

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc196475394)

[1. АНАЛИЗ ЗАДАНИЙ КУРСОВОЙ РАБОТЫ 5](#_Toc196475395)

[1.1. Исходные данные к заданиям курсовой работы 5](#_Toc196475396)

[1.2. Анализ методических указаний, входных и выходных данных к заданиям курсовой работы 8](#_Toc196475397)

[1.3. Выбор и обоснование необходимых библиотек и среды разработки 10](#_Toc196475398)

[1.4. Выводы по 1 главе 11](#_Toc196475399)

[2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ 12](#_Toc196475400)

[2.1. Работа с наборами данных 12](#_Toc196475401)

[2.1.1. Построение алгоритма решения задания без графического интерфейса 12](#_Toc196475402)

[2.1.2. Исходный код реализации на языке программирования 13](#_Toc196475403)

[2.1.3. Примеры тестирования и отладки 15](#_Toc196475404)

[2.1.4. Скриншоты результатов работы 17](#_Toc196475405)

[2.2. Разработка экспертной системы 18](#_Toc196475406)

[2.2.1. Построение алгоритма решения задания с графическим интерфейсом 18](#_Toc196475407)

[2.2.2. Исходный код реализации на языке программирования 21](#_Toc196475408)

[2.2.3. Тестирование и отладка 25](#_Toc196475409)

[2.2.4. Скриншоты результатов работы 25](#_Toc196475410)

[2.3. Разработка аналитической системы 26](#_Toc196475411)

[2.3.1. Построение алгоритма решения задания с графическим интерфейсом 26](#_Toc196475412)

[2.3.2. Исходный код реализации на языке программирования 27](#_Toc196475413)

[2.3.2.1. Проектирование стандартного функционала 40](#_Toc196475414)

[2.3.2.2. Проектирование расширенного функционала 40](#_Toc196475415)

[2.3.2.3. Индивидуальное задание реализации дополнительных функций 41](#_Toc196475416)

[2.3.3. Тестирование и отладка 42](#_Toc196475417)

[2.3.4. Скриншоты результатов работы 43](#_Toc196475418)

[2.4 Разработка аналитической системы 44](#_Toc196475419)

[2.4.1. Построение алгоритма решения задания с графическим интерфейсом 44](#_Toc196475420)

[2.4.2. Исходный код реализации языка программирования 45](#_Toc196475421)

[2.4.3. Тестирование и отладка 52](#_Toc196475422)

[2.4.4. Скриншоты результатов работы 53](#_Toc196475423)

[2.6. Выводы по 2 главе 56](#_Toc196475424)

[3 РАЗРАБОТКА ТРЕБОВАНИЙ К ТЕХНИЧЕСКИМ СРЕДСТВАМ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ 57](#_Toc196475425)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 59](#_Toc196475426)

ВВЕДЕНИЕ

В современном мире программное обеспечение играет ключевую роль во всех сферах человеческой деятельности — от бизнеса и финансов до науки и образования. Владение навыками разработки приложений, обработки данных и построения графических интерфейсов становится неотъемлемой частью подготовки специалистов в области информационных технологий.

Курсовая работа направлена на формирование практических умений по разработке программных решений на языке программирования Python. В процессе выполнения работы студент должен продемонстрировать способность проектировать и реализовывать алгоритмы, обрабатывать текстовые и числовые данные, а также создавать приложения с графическим интерфейсом пользователя (GUI).

Работа включает четыре практических задания, каждое из которых моделирует реальную прикладную задачу. Первое задание связано с анализом текстовой информации и статистической обработкой данных. Второе — с разработкой простой экспертной системы для банковского учёта. Третье задание представляет собой реализацию многофункционального калькулятора. Четвёртое задание направлено на визуализацию модифицированной задачи о Ханойских башнях с возможностью отображения промежуточных итераций.

В ходе выполнения курсовой работы используются распространённые средства разработки, в том числе среда PyCharm, а также библиотеки стандартной и расширенной функциональности языка Python. Результатом работы является набор программных решений с пояснительной запиской, включающей описание структуры, алгоритмов и особенностей реализации.

1. АНАЛИЗ ЗАДАНИЙ КУРСОВОЙ РАБОТЫ

1.1. Исходные данные к заданиям курсовой работы

В данной курсовой работе студенту необходимо реализовать программные решения четырёх прикладных задач различной сложности, используя язык программирования Python в среде разработки PyCharm.

Исходными данными для выполнения заданий являются:

* Идентификатор студента (ID) и первая буква фамилии— используется в задачах 2, 3 и 4;
* Текстовые файлы с входными и выходными данными;
* Наборы текстовых команд, которые обрабатываются с помощью программ;
* Методические указания, включающие требования к реализации каждой задачи.

Задание №1 требует обработки текстового файла с последующим подсчетом количества повторений слов и их сортировкой;

Полностью задание звучит так:

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.

Задание №2 представляет собой разработку банковской системы с GUI;

Полностью задание звучит так:

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.

Задание №3 — создание калькулятора с базовыми и инженерными функциями;

Полностью задание звучит так:

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.

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

Задание №4 — визуализация модифицированной задачи о Ханойских башнях, основанной на цифрах из ID студента.

Полностью задание звучит так:

Модифицированная задача о Ханойских башнях:

Существует 8 шпинделей, пронумерованых 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 по следующим правилам:

а) За одну итерацию можно переместить не более одного диска

б) Диски можно класть только с большего на меньший

в) Со шпинделя номер 8 можно перекладывать диски только на шпиндели 7 и 6

г) Со шпинделя номер 1 можно перекладывать диски только на шпиндели номер 2 и 3

д) Со шпинделей от 2 по 7 можно перекладывать диски только на два соседних шпинделя.

