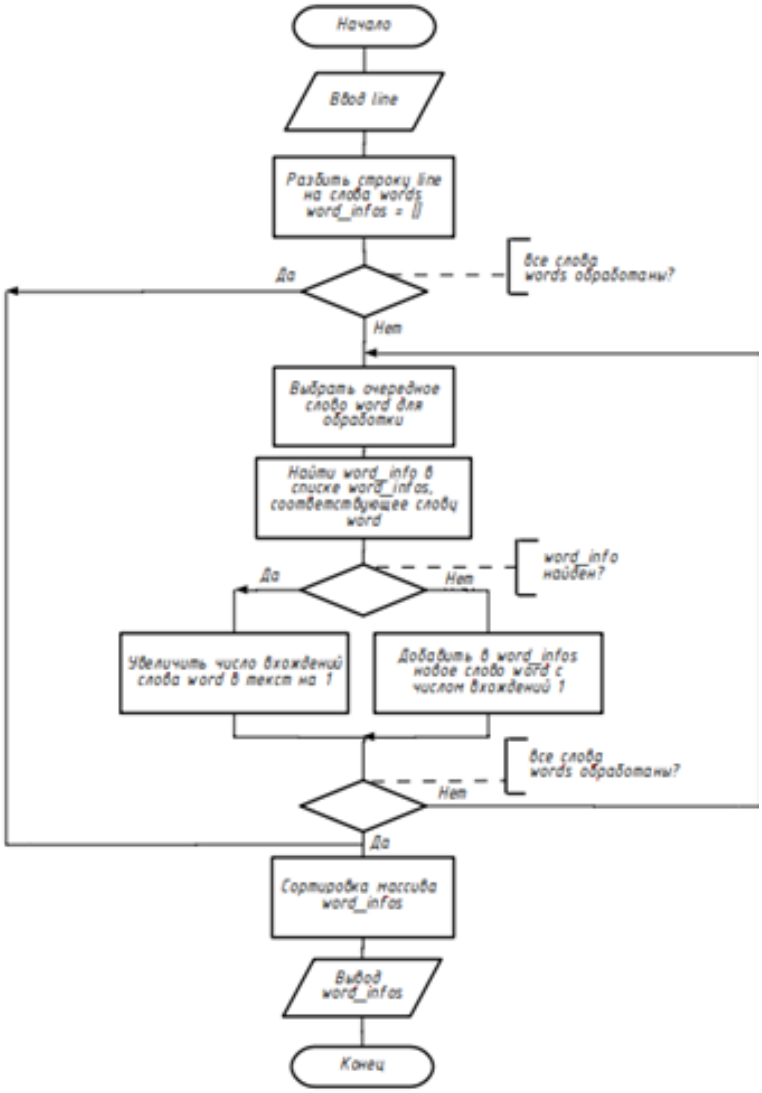


Рисунок 3 – Блок-схема алгоритма решения первого задания курсовой работы

2.1.2. Разработка программной реализации на языке программирования

Для разбивки входной строки на массив слов используется функция split. Данная функция хорошо выполняет свою задачу, но при этом в строках выделенных слов могут содержаться спецсимволы. Спецсимволами считаются, например, знаки препинания. Для того чтобы удалить из выделенных слов спецсимволы, используется замена с помощью регулярных выражений. Все символы, из которых могут состоять слова, соответствуют в регулярных выражениях специальной последовательности \w. Замена во всех словах осуществляется с помощью функции map, что демонстрирует применение методов функционального программирования.

Массив word_infos содержит кортежи, как и указано в методических рекомендациях. При этом первым элементом кортежа является количество повторений слова в тексте, а вторым – само это слово. Сортировка массива кортежей word_infos выполняется с использованием функции-компаратора. Как и указано в методических рекомендациях, в первую очередь сортировка идет по количеству вхождений слова в текст, а во вторую – по алфавитному порядку слов (если число вхождений в текст одинаково).

Текст программы приведен в приложении А.

2.1.3. Тестирование и отладка

Первый контрольный пример соответствует данным из методических рекомендаций:

```
hi
hi
what is your name
my name is bond
```

james bond

my name is damme

van damme

claude van damme

jean claude van damme

Выходные данные для этого текста:

damme 4

is 3

name 3

van 3

bond 2

claude 2

hi 2

my 2

james 1

jean 1

what 1

2.1.4. Формирование выходных файлов

На эталонных входных данных подготовлен выходной файл result_1.txt со следующим содержимым:

damme 4

is 3

name 3

van 3

bond 2

claude 2

hi 2

my 2

james 1

jean 1

what 1
8
your 1

2.2. Разработка экспертной системы

3

В данном разделе рассматривается решение второго задания курсовой работы.

2.2.1. Построение алгоритма решения задания с пользовательским интерфейсом

Блок-схема алгоритма работы программы с решением второго задания курсовой работы представлена на рисунке 4. Обработка отдельных команд на рисунке не детализована, так как алгоритмы обработки этих команд достаточно тривиальны. Текстовое описание алгоритма работы программы представлено ниже:

1. Разбить команду command на отдельные слова в массив.
2. Если команда command начинается со строки DEPOSIT, то обработать команду DEPOSIT, иначе перейти к шагу 3.
3. Если команда command начинается со строки WITHDRAW, то обработать команду WITHDRAW, иначе перейти к шагу 4.
4. Если команда command начинается со строки BALANCE, то обработать команду BALANCE, иначе перейти к шагу 5.
5. Если команда command начинается со строки TRANSFER, то обработать команду TRANSFER, иначе перейти к шагу 6.
6. Если команда command начинается со строки INCOME, то обработать команду INCOME.

Текстовое описание алгоритма работы метода DEPOSIT представлено ниже:

1. Присвоить в переменную account_found значение False.

2. Если еще есть счета для обработки в списке self.accounts, то перейти к шагу 3, иначе перейти к шагу 8.

3. Записать в переменную account очередной счет для обработки.

4. Если account.name совпадает с именем держателя счета для обработки, то перейти к шагу 5, иначе перейти к шагу 2.

5. Вывести информацию о счете в лог.

6. account.balance += sum.

7. account_found = True.

8. Если account_found истинно, то перейти к шагу 9, иначе закончить.

9. Вывести информацию о добавлении нового счета в лог.

10. self.accounts.append(account).

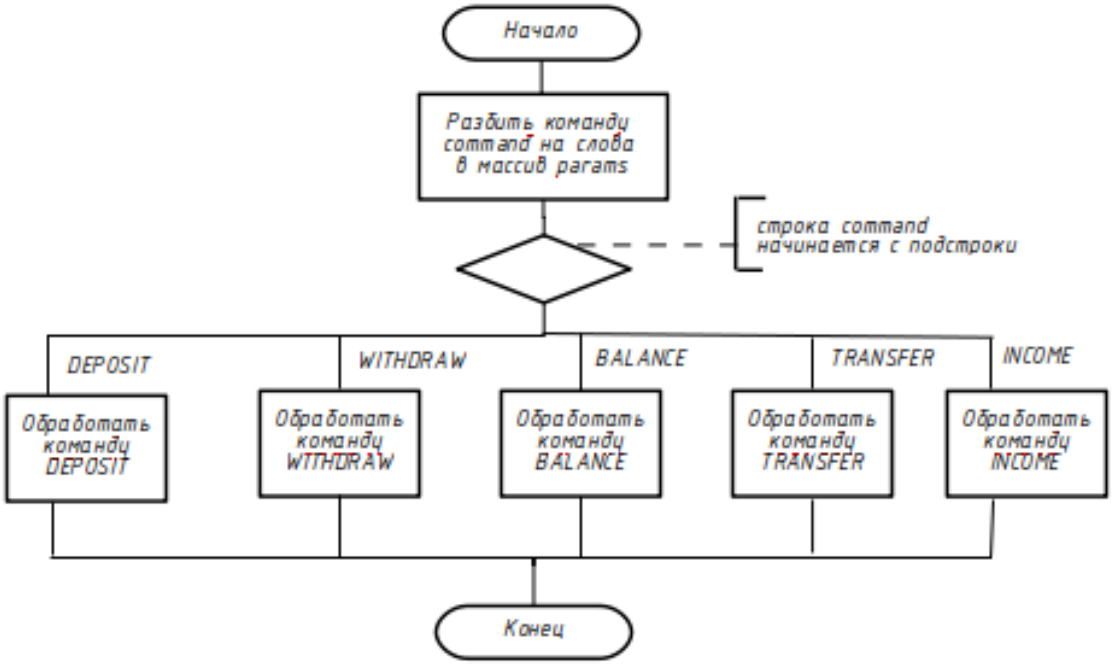


Рисунок 4 – Блок-схема алгоритма решения второго задания курсовой работы

Блок-схема метода DEPOSIT представлена на рисунке 5.

Блок-схема алгоритма обработки команды WITHDRAW представлена на рисунке 6.

Текстовое описание алгоритма метода WITHDRAW представлено ниже:

1. Присвоить в переменную account_found значение False.

2. Если еще есть счета для обработки в списке self.accounts, то перейти к шагу 3, иначе перейти к шагу 8.
3. Записать в переменную account очередной счет для обработки.
4. Если account.name совпадает с именем держателя счета для обработки, то перейти к шагу 5, иначе перейти к шагу 2.
5. Вывести информацию о счете в лог.
6. account.balance -= sum.
7. account_found = True.
8. Если account_found истинно, то перейти к шагу 9, иначе закончить.
9. Вывести информацию о добавлении нового счета в лог.
10. self.accounts.append(account).

