

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc128694765)

[1. АНАЛИЗ КУРСОВОЙ РАБОТЫ 4](#_Toc128694766)

[1.1. Исходные данные к задаче курсовой работы. 4](#_Toc128694767)

[1.2. Анализ методических указаний, входных и выходных данных к заданиям курсовой работы 6](#_Toc128694768)

[1.3. Выбор и обоснование необходимых библиотек и среды разработки 9](#_Toc128694769)

[1.4. Выводы по 1 главе 10](#_Toc128694770)

[2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ. 10](#_Toc128694771)

[2.1. Работа с наборами данных 10](#_Toc128694772)

[2.2. Разработка экспертной системы 13](#_Toc128694773)

[2.3. Разработка аналитической системы 16](#_Toc128694774)

[2.4 Задача о Ханойских башнях 19](#_Toc128694775)

[2.5. Вывод по главе 2 23](#_Toc128694776)

[3. РАЗРАБОТКА ТРЕБОВАНИЙ К ТЕХНИЧЕСКИМ СРЕДСТВАМ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ 24](#_Toc128694777)

[Выводы 25](#_Toc128694778)

[СПИСОК ЛИТЕРАТУРЫ 26](#_Toc128694779)

[ПРИЛОЖЕНИЯ 27](#_Toc128694780)

**ВВЕДЕНИЕ**

В современной экономике, которая быстро развивается и основана на высоких технологиях, важно создавать автоматизированные системы, которые позволяют использовать современные средства информационной вычислительной техники для эффективного решения прикладных задач. Для этого необходимо использовать высокоуровневые средства разработки и декомпозировать задачу.

В данной курсовой работе решаются задачи высокоуровневого программирования в соответствии с индивидуальным заданием.

Ее актуальность обусловлена наличием множества прикладных задач, которые требуют эффективного решения с помощью инструментов высокоуровневых методов программирования.

# 1. АНАЛИЗ КУРСОВОЙ РАБОТЫ

## 1.1. Исходные данные к задаче курсовой работы.

Работа с наборами данных.

Формулировка: Во внешнем файле resourse\_1.txt дан текст. Выведите все слова, встречающиеся в тексте, по одному на каждую строку, через пробел укажите количество повторений. Слова должны быть отсортированы по убыванию их количества появления в тексте, а при одинаковой частоте появления — в лексикографическом порядке. Вывод должен осуществляться в текстовый файл result\_1.txt. При необходимости можно продублировать вывод в консоль.

Разработка экспертной системы.

Формулировка: Некоторый банк хочет внедрить систему управления счетами клиентов, поддерживающую следующие операции:

1. Пополнение счета клиента.

2. Снятие денег со счета.

3. Запрос остатка средств на счете.

4. Перевод денег между счетами клиентов.

5. Начисление процентов всем клиентам.

Разработка аналитической системы.

Формулировка: разработать калькулятор со стандартным и расширенным функционалом.

Стандартный функционал

1. Арифметические действия +-\*/.

2. Возможность ввода отрицательного числа

3. Возведение в степень.

4. Извлечение квадратного корня.

5. Работа с памятью, состоящей из одной ячейки.

6. Должна быть кнопка сброса и кнопка «=» (равно).

Расширенный функционал

1. Наличие кнопки/меню перехода в расширенный режим

2. Возможность работы с несколькими ячейками памяти. Количество ячеек памяти выбирается согласно методическим указаниям.

3. Отображение последовательности математических операций и цифр в n-строчном «дисплее», с возможностью «прокрутки». Количество строк «дисплея» калькулятора выбирается согласно методическим указаниям.

4. Реализация «инженерных» функций расширенного режима. Конкретный перечень функций выбирается согласно методическим указаниям.

Задача о Ханойских башнях:

Модифицированная задача о Ханойских башнях:

Существует 8 шпинделей, пронумерованных от 8 до 1 слева направо. На каждом шпинделе надеты диски, в количестве, равном соответствующей цифре из ID студента. Все диски имеют разные диаметры. Диаметр диска равен M \* 10 + N, где М – номер шпинделя, на котором надет диск, а N – это номер диска на шпинделе, считая сверху вниз.

1. Необходимо визуально изобразить предложенную задачу. Диски на шпинделях сделать случайных цветов. На каждом диске отображать цифру, равную его диаметру. Диаметр диска также показывать его фактическим размером в пикселях.

2. Необходимо вычислить, за какое минимальное количество итераций переместятся все диски на шпиндель номер 1

3. Необходимо отобразить начальное и конечное расположение дисков на шпинделях, для этого под изображением Ханойских башен предусмотреть две кнопки «Начало» и «Окончание».