3. Необходимо отобразить начальное и конечное расположение дисков на шпинделях, для этого под изображением Ханойских башен предусмотреть две кнопки «Начало» и «Окончание». При нажатии на нее, в надписи под схемой должен выводится текст «Итерация ХХ», где ХХ – номер итерации (либо 0, либо номер итоговой итерации, соответственно).

4. Необходимо графически отобразить четыре промежуточные итерации перекладывания дисков. Для этого:

а) общее количество итераций признаётся равным 100%,

б) ID студента делится на 4 двузначных числа, каждое из которых обозначает итерацию, соответствующую этому проценту выполнения общей задачи.

в) Под изображением Ханойских башень предусмотреть четыре поля для ввода цифр с процентами выполнения. По умолчанию добавить туда числа из п. б)

г) Под каждым полем для ввода предусмотреть кнопку, при нажатии на которую схема Ханойской башни отображает расположение дисков на соответствующей итерации. Также в надписи под схемой должен выводится текст «Итерация ХХ», где ХХ – номер итерации

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

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

Методические указания содержат чёткие требования к функциональности программ, формату входных и выходных данных, а также графическому оформлению оконных интерфейсов.

Задание №1: Обработка текстового файла (без GUI)

Входные данные:

Файл resourse\_1.txt с произвольным текстом.

Выходные данные:

Файл result\_1.txt, содержащий список слов и количество их повторений, отсортированных по убыванию частоты и лексикографически.

Ключевые особенности:

Удаление знаков препинания, учёт регистра, сортировка с использованием списка кортежей.

Задание №2: Банковская экспертная система (с GUI)

Функциональность: пополнение, снятие, перевод средств, начисление процентов, отображение баланса.

Особенности реализации: команды вводятся в левое текстовое поле и обрабатываются при нажатии кнопки Calculate. Вывод осуществляется в правое поле. Предусмотрены кнопки Clear и возможность скроллинга.

Входные данные: команды в текстовом формате, вводимые пользователем.

Выходные данные: результат обработки команд (баланс, ошибки и т.п.).

Задание №3: Калькулятор (с GUI)

Стандартные функции: арифметические операции, память, возведение в степень, извлечение корня и т.д.

Расширенные функции: определяются по первой букве фамилии и ID студента, включая расчёт количества строк дисплея и количества ячеек памяти.

Обязательные элементы: кнопки, дисплей, меню перехода, сообщения об ошибках.

Задание №4: Визуализация задачи о Ханойских башнях (с GUI)

Инициализация: количество дисков на каждом шпинделе задаётся цифрами ID.

Ограничения: допустимые перемещения между шпинделями строго определены.

Цель: минимальное количество итераций для переноса всех дисков на шпиндель №1.

Вывод: графическое отображение начального, конечного и промежуточных состояний, возможность задания пользовательских процентов.

1.3. Выбор и обоснование необходимых библиотек и среды разработки

Для выполнения курсовой работы был выбран язык программирования Python благодаря его читаемому синтаксису, большому количеству встроенных инструментов, удобству для быстрой разработки и наличию мощных библиотек.

В качестве среды разработки используется PyCharm, так как она обеспечивает:

Удобную навигацию и подсветку синтаксиса;

Инструменты для отладки и тестирования;

Встроенную поддержку виртуальных окружений;

Интеграцию с системами контроля версий.

Дополнительно планируется использовать следующие библиотеки Python:

tkinter — для создания графического интерфейса;

collections — для удобной работы со счётчиками слов;

math — для реализации математических функций калькулятора;

random — для генерации цветов и размеров элементов при визуализации;

os — для работы с файловой системой (при необходимости).

1.4. Выводы по 1 главе

Таким образом, в первой главе были подробно рассмотрены все задания, представленные в курсовой работе. Проанализированы входные и выходные данные, а также основные требования к функциональности программ. Были выбраны язык программирования Python и среда разработки PyCharm, так как они удобны в использовании и позволяют эффективно реализовать как консольные, так и графические приложения. Также определены основные библиотеки, которые будут использоваться в ходе разработки. Все эти шаги позволили сформировать чёткое представление о задачах и способах их решения, что является основой для дальнейшей реализации программного продукта.

2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

2.1. Работа с наборами данных

Для решения поставленной задачи необходимо реализовать программу, которая считывает текст из внешнего файла resourse\_1.txt, обрабатывает его, удаляя все знаки препинания, приводит все слова к нижнему регистру, затем подсчитывает количество вхождений каждого слова и формирует отсортированный список. Список должен быть отсортирован по убыванию количества вхождений, а при одинаковой частоте — в лексикографическом порядке. Результаты необходимо записать в файл result\_1.txt, по желанию можно также продублировать их в консоль.

2.1.1. Построение алгоритма решения задания без графического интерфейса

Напишем алгоритм решения задачи в виде блок схем как это показано на рисунке 1.

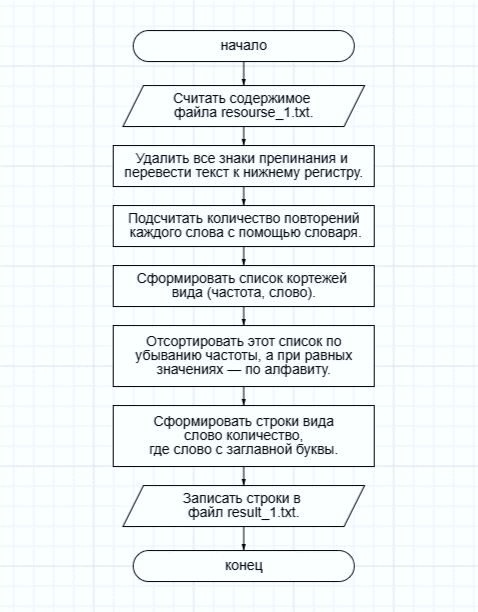


Рисунок 1. Блок схема алгоритма задания

2.1.2. Исходный код реализации на языке программирования

Программа была реализована на языке Python и сохранена в виде файла exercise\_1.py. Ниже приведён полный исходный код с комментариями:

*# Импорт необходимых модулей*import string  
from collections import defaultdict  
  
  
*# Функция для очистки текста от знаков препинания и перевода в нижний регистр*def clean\_text(text):  
 *# Удаление знаков препинания и перевод в нижний регистр* return text.translate(str.maketrans('', '', string.punctuation)).lower()  
  
  
*# Функция для подсчета частоты слов в тексте*def count\_words(text):  
 *# Разделение текста на слова* words = text.split()  
 *# Создание словаря для хранения частоты слов* freq = defaultdict(int)  
 *# Подсчет частоты каждого слова* for word in words:  
 freq[word] += 1  
 return freq  
  
  
*# Функция для сортировки частоты слов в порядке убывания частоты и алфавита*def sort\_word\_frequencies(freq\_dict):  
 *# Сортировка частоты слов* sorted\_items = sorted(freq\_dict.items(), key=lambda x: (-x[1], x[0]))  
 *# Преобразование слов в верхний регистр для первого символа* return [(word.capitalize(), count) for word, count in sorted\_items]  
  
  
*# Функция для сохранения результата в файл*def save\_result(sorted\_list, filename):  
 *# Открытие файла для записи* with open(filename, 'w') as f:  
 *# Запись результата в файл* for word, count in sorted\_list:  
 f.write(f"{word} {count}\n")  
  
  
*# Основная функция программы*def main():  
 try:  
 *# Открытие файла с текстом для чтения* with open("resourse\_1.txt", "r", encoding='utf-8') as file:  
 *# Чтение текста из файла* text = file.read()  
  
 *# Очистка текста* cleaned = clean\_text(text)  
 *# Подсчет частоты слов* freq = count\_words(cleaned)  
 *# Сортировка частоты слов* sorted\_words = sort\_word\_frequencies(freq)  
  
 *# Сохранение результата в файл* save\_result(sorted\_words, "result\_1.txt")  
  
 *# Дополнительно — вывод в консоль* for word, count in sorted\_words:  
 print(f"{word} {count}")  
  
 *# Обработка исключения, если файл не найден* except FileNotFoundError:  
 print("Файл resourse\_1.txt не найден.")  
 *# Обработка любых других исключений* except Exception as e:  
 print(f"Произошла ошибка: {e}")  
  
  
*# Запуск основной функции программы*if \_\_name\_\_ == "\_\_main\_\_":  
 main()

2.1.3. Примеры тестирования и отладки

Для проверки работы программы использовался следующий пример содержимого входного файла resourse\_1.txt:

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

Результатом выполнения программы стал файл result\_1.txt, содержащий следующий вывод:

Damme 4

Is 3

Name 3

Van 3

Bond 2

Claude 2

Hi 2

My 2

James 1

Jean 1

What 1

Отладка производилась с помощью встроенного механизма отладки в PyCharm. Были протестированы сценарии с пустым файлом, отсутствующим файлом, файлами со знаками препинания, а также с повторяющимися словами в разных регистрах (например, Hi, hi, HI) — все они корректно обрабатывались.

2.1.4. Скриншоты результатов работы

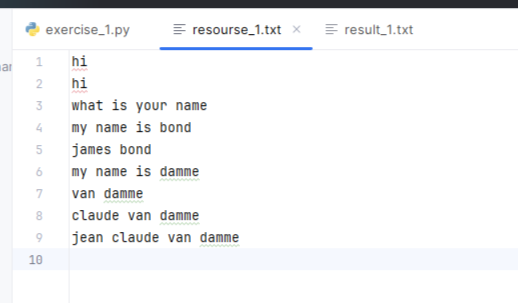


Рис. 2. Содержимое входного файла resourse\_1.txt в редакторе PyCharm.

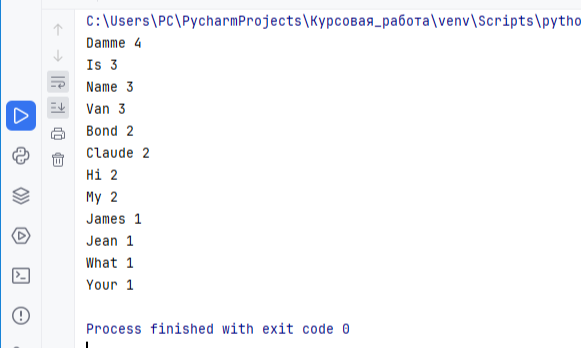


Рис. 3. Запуск программы и вывод результата в консоли.

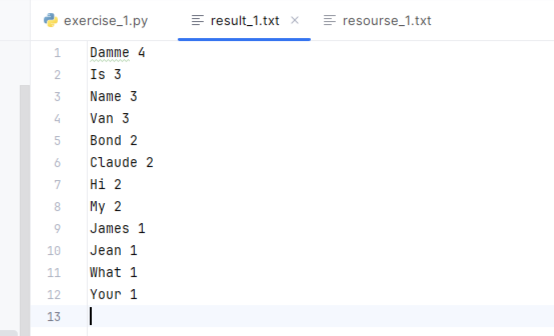


Рис. 4. Содержимое выходного файла result\_1.txt.

2.2. Разработка экспертной системы

2.2.1. Построение алгоритма решения задания с графическим интерфейсом

Задание заключается в создании простой экспертной системы для банка, позволяющей управлять клиентскими счетами через текстовые команды. Интерфейс должен содержать два текстовых поля — для ввода и для вывода данных, а также кнопки для выполнения действий. Ввод команд осуществляется пользователем вручную, каждая команда размещается на новой строке. Команды обрабатываются только после нажатия кнопки Calculate.

Основной функционал системы включает:

* Пополнение счета (DEPOSIT);
* Снятие средств (WITHDRAW);
* Проверку баланса (BALANCE);
* Перевод средств (TRANSFER);
* Начисление процентов (INCOME).