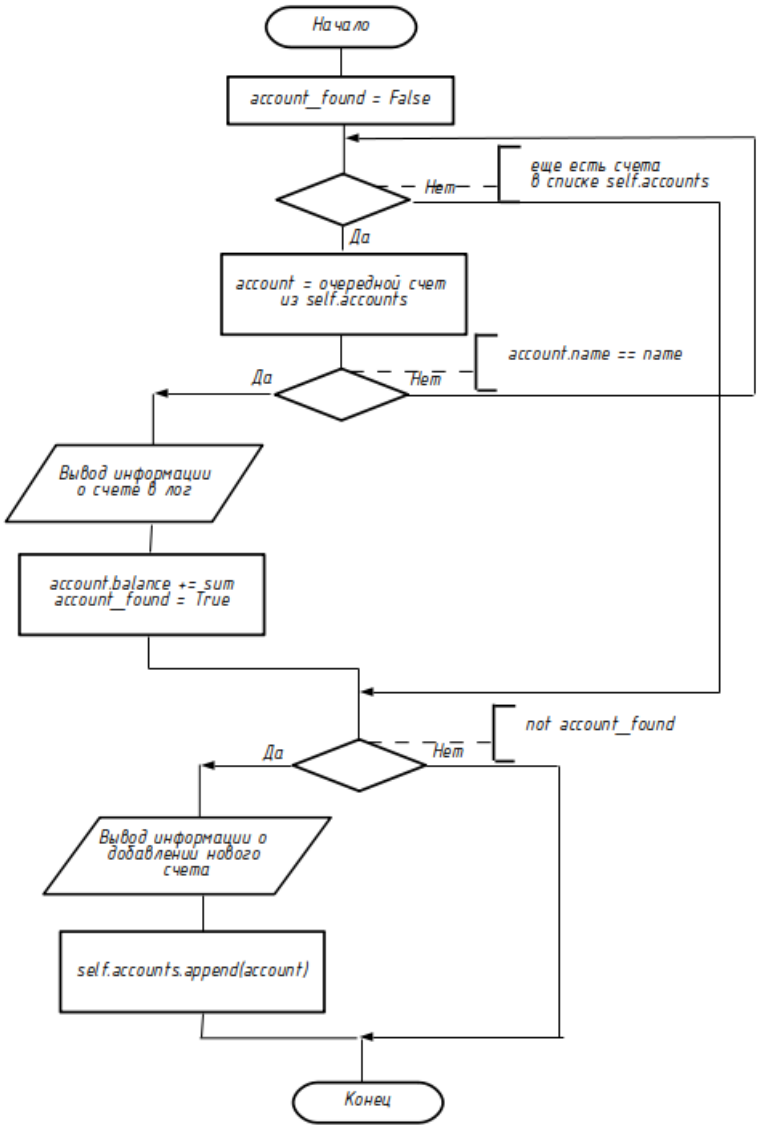


Рисунок 5 – Блок-схема метода DEPOSIT

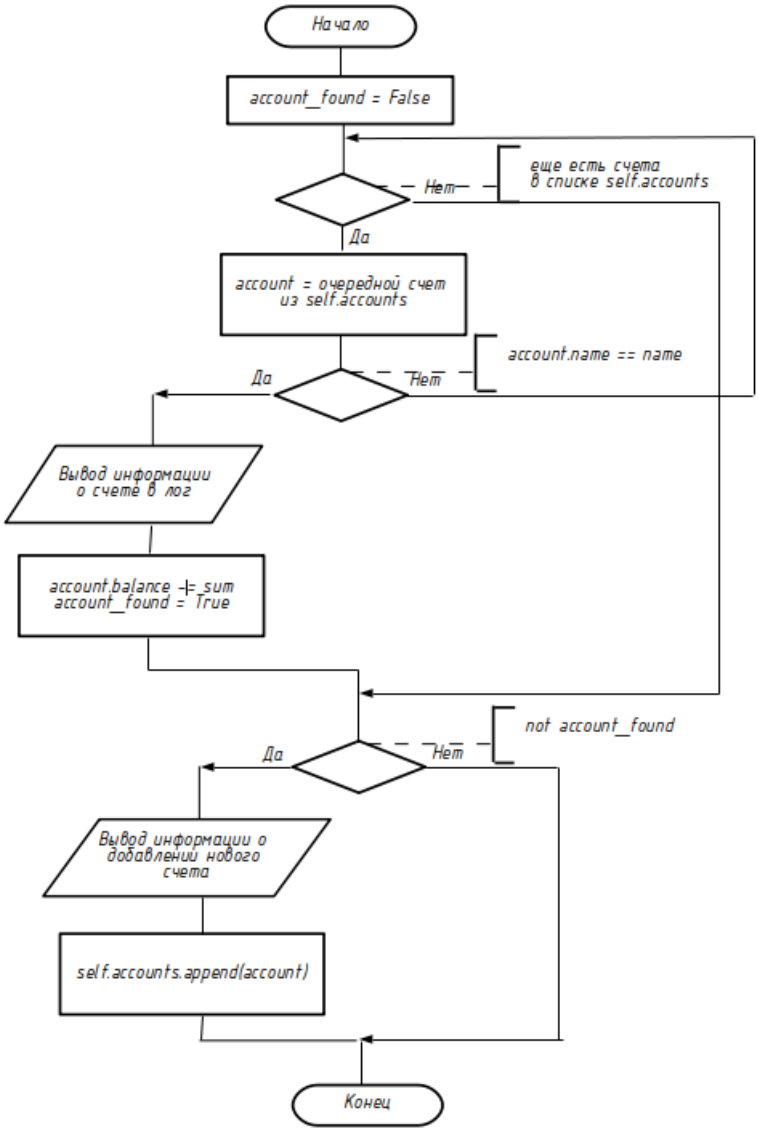


Рисунок 6 – Блок-схема метода WITHDRAW

2.2.2. Разработка программной реализации на языке программирования и с использованием дополнительных библиотек

Программа разработана с использованием объектно-ориентированного подхода. В прогремме используются следующие классы:

- Account — класс, представляющий счет, содержит ФИО клиента и его текущий остаток
- Bank — класс, представляющий банк, содержит список клиентов и предоставляет основные методы по работе со счетами

- CommandEngine — класс, предназначенный для разбора команд, введенных в поле ввода
- BankWindow — класс главного окна приложения

Текст программы приведен в приложении А.

2.2.3. Тестирование и отладка

До создания каких-либо дополнительных счетов введем команду запроса баланса:

BALANCE

В окне вывода получитм единственный счет, зачисленный на имя автора курсовой работы:

BALANCE: найден клиент Knyazev, на его счету 70143770

Заведем несколько новых счетов (в том числе с отрицательным балансом):

DEPOSIT Мурынов 700

DEPOSIT Аверкиев 400

DEPOSIT Дружкин -200

В окне вывода получим информацию орезультатах создания счетов:

DEPOSIT: добавлен новый клиент Мурынов, на его счете 700

DEPOSIT: добавлен новый клиент Аверкиев, на его счете 400

DEPOSIT: добавлен новый клиент Дружкин, на его счете -200

2.2.4. Формирование выходных файлов

Программа не формирует выходные файлы, все взаимодействие с пользователем осуществляется с помощью пользовательского интерфейса. Внешний вид интерфейса программы приведен в приложении Б.

2.3. Разработка аналитической системы

В данном разделе рассматривается решение третьего задания курсовой работы.

2.3.1. Построение алгоритма решения задания с пользовательским интерфейсом

Операции, выполняемые калькулятором, подразделяются на унарные и бинарные. Унарные операции вычисляются сразу «на месте». А для вычисления бинарных операций используется запоминание предыдущего операнда и операции. Вычисление бинарной операции выполняется при нажатии на кнопку «=». Блок-схема алгоритма работы кнопки «=» (наиболее сложного из общего перечня функция калькулятора) представлена на рисунке 7.

Текстовое описание алгоритма обработки нажатия кнопки «равно» представлено ниже:

1. Если текущая операция означена, то перейти к шагу 2, иначе перейти к шагу 3.
2. Сохранить в истории текущую строку дисплея и строку «=».
3. is_error = False.
4. result = «nop».
5. Выполнить текущую операцию (+, -, *, /, log).
6. Если result != «nop», то перейти к шагу 7, иначе перейти к шагу 11.
7. Если result целочисленный, то перейти к шагу 8, иначе перейти к шагу 10.
8. Запомнить текущим операндом str(int(result)).
9. Перейти к шагу 11.
10. Запомнить текущим операндом str(result).
11. operation = «».
12. Если is_error, то перейти к шагу 13, иначе перейти к шагу 15.
13. Подготовить к выводу ошибку.
14. Закончить.

15. Подготовить к выводу текущий операнд.

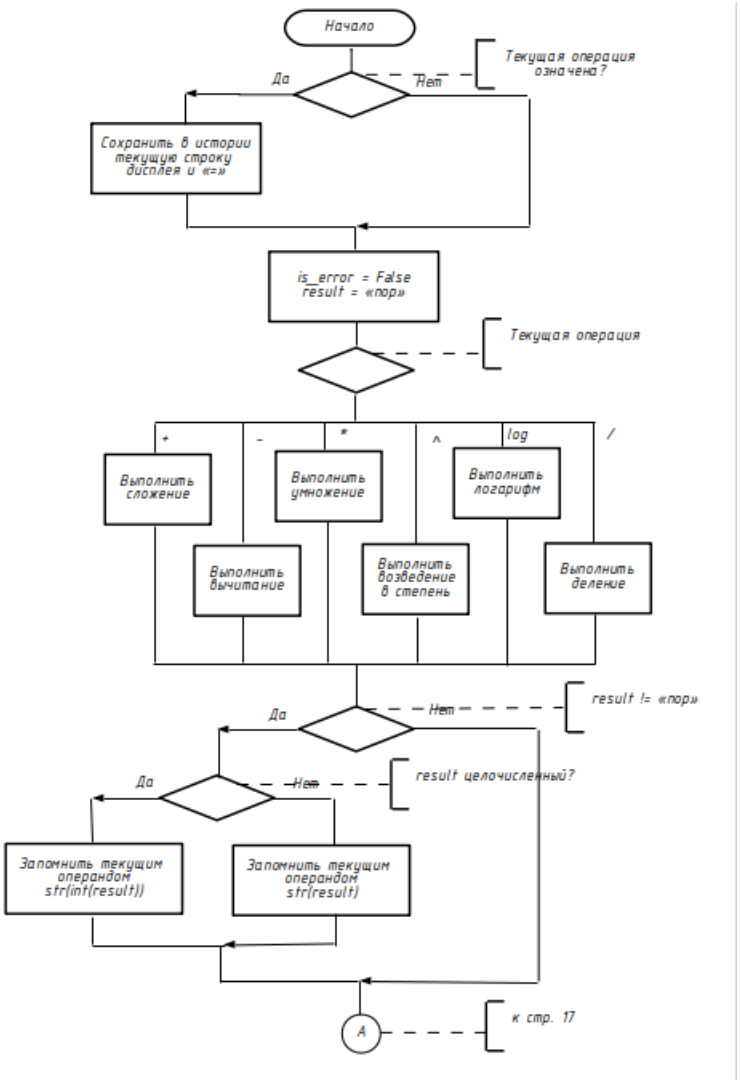


Рисунок 7 – Блок-схема алгоритма обработки нажатия кнопки «равно»

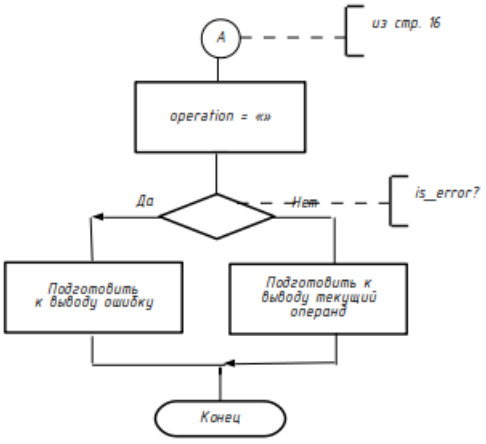


Рисунок 7 – Продолжение

Блок-схема алгоритма рекурсивного вычисления суммы цифр числа представлена на рисунке 8.

Текстовое описание алгоритма вычисления суммы цифр числа представлено ниже:

1. Если «.» содержится в self.current_operand, то перейти к шагу 2, иначе перейти к шагу 6.
2. self.display = «error»
3. self.current_operand = «0»
4. current_number_string = self.current_operand
5. Закончить.
6. current_result = 0
7. Если еще есть символы в current_number_string, то перейти к шагу 8, иначе перейти к шагу 11.
8. character = очередной символ.
9. current_result += int(character).
10. Перейти к шагу 6.
11. Если current_result < 10, то перейти к шагу 12, иначе перейти к шагу 15.
12. self.current_operand = str(current_result).
13. self.display = self.current_operand.
14. Закончить.
15. current_number_string = str(current_result).

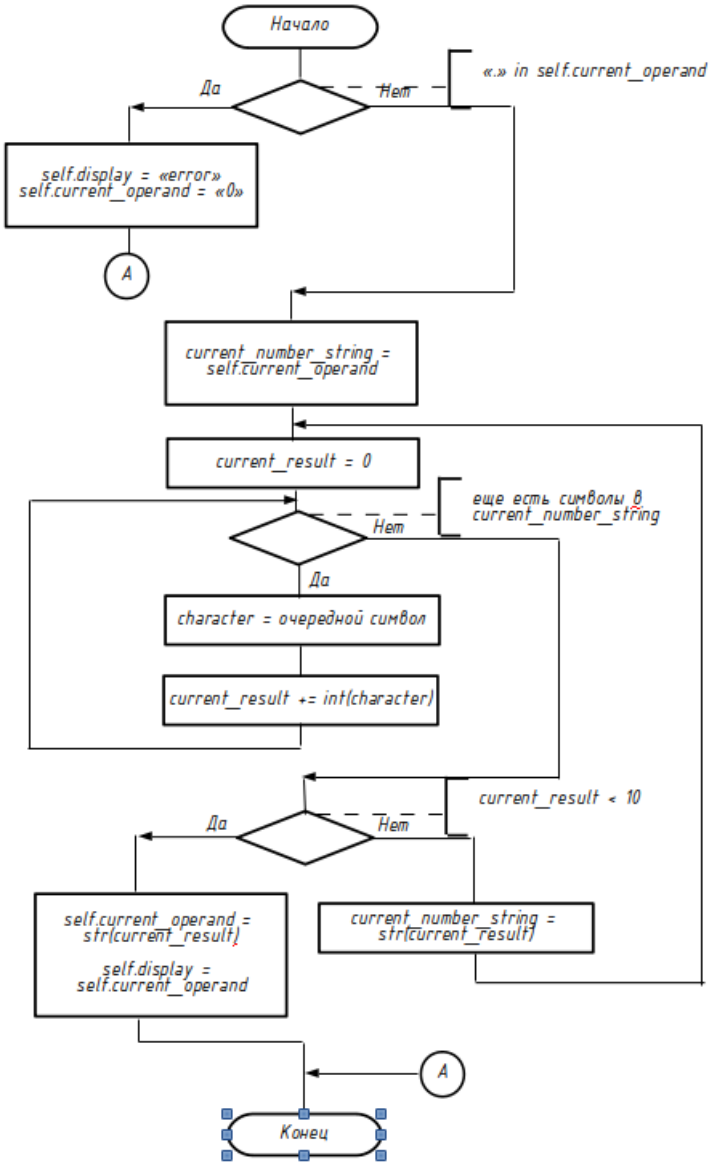


Рисунок 8 - Блок-схема алгоритма рекурсивного вычисления суммы цифр числа

2.3.2. Разработка программной реализации на языке программирования

Внешний вид двух окон программы-калькулятора разработан с использованием инструмента Qt Designer (то есть путем графического перетаскивания виджетов). На выходе инструмента Qt Designer получается файл с расширением ui. Этот файл преобразуется в код на языке Python с использованием утилиты pyuic5.