4. Необходимо графически отобразить четыре промежуточные итерации перекладывания дисков

5. Дать возможность пользователю изменять проценты в полях для ввода цифр, и по нажатию соответствующей кнопки просматривать расположение дисков на данной итерации.

Таким образом, задания включают в себя основные прикладные задачи, которые решаются разработчиками программного обеспечения, это и работа с данными, использование алгоритмов и структур данных и построение ПО с графическим пользовательским интерфейсом.

## 1.2. Анализ методических указаний, входных и выходных данных к заданиям курсовой работы

Необходимо ознакомиться с методическими указаниями и входными данными для выполнения курсовой работы.

Первое задание предполагает создание словаря всех слов и их сортировку по частоте встречаемости. Для этого следует создать список кортежей, где первый элемент - частота встречаемости слова, а второй - само слово. Знаки препинания не учитываются.

Входные данные - текстовый файл resourse\_1.txt с 5-6 абзацами, который преподаватель сохраняет.

Выходные данные - текстовый файл result\_1.txt.

Для второго задания методические указания звучат следующим образом: необходимо реализовать такую систему. Первоначально у банка 1 клиент. Клиент(ы) банка идентифицируются именами (уникальная строка, не содержащая пробелов). Вам необходимо задать в качестве имени клиента – свою фамилию на английском языке с большой буквы. На вашу фамилию должен 2 Приведены в сокращенном виде. Полный текст в методических материалах 37 быть открыт счет с суммой равной вашему ID. В отдельном поле должна быть предусмотрена возможность ввода простых команд:

• DEPOSIT name sum - Зачислить сумму sum на счет клиента name. Если клиента нет, то он создается и на него заводится счет с указанной суммой.

• WITHDRAW name sum - Снять сумму sum со счета клиента name. Если клиента, то счет создается. Баланс при выполнении такой операции у вновь созданного клиента должен быть отрицательный.

• BALANCE name - Узнать остаток средств на счету клиента name. Для каждого запроса BALANCE программа должна вывести остаток на счету данного клиента. Если же у клиента с запрашиваемым именем не открыт счет в банке, выводится сообщение «NO CLIENT». Если пользователь не указал имя клиента – то выводится баланс всех существующих клиентов.

• TRANSFER name1 name2 sum - Перевести сумму sum со счета клиента name1 на счет клиента name2. Если у какого-либо клиента, то он заводится в системе и ему создается счет с переведенной суммой.

• INCOME p - Начислить всем клиентам, у которых открыты счета, p% от суммы счета. Проценты начисляются только клиентам с положительным остатком на счету, если у клиента остаток отрицательный, то его счет не меняется. После начисления процентов сумма на счету остается целой, то есть начисляется только целое число денежных единиц. Дробная часть начисленных процентов отбрасывается.

Программа обрабатывает текстовые команды из левого поля только после нажатия кнопки «Calculate». То есть, пользователь СНАЧАЛА вводит желаемые команды, при этом каждая новая команда вводится с новой строки, а ПОТОМ нажимает на кнопку «Calculate». Результат должен быть выведен в поле справа.

Входные данные: Преподаватель вводит в поле ввода команды Выходные данные: Преподаватель наблюдает вывод

Методические указания к третьему заданию: необходимо разработать программу и GUI для реализации стандартных функций калькулятора. Примерные вид внешнего интерфейса – стандартный.

Функционал «инженерного» режима работы калькулятора добавляется при нажатии на кнопку перехода в расширенный режим, либо выбора соответствующего пункта меню.

Расширенный режим работы определяется внешним видом «цифрового дисплея», количеством ячеек памяти (кнопки M+, M-, MS, MR, MS) и кнопками, отвечающими за дополнительные функции. Количество строк «цифрового дисплея» должно определяться как последовательная сумма всех цифр ID студента. Суммирование отдельных цифр числа должно осуществляться до получения однозначного числа, состоящей из 1-й цифры.

Количество ячеек памяти калькулятора должно определяться как последовательная сумма последних 3-х чисел ID. Суммирование отдельных цифр числа должно осуществляться до получения однозначного числа, состоящей из 1-й цифры.

Дополнительные функции расширенного режима: Ln, ,

Dms, sin переводит из десятичного вида в формат в градусы, минуты, секунды; возведение произвольность числа в произвольную степень, число Пи, гиперболический тангенс, натуральный логарифм.

Входные данные: Преподаватель с помощью мыши вводит числа и выполняет арифметические операции.

Преподаватель с помощью мыши вводит числа и выбирает заявленные дополнительные функции. Преподаватель вводит намеренно некорректные операции (попытка деления на 0)

Выходные данные: на цифровом дисплее должен отображаться результат, либо сообщение об ошибке при некорректной операции.