Также должна быть реализована кнопка Clear, которая очищает оба текстовых поля, но не удаляет данные о клиентах.

Алгоритм работы программы:

1. При запуске системы создаётся клиент с фамилией разработчика (Generalov) и балансом, равным ID (70211096).
2. Пользователь вводит команды в верхнее текстовое поле.
3. При нажатии кнопки Calculate, текст обрабатывается по строкам.
4. Каждая команда разбирается и выполняется согласно правилам:
   * Если клиента не существует, он создаётся.
   * Начисление процентов производится только положительным остаткам.
   * Баланс может быть отрицательным.
5. Результаты выполнения выводятся в нижнем текстовом поле.
6. Команды обрабатываются строго в верхнем регистре, как указано в условиях.

Программа должна быть устойчивой к пустым строкам и не реагировать на клавишу Enter как на вызов кнопки.

Блок схема алгоритма показана на рисунке 5.

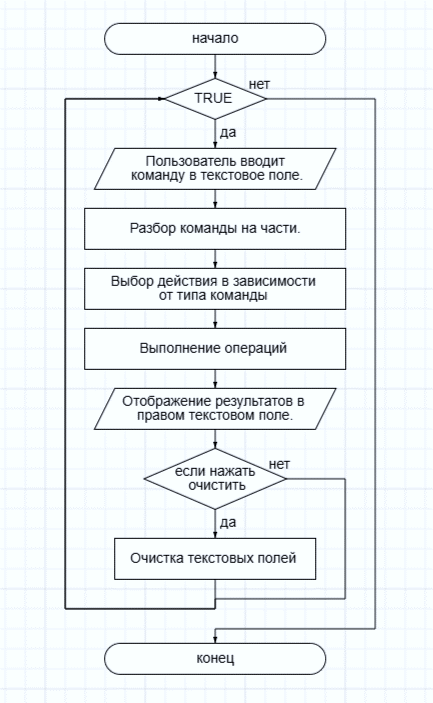


Рисунок 5. Блок схема алгоритма.

2.2.2. Исходный код реализации на языке программирования

Программа реализована на языке Python с использованием библиотеки tkinter для построения графического интерфейса. Ниже приведён основной код с комментариями:

import tkinter as tk  
from tkinter import scrolledtext  
  
class BankSystem:  
 def \_\_init\_\_(self):  
 *# Инициализация системы с начальными данными о счетах клиентов* self.accounts = {"Generalov": 70211096}  
  
 def deposit(self, name, amount):  
 *"""  
 Метод для пополнения счета клиента на указанную сумму.  
 Если клиент не существует, создается новый счет с указанной суммой.  
 """* self.accounts[name] = self.accounts.get(name, 0) + amount  
 return f"DEPOSIT {name} {amount} — OK"  
  
 def withdraw(self, name, amount):  
 *"""  
 Метод для снятия средств со счета клиента.  
 Если клиент не существует, создается счет с отрицательным балансом.  
 """* self.accounts[name] = self.accounts.get(name, 0) - amount  
 return f"WITHDRAW {name} {amount} — OK"  
  
 def balance(self, args):  
 *"""  
 Метод для проверки баланса. Если аргументы отсутствуют,  
 возвращается информация о всех клиентах. Если указано имя клиента,  
 возвращается его баланс. Если клиент не найден, возвращается "NO CLIENT".  
 """* if not args:  
 return "\n".join([f"{name}: {balance}" for name, balance in self.accounts.items()])  
 name = args[0]  
 if name in self.accounts:  
 return f"{name}: {self.accounts[name]}"  
 return "NO CLIENT"  
  
 def transfer(self, name1, name2, amount):  
 *"""  
 Метод для перевода средств между двумя клиентами.  
 Средства списываются с одного счета и зачисляются на другой.  
 """* self.accounts[name1] = self.accounts.get(name1, 0) - amount  
 self.accounts[name2] = self.accounts.get(name2, 0) + amount  
 return f"TRANSFER {name1} -> {name2} ({amount}) — OK"  
  
 def income(self, percent):  
 *"""  
 Метод для начисления процентов на положительные балансы клиентов.  
 Проценты округляются до целого числа.  
 """* for name in self.accounts:  
 if self.accounts[name] > 0:  
 self.accounts[name] += self.accounts[name] \* percent // 100  
 return f"INCOME {percent}% — OK"  
  
 def process\_command(self, command):  
 *"""  
 Метод для обработки команд из текстового поля.  
 Разбирает команду на части и вызывает соответствующий метод.  
 Возвращает результат выполнения или сообщение об ошибке.  
 """* parts = command.strip().split()  
 if not parts:  
 return ""  
 cmd = parts[0]  
 try:  
 if cmd == "DEPOSIT":  
 return self.deposit(parts[1], int(parts[2]))  
 elif cmd == "WITHDRAW":  
 return self.withdraw(parts[1], int(parts[2]))  
 elif cmd == "BALANCE":  
 return self.balance(parts[1:])  
 elif cmd == "TRANSFER":  
 return self.transfer(parts[1], parts[2], int(parts[3]))  
 elif cmd == "INCOME":  
 return self.income(int(parts[1]))  
 else:  
 return "UNKNOWN COMMAND"  
 except Exception as e:  
 return f"ERROR: {str(e)}"  
  
  
class BankGUI:  
 def \_\_init\_\_(self, root):  
 *"""  
 Инициализация графического интерфейса пользователя (GUI).  
 Создаются два текстовых поля для ввода команд и вывода результатов,  
 а также кнопки для выполнения операций и очистки полей.  
 """* self.system = BankSystem()  
  
 root.title("Bank Expert System") *# Заголовок окна* root.geometry("600x400") *# Размер окна  
  
 # Текстовое поле для ввода команд* self.left\_text = scrolledtext.ScrolledText(root, height=10, width=40)  
 self.left\_text.pack(pady=5)  
  