Программа разработана с использованием объектно-ориентированного подхода: модель калькулятора отделена от реализации на уровне библиотеки для работы с пользовательским интерфейсом. Сам класс, реализующий бизнес-логику калькулятора, называется Calculator. Класс окна программы, реализующего стандартный режим работы калькулятора, называется CalculatorStandardModeWindow. Класс окна программы, реализующего расширенный режим работы калькулятора, называется CalculatorExtendedModeWindow.

При переходе между стандартным и расширенным режимами работы калькулятора окно калькулятора с предыдущим режимом работы скрывается и на его месте показывается окно калькулятора с новым режимом работы. Положение нового окна на экране копируется из положения старого окна на экране.

Текст программы приведен в приложении А.

2.3.3. Тестирование и отладка

Составим перечень тестовых ситуаций:

1) Стандартный режим работы калькулятора:

- ввод числа с помощью кнопок-цифр и точки
- унарные операции: плюс/минус и извлечение квадратного корня
- бинарные операции: плюс, минус, умножить, делить, возведение в степень
- цепочки бинарных операций (без промежуточного нажатия на кнопку равно)
- работа с памятью

2) Расширенный режим работы калькулятора:

- ввод числа с помощью кнопок-цифр и точки

- унарные операции: плюс/минус, квадратный корень, арксинус, арккосинус, арктангенс, факториал, сумма цифр числа
- бинарные операции: плюс, минус, умножить, делить, возведение в степень, логарифм
- работа с памятью
- кнопка «равно» и кнопка «сброс»

2.3.4. Формирование выходных файлов

Программа не формирует выходных файлов, все взаимодействие с программой осуществляется через интерфейс пользователя. Внешний вид интерфейса пользователя для программы демонстрируется в приложении Б.

2.4. Разработка логико-аналитической системы

В данном разделе рассматривается решение четвертого задания курсовой работы.

2.4.1. Построение алгоритма решения задания «Ханойские башни»

Для решения задачи «Ханойские башни» с числом шпинделей более трех существует алгоритм Фрейма-Стюарта, однако в настоящее время неизвестно точно, является он оптимальным или нет. Поскольку решение задачи с помощью полного перебора (поиск в ширину) занимает достаточно длительное время, было решено взять за основу алгоритм Фрейма-Стюарта, при этом был сделан ряд допущений с учетом известных начальных условий задачи. Поскольку два шпинделя из восьми совсем не содержат дисков, их можно использовать для промежуточных перекладываний дисков, разбивая большую

исходную задачу с восемью шпинделями на более мелкие задачи с тремя шпинделями. Первым этапом с помощью свободных шпинделей освбождается восьмой шпиндель. Затем поочередно перекладываются диски, начиная с наибольшего диаметра и заканчивая наименьшим. На каждом этапе рассматривается меньшая задача из трех шпинделей. Алгоритм решения задачи с тремя шпинделями представлен на рисунке 9.

Текстовое описание алгоритма решения задачи «Ханойские башни» с тремя шпинделями представлено ниже:

1. Если disk_count = 1, то перейти к шагу 2, иначе перейти к шагу 4.
2. Перенести элемент из колонки source в колонку destination.
3. Закончить.
4. Сделать рекурсивный вызов решения задачи для переноса disk_count – 1 дисков с колонки source на колонку auxiliary, используя в качестве вспомогательной колонки колонку destination.
5. Перенести элемент из колонки source в колонку destination.
6. Сделать рекурсивный вызов решения задачи для переноса disk_count – 1 дисков с колонки auxiliary на колонку destination, используя в качестве вспомогательной колонку source.
7.

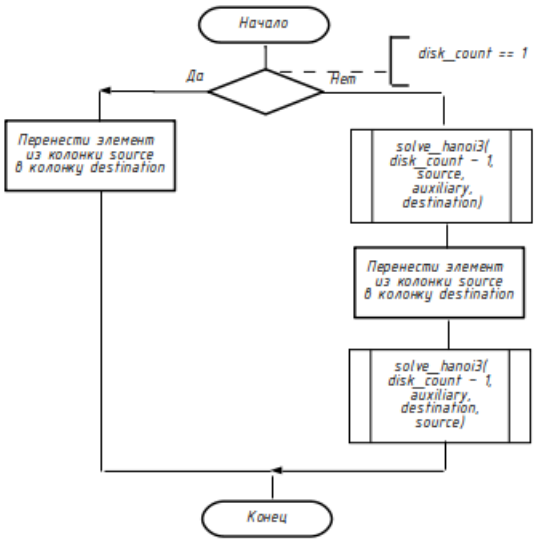


Рисунок 9 – Блок-схема алгоритма решения задачи «Ханойские башни» с тремя

Алгоритм отрисовки текущей итерации решения задачи «Ханойские башни» заключается в последовательной отрисовке восьми шпинделей, дисков на них с подписями и поверхности, на которой находятся шпиндели с дисками. Если текущая итерация является дробной (то есть имеется диск между шпинделями), то рисуется также диск между шпинделями. Алгоритм отрисовки текущей итерации решения задачи «Ханойские башни» представлен на рисунке 10.

Текстовое описание алгоритма отрисовки текущей итерации представлено ниже:

1. column_index = 0.
2. Если column_index <= 7, перейти к шагу 3, иначе перейти к шагу 9.
3. Нарисовать шпиндель.
4. column_index += 1.
5. Если еще есть диски в колонке column_index, то перейти к шагу 6, иначе перейти к шагу 8.
6. Нарисовать текущий диск.
7. Перейти к шагу 5.
8. Перейти к шагу 2.
9. Если есть диск между шпинделями, перейти к шагу 10, иначе перейти к шагу 11.
10. Нарисовать диск между шпинделями.
11. Нарисовать поверхность, на которой находятся шпиндели.

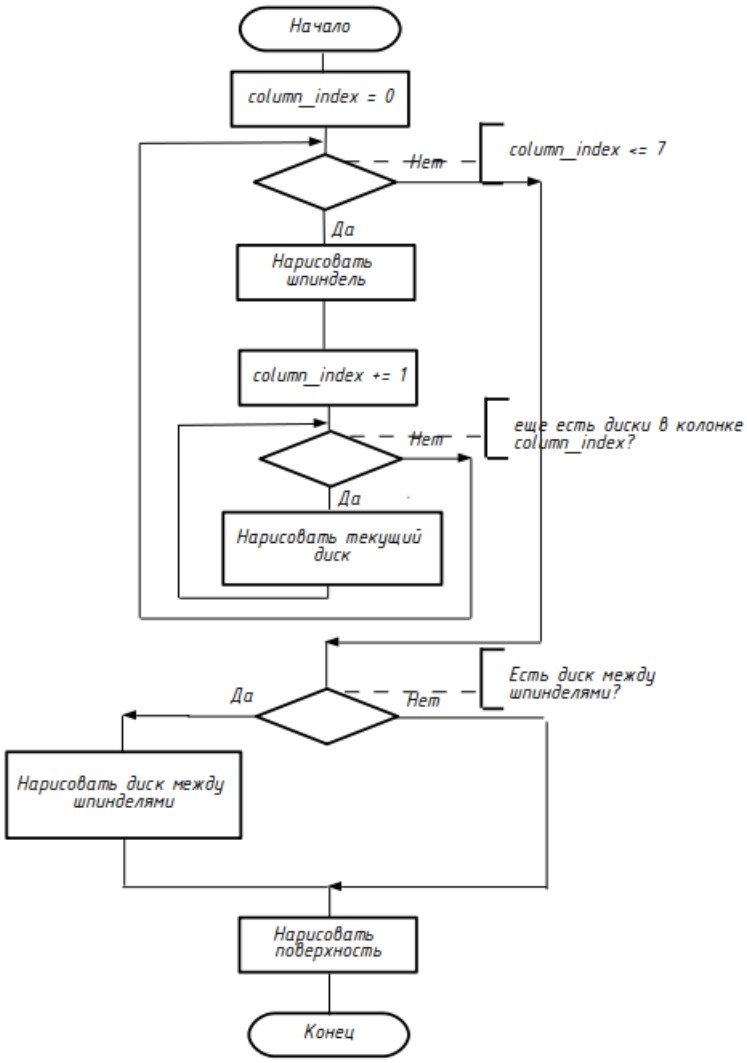


Рисунок 10 – Блок-схема алгоритма отрисовки текущей итерации решения задачи «Ханойские башни»

2.4.2. Разработка программной реализации

Внешний вид окна программы для решения задачи «Ханойские башни» разработан с использованием инструмента Qt Designer (то есть путем графического перетаскивания виджетов). На выходе инструмента Qt Designer получается файл с расширением ui. Этот файл преобразуется в код на языке Python с использованием утилиты pyuic5.

При разработке программы использован объектно-ориентированный подход. Для каждой итерации решения задачи «Ханойские башни» создается экземпляр класса HanoiIteration. Главное окно программы представлено классом HanoiMainWindow.

Текст программы приведен в приложении А.

2.4.3. Тестирование и отладка

На рисунке 11 представлен внешний вид окна программы после щелчка по кнопке «Начало». В этом окне отображено начальное состояние решения задачи, когда диски расположены на шпинделях согласно ID студента.

На рисунке 12 представлен внешний вид окна программы после щелчка по кнопке «Окончание». В этом окне отображно конечное состояние решения задачи, когда все диски собраны на последнем шпинделе.

На рисунке 13 представлен внешний вид окна программы после щелчка по кнопке «П.1», когда введено 70%.

2.4.4. Формирование выходных файлов

Программа не формирует выходные файлы, все взаимодействие пользователя с программой осуществляется посредством пользовательского интерфейса. Примеры внешнего вида окна программы приведены в приложении Б.

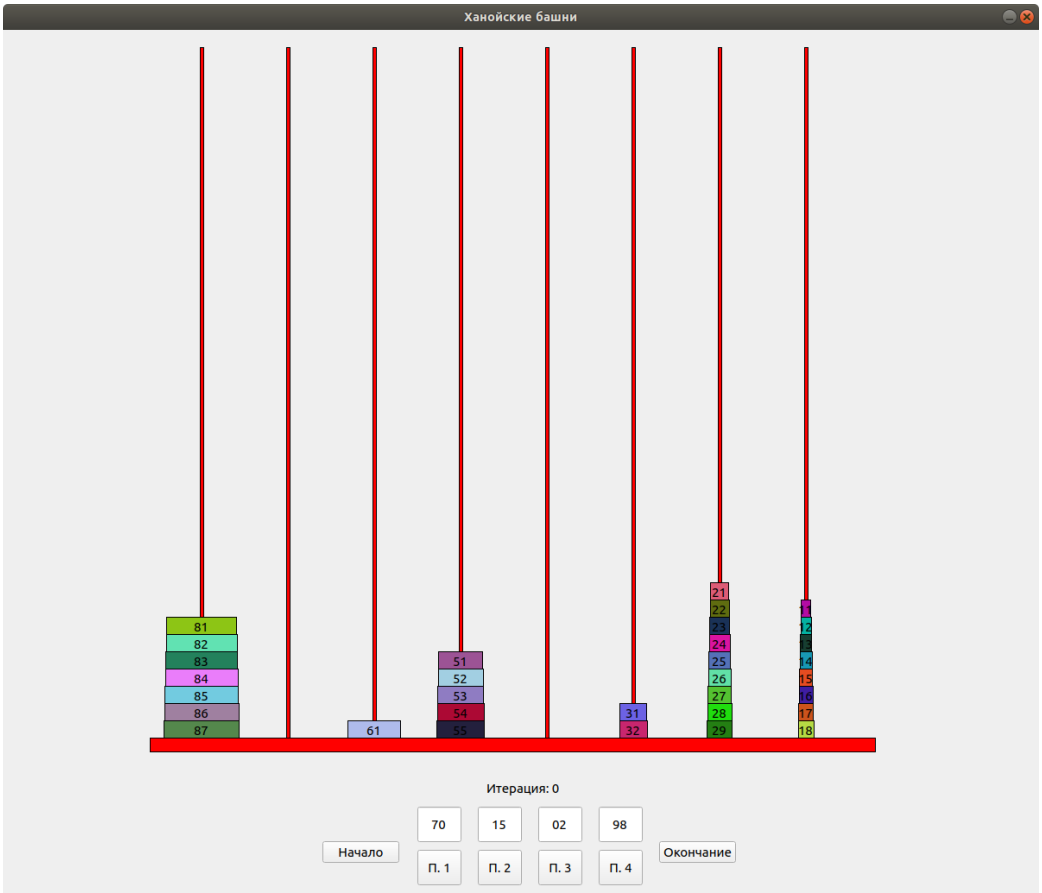


Рисунок 11 – Внешний вид окна программы после щелчка по кнопке «Начало»

2.5. Выводы по второй главе

Таким образом с помощью языка программирования Python и графической библиотеки PyQt5 были разработаны четыре программы с решением заданий курсовой работы. Благодаря выполнению этих заданий, удалось на практике убедиться в том, что язык Python обладает большим активным сообществом, вопросы по использованию языка находят свое решение в соответствующих бесплатных интернет-ресурсах. В рамках данной курсовой работы удалось освоить навыки объектно-ориентированного и функционального программирования.