Проверяется количество строк в «цифровом дисплее» калькулятора и количество ячеек памяти.

Для выполнения четвертого задания необходимо правильно визуализировать полученные результаты и оптимально искать промежуточные результаты.

Для контроля на каждом диске необходимо проставить его диаметр в виде цифры, так как диски близких размеров будут отличаться всего на один пиксель. Рекомендуется сделать толщину одного диска примерно равной 10-12 пикселям для указанного окна, так как в конце задачи все диски будут находиться на первом шпинделе, а общее количество дисков теоретически может быть равно 72.

Входные данные представляются идентификатором студента и промежуточными процентами, вводимыми преподавателем в соответствующие поля. Выходные данные должны содержать окно с начальным расположением дисков на шпинделях Ханойских башен, пронумерованными шпинделями и обозначенными диаметрами на дисках.

Также должны быть представлены шесть кнопок и четыре поля для ввода цифр, а нижняя часть экрана должна демонстрировать надпись «Итерация 0». Методические материалы содержат указания и подсказки, примеры интерфейсов (вплоть до пикселей) и структуры данных, которые следует использовать, что упрощает выполнение заданий.

### 

## 1.3. Выбор и обоснование необходимых библиотек и среды разработки

Поскольку в методических указаниях к выполнению курсовой работы указано использование языка программирования Python, среды разработки PyCharm, а также стандартных библиотек и модулей стандартной библиотеки, то для решения заданий курсовой работы были выбраны следующие средства:

Язык программирования Python3 – согласно рекомендациям;

Интегрированная среда разработки PyCharm – согласно рекомендациям.

Графическая библиотека Tkinter – как наиболее простая и по причине того, что предыдущие задания по дисциплине выполнялись с помощью неё, соответственно, есть некоторый опыт использования этой библиотеки

## 1.4. Выводы по 1 главе

Получив информацию из первой главы, можно заключить, что задания включают в себя наиболее распространенные прикладные задачи высокоуровневого программирования, такие как работа с файлами, использование структур данных и алгоритмов, создание приложений с графическим интерфейсом и работа в популярной среде разработки Python. Выполнение этих заданий демонстрирует полное понимание материала по дисциплине «Методы высокоуровневого программирования» и способность решать задачи различной сложности.

# 2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ.

## 2.1. Работа с наборами данных

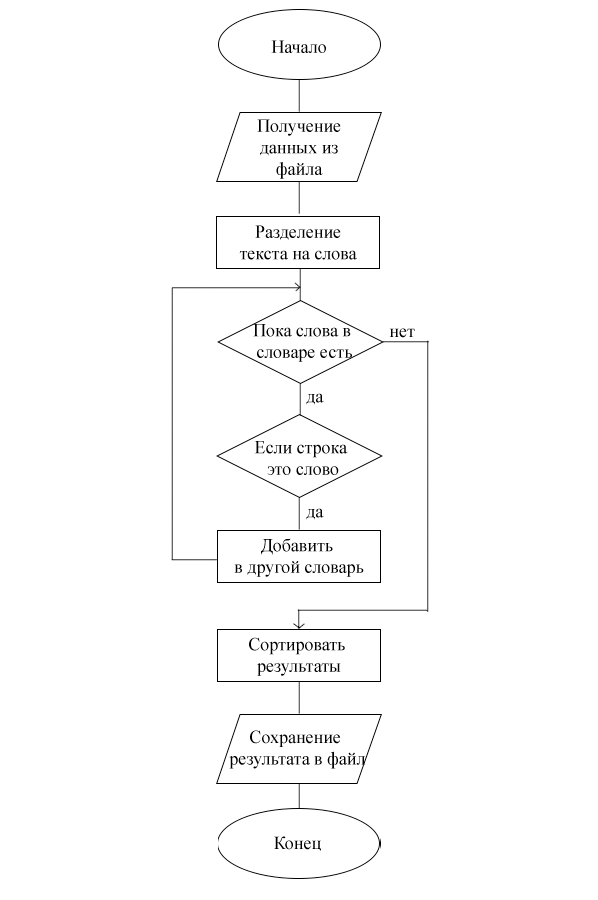
При работе с наборами данных обычно используется оператора open. После чего для записи используется оператор with, который автоматически закрывает файл, по окончанию работы в нем. Благодаря этому код становится более читаемым, а также предотвращаются возможные ошибки. Это особенно важно, если не используется анализ кода.

Для более удобной работы с файлами был создан словарь «wdict», представляющая собой пару слово – количество сколько раз оно встречается в тексте.