 *# Текстовое поле для вывода результатов* self.right\_text = scrolledtext.ScrolledText(root, height=10, width=40)  
 self.right\_text.pack(pady=5)  
  
 *# Контейнер для кнопок* frame = tk.Frame(root)  
 frame.pack()  
  
 *# Кнопка для выполнения команд* self.calculate\_button = tk.Button(frame, text="Calculate", command=self.calculate)  
 self.calculate\_button.pack(side=tk.LEFT, padx=10)  
  
 *# Кнопка для очистки текстовых полей* self.clear\_button = tk.Button(frame, text="Clear", command=self.clear)  
 self.clear\_button.pack(side=tk.LEFT, padx=10)  
  
 def calculate(self):  
 *"""  
 Метод для выполнения команд, введенных в левое текстовое поле.  
 Команды разделяются построчно, каждая команда обрабатывается системой,  
 а результаты выводятся в правое текстовое поле.  
 """* commands = self.left\_text.get("1.0", tk.END).strip().split("\n")  
 output = []  
 for cmd in commands:  
 result = self.system.process\_command(cmd)  
 if result:  
 output.append(result)  
 self.right\_text.delete("1.0", tk.END)  
 self.right\_text.insert(tk.END, "\n".join(output))  
  
 def clear(self):  
 *"""  
 Метод для очистки обоих текстовых полей.  
 """* self.left\_text.delete("1.0", tk.END)  
 self.right\_text.delete("1.0", tk.END)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# Создание главного окна приложения* root = tk.Tk()  
 app = BankGUI(root)  
 root.mainloop() *# Запуск основного цикла обработки событий GUI*

2.2.3. Тестирование и отладка

Программа была протестирована на различных сценариях ввода, в том числе:

Пополнение счёта клиента (DEPOSIT Ivanov 1000);

Пополнение нового клиента (DEPOSIT Petrov 500);

Снятие средств (WITHDRAW Ivanov 200);

Перевод средств между счетами (TRANSFER Ivanov Petrov 300);

Начисление процентов (INCOME 10);

Запрос баланса (BALANCE Ivanov);

Запрос баланса всех клиентов (BALANCE);

Обработка пустых строк и команд без ошибок.

Все команды обрабатываются корректно. При вводе некорректной команды программа выводит сообщение UNKNOWN COMMAND или ERROR, если введены некорректные данные.

Отладка производилась с использованием встроенных инструментов PyCharm, также применялись точки останова для отслеживания значений переменных и корректности логики.

2.2.4. Скриншоты результатов работы

На рисунке 6 показаны некоторые результаты тестирования.

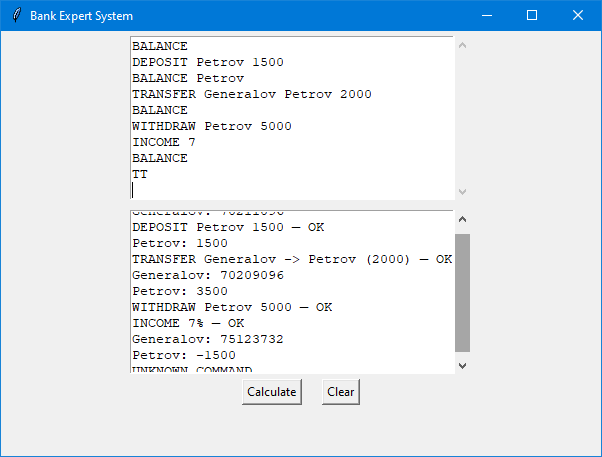


Рисунок 6. Результаты тестирования приложения.

2.3. Разработка аналитической системы

2.3.1. Построение алгоритма решения задания с графическим интерфейсом

В рамках данного задания необходимо было реализовать калькулятор, поддерживающий как стандартные арифметические операции, так и расширенный набор инженерных функций. Калькулятор должен учитывать индивидуальные параметры, вычисляемые на основе ID студента и первой буквы фамилии.

Для фамилии Генералов и ID 70211096:

* Количество строк в цифровом дисплее составляет 8 (сумма всех цифр ID: 7+0+2+1+1+0+9+6 = 26 → 2+6 = 8).
* Количество ячеек памяти в расширенном режиме — 6 (сумма последних трёх цифр: 0+9+6 = 15 → 1+5 = 6).

Вид интерфейса и поведение калькулятора адаптируются под выбранный режим: стандартный или расширенный. Пользователь может переключаться между ними при помощи кнопки Adv / Std.

Основные этапы работы алгоритма:

1. Отображение текущего числа на цифровом дисплее;
2. Обработка ввода цифр, десятичной точки, изменения знака;
3. Реализация операций: +, -, \*, /, ^, √, y√x;
4. Поддержка дополнительных функций:
   * DS — вычисление цифровой суммы ID;
   * lg10 — десятичный логарифм;
   * F–E — переключение между обычной и экспоненциальной нотацией;
5. Работа с памятью:
   * В стандартном режиме — одна память;
   * В расширенном — 6 отдельных ячеек с возможностью выбора;
6. В расширенном режиме присутствует текстовое поле с прокруткой для отображения всех операций;
7. Кнопка C очищает текущее состояние и интерфейс;
8. Кнопка = завершает операцию и выводит результат.