В первом задании курсовой работы была разработана программа для обработки текстового файла с использованием стандартных библиотек языка

программирования Python. Во втором задании курсовой работы была разработана диалоговая банковская система. Для разработки пользовательского интерфейса использовалась библиотека PyQt5. В третьем задании курсовой работы была разработана программа-калькулятор. В четвертом задании курсовой работы была разработана программа для решения задачи «Ханойские башни». Сами тексты программ приведены в приложении А.

3. Разработка требований к техническим средствам реализации программного обеспечения для решения прикладных задач

Для работы разработанных программ требуется операционная система Windows (32-битная или 64-битная), MacOS (64-битная) или Linux (64-битная). Для запуска программ в операционной системе должен быть установлен интерпретатор Python 3 и библиотека PyQt5 (конкретный порядок установки зависит от используемой операционной системы). Библиотека PyQt5 в свою очередь требует установленной библиотеки Qt5 (конкретный порядок установки также зависит от используемой операционной системы).

Что касается аппаратных средств, то рекомендуется использовать Intel-совместимый процессор с тактовой частотой не ниже 1 ГГц. Рекомендуемый объем оперативной памяти не менее 4 Гб.

Выводы

Таким образом в рамках данной курсовой работы были разработаны четыре программы. Первая программа выполняет обработку текстового файла, подсчитывая, сколько раз в тексте встречается каждое слово. Вторая программа представляет из себя диалоговую банковскую систему, предназначенную для создания вкладов, переводов средств между счетами, начисления процентов и т.д. Третья программа представляет из себя программу-калькулятор, поддерживающую стандартный и расширенный режимы работы. Четвертая программа представляет из себя решение головоломки «Ханойская башня» с восемью шпинделями. В результате автору удалось освоить навыки программирования на языке программирования Python, познакомиться с библиотекой PyQt5 для разработки пользовательского интерфейса.

Список литературы

1. Введение в Python. Язык программирования Python: [Электронный ресурс]. URL: https://metanit.com/python/tutorial/1.1.php. (Дата обращения: 11.05.2021)

2. Lutz M. Learning Python, Fifth Edition. – Sebastopol: O'Reilly Media, 2013. – 1540 p.

3. Россум Г. Язык программирования Python [Электронный ресурс] / Россум Г., Дрейк Ф.Л.Дж., Откидач Д.С. – URL: http://prosmart.by/programimng/895-yazyk-programmirovaniya-python-rossum-g-drejk.html. (Дата обращения: 11.05.2021)

4. Ромальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс, 2016. – 768 с.

5. Седжевик Р. Программирование на языке Python: учебный курс. / Седжевик Р., Уэйн К., Дондеро Р. – СПб.: ООО «Альфа-книга», 2017. – 736 с.

6. Официальный сайт Python [Электронный ресурс]. URL: http://www.python.org. (Дата обращения: 11.05.2021)

7. Официальная страница фреймворка PyQt [Электронный ресурс]. URL: https://riverbankcomputing.com/software/pyqt/intro. (Дата обращения: 11.05.2021)

8. Официальная страница фреймворка wxPython [Электронный ресурс]. URL: https://www.wxpython.org. (Дата обращения: 11.05.2021)

9. Официальная страница библиотеки PySimpleGUI на GitHub [Электронный ресурс]. URL: https://github.com/PySimpleGUI/PySimpleGUI. (Дата обращения: 11.05.2021)

10. Официальная страница библиотеки PySimpleGUI [Электронный ресурс]. URL: https://pysimplegui.readthedocs.io. (Дата обращения: 11.05.2021)

Приложение А. Листинг текстов заданий

А.1. Текст программы exercise_1.py

```
# -*- coding: utf-8 -*-
# Назначение: программа по входному файлу выводит список использованных в нем слов и по каждому слову - сколько раз оно встречается в тексте.
# Входные данные: файл resource_1.txt в одном каталоге с данной программой.
# Результаты: список слов, использованных во входном файле, и по каждому слову - сколько раз оно встречается в тексте.

import functools
import re

# Индекс числа использований слова в кортеже.
COUNT_VALUE_INDEX = 0
# Индекс слова в кортеже.
WORD_VALUE_INDEX = 1

# Назначение: функция сравнения элементов индекса, содержащего слова и сколько раз они встречаются в исходном тексте.
# Входные данные:
#  left_operand - левый операнд операции сравнения (кортеж)
#  right_operand - правый операнд операции сравнения (кортеж)
# Результаты: 1, если левый операнд больше, 0, если операнды равны, -1, если правый операнд больше.
# Дата написания: 26.04.2021.
def word_info_comparator(left_operand, right_operand):
    if left_operand[COUNT_VALUE_INDEX] > right_operand[COUNT_VALUE_INDEX]:
        return -1
    elif left_operand[COUNT_VALUE_INDEX] < right_operand[COUNT_VALUE_INDEX]:
        return 1
    else:
        if left_operand[WORD_VALUE_INDEX] > right_operand[WORD_VALUE_INDEX]:
            return 1
        elif left_operand[WORD_VALUE_INDEX] < right_operand[WORD_VALUE_INDEX]:
            return -1
        else:
            return 0
```

```
with open("resource_1.txt", "r", encoding="utf-8") as f:
    word_infos = []
        for  line  in  f:
                # Разбить входную строку на слова и удалить ведущие и ведомые пробелы.
        words = line.strip().split()
        # Удалить из строки символы, из которых не могут состоять слова (в частности, спецсимволы).
                words = map(lambda line: re.sub(r'[^\w]', '', line), words)
                # Удалить из списка полностью пустые слова.
                words = list(filter(None, words))
                for word in words:
            # Если слово уже есть в индексе, то увеличить счетчик числа использований.
                        word_found = False
            word_info_index = 0
                        for word_info in word_infos:
                                if word_info[WORD_VALUE_INDEX] == word:
                    word_found = True
                                        break
                word_info_index += 1
                        # Если слова в индексе еще нет, то добавить его в индекс со значением счетчика 1.
                        if word_found:
                word_infos[word_info_index] = (word_infos[word_info_index][COUNT_VALUE_INDEX] + 1, word)
                        else:
                word_infos.append((1, word))

word_infos = sorted(word_infos, key=functools.cmp_to_key(word_info_comparator))

with open('result_1.txt', 'w') as output_file:
        for word_info in word_infos:
        output_file.write(word_info[WORD_VALUE_INDEX])
        output_file.write(' ')
        output_file.write(str(word_info[COUNT_VALUE_INDEX]))
        output_file.write('\n')
```

A.2. Текст программы exercise_2.py

```
# -*- coding: utf-8 -*-
# Назначение: программа для управления банком.
# Входные данные: допускается ввод команд, расположенных в левом поле ввода.
# Результаты: результаты выполнения команд выводятся в правом текстовом поле.

from PyQt5 import QtWidgets
from PyQt5.QtCore import Qt, QEvent
```

```
from bank import Ui_MainWindow
import sys

# Класс, представляющий клиента банка.
class Account:
    # Назначение: конструктор экземпляра клиента банка.
    # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    #   name - имя клиента банка.
    #   balance - баланс счета.
    # Результаты: инициализированный экземпляр класса клиента банка.
        def __init__(self, name, balance):
        self.name = name
        self.balance = balance

# Класс, представляющий банк.
class Bank:
    # Назначение: конструктор экземпляра банка.
    # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    #   log_info - функция записи информации о ходе процесса.
    # Результаты: инициализированный экземпляр класса банка.
        def __init__(self, log_info):
        self.accounts = [Account("Knyazev", 70143770)]
        self.log_info = log_info

        # Назначение: занести указанную сумму на счет указанного клиента.
    # Входные данные:
    #   self - ссылка на экземпляр банка.
    #   name - имя клиента банка.
    #   sum - зачисляемая сумма.
    # Результаты: сумма зачислена на счет клиента. Если такого клиента еще нет,
    #   то он создается и на его счет заносится указанная сумма.
        def deposit(self, name, sum):
        self.current_operation = "DEPOSIT"
        self.internal_deposit(name, sum)

        # Назначение: занести указанную сумму на счет указанного клиента (внутренний метод).
    # Входные данные:
    #   self - ссылка на экземпляр банка.
```

```
#   name - имя клиента банка.
#   sum - зачисляемая сумма.
# Результаты: сумма зачислена на счет клиента. Если такого клиента еще нет,
#   то он создается и на его счет заносится указанная сумма.
    def internal_deposit(self, name, sum):
  account_found = False
        for account in self.accounts:
            if account.name == name:
      self.log_info(self.current_operation + ": найден  клиент " + account.name + "\n")
      self.log_info(
        self.current_operation + ": текущий  остаток  на  счету  клиента " +
        account.name + " равен " + str(account.balance) + "\n")
      account.balance += sum
      self.log_info(self.current_operation + ": клиенту " + account.name + " зачислено " + str(sum) + "\n")
      self.log_info(
        self.current_operation + ": текущий  остаток  на  счету  клиента " +
        account.name + " равен " + str(account.balance) + "\n")
      account_found = True
                break
        if not account_found:
    self.log_info(self.current_operation + ": добавлен  новый  клиент " + name + ", на  его  счете " +
str(sum) + "\n")
            account = Account(name, sum)
            self.accounts.append(account)

    # Назначение: снять указанную сумму со счета указанного клиента.
# Входные данные:
#   self - ссылка на экземпляр банка.
#   name - имя клиента банка.
#   sum - снимаемая сумма.
# Результаты: сумма снимается со счета клиента (это может привести к отрицательному балансу).
#  Если такого клиента нет, то он создается с соответствующим отрицательным балансом.
    def withdraw(self, name, sum):
  self.current_operation = "WITHDRAW"
  self.internal_withdraw(name, sum)

    # Назначение: снять указанную сумму со счета указанного клиента (внутренний метод).
# Входные данные:
#   self - ссылка на экземпляр банка.
```

```
        #   name - имя клиента банка.
        #   sum - снимаемая сумма.
        # Результаты: сумма снимается со счета клиента (это может привести к отрицательному балансу).
        #  Если такого клиента нет, то он создается с соответствующим отрицательным балансом.
            def internal_withdraw(self, name, sum):
          account_found = False
                    for account in self.accounts:
                            if account.name == name:
                self.log_info(self.current_operation + ": найден   клиент " + account.name + "\n")
                self.log_info(
                  self.current_operation + ": текущий   остаток   на   счету   клиента " +
                  account.name + " равен " + str(account.balance) + "\n")
                account.balance -= sum
                self.log_info(
                  self.current_operation + ": со   счета   клиента " + account.name + " снято " + str(sum) + "\n")
                self.log_info(
                  self.current_operation + ": текущий   остаток   на   счету   клиента " +
                  account.name + " равен " + str(account.balance) + "\n")
                account_found = True
                                break
                    if not account_found:
             self.log_info(self.current_operation + ": добавлен    новый    клиент " + account.name + ", на    его
счете " + -sum + "\n")
                                account = Account(name, -sum)
                                self.accounts.append(account)

            # Назначение: получить баланс на счету указанного клиента.
        # Входные данные:
        #   self - ссылка на экземпляр банка.
        #   name - имя клиента банка.
        # Результаты: возвращается баланс на счету указанного клиента. Если клиента нет, то возвращается 0.
            def   get_balance(self, name):
                    for account in self.accounts:
                            if (account.name == name) or (name == ""):
                self.log_info("BALANCE:   найден      клиент   "   +   account.name   +   ",   на      его      счету   "   +
str(account.balance) + "\n")
                if name != "":
                                        return account.balance
          if name != "":
```

```
            self.log_info("BALANCE: NO CLIENT\n")
                return 0

        # Назначение: перевести указанную сумму со счета одного клиента на счет другого клиента.
        # Входные данные:
        #   self - ссылка на экземпляр банка.
        #   name1 - имя клиента банка, со счета которого переводятся средства.
        #   name2 - имя клиента банка, на счет которого переводятся средства.
        #   sum - переводимая сумма.
        # Результаты: указанная сумма переведена со счета одного клиента на счет другого клиента.
        #   Если одного из указанных клиентов нет, то он создается.
            def transfer(self, name1, name2, sum):
           self.current_operation = "TRANSFER"
           self.internal_withdraw(name1, sum)
           self.internal_deposit(name2, sum)

            # Назначение: начислить процент по вкладу всем клиентам банка.
        # Входные данные:
        #   self - ссылка на экземпляр банка.
        #   percent - величина процента.
        # Результаты: всем клиентам банка начислен указанный процент.
            def apply_income(self, percent):
                for account in self.accounts:
             self.log_info("INCOME: на счету клиента " + account.name + " до начисления процентов " + str(account.balance) + "\n")
                    if account.balance > 0:
                account.balance += int(account.balance * percent / 100)
                self.log_info("INCOME: на счету клиента " + account.name + " после начисления процентов " + str(account.balance) + " (начислено " + str(percent) + "%)\n")
                    else:
                self.log_info("INCOME: клиенту " + account.name + " проценты не начислены, так как остаток на счету отрицательный\n")

    # Класс, представляющий "движок" выполнения команд банковской системы.
    class CommandEngine:
        # Назначение: инициализировать экземпляр движка.
        # Входные данные:
        #   self - ссылка на неинициализированный экземпляр движка.
        #   bank - ссылка на экземпляр банка.
        # Результат: инициализированный движок.
```

```
    def __init__(self, bank):
        self.bank = bank

    # Назначение: выполнить команду банковской системы.
    # Входные данные:
    #   self - ссылка на экземпляр движка.
    #   command - выполняемая команда.
    # Результаты: команды выполнена.
        def execute_command(self, command):
                params = command.split()
                if command.startswith("DEPOSIT"):
                        self.bank.deposit(params[1], int(params[2]))
                elif command.startswith("WITHDRAW"):
                        self.bank.withdraw(params[1], int(params[2]))
                elif command.startswith("BALANCE"):
                        if len(params) > 1:
                                name = params[1]
                        else:
                name = ""
            self.bank.get_balance(name)
                elif command.startswith("TRANSFER"):
                        self.bank.transfer(params[1], params[2], int(params[3]))
                elif command.startswith("INCOME"):
            self.bank.apply_income(int(params[1]))

# Класс  окна  приложения.
class BankWindow(QtWidgets.QMainWindow):
        # Назначение: инициализировать  экземпляр  окна  приложения.
        # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    # Результат: инициализированный экземпляр класса.
        def __init__(self):
                super(BankWindow, self).__init__()
        self.ui = Ui_MainWindow()
                self.ui.setupUi(self)
                self.ui.commandsEdit.installEventFilter(self)
                self.ui.calculateButton.clicked.connect(self.calculate)
                self.ui.clearButton.clicked.connect(self.clear)
        self.bank = Bank(self.log)
```

```
        self.command_engine = CommandEngine(self.bank)

    # Назначение: фильтр событий.
# Входные данные:
#   self - ссылка на экземпляр класса.
#   obj - ылка на объект, в котором произошло событие.
#   event - событие.
# Результат: запрещено вводить Enter, если число строк превышает определенный лимит.
    def eventFilter(self, obj, event):
        if event.type() == QEvent.KeyPress and obj is self.ui.commandsEdit:
            line_count = self.ui.commandsEdit.toPlainText().count('\n')
            if line_count < 19 or not event.key() in (Qt.Key_Enter, Qt.Key_Return):
                QtWidgets.QTextEdit.keyPressEvent(obj, event)
            return True
        return super(QtWidgets.QMainWindow, self).eventFilter(obj, event)

    # Назначение: обработать щелчок по кнопке "Calculate".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: выполнение команд банковской системы и запись результата в правое поле ввода.
    def calculate(self):
        command_strings = self.ui.commandsEdit.toPlainText()
        for command_string in command_strings.splitlines():
            self.command_engine.execute_command(command_string)

    # Назначение: очистить поля ввода для команд и для выходных данных.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: очистка поля ввода для команд и для выходных данных.
    def clear(self):
        self.ui.commandsEdit.clear()
        self.ui.outputBrowser.clear()

# Назначение: вывести сообщение в поле для вывода сообщений.
# Входные данные:
#   self - ссылка на экземпляр класса.
#   message - логируемое сообщение.
# Результат: логируемое сообщение заносится в поле для выходныхъ данных.
    def log(self, message):
        self.ui.outputBrowser.append(message)
```

```
application = QtWidgets.QApplication([])
window = BankWindow()
window.show()

sys.exit(application.exec())
```