#### 2.1.1. Построение алгоритма решения задания без графического интерфейса

Задача состоит из нескольких этапов. Сначала следует взять данные из файла, далее идет анализ, включающий в себя: подсчет количества встречаемых слов, сортировка и запись в файл.

Блок-схема алгоритма изображена на рисунке



#### 2.1.2. Разработка программной реализации на языке программирования.

Для создания программы будет использоваться язык программирования Python.

Реализация алгоритма занимает всего 15 строк, поэтому не требует классов, которые могут усложнить код, если планируется его расширение.

Код программы с комментариями приведен ниже.

­­­­­­ from collections import defaultdict  
  
with open("resourse\_1.txt", "r", encoding="utf-8") as input\_file:  
 # Открываем файл и разбиваем текст на слова  
 words = input\_file.read().split()  
 wdict = defaultdict(int)  
 # Записываем из послученного списка в словарь после чего перебираем их в цикле  
 for word in words:  
 if word.isalpha():  
 wdict[word] += 1  
 # Сортируем сравнивая второе значение в обратном порядке (от большего к меньшему), иначе по первому  
 sorted\_words = sorted(wdict.items(), key=lambda x: (-x[1], x[0]))  
  
 # Записываем полученный результат в файл  
 with open("result\_1.txt", "w", encoding="utf-8") as output\_file:  
 for word, count in sorted\_words:  
 print(word, count, file=output\_file)

## 2.2. Разработка экспертной системы

#### 2.2.1. Построение алгоритма решения задания с графическим интерфейсом

Один из подходящих алгоритмов для решения задачи является следующая последовательность действий:

• нажатие клавиши «Calculate»;

• чтение данных из поля ввода;

• разбиение на команды;

• выполнение команды;

• вывод данных.

#### 2.2.2. Разработка программной реализации на языке программирования

Для создания программной реализации графического интерфейса были использованы виджеты библиотеки tkinter.

Программная реализация использует объектно-ориентированный подход. Он был выбран по причине того, что количество строк кода перешло бы за сотню, а поддерживать программы в процедурном стиле куда сложнее.

Для обработки всех операций был создан класс Bank, который берет на себя основную функциональность приложения. Именно в нем прописаны все методы для обработки команд.

Сделано это было для того, чтобы код был менее загружен и более понятен, а также если, например, появилась необходимость изменить сам текст команд, при этом не меняя их основной функционал.

class Bank:  
 def \_\_init\_\_(self):  
 self.clientsdict = defaultdict(int)  
  
 # Функционал всех команд прописан именно здесь  
 def deposit(self, name: str, money: int):  
 self.clientsdict[name] += money  
  
 def withdraw(self, name: str, money: int):  
 self.clientsdict[name] -= money  
  
 def balance(self, name: str):  
 if name != "":  
 return self.clientsdict.get(name, "NO CLIENT")  
 else:  
 return self.clientsdict.items()  
  
 def transfer(self, name1: str, name2: str, money: int):  
 self.clientsdict[name1] -= money  
 self.clientsdict[name2] += money  
  
 def income(self, p: int):  
 for name in self.clientsdict:  
 if self.clientsdict[name] > 0:  
 self.clientsdict[name] = int(self.clientsdict[name] \* (1 + p / 100))

Взаимодействие пользователя с интерфейсом было реализовано благодаря 2 функциям, для кнопок Calculate и Clear.

def clicked\_calculate():  
 input\_text = str(text.get(1.0, "end"))  
 lbl\_text = ""  
  
 # Сначала из поля ввода текст делится на строки, после чего на отдельные слова  
 for line in input\_text.split('\n'):  
  
 words = line.split(' ')  
  
 if words[0] == "DEPOSIT":  
 bank\_app.deposit(words[1], int(words[2]))  
 elif words[0] == "WITHDRAW":  
 bank\_app.withdraw(words[1], int(words[2]))  
  
 elif words[0] == "BALANCE":  
 # Проверка на ввод конкретного пользователя, либо отображаем баланс всех пользователей  
 if len(words) > 1 and words[1] != '':  
 output\_text = bank\_app.balance(words[1])  
 lbl\_text += words[1] + ' ' + str(output\_text) + '\n'  
 else:  
 data = bank\_app.balance("")  
 output\_text = ""  
 for key, value in data:  
 output\_text += key + ' ' + str(value) + '\n'  
 lbl\_text += output\_text  
  
 elif words[0] == "TRANSFER":  
 bank\_app.transfer(words[1], words[2], int(words[3]))  
 elif words[0] == "INCOME":  
 bank\_app.income(int(words[1]))  
  
 text\_label.configure(state='normal')  
 text\_label.insert(1.0, lbl\_text)  
 text\_label.configure(state='disabled')  
  