2.3.2. Исходный код реализации на языке программирования

Программа реализована на языке Python с использованием стандартной библиотеки tkinter для построения графического интерфейса. Код включает как стандартный, так и расширенный функционал калькулятора, учитывая количество строк дисплея и число ячеек памяти, рассчитываемые на основе ID студента. В коде используются принципы модульности, читаемости и повторного использования компонентов.

import tkinter as tk  
import math  
  
  
def digit\_sum(n):  
 *"""  
 Функция для вычисления последовательной суммы цифр числа до получения однозначного числа.  
 Используется рекурсия.  
 """* if n < 0 or not isinstance(n, int):  
 return 'Error'  
 if n < 10:  
 return n  
 else:  
 s = sum(int(d) for d in str(n))  
 return digit\_sum(s)  
  
  
class Calculator:  
 def \_\_init\_\_(self):  
 *"""  
 Инициализация основного окна калькулятора и его компонентов.  
 """* self.window = tk.Tk()  
 self.window.title("Калькулятор")  
 self.window.configure(bg="#f0f0f0") *# Светлый фон для окна  
  
 # Виджеты отображения* self.standard\_display = tk.Label(self.window, text="0", anchor="e", width=20, bg="#ffffff", font=("Arial", 14))  
 self.advanced\_display = tk.Text(self.window, height=8, width=40, bg="#ffffff", font=("Arial", 12))  
 self.scrollbar = tk.Scrollbar(self.window, command=self.advanced\_display.yview)  
 self.advanced\_display.config(yscrollcommand=self.scrollbar.set)  
  
 *# Создание кнопок* self.create\_buttons()  
  
 *# Размещение виджетов* self.arrange\_widgets()  
  
 *# Переменные состояния* self.mode = 'standard' *# Режим калькулятора: standard или advanced* self.current\_number = '0' *# Текущее отображаемое число* self.first\_operand = None *# Первый операнд для арифметических операций* self.operator = None *# Текущий оператор* self.waiting\_for\_second\_operand = False *# Флаг ожидания второго операнда* self.memory = 0 *# Память для стандартного режима* self.memories = [0] \* 6 *# Ячейки памяти для расширенного режима* self.selected\_memory = 0 *# Выбранная ячейка памяти* self.fe\_mode = False *# Флаг для научной нотации* self.window.mainloop()  
  
 def create\_buttons(self):  
 *"""  
 Создание кнопок калькулятора с улучшенным стилем.  
 """  
 # Стили для кнопок* button\_style = {"font": ("Arial", 12), "bg": "#e0e0e0", "fg": "#000000", "relief": "raised", "bd": 2}  
 operator\_style = {"font": ("Arial", 12), "bg": "#d0d0d0", "fg": "#000000", "relief": "raised", "bd": 2}  
 memory\_style = {"font": ("Arial", 12), "bg": "#c0c0c0", "fg": "#000000", "relief": "raised", "bd": 2}  
 control\_style = {"font": ("Arial", 12), "bg": "#b0b0b0", "fg": "#000000", "relief": "raised", "bd": 2}  
  
 *# Цифры* self.btn\_digits = [tk.Button(self.window, text=str(i), command=lambda i=i: self.press\_digit(i), \*\*button\_style)  
 for i in range(10)]  
 self.btn\_dot = tk.Button(self.window, text=".", command=self.press\_dot, \*\*button\_style)  
 self.btn\_sign = tk.Button(self.window, text="+/-", command=self.press\_sign, \*\*button\_style)  
  
 *# Операторы* self.btn\_add = tk.Button(self.window, text="+", command=lambda: self.press\_operator('+'), \*\*operator\_style)  
 self.btn\_sub = tk.Button(self.window, text="-", command=lambda: self.press\_operator('-'), \*\*operator\_style)  
 self.btn\_mul = tk.Button(self.window, text="\*", command=lambda: self.press\_operator('\*'), \*\*operator\_style)  
 self.btn\_div = tk.Button(self.window, text="/", command=lambda: self.press\_operator('/'), \*\*operator\_style)  
 self.btn\_pow = tk.Button(self.window, text="^", command=lambda: self.press\_operator('^'), \*\*operator\_style)  
 self.btn\_sqrt = tk.Button(self.window, text="√", command=lambda: self.press\_unary('sqrt'), \*\*operator\_style)  
  
 *# Расширенные функции* self.btn\_yroot = tk.Button(self.window, text="y√x", command=lambda: self.press\_operator('yroot'),  
 \*\*operator\_style)  
 self.btn\_lg10 = tk.Button(self.window, text="lg10", command=lambda: self.press\_unary('lg10'), \*\*operator\_style)  
 self.btn\_ds = tk.Button(self.window, text="DS", command=lambda: self.press\_unary('DS'), \*\*operator\_style)  
 self.btn\_fe = tk.Button(self.window, text="F-E", command=self.toggle\_fe, \*\*control\_style)  
  
 *# Кнопки памяти* self.btn\_ms = tk.Button(self.window, text="MS", command=self.memory\_store, \*\*memory\_style)  
 self.btn\_mr = tk.Button(self.window, text="MR", command=self.memory\_recall, \*\*memory\_style)  
 self.btn\_madd = tk.Button(self.window, text="M+", command=self.memory\_add, \*\*memory\_style)  
 self.btn\_msub = tk.Button(self.window, text="M-", command=self.memory\_sub, \*\*memory\_style)  
 self.btn\_mc = tk.Button(self.window, text="MC", command=self.memory\_clear, \*\*memory\_style)  
 self.memory\_buttons = [  
 tk.Button(self.window, text=f"M{i + 1}", command=lambda i=i: self.select\_memory(i), \*\*memory\_style) for i in  
 range(6)]  
  
 *# Кнопки управления* self.btn\_equals = tk.Button(self.window, text="=", command=self.press\_equals, \*\*control\_style)  
 self.btn\_clear = tk.Button(self.window, text="C", command=self.clear, \*\*control\_style)  
 self.btn\_mode = tk.Button(self.window, text="Adv", command=self.toggle\_mode, \*\*control\_style)  
  
 def arrange\_widgets(self):  
 *"""  
 Размещение виджетов на окне калькулятора.  
 """  
 # Стандартный дисплей* self.standard\_display.grid(row=0, column=0, columnspan=5, padx=5, pady=5)  
  