A.3. Текст программы exercise_3.py

```
# -*- coding: utf-8 -*-
# Назначение: программа-калькулятор.
# Входные данные: допускается ввод цифр и команд с помощью кнопок.
# Результаты: результаты выполнения команд выводятся на цифровом дисплее.
# Дата   написания: 04.05.2021.

from PyQt5 import QtWidgets
from calculator_standard_mode import Ui_CalculatorStandardModeWindow
from calculator_extended_mode import Ui_CalculatorExtendedModeWindow
import math
import sys

# Класс   калькулятора.
class Calculator:
   # Назначение: инициализировать экземпляр калькулятора.
   # Входные данные:
   #   self - ссылка на неинициализированный экземпляр класса.
   # Результат: инициализированный экземпляр класса.
   def __init__(self):
                self.memory = ["0", "0"]
                self.history = []
                self.internal_clear()

        # Назначение: очистить введенные операнды и операцию.
   # Входные данные:
   #   self - ссылка на калькулятор.
   # Результат: введенные операнды и операция очищены.
        def internal_clear(self):
      self.previous_operand = "0"
      self.current_operand = "0"
      self.display = "0"
                self.operation = ""
```

```
# Назначение: очистить введенные операнды и операцию.
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: введенные операнды и операция очищены.
     def clear(self):
   self.append_history(self.get_last_display_string())
   self.append_history("C")
   self.internal_clear()

     # Назначение: добавить строку в историю.
# Входные данные:
#   self - ссылка на калькулятор.
#   history_string - строка истории.
# Результат: строка добавлена в историю.
     def append_history(self, history_string):
          if len(self.history) > 3:
               self.history.pop(0)
          self.history.append(history_string)

     # Назначение: ввести цифру.
# Входные данные:
#   self - ссылка на калькулятор.
#   digit - число, представляющее введенную цифру.
# Результат: результат обработки введения цифры калькулятором.
     def enter_digit(self, digit):
          if self.current_operand == "0":
      self.current_operand = str(digit)
          else:
      self.current_operand = self.current_operand + str(digit)
   self.display = self.current_operand

# Назначение: ввести   точку.
     # Входные данные:
#   self - ссылка на калькулятор.
# Результат: результатввода точки.
     def   enter_point(self):
          if not "." in self.current_operand:
      self.current_operand = self.current_operand + "."
               self.display = self.current_operand
```

```
# Назначение: сохранить текущий введенный операнд.
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: текущий введенный операнд сохранен.
    def store_current_operand(self):
  self.previous_operand = self.current_operand
            self.current_operand = "0"

# Назначение: обработать ввод кнопки "плюс".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "плюс".
    def plus(self):
            self.equals()
  self.store_current_operand()
  self.operation = "+"

    # Назначение: обработать ввод кнопки "минус".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "минус".
    def minus(self):
            self.equals()
  self.store_current_operand()
  self.operation = "-"

    # Назначение: обработать ввод кнопки "умножить".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "умножить".
    def multiply(self):
            self.equals()
  self.store_current_operand()
  self.operation = "*"

    # Назначение: обработать ввод кнопки "делить".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "делить".
    def divide(self):
```

```
            self.equals()
        self.store_current_operand()
        self.operation = "/"

    # Назначение: обработать ввод кнопки "возведение в степень".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "возведение в степень".
    def power(self):
            self.equals()
        self.store_current_operand()
        self.operation = "^"

    # Назначение: обработать ввод кнопки "логарифм".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат ввода кнопки "логарифм".
    def log(self):
            if float(self.current_operand) <= 0.0:
        self.current_operand = "0"
        self.display = "error"
            else:
                    self.equals()
        self.store_current_operand()
                    self.operation = "log"

# Назначение: сменить знак текущего операнда.
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: знак текущего операнда изменен на противоположный.
    def plus_minus(self):
            if (len(self.current_operand) > 1) and (self.current_operand[0] == "-"):
        self.current_operand = self.current_operand[1:]
            else:
        self.current_operand = "-" + self.current_operand
            self.display = self.current_operand

# Назначение: извлечь квадратный корень из текущего операнда.
# Входные данные:
#   self - ссылка на калькулятор.
```

```
# Результат: в текущем операнде значение квадратного корня из текущего операнда.
    def square_root(self):
            if float(self.current_operand) < 0.0:
     self.current_operand = "0"
     self.display = "error"
            else:
                    result = math.sqrt(float(self.current_operand))
                    if result.is_integer():
       self.current_operand = str(int(result))
                    else:
       self.current_operand = str(result)
     self.display = self.current_operand

# Назначение: арксинус.
    # Входные данные:
#   self - ссылка на калькулятор.
# Результат: вычисленный арксинус.
    def   arcsin(self):
            argument = float(self.current_operand)
            if math.fabs(argument) > 1.0:
     self.current_operand = "0"
     self.display = "error"
            else:
     self.current_operand = str(math.asin(argument))
     self.display = self.current_operand

# Назначение: арккосинус.
    # Входные данные:
#   self - ссылка на калькулятор.
# Результат: вычисленный арккосинус.
    def   arccos(self):
            argument = float(self.current_operand)
            if math.fabs(argument) > 1.0:
     self.current_operand = "0"
     self.display = "error"
            else:
     self.current_operand = str(math.acos(argument))
     self.display = self.current_operand

# Назначение: арктангенс.
```

```
    # Входные данные:
#   self - ссылка на калькулятор.
# Результат: вычисленный арктангенс.
    def   arctan(self):
            argument = float(self.current_operand)
   self.current_operand = str(math.atan(argument))
   self.display = self.current_operand

# Назначение: факториал.
    # Входные данные:
#   self - ссылка на калькулятор.
# Результат: вычисленный факториал.
    def   fact(self):
            argument = float(self.current_operand)
            if argument < 0.0 or not argument.is_integer():
     self.current_operand = "0"
     self.display = "error"
            else:
     self.current_operand = str(math.factorial(int(self.current_operand)))
                    self.display = self.current_operand


# Назначение: обработать нажатие на кнопку "равно".
# Входные данные:
#   self - ссылка на калькулятор.
# Результат: результат нажатия на кнопку "равно".
    def equals(self):
   if self.operation != "":
     self.append_history(self.get_last_display_string())
     self.append_history("=")

   is_error = False
            result = "nop"

            if self.operation == "+":
                    result = float(self.previous_operand) + float(self.current_operand)
            elif self.operation == "-":
                    result = float(self.previous_operand) - float(self.current_operand)
            elif self.operation == "*":
                    result = float(self.previous_operand) * float(self.current_operand)
```

```
            elif self.operation == "^":
                result = float(self.previous_operand) ** float(self.current_operand)
            elif self.operation == "log":
                if (float(self.current_operand) <= 0.0) or (float(self.previous_operand) <= 0):
        is_error = True
                else:
                    result = math.log(float(self.current_operand), float(self.previous_operand))
            elif self.operation == "/":
      current_operand_float = float(self.current_operand)
                if current_operand_float.is_integer() and int(current_operand_float) == 0:
        is_error = True
                else:
                    result = float(self.previous_operand) / float(self.current_operand)

            if result != "nop":
                if result.is_integer():
        self.current_operand = str(int(result))
                else:
        self.current_operand = str(result)

    self.operation = ""
            if is_error:
      self.display = "error"
            else:
      self.display = self.current_operand

     # Назначение: сохранить в ячейку памяти с заданным индексом значение с экрана.
# Входные данные:
#   self - ссылка на калькулятор.
#   index - индекс ячейки памяти.
# Результат: значение с экрана сохранено в ячейку памяти с заданным индексом.
     def memory_save(self, index):
            self.memory[index] = self.display

# Назначение: очистить ячейку памяти с заданным индексом.
# Входные данные:
#   self - ссылка на калькулятор.
#   index - индекс ячейки памяти.
# Результат: ячейка памяти с заданным индексом очищена.
     def memory_clear(self, index):
```

```
            self.memory[index] = "0"

        # Назначение: считать на экран значение из ячейки памяти с указанным индексом.
        # Входные данные:
        #   self - ссылка на калькулятор.
        #   index - индекс ячейки памяти.
        # Результат: ячейка памяти с заданным индексом считана на экран.
         def memory_read(self, index):
       self.current_operand = self.memory[index]
            self.display = self.current_operand

        # Назначение: записать результат в ячейку памяти с заданным индексом.
        # Входные данные:
        #   self - ссылка на калькулятор.
        #   index - индекс ячейки памяти.
        #   result - текущий результат.
        # Результат: результат записан в ячейку памяти с заданным индексом.
         def put_result_to_memory(self, index, result):
            if result.is_integer():
                self.memory[index] = str(int(result))
            else:
                self.memory[index] = str(result)

        # Назначение: записать в память разность между текущим значением на экране и значением из ячейки памяти.
        # Входные данные:
        #   self - ссылка на калькулятор.
        #   index - индекс ячейки памяти.
        # Результат: в память записана разность между текущим значением на экране и значением из ячейки памяти.
         def memory_minus(self, index):
            result = float(self.display) - float(self.memory[index])
       self.put_result_to_memory(index, result)

        # Назначение: записать в память сумму между текущим значением на экране и значением из ячейки памяти.
        # Входные данные:
        #   self - ссылка на калькулятор.
        #   index - индекс ячейки памяти.
```

```
        # Результат: в память записана сумма между текущим значением на экране и значением из ячейки памяти.
        def memory_plus(self, index):
            result = float(self.display) + float(self.memory[index])
            self.put_result_to_memory(index, result)

        # Назначение: вычислить сумму цифр числа рекурсивно до получения числа из одной цифры.
        # Входные данные:
        #   self - ссылка на калькулятор.
        # Результат: сумма цифр числа в виде числа из одной цифры.
        def sum_of_digits(self):
            if "." in self.current_operand:
                self.display = "error"
                self.current_operand = "0"
                return
            current_number_string = self.current_operand
            while True:
                current_result = 0
                for character in current_number_string:
                    current_result += int(character)
                if current_result < 10:
                    self.current_operand = str(current_result)
                    self.display = self.current_operand
                    break
                else:
                    current_number_string = str(current_result)

        # Назначение: получить содержимое дисплея.
        # Входные данные:
        #   self - ссылка на калькулятор.
        # Результат: содержимое дисплея.
        def get_display(self):
            return self.display

        # Назначение: получить последнюю строку цифрового дисплея.
        # Входные данные:
        #   self - ссылка на калькулятор.
        # Результат: последняя строка цифрового дисплея.
        def get_last_display_string(self):
            if self.display == "error":
```

```
                return "error"

        last_string = ""
        if self.operation != "":
                if self.operation == "log":
            last_string = "log_" + self.previous_operand
                else:
            last_string = self.previous_operand + self.operation
                if (last_string != "") and ("-" in self.current_operand) and (self.operation != "log"):
            current_operand = "(" + self.current_operand + ")"
                else:
            current_operand = self.current_operand
                if self.operation == "log":
            last_string = last_string + "(" + current_operand + ")"
                else:
            last_string = last_string + current_operand
                return last_string

    # Назначение: получить содержимое цифрового дисплея.
    # Входные данные:
    #   self - ссылка на калькулятор.
    # Результат: содержимое цифрового дисплея.
        def  get_digital_display(self):
                open_paragraph_tag = "<p align=\"right\" style=\"margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">"
        result = ""
                for history_string in self.history:
                        result = result + open_paragraph_tag + history_string + "</p>\n"
                result = result + open_paragraph_tag + self.get_last_display_string() + "</p>"

                return result

# Класс окна калькулятора в стандартном режиме.
class CalculatorStandardModeWindow(QtWidgets.QMainWindow):
    # Назначение: инициализировать экземпляр окна приложения.
    # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    # Результат: инициализированный экземпляр класса.
        def __init__(self):
                super(CalculatorStandardModeWindow, self).__init__()
```

```
        self.ui = Ui_CalculatorStandardModeWindow()
        self.ui.setupUi(self)
        self.calculator = Calculator()
        self.ui.resetButton.clicked.connect(self.reset)
        self.ui.nineButton.clicked.connect(self.nine)
        self.ui.eightButton.clicked.connect(self.eight)
        self.ui.sevenButton.clicked.connect(self.seven)
        self.ui.sixButton.clicked.connect(self.six)
        self.ui.fiveButton.clicked.connect(self.five)
        self.ui.fourButton.clicked.connect(self.four)
        self.ui.threeButton.clicked.connect(self.three)
        self.ui.twoButton.clicked.connect(self.two)
        self.ui.oneButton.clicked.connect(self.one)
        self.ui.zeroButton.clicked.connect(self.zero)
        self.ui.pointButton.clicked.connect(self.point)
        self.ui.divButton.clicked.connect(self.divide)
        self.ui.multButton.clicked.connect(self.multiply)
        self.ui.minusButton.clicked.connect(self.minus)
        self.ui.plusButton.clicked.connect(self.plus)
        self.ui.plusMinusButton.clicked.connect(self.plus_minus)
        self.ui.squareRootButton.clicked.connect(self.square_root)
        self.ui.equalsButton.clicked.connect(self.equals)
        self.ui.powerButton.clicked.connect(self.power)
        self.ui.memorySaveButton.clicked.connect(self.memory_save)
        self.ui.memoryClearButton.clicked.connect(self.memory_clear)
        self.ui.memoryReadButton.clicked.connect(self.memory_read)
        self.ui.memoryMinusButton.clicked.connect(self.memory_minus)
        self.ui.memoryPlusButton.clicked.connect(self.memory_plus)
        self.ui.extendedModeButton.clicked.connect(self.extended_mode)

    # Назначение: обновить дисплей калькулятора.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обновленный дисплей калькулятора.
    def   refresh(self):
        self.ui.lineEdit.setText(self.calculator.get_display())

    # Назначение: перевести калькулятор в расширенный режим.
# Входные данные:
```

```
#   self - ссылка на экземпляр класса.
# Результат: калькулятор переведен в расширенный режим.
     def extended_mode(self):