# Функция для кнопки Clear (очистка полей)  
def clicked\_clear():  
 text.delete(1.0, "end")  
  
 text\_label.configure(state='normal')  
 text\_label.delete(1.0, "end")  
 text\_label.configure(state='disabled')

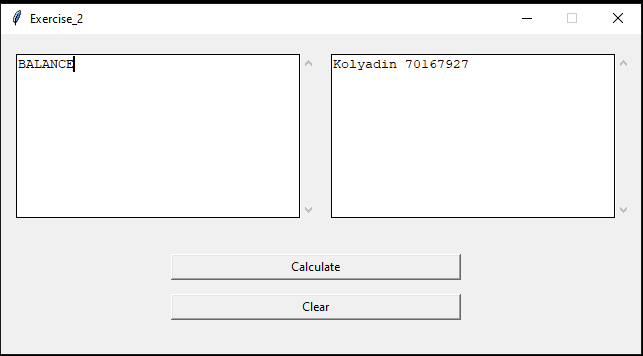
Реализация поставленной задачи сложнее прошлой, однако для ее решения данного алгоритма вполне достаточно, благодаря простоте и удобству выбранного языка программирования.

Полный код программы можно найти по ссылке в приложении.

#### 2.2.3. Построение алгоритма решения задания с графическим интерфейсом.

В задаче требуется 2 текстовых поля: для ввода команд и для отображения результатов, также для взаимодействия пользователя с интерфейсом были созданы 2 кнопки.

Интерфейс готового приложения изображен ниже



## 2.3. Разработка аналитической системы

#### 2.3.1. Построение алгоритма решения задания с графическим интерфейсом

Сам алгоритм реализован относительно ввода данных непосредственно пользователем с помощью нажатий на клавиши калькулятора, где уже отдельные функции возьмутся за обработку вычислений разных арифметических действий и последующему отображению их результата.

Так же для отображения истории вычислений подобно популярному калькулятору на Windows, будет производится запись предыдущих нажатий в отдельные переменные и отображаем их сверху.

Алгоритм для расширенной версии калькулятора схож с тем же алгоритмом, который был использован для предыдущей системы, где строка разбивается на символы и алгоритм обрабатывал команды.

#### 2.3.2. Разработка программной реализации на языке программирования

Разработке графического интерфейса аналогично использовалась библиотека tkinter.

Для расширенной работы с числами, а также с математическими функциями, была использована встроенная библиотеки math.

Реализация данной программы использует также использует объектно-ориентированный подход. Класс Calculator был создан для хранения всех данных о введенных числах, ячейках памяти, а также проводимых операциях.

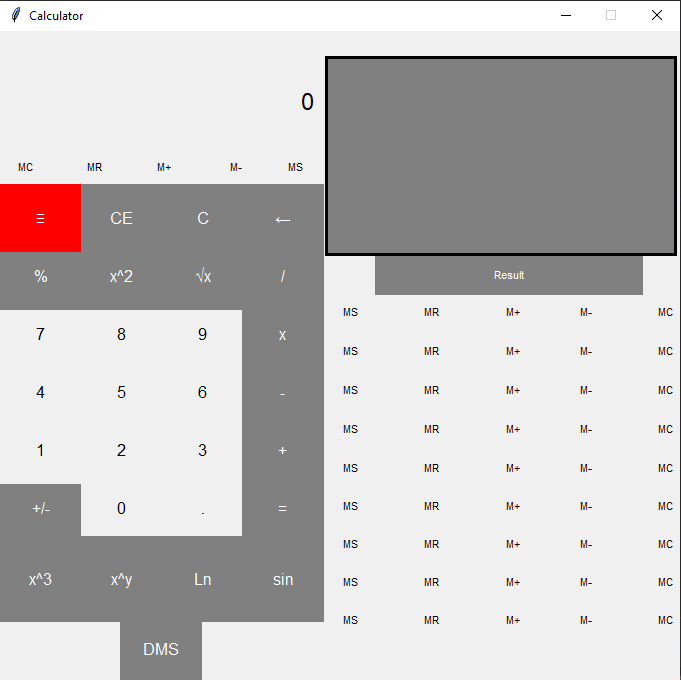
# Объявление класса калькулятора, который хранит все необходимые данные для работы  
class Calculator:  
 def \_\_init\_\_(self) -> None:  
 self.text\_current = "0"  
 self.text\_history = ""  
 self.text\_display = "0"  
 self.small\_mode = True  
 self.prev = ""  
 self.mcells = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, }  
 self.value = 0  
 self.history = ""  
 self.type = "int"  
 self.operation = None