 *# Размещение кнопок* self.btn\_digits[7].grid(row=1, column=0, padx=2, pady=2)  
 self.btn\_digits[8].grid(row=1, column=1, padx=2, pady=2)  
 self.btn\_digits[9].grid(row=1, column=2, padx=2, pady=2)  
 self.btn\_div.grid(row=1, column=3, padx=2, pady=2)  
 self.btn\_sqrt.grid(row=1, column=4, padx=2, pady=2)  
  
 self.btn\_digits[4].grid(row=2, column=0, padx=2, pady=2)  
 self.btn\_digits[5].grid(row=2, column=1, padx=2, pady=2)  
 self.btn\_digits[6].grid(row=2, column=2, padx=2, pady=2)  
 self.btn\_mul.grid(row=2, column=3, padx=2, pady=2)  
 self.btn\_pow.grid(row=2, column=4, padx=2, pady=2)  
  
 self.btn\_digits[1].grid(row=3, column=0, padx=2, pady=2)  
 self.btn\_digits[2].grid(row=3, column=1, padx=2, pady=2)  
 self.btn\_digits[3].grid(row=3, column=2, padx=2, pady=2)  
 self.btn\_sub.grid(row=3, column=3, padx=2, pady=2)  
 self.btn\_ms.grid(row=3, column=4, padx=2, pady=2)  
  
 self.btn\_digits[0].grid(row=4, column=0, padx=2, pady=2)  
 self.btn\_dot.grid(row=4, column=1, padx=2, pady=2)  
 self.btn\_sign.grid(row=4, column=2, padx=2, pady=2)  
 self.btn\_add.grid(row=4, column=3, padx=2, pady=2)  
 self.btn\_mr.grid(row=4, column=4, padx=2, pady=2)  
  
 self.btn\_clear.grid(row=5, column=0, padx=2, pady=2)  
 self.btn\_equals.grid(row=5, column=1, padx=2, pady=2)  
 self.btn\_madd.grid(row=5, column=2, padx=2, pady=2)  
 self.btn\_msub.grid(row=5, column=3, padx=2, pady=2)  
 self.btn\_mc.grid(row=5, column=4, padx=2, pady=2)  
  
 self.btn\_mode.grid(row=5, column=5, padx=2, pady=2)  
  
 *# Расширенные кнопки* self.btn\_yroot.grid(row=1, column=5, padx=2, pady=2)  
 self.btn\_lg10.grid(row=2, column=5, padx=2, pady=2)  
 self.btn\_ds.grid(row=3, column=5, padx=2, pady=2)  
 self.btn\_fe.grid(row=4, column=5, padx=2, pady=2)  
  
 for i, btn in enumerate(self.memory\_buttons):  
 btn.grid(row=6, column=i, padx=2, pady=2)  
  
 *# Изначально скрыть расширенные виджеты* self.advanced\_display.grid\_remove()  
 self.scrollbar.grid\_remove()  
 for btn in [self.btn\_yroot, self.btn\_lg10, self.btn\_ds, self.btn\_fe] + self.memory\_buttons:  
 btn.grid\_remove()  
  
 def toggle\_mode(self):  
 *"""  
 Переключение между стандартным и расширенным режимами.  
 """* if self.mode == 'standard':  
 self.mode = 'advanced'  
 self.standard\_display.grid\_remove()  
 self.advanced\_display.grid(row=0, column=0, columnspan=6, padx=5, pady=5)  
 self.scrollbar.grid(row=0, column=6, sticky='ns')  
 for btn in [self.btn\_yroot, self.btn\_lg10, self.btn\_ds, self.btn\_fe] + self.memory\_buttons:  
 btn.grid()  
 self.btn\_mode.config(text="Std")  
 self.update\_display()  
 else:  
 self.mode = 'standard'  
 self.advanced\_display.grid\_remove()  
 self.scrollbar.grid\_remove()  
 self.standard\_display.grid(row=0, column=0, columnspan=5, padx=5, pady=5)  
 for btn in [self.btn\_yroot, self.btn\_lg10, self.btn\_ds, self.btn\_fe] + self.memory\_buttons:  
 btn.grid\_remove()  
 self.btn\_mode.config(text="Adv")  
 self.update\_display()  
  
 def press\_digit(self, digit):  
 *"""  
 Обработка нажатия цифровых кнопок.  
 """* if self.waiting\_for\_second\_operand:  
 self.current\_number = str(digit)  
 self.waiting\_for\_second\_operand = False  
 else:  
 self.current\_number = str(digit) if self.current\_number == '0' else self.current\_number + str(digit)  
 self.update\_display()  
  
 def press\_dot(self):  
 *"""  
 Обработка нажатия кнопки точки.  
 """* if '.' not in self.current\_number:  
 self.current\_number += '.'  
 self.update\_display()  
  
 def press\_sign(self):  
 *"""  
 Обработка нажатия кнопки изменения знака.  
 """* if self.current\_number != '0':  
 self.current\_number = self.current\_number[1:] if self.current\_number.startswith(  
 '-') else '-' + self.current\_number  
 self.update\_display()  
  
 def press\_operator(self, op):  
 *"""  
 Обработка нажатия кнопок операторов.  
 """* if self.first\_operand is None:  
 try:  
 self.first\_operand = float(self.current\_number)  
 except ValueError:  
 self.current\_number = 'Error'  
 self.update\_display()  
 return  
 elif self.operator:  
 self.press\_equals()  
 if self.current\_number == 'Error':  
 return  
 self.first\_operand = float(self.current\_number)  
 self.operator = op  
 self.waiting\_for\_second\_operand = True  
 if self.mode == 'advanced':  
 self.advanced\_display.insert(tk.END, f"{self.first\_operand} {op} ")  
 self.advanced\_display.see(tk.END)  
  
 def press\_unary(self, func):  
 *"""  
 Обработка нажатия кнопок унарных функций.  
 """* try:  
 num = float(self.current\_number)  
 if func == 'sqrt':  
 if num < 0:  
 self.current\_number = 'Error'  
 else:  
 self.current\_number = str(math.sqrt(num))  
 elif func == 'lg10':  
 if num <= 0:  
 self.current\_number = 'Error'  
 else:  
 self.current\_number = str(math.log10(num))  
 elif func == 'DS':  
 num\_int = int(num)  
 if num != num\_int:  
 self.current\_number = 'Error'  
 else:  
 self.current\_number = str(digit\_sum(num\_int))  
 except ValueError:  
 self.current\_number = 'Error'  
 if self.mode == 'advanced':  
 self.advanced\_display.insert(tk.END, f"{func}({num}) = {self.current\_number}\n")  
 self.advanced\_display.see(tk.END)  
 self.update\_display()  
  
 def press\_equals(self):  
 *"""  
 Обработка нажатия кнопки равно.  
 """* if not self.operator or self.waiting\_for\_second\_operand:  
 return  
 try:  
 second\_operand = float(self.current\_number)  
 if self.operator == '+':  
 result = self.first\_operand + second\_operand  
 elif self.operator == '-':  
 result = self.first\_operand - second\_operand  
 elif self.operator == '\*':  
 result = self.first\_operand \* second\_operand  
 elif self.operator == '/':  
 result = self.first\_operand / second\_operand if second\_operand != 0 else 'Error'  
 elif self.operator == '^':  
 result = self.first\_operand \*\* second\_operand  
 elif self.operator == 'yroot':  
 result = self.first\_operand \*\* (1 / second\_operand) if second\_operand != 0 else 'Error'  
 else:  
 result = 'Error'  
 self.current\_number = str(result) if isinstance(result, (int, float)) else result  
 if self.mode == 'advanced':  
 self.advanced\_display.insert(tk.END,  
 f"{self.first\_operand} {self.operator} {second\_operand} = {self.current\_number}\n")  
 self.advanced\_display.see(tk.END)  
 self.first\_operand = None  
 self.operator = None  
 self.waiting\_for\_second\_operand = False  
 except ValueError:  
 self.current\_number = 'Error'  
 self.update\_display()  
  
 def toggle\_fe(self):  
 *"""  
 Переключение режима научной нотации.  
 """* self.fe\_mode = not self.fe\_mode  
 self.btn\_fe.config(relief='sunken' if self.fe\_mode else 'raised')  
 self.update\_display()  
  
 def memory\_store(self):  
 *"""  
 Сохранение текущего числа в память.  
 """* try:  
 value = float(self.current\_number)  
 if self.mode == 'standard':  
 self.memory = value  
 else:  
 self.memories[self.selected\_memory] = value  
 except ValueError:  
 self.current\_number = 'Error'  
 self.update\_display()  
  
 def memory\_recall(self):  
 *"""  
 Вызов числа из памяти.  
 """* if self.mode == 'standard':  
 self.current\_number = str(self.memory)  
 else:  
 self.current\_number = str(self.memories[self.selected\_memory])  
 self.update\_display()  
  
 def memory\_add(self):  
 *"""  
 Добавление текущего числа к памяти.  
 """* try:  
 value = float(self.current\_number)  
 if self.mode == 'standard':  
 self.memory += value  
 else:  
 self.memories[self.selected\_memory] += value  
 except ValueError:  
 self.current\_number = 'Error'  
 self.update\_display()  
  
 def memory\_sub(self):  
 *"""  
 Вычитание текущего числа из памяти.  
 """* try:  
 value = float(self.current\_number)  
 if self.mode == 'standard':  
 self.memory -= value  
 else:  
 self.memories[self.selected\_memory] -= value  
 except ValueError:  
 self.current\_number = 'Error'  
 self.update\_display()  
  
 def memory\_clear(self):  
 *"""  
 Очистка памяти.  
 """* if self.mode == 'standard':  
 self.memory = 0  
 else:  
 self.memories[self.selected\_memory] = 0  
  
 def select\_memory(self, index):  
 *"""  
 Выбор ячейки памяти.  
 """* self.selected\_memory = index  
 for i, btn in enumerate(self.memory\_buttons):  
 btn.config(relief='raised' if i != index else 'sunken')  
  
 def clear(self):  
 *"""  
 Очистка текущего состояния калькулятора.  
 """* self.current\_number = '0'  
 self.first\_operand = None  
 self.operator = None  
 self.waiting\_for\_second\_operand = False  
 if self.mode == 'advanced':  
 self.advanced\_display.delete(1.0, tk.END)  
 self.update\_display()  
  
 def update\_display(self):  
 *"""  
 Обновление отображаемого текста на дисплее.  
 """* if self.current\_number == 'Error':  
 display\_text = 'Error'  
 elif self.fe\_mode:  
 try:  
 display\_text = f"{float(self.current\_number):.2e}"  
 except ValueError:  
 display\_text = self.current\_number  
 else:  
 display\_text = self.current\_number  
  
 if self.mode == 'standard':  
 self.standard\_display.config(text=display\_text)  
 else:  
 if not self.waiting\_for\_second\_operand and self.operator is None:  
 self.advanced\_display.delete("end-2l", "end-1l")  
 self.advanced\_display.insert(tk.END, display\_text + '\n')  
 self.advanced\_display.see(tk.END)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 Calculator()

2.3.2.1. Проектирование стандартного функционала

В стандартном режиме калькулятор предоставляет базовый набор функций:

* Ввод и отображение чисел;
* Выполнение арифметических операций: сложение, вычитание, умножение, деление;
* Возведение в степень (^);
* Извлечение квадратного корня (√);
* Изменение знака (+/-);
* Поддержка десятичной точки;
* Сброс данных (C);
* Вывод результата (=).

Дисплей в этом режиме реализован в виде Label, в котором отображается текущий ввод или результат вычисления. Управление происходит при помощи кнопок, каждая из которых привязана