   extended_mode_window.show()
   extended_mode_window.move(self.x(), self.y())
            self.hide()

# Назначение: обработать щелчок по кнопке "сохранить в памяти".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат обработки щелчка по кнопке "сохранить в памяти".
     def memory_save(self):
   self.calculator.memory_save(0)
            self.refresh()

# Назначение: обработать щелчок по кнопке "очистить память".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат обработки щелчка по кнопке "очистить память".
     def memory_clear(self):
   self.calculator.memory_clear(0)
            self.refresh()

# Назначение: обработать щелчок по кнопке "считать из памяти".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат обработки щелчка по кнопке "считать из памяти".
     def memory_read(self):
   self.calculator.memory_read(0)
            self.refresh()

# Назначение: обработать щелчок по кнопке "M-".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат обработки щелчка по кнопке "M-".
     def memory_minus(self):
   self.calculator.memory_minus(0)
            self.refresh()

# Назначение: обработать щелчок по кнопке "M+".
```

```
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат обработки щелчка по кнопке "M+".
     def memory_plus(self):
   self.calculator.memory_plus(0)
            self.refresh()

# Назначение: обработать нажатие на кнопку "возведение в степень".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат возведения в степень.
     def power(self):
            self.calculator.power()
            self.refresh()

# Назначение: обработать щелчок по кнопке "равно".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат щелчка по кнопке "равно".
     def equals(self):
            self.calculator.equals()
            self.refresh()

# Назначение: обработать нажатие на кнопку "плюс-минус".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "плюс-минус".
     def plus_minus(self):
   self.calculator.plus_minus()
            self.refresh()

# Назначение: обработать нажатие на кнопку "квадратный корень".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "квадратный корень".
     def square_root(self):
   self.calculator.square_root()
            self.refresh()

# Назначение: обработать нажатие на кнопку "делить".
```

```
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "делить".
    def divide(self):
            self.calculator.divide()
            self.refresh()

# Назначение: обработать нажатие на кнопку "умножить".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "умножить".
    def multiply(self):
            self.calculator.multiply()
            self.refresh()

# Назначение: обработать нажатие на кнопку "минус".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "минус".
    def minus(self):
            self.calculator.minus()
            self.refresh()

# Назначение: обработать нажатие на кнопку "плюс".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат нажатия на кнопку "плюс".
    def plus(self):
            self.calculator.plus()
            self.refresh()

# Назначение: обработать щелчок по кнопке "точка".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат щелчка по кнопке "точка".
    def point(self):
  self.calculator.enter_point()
            self.refresh()

# Назначение: сбросить калькулятор.
```

```
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: сброс калькулятора.
    def   reset(self):
            self.calculator.clear()
            self.refresh()

    # Назначение: ввести цифру "девять".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "девять".
    def   nine(self):
            self.calculator.enter_digit(9)
            self.refresh()

    # Назначение: ввести цифру "восемь".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "восемь".
    def   eight(self):
            self.calculator.enter_digit(8)
            self.refresh()

    # Назначение: ввести цифру "семь".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "семь".
    def   seven(self):
            self.calculator.enter_digit(7)
            self.refresh()

    # Назначение: ввести цифру "шесть".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "шесть".
    def   six(self):
            self.calculator.enter_digit(6)
            self.refresh()

    # Назначение: ввести цифру "пять".
```

```
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "пять".
    def   five(self):
            self.calculator.enter_digit(5)
            self.refresh()

    # Назначение: ввести цифру "четыре".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "четыре".
    def   four(self):
            self.calculator.enter_digit(4)
            self.refresh()

    # Назначение: ввести цифру "три".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "три".
    def   three(self):
            self.calculator.enter_digit(3)
            self.refresh()

    # Назначение: ввести цифру "два".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "два".
    def   two(self):
            self.calculator.enter_digit(2)
            self.refresh()

    # Назначение: ввести цифру "один".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: ввод цифры "один".
    def   one(self):
            self.calculator.enter_digit(1)
            self.refresh()

    # Назначение: ввести цифру "ноль".
```

```
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    # Результат: ввод цифры "ноль".
        def   zero(self):
                self.calculator.enter_digit(0)
                self.refresh()

# Класс окна калькулятора в расширенном режиме.
class CalculatorExtendedModeWindow(QtWidgets.QMainWindow):
    # Назначение: инициализировать экземпляр окна приложения.
    # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    # Результат: инициализированный экземпляр класса.
        def __init__(self):
                super(CalculatorExtendedModeWindow, self).__init__()
        self.ui = Ui_CalculatorExtendedModeWindow()
                self.ui.setupUi(self)
        self.calculator = Calculator()
                self.ui.standardModeButton.clicked.connect(self.standard_mode)
                self.ui.nineButton.clicked.connect(self.nine)
                self.ui.eightButton.clicked.connect(self.eight)
                self.ui.sevenButton.clicked.connect(self.seven)
                self.ui.sixButton.clicked.connect(self.six)
                self.ui.fiveButton.clicked.connect(self.five)
                self.ui.fourButton.clicked.connect(self.four)
                self.ui.threeButton.clicked.connect(self.three)
                self.ui.twoButton.clicked.connect(self.two)
                self.ui.oneButton.clicked.connect(self.one)
                self.ui.zeroButton.clicked.connect(self.zero)
                self.ui.pointButton.clicked.connect(self.point)
                self.ui.resetButton.clicked.connect(self.reset)
                self.ui.equalsButton.clicked.connect(self.equals)
                self.ui.sumOfDigitsButton.clicked.connect(self.sum_of_digits)
                self.ui.squareRootButton.clicked.connect(self.square_root)
                self.ui.plusMinusButton.clicked.connect(self.plus_minus)
                self.ui.plusButton.clicked.connect(self.plus)
                self.ui.minusButton.clicked.connect(self.minus)
                self.ui.multiplyButton.clicked.connect(self.multiply)
                self.ui.divideButton.clicked.connect(self.divide)
```

```
        self.ui.powerButton.clicked.connect(self.power)
        self.ui.asinButton.clicked.connect(self.arcsin)
        self.ui.acosButton.clicked.connect(self.arccos)
        self.ui.atgButton.clicked.connect(self.arctan)
        self.ui.factorialButton.clicked.connect(self.fact)
        self.ui.memryClearButton0.clicked.connect(self.memory_clear0)
        self.ui.memoryReadButton0.clicked.connect(self.memory_read0)
        self.ui.memorySaveButton0.clicked.connect(self.memory_save0)
        self.ui.memryPlusButton0.clicked.connect(self.memory_plus0)
        self.ui.memoryMinusButton0.clicked.connect(self.memory_minus0)
        self.ui.memryClearButton1.clicked.connect(self.memory_clear1)
        self.ui.memoryReadButton1.clicked.connect(self.memory_read1)
        self.ui.memorySaveButton1.clicked.connect(self.memory_save1)
        self.ui.memryPlusButton1.clicked.connect(self.memory_plus1)
        self.ui.memoryMinusButton1.clicked.connect(self.memory_minus1)
        self.ui.logButton.clicked.connect(self.log)

    # Назначение: обновить дисплей калькулятора.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обновлен дисплей калькулятора.
    def   refresh(self):
        self.ui.digitalDisplay.setText(self.calculator.get_digital_display())

    # Назначение: обработать щелчок по кнопке вычисления логарифма.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке вычисления логарифма.
    def log(self):
        self.calculator.log()
        self.refresh()

# Назначение: обработать нажатие кнопки "MC" для нулевой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MC".
    def memory_clear0(self):
  self.calculator.memory_clear(0)
        self.refresh()
```

```
# Назначение: обработать нажатие кнопки "MR" для нулевой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MR"
     def memory_read0(self):
   self.calculator.memory_read(0)
            self.refresh()

# Назначение: обработать нажатие кнопки "MS" для нулевой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MS"
     def memory_save0(self):
   self.calculator.memory_save(0)
            self.refresh()

# Назначение: обработать нажатие кнопки "M+" для нулевой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "M+"
     def memory_plus0(self):
   self.calculator.memory_plus(0)
            self.refresh()

# Назначение: обработать нажатие кнопки "M-" для нулевой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "M-"
     def memory_minus0(self):
   self.calculator.memory_minus(0)
            self.refresh()

# Назначение: обработать нажатие кнопки "MC" для первой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MC".
     def memory_clear1(self):
   self.calculator.memory_clear(1)
            self.refresh()
```

```
# Назначение: обработать нажатие кнопки "MR" для первой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MR"
     def memory_read1(self):
   self.calculator.memory_read(1)
            self.refresh()

# Назначение: обработать нажатие кнопки "MS" для первой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "MS"
     def memory_save1(self):
   self.calculator.memory_save(1)
            self.refresh()

# Назначение: обработать нажатие кнопки "M+" для первой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "M+"
     def memory_plus1(self):
   self.calculator.memory_plus(1)
            self.refresh()

# Назначение: обработать нажатие кнопки "M-" для первой ячейки памяти.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработано нажатие кнопки "M-"
     def memory_minus1(self):
   self.calculator.memory_minus(1)
            self.refresh()

# Назначение: арксинус.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: вычислен арксинус.
     def   arcsin(self):
            self.calculator.arcsin()
            self.refresh()
```

```
    # Назначение: арккосинус.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: вычислен арккосинус.
    def   arccos(self):
            self.calculator.arccos()
            self.refresh()

    # Назначение: арктангенс.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: вычислен арктангенс.
    def   arctan(self):
            self.calculator.arctan()
            self.refresh()

    # Назначение: факториал.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: вычислен факториал.
    def   fact(self):
            self.calculator.fact()
            self.refresh()

    # Назначение: обработать щелчок по кнопке "плюс".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "плюс".
    def plus(self):
            self.calculator.plus()
            self.refresh()

# Назначение: обработать щелчок по кнопке "минус".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "минус".
    def minus(self):
            self.calculator.minus()
            self.refresh()
```

```
# Назначение: обработать щелчок по кнопке "умножить".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "умножить".
    def multiply(self):
        self.calculator.multiply()
        self.refresh()

# Назначение: обработать щелчок по кнопке "разделить".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "разделить".
    def divide(self):
        self.calculator.divide()
        self.refresh()

# Назначение: обработать щелчок по кнопке "степень".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "степень".
    def power(self):
        self.calculator.power()
        self.refresh()

# Назначение: обработать щелчок по кнопке "равно".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "равно".
    def equals(self):
        self.calculator.equals()
        self.refresh()

# Назначение: обработать щелчок по кнопке "сброс".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: результат щелчка по кнопке "сброс".
    def reset(self):
        self.calculator.clear()
        self.refresh()
```

```
# Назначение: обработать щелчок по кнопке "точка".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "точка".
     def point(self):
   self.calculator.enter_point()
            self.refresh()

# Назначение: обработать щелчок по кнопке "плюс-минус".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "плюс-минус".
     def plus_minus(self):
   self.calculator.plus_minus()
            self.refresh()

# Назначение: извлечь квадратный корень из числа.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: извлечен квадратный корень.
     def   square_root(self):
            self.calculator.square_root()
            self.refresh()

     # Назначение: подсчитать сумму цифр числа (курсивно до получения числа из одной цифры).
# Входные данные:
#   self - ссылка на экземпляр класса.
     # Результат: сумма   цифр   числа.
     def sum_of_digits(self):
   self.calculator.sum_of_digits()
            self.refresh()

# Назначение: обработать щелчок по кнопке "девять".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "девять".
     def nine(self):
   self.calculator.enter_digit(9)
            self.refresh()
```

```
# Назначение: обработать щелчок по кнопке "восемь".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "восемь".
    def eight(self):
  self.calculator.enter_digit(8)
            self.refresh()

# Назначение: обработать щелчок по кнопке "семь".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "семь".
    def seven(self):
  self.calculator.enter_digit(7)
            self.refresh()

# Назначение: обработать щелчок по кнопке "шесть".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "шесть".
    def six(self):
  self.calculator.enter_digit(6)
            self.refresh()

# Назначение: обработать щелчок по кнопке "пять".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "пять".
    def five(self):
  self.calculator.enter_digit(5)
            self.refresh()

# Назначение: обработать щелчок по кнопке "четыре".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "четыре".
    def four(self):
  self.calculator.enter_digit(4)
            self.refresh()
```

```
# Назначение: обработать щелчок по кнопке "три".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "три".
    def three(self):
  self.calculator.enter_digit(3)
            self.refresh()

# Назначение: обработать щелчок по кнопке "два".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "два".
    def two(self):
  self.calculator.enter_digit(2)
            self.refresh()

# Назначение: обработать щелчок по кнопке "один".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "один".
    def one(self):
  self.calculator.enter_digit(1)
            self.refresh()

# Назначение: обработать щелчок по кнопке "ноль".
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: обработан щелчок по кнопке "ноль".
    def zero(self):
  self.calculator.enter_digit(0)
            self.refresh()

# Назначение: переключиться в окно стандартного режима калькулятора.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: показано окно калькулятора в стандартном режимме.
    def standard_mode(self):
  standard_mode_window.show()
  standard_mode_window.move(self.x(), self.y())
            self.hide()
```

```
application = QtWidgets.QApplication([])
standard_mode_window = CalculatorStandardModeWindow()
extended_mode_window = CalculatorExtendedModeWindow()
standard_mode_window.show()

sys.exit(application.exec())
```