Для обработки конкретного арифметического действия были созданы отдельные функции. Ниже представлен отрывок кода для сложения.

def click\_plus():  
 # Проверяем если уже вводили операнд или же вводится ноль  
 # calc.prev записывает значение прошлой нажатой клавиши  
  
 if calculate.operation == "+" and calculate.text\_current == "0":  
 calculate.text\_display = calculate.prev  
 pass  
 elif calculate.operation != "+":  
 calculate.operation = "+"  
 calculate.prev = calculate.text\_display  
 calculate.text\_current = "0"  
 calculate.text\_history = f"{calculate.text\_display} + "  
 lbl\_history.configure(text=calculate.text\_history)  
 else:  
  
 # Если число не целое то округяем полученное значение  
  
 tmp = round(float(calculate.text\_current) + float(calculate.prev), 3)  
 if tmp % 1 == 0.0:  
 calculate.text\_display = str(int(tmp))  
 else:  
 calculate.text\_display = str(tmp)  
 calculate.prev = calculate.text\_display  
 calculate.text\_history = f"{calculate.text\_display} + "  
 calculate.text\_current = "0"  
 lbl\_history.configure(text=calculate.text\_history)  
 lbl\_current.configure(text=calculate.text\_display)

Полный код с комментариями находится в репозитории по ссылке прикрепленной в приложении

#### 2.3.3. Проектирование интерфейса приложения

Часто при планировании стандартного функционала возникает значительная трудность именно при создании удобного интерфейса для пользователя. В качестве образца был использован стандартный калькулятор ОС Windows 10.

## 

## 2.4 Задача о Ханойских башнях

#### 2.4.1. Построение алгоритма решения задания с графическим интерфейсом

Для решения стандартной задачи с 3 штырями часто используется рекурсивный метод, который заключается в переносе башни из n-1 диска на 2-й штырь, затем переносе самого большого диска на 3-й штырь и, наконец, переносе башни из n-1 диска на 3-й штырь.

Однако, при усложненной задаче мы рассматриваем промежуток из трех центральных штырей как стандартную задачу, а крайние штыри рассматриваем отдельно, где предыдущий штырь выступает в качестве промежуточного, а средний штырь - начальным, а все крайние - конечными.

#### 2.4.2. Разработка программной реализации на языке программирования.

Для удобной работы с башнями для самих башен и дисков на них были созданы отдельные классы.

class Disk:  
 def \_\_init\_\_(self):  
 self.size = 0  
 self.color = ""  
  
 def add\_size(self, m: int, n: int):  
 self.size = m \* 10 + n  
  
 def color\_gen(self, used\_colors: defaultdict):  
 self.color = generate\_colors(used\_colors)  
  
 def draw(self, canvas, x, y):  
 canvas.create\_rectangle(  
 x - self.size / 2,  
 y,  
 x + self.size / 2,  
 y + DISK\_HEIGHT,  
 outline="",  
 fill=self.color,  
 )  
 canvas.create\_text(x, y + 6, text=str(self.size))

def Tower(id):  
 towers = [[] for \_ in range(9)]  
 disk\_array = id\_parsing(id)  
 used\_colors = defaultdict(int)  
  
 for i in range(8, 0, -1):  
 for j in range(disk\_array[i - 1]):  
 tmp = Disk()  
 tmp.color\_gen(used\_colors)  
 tmp.add\_size(9 - i, j + 1)  
 towers[i].append(tmp)  
 towers[i] = towers[i][::-1]  
 return towers

Ниже представлен отрывок кода, включающий в себя непосредственно ту самую рекурсию необходимую для решения поставленной задачи.

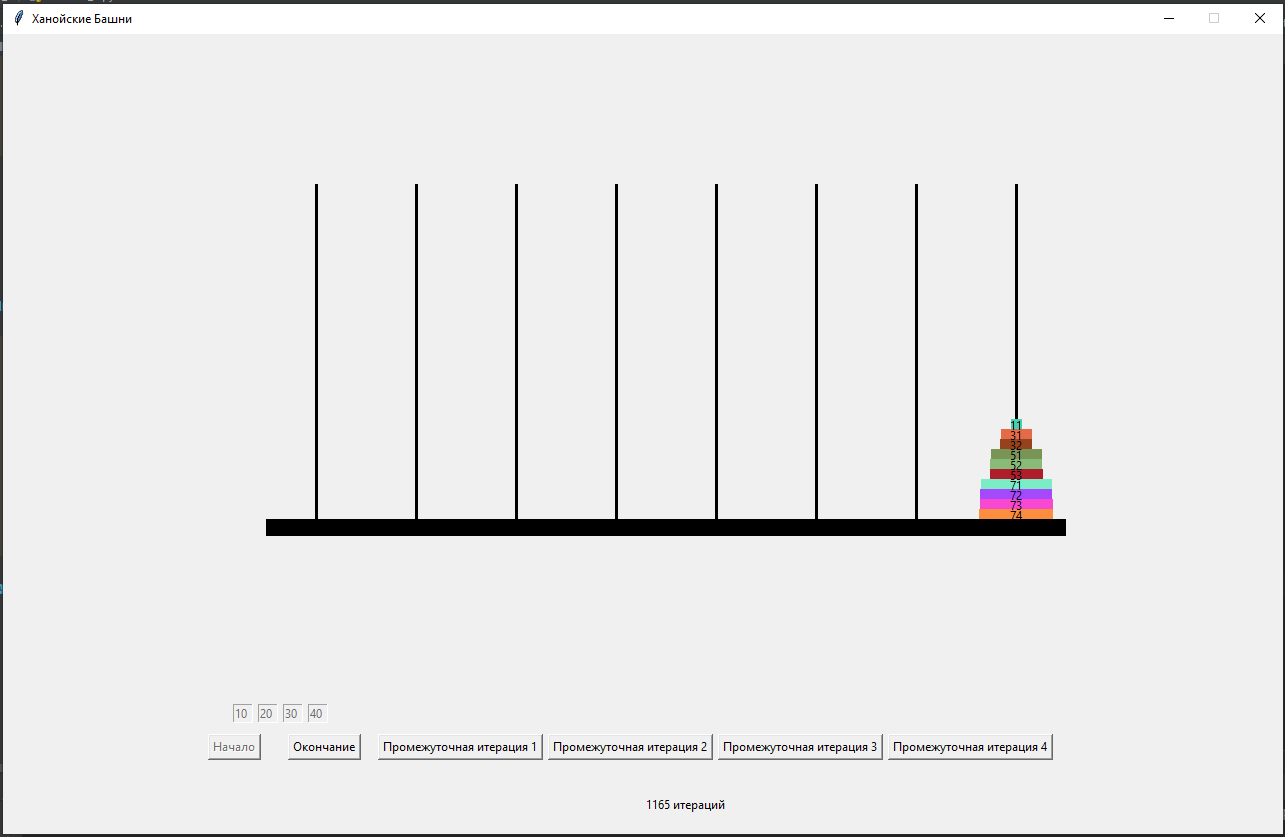
# Функция simple\_towers реализует алгоритм Ханойских башен  
# n - количество дисков, start - начальный стержень, finish - конечный стержень, sum\_rods - сумма номеров стержней  
# tower - список стержней, cnt - объект класса Cnt для подсчета количества ходов  
# canvas - объект Canvas для отрисовки графики, stop - список точек останова для записи результатов  
# results - словарь для сохранения результатов на определенных шагах, если record\_iterations=True  
def simple\_towers(  
 n: int,  
 start: int,  
 finish: int,  
 sum\_rods: int,  
 tower: list,  
 step: Step,  
 canvas,  
 stop,  
 results,  
):  
 global display\_number, last\_display\_time  
 # Если количество дисков меньше или равно 0, то выходим из функции  
 if n <= 0:  
 return  
 # Определяем промежуточный стержень  
 middle = sum\_rods - start - finish  
  
 # Вызываем рекурсию для n-1 диска и перемещаем его на промежуточный стержень  
 simple\_towers(n - 1, start, middle, sum\_rods, tower, step, canvas, stop, results)  
 # Увеличиваем счетчик ходов  
 step.count += 1  
 # Перемещаем верхний диск со стартового стержня на конечный стержень  
 tower[finish].append(tower[start][-1])  
 tower[start].pop(-1)  
 # Увеличиваем счетчик отображения результатов  
 display\_number += 1  
 # Если достигнуто CHECK\_EVERY ходов, то отображаем результаты на экране  
 if display\_number >= CHECK\_EVERY:  
 if time.time() - last\_display\_time >= DISPLAY\_EVERY\_N\_SEC:  
 draw\_frame(canvas, tower, None)  
 last\_display\_time = time.time()  
 display\_number = 0  
  
 # Если record\_iterations=True и текущий ход находится в списке точек остановки,  
 # то сохраняем результаты в словарь results  
 for stopPoint in stop:  
 if stopPoint - ANIMATION\_TOTAL\_MOVES <= step.count <= stopPoint:  
 results[stopPoint].append(  
 (deepcopy(tower), start, finish, tower[finish][-1])  
 )  
 # Вызываем рекурсию для прошлого диска и перемещаем его с промежуточного стержня на конечный стержень  
 simple\_towers(n - 1, middle, finish, sum\_rods, tower, step, canvas, stop, results)  
  