A.4. Текст программы exercise_4.py

```
# -*- coding: utf-8 -*-
# Назначение: программа, визуализирующая решение задачи про ханойские башни.
# Входные данные: допускается ввод процента завершения задачи.
# Результаты: отображается подходящая итерация решения задачи.
# Дата   написания: 07.05.2021

from PyQt5 import QtWidgets
from PyQt5.QtGui import QPainter, QColor, QFontMetrics
from hanoi import Ui_HanoiMainWindow
import copy
import math
import random
import sys

# Класс, представляющий итерацию решения задачи о Ханойских башнях.
class HanoiIteration:
    # Назначение: инициализировать экземпляр класса.
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    #   iteration - номер итерации.
    #   state - состояние решения задачи.
    #   from_column - с какого шпинделя.
    #   to_column - какой шпиндель.
    #   disk_diameter - диаметр диска.
        # Результат: инициализированный  экземпляр  класса.
        def __init__(self, iteration, state, from_column = None, to_column = None, disk_diameter = None):
        self.iteration = iteration
        self.state = state
        self.from_column = from_column
        self.to_column = to_column
                self.disk_diameter = disk_diameter
```

```
    # Назначение: вывести состояние решения задачи.
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    # Результат: состояние решения задачи выведено в консоль.
        def print(self):
            print(self.state)

        # Назначение: проверить на равенство с другой итерацией.
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    #   other - другая итерация.
    # Результат: результат проверки на равенство с другой итерацией.
        def equals_to_by_state(self, other):
            return self.state == other.state

    @staticmethod
        def start():
            return HanoiIteration(0,
         [[87, 86, 85, 84, 83, 82, 81], [], [61], [55, 54, 53, 52, 51], [], [32, 31],
            [29, 28, 27, 26, 25, 24, 23, 22, 21], [18, 17, 16, 15, 14, 13, 12, 11]])

    @staticmethod
        def final():
            return   HanoiIteration(0,
                [[], [], [], [], [], [], [], [87, 86, 85, 84, 83, 82, 81, 61, 55, 54, 53, 52,
                    51, 32, 31, 29, 28, 27, 26, 25, 24, 23, 22, 21, 18, 17, 16, 15, 14, 13, 12, 11]])

# Назначение: решить головоломку Ханойская башня для трех указанных башен.
# Входные данные:
#   disk_count - количество дисков для перемещения.
#   current_iteration - текущая итерация решения задачи.
#   iteration_list - список итераций (передается для накопления результата).
#   source - номер исходной башни.
#   destination - номер целевой башни.
#   auxiliary - номер вспомогательной башни.
# Результат: список итераций по перемещению дисокв из source в destination.
def solve_hanoi3(disk_count, current_iteration, iteration_list, source, destination, auxiliary):
        if (disk_count == 1):
      new_state = copy.deepcopy(current_iteration.state)
            element = new_state[source].pop()
```

```
            new_state[destination].append(element)
            new_iteration = HanoiIteration(current_iteration.iteration + 1, new_state, source, destination, element)
            iteration_list.append(new_iteration)
                return iteration_list
          else:
            result_iteration_list = solve_hanoi3(disk_count - 1, current_iteration, iteration_list, source, auxiliary, destination)
            current_iteration = result_iteration_list[-1]

            new_state = copy.deepcopy(current_iteration.state)
                element = new_state[source].pop()
            new_state[destination].append(element)
            new_iteration = HanoiIteration(current_iteration.iteration + 1, new_state, source, destination, element)
            result_iteration_list.append(new_iteration)

            result_iteration_list = solve_hanoi3(disk_count - 1, new_iteration, result_iteration_list, auxiliary, destination, source)
                return result_iteration_list

        # Назначение: построить решение головоломки Ханойская башня для восьми башен (исходное задание).
        # Результат: список итераций с решением головоломки от начальной до конечной.
        def solve_hanoi8():
          start_iteration = HanoiIteration.start()
          disk_count = len(start_iteration.state[7])
          iteration_list = solve_hanoi3(disk_count, start_iteration, [start_iteration], 7, 1, 4)
          current_iteration = iteration_list[-1]
          disk_count = len(current_iteration.state[0])
          iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 0, 7, 4)
          current_iteration = iteration_list[-1]
          disk_count = len(current_iteration.state[2])
          iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 2, 7, 4)
          current_iteration = iteration_list[-1]
          disk_count = len(current_iteration.state[3])
          iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 3, 7, 4)
          current_iteration = iteration_list[-1]
          disk_count = len(current_iteration.state[5])
          iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 5, 7, 4)
          current_iteration = iteration_list[-1]
          disk_count = len(current_iteration.state[6])
```

```
    iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 6, 7, 4)
    current_iteration = iteration_list[-1]
    disk_count = len(current_iteration.state[1])
    iteration_list = solve_hanoi3(disk_count, current_iteration, iteration_list, 1, 7, 4)
        return iteration_list

# Класс главного окна программы.
class HanoiMainWindow(QtWidgets.QMainWindow):
    # Назначение: инициализировать экземпляр окна приложения.
    # Входные данные:
    #   self - ссылка на неинициализированный экземпляр класса.
    # Результат: инициализированный экземпляр класса.
        def __init__(self):
                super(HanoiMainWindow, self).__init__()
        self.ui = Ui_HanoiMainWindow()
                self.ui.setupUi(self)
                self.ui.startButton.clicked.connect(self.start)
                self.ui.finalButton.clicked.connect(self.final)
                self.ui.firstPercentButton.clicked.connect(self.first_percent)
                self.ui.secondPercentButton.clicked.connect(self.second_percent)
                self.ui.thirdPercentButton.clicked.connect(self.third_percent)
                self.ui.fourthPercebtButton.clicked.connect(self.fourth_percent)
        self.iterations = solve_hanoi8()
        self.is_disk_between = False
                self.start()

    # Назначение: инициализировать задачу в начальное состояние.
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    # Результат: задача инициализирована в начальное состояние и окно перерисовано.
        def start(self):
        self.goto_iteration(0)

        # Назначение: перевести задачу в конечное состояние.
    # Входные данные:
    #   self - ссылка на экземпляр класса.
    # Результат: задача переведена в конечное состояние и окно перерисовано.
        def final(self):
        self.goto_iteration(len(self.iterations) - 1)
```

```
    # Назначение: перейти к итерации по проценту в заданном поле ввода.
# Входные данные:
#   self - ссылка на экземпляр класса.
#   edit - поле ввода.
# Результат: задача переведена к заданной итерации и окно перерисовано.
    def apply_percent(self, edit):
  iteration_percent = int(edit.text())
            iteration = (len(self.iterations) - 1) * iteration_percent / 100
            self.goto_iteration(iteration)

# Назначение: перейти к итерации по проценту в первом поле ввода.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: задача переведена к заданной итерации и окно перерисовано.
    def first_percent(self):
  self.apply_percent(self.ui.firstPercentEdit)

    # Назначение: перейти к итерации по проценту во втором поле ввода.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: задача переведена к заданной итерации и окно перерисовано.
    def second_percent(self):
  self.apply_percent(self.ui.secondPercentEdit)

    # Назначение: перейти к итерации по проценту в третьем поле ввода.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: задача переведена к заданной итерации и окно перерисовано.
    def third_percent(self):
  self.apply_percent(self.ui.thirdPercentEdit)

    # Назначение: перейти к итерации по проценту в четвертом поле ввода.
# Входные данные:
#   self - ссылка на экземпляр класса.
# Результат: задача переведена к заданной итерации и окно перерисовано.
    def fourth_percent(self):
  self.apply_percent(self.ui.fourthPercentEdit)

    # Назначение: перейти к заданной итерации.
# Входные данные:
```

```
        #   self - ссылка на экземпляр класса.
        #   number - номер итерации.
        # Результат: задача переведена к заданной итерации и окно перерисовано.
            def goto_iteration(self, number):
                if float(number).is_integer():
                int_number = int(number)
                self.current_hanoi = self.iterations[int_number].state
                self.is_disk_between = False
                        self.ui.iterationLabel.setText("Итерация: " + str(int(number)))
                else:
                prev_number = math.floor(number)
                next_number = prev_number + 1
                self.current_hanoi = list(self.iterations[prev_number].state)
                self.is_disk_between = True
                self.disk_diameter = self.iterations[next_number].disk_diameter
                self.from_column = self.iterations[next_number].from_column
                self.to_column = self.iterations[next_number].to_column
                self.current_hanoi = [[element for element in sublist if element != self.disk_diameter] for sublist in
self.current_hanoi]
                        self.ui.iterationLabel.setText("Итерация: " + "%.3f" % number)
                self.repaint()

        # Назначение: перерисовать окно.
        # Входные данные:
        #   self - ссылка на экземпляр класса.
        #   e - событие.
        # Результат: окно перерисовано.
            def paintEvent(self, e):
                qp = QPainter()
                qp.begin(self)

           disk_height = 20
                for column_index in range(0, 8):
                current_hanoi_column = self.current_hanoi[column_index]
                column_center_x = 230 + 100 * column_index

                        # Нарисовать  шпиндель.
                        qp.setBrush(QColor(255, 0, 0))
                        qp.drawRect(column_center_x - 2, 20, 4, 800)
```

```
                    # Нарисовать  диски.
                disk_index = 0
                    for current_disk_diameter in current_hanoi_column:
                        qp.setBrush(QColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)))
                        qp.drawRect(column_center_x - current_disk_diameter / 2, 800 - disk_height * disk_index, current_disk_diameter, disk_height)
                        disk_index += 1

                # Нарисовать подписи к дискам.
                    disk_index = 0
                    for current_disk_diameter in current_hanoi_column:
                        text = str(current_disk_diameter)
                        fm = QFontMetrics(qp.font())
                        qp.drawText(column_center_x - fm.width(text) / 2, 800 - disk_height * (disk_index - 1) - 3, text)
                        disk_index += 1

            # Если есть диск между шпинделями, то нарисовать его.
                if self.is_disk_between:
                    qp.setBrush(QColor(random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)))
                disk_center_x = 230 + 100 * self.from_column + 50
                    qp.drawRect(disk_center_x - self.disk_diameter / 2, 20, self.disk_diameter, disk_height)
                    text = str(self.disk_diameter)
                    fm = QFontMetrics(qp.font())
                    qp.drawText(disk_center_x - fm.width(text) / 2, 37, text)

                # Нарисовать поверхность, на которой находятся шпиндели.
                qp.setBrush(QColor(255, 0, 0))
                qp.drawRect(170, 820, 840, 16)
                qp.end()

        application = QtWidgets.QApplication([])
        window = HanoiMainWindow()
        window.show()

        sys.exit(application.exec())
```

Приложение Б. Образцы GUI заданий

Б.1. Образец GUI по второму заданию

Образец GUI по второму заданию представлен на рисунке 12.

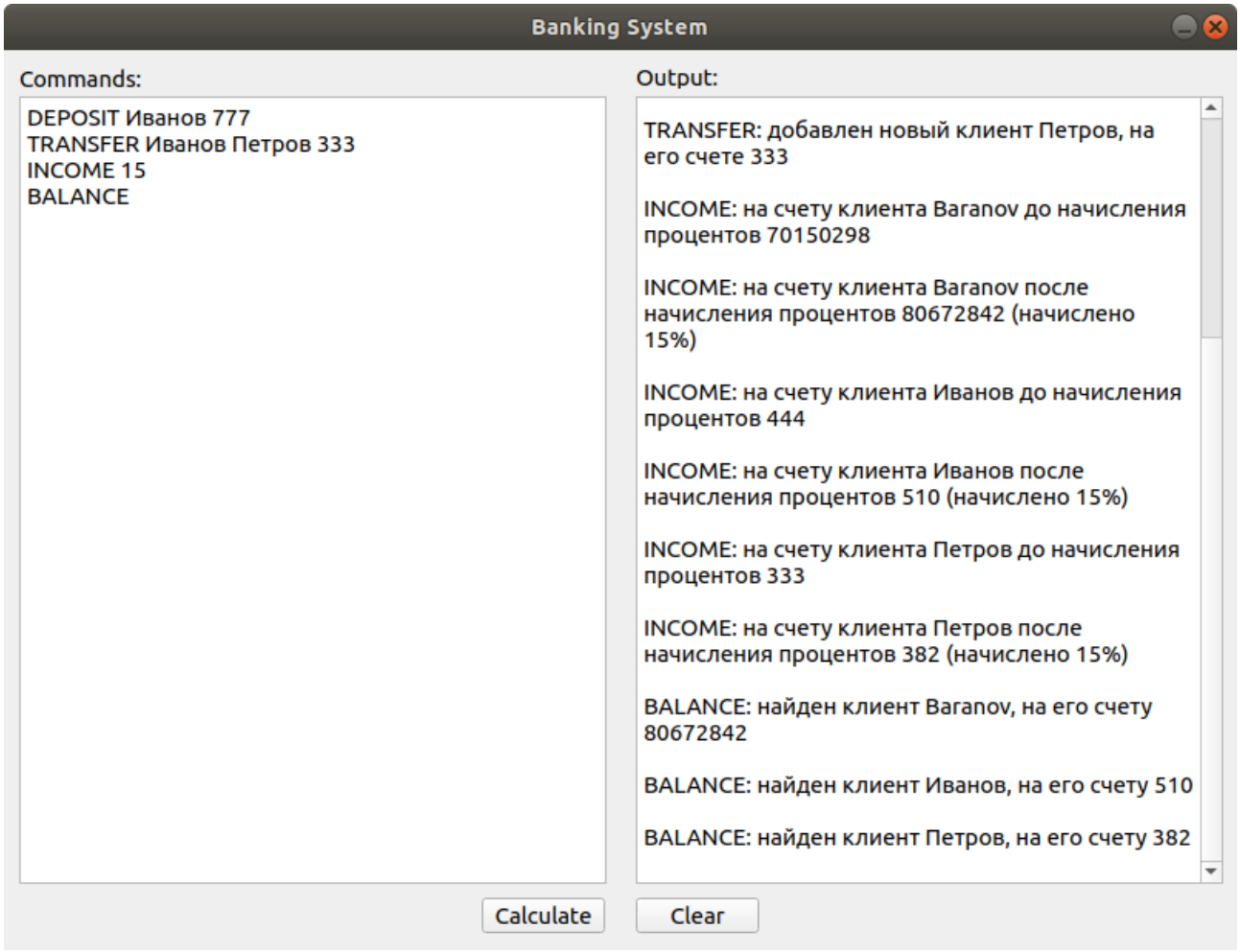


Рисунок 12 – Образец GUI ко второму заданию

Б.2. Образец GUI по третьему заданию

Образец GUI по третьему заданию представлен на рисунках 13 и 14.

Б.3. Образец GUI по четвертому заданию

Образец GUI по четвертому заданию представлен на рисунке 15.

Рисунок 13 – Образец GUI по третьему заданию (стандартный режим калькулятора)

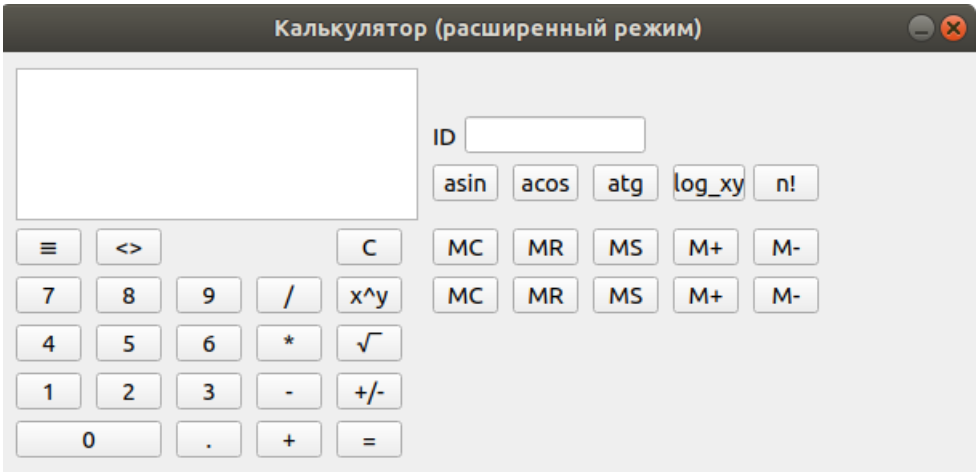


Рисунок 14 – Образец GUI к третьему заданию (расширенный режим)

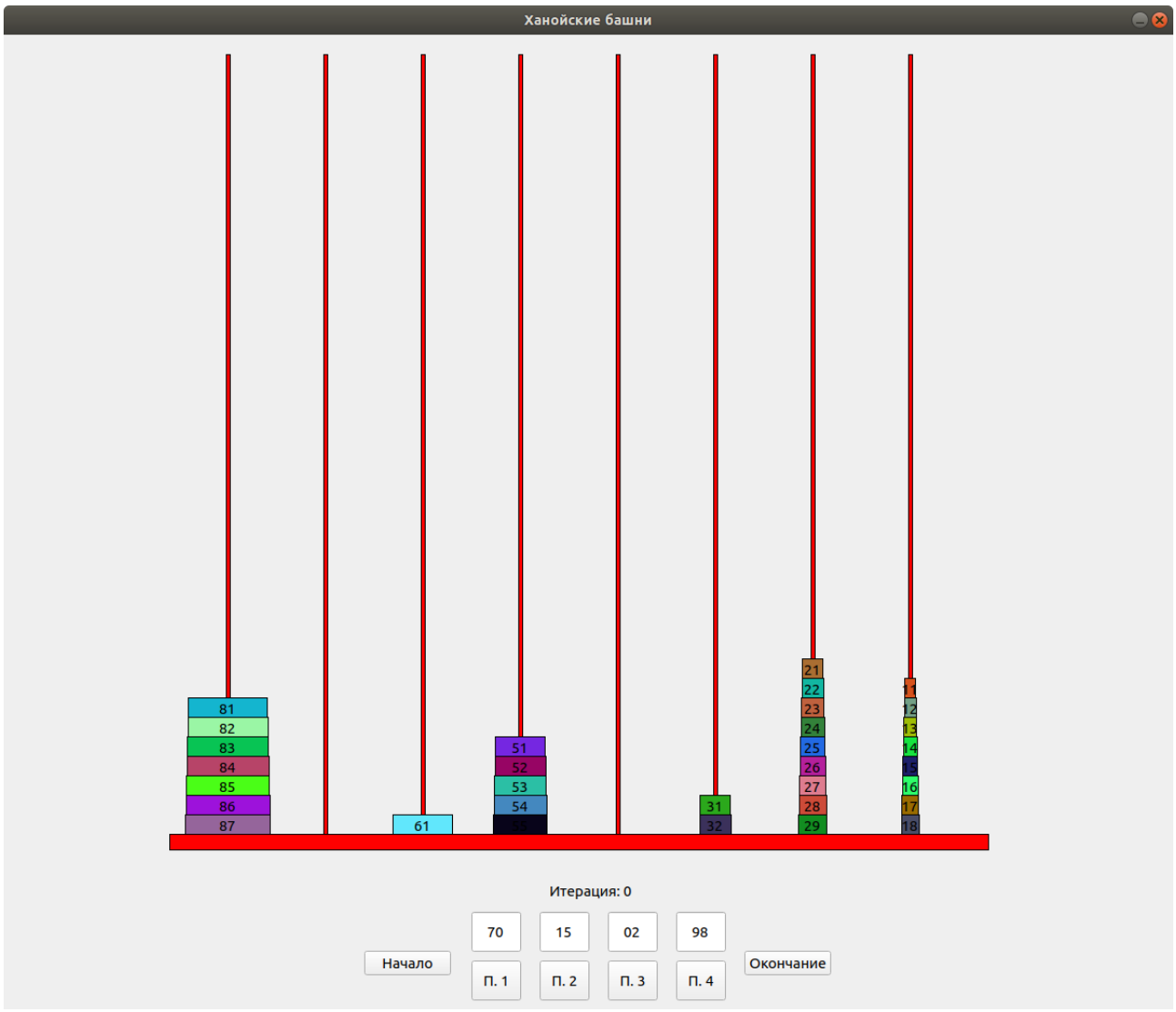


Рисунок 15 – Образец GUI к четвертому заданию