  
# Функция start\_hanoi запускает алгоритм программы для каждого стержня  
# theight - список стержней, cnt - объект класса Cnt для подсчета количества ходов,  
# canvas - объект Canvas для отрисовки графики,  
# record\_iterations - флаг для сохранения результатов на определенных шагах в словарь results  
  
  
def start\_hanoi(theight: list, cnt, canvas, record\_iterations):  
 stop = []  
 results = None  
 # Если record\_iterations=True, то определяем список точек останова и создаем пустой словарь c результатами  
 # для записи в последующих рекурсиях  
 if record\_iterations:  
 stop = tkState.stop  
 results = defaultdict(list)  
 # Запускаем алгоритм Ханойских башен для самого высокого стержня (8)  
 length = len(theight[8])  
 simple\_towers(length, 8, 7, 21, theight, cnt, canvas, stop, results)  
 # Запускаем алгоритм Ханойских башен для оставшихся стержней в порядке убывания (7-3)  
 for i in range(7, 2, -1):  
 length = len(theight[i])  
 simple\_towers(length, i, i - 1, 3 \* i, theight, cnt, canvas, stop, results)  
 length = len(theight[2])  
 # Запускаем алгоритм Ханойских башен для промежуточного стержня (2)  
 simple\_towers(length, 2, 1, 6, theight, cnt, canvas, stop, results)  
 # Отображаем результаты на экране и возвращаем словарь results (если он был создан)  
 draw\_frame(canvas, theight, None)  
  
 return results

Полный листинг программы предоставлен в приложении.

#### 2.4.3. Проектирование интерфейса приложения

Для достижения нужного результата в разработке пользовательского интерфейса был выбран виджет Canvas из библиотеки tkinter, который уже был использован ранее. Визуализация задачи состояла в создании прямоугольников нужных размеров и цветов и их размещении на виджете.

В интерфейсе приложения было важно изобразить сами башни с дисками, находящимися на штырях. Для взаимодействия пользователя с интерфейсом были созданы кнопки для выбора номера итерации, поля ввода данных и кнопка запуска программы.

Интерфейс готовой программы изображен ниже.

## 2.5. Вывод по главе 2

Python имеет обширную стандартную библиотеку, которая помогает разработчикам решать задачи различной сложности и направленности. Правильное использование алгоритмов и структур данных упрощает разработку, а модульное тестирование позволяет следить за качеством кода. Вокруг языка Python сложилась богатая инфраструктура, включая инструменты для отладки программного обеспечения. PEP 8 - руководство по написанию кода на Python, которое позволяет начинающим разработчикам писать понятный код для всего сообщества программистов на Python. Данное руководство использовалось при написании данной курсовой работы.

# 3. РАЗРАБОТКА ТРЕБОВАНИЙ К ТЕХНИЧЕСКИМ СРЕДСТВАМ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

Минимальный состав используемых технических средств:

Версия ОС: Windows 7 / 8 /10 (64bit)

Процессор: Intel Core 2 Duo 2.6 GHz / AMD Athlon 64 X2 6000+ Оперативная память: 4 Gb

Наличие свободного места на жестком диске более 500 Мбайт. Установленный интерпретатор языка Python> 3.7.

Для поддержки, дальнейшего развития и отладки рекомендуется использовать среду разработки PyCharm.

# Выводы

Основной целью данной курсовой работы являлось решение прикладных задач. Осуществлена попытка решения предложенных задач, в ходе решения которых были углублены знания языка Python и его инфраструктуры, были реализованы алгоритмы и графические пользовательские интерфейсы программ, использованы некоторые дополнительные библиотеки. Помимо этого, было уделено внимание тестированию и отладке программного кода, что является неотъемлемой частью разработки программного обеспечения. Использовано руководство PEP 8 для правильного оформления кода, призванного упростить поддержку и развитие программных продуктов.

Получившиеся в результате разработки программы успешно выполняются интерпретатором языка Python.

# СПИСОК ЛИТЕРАТУРЫ

1. PEP 8 – руководство по Python - <https://peps.python.org/pep-0008/>
2. Референсы для библиотеки tkinter - <https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/>
3. Документация по Python 3 - <https://docs.python.org/3/>

# ПРИЛОЖЕНИЯ

Приложение 1

Ссылка на репозиторий GitHub:

<https://github.com/helpme2022/kursovaya2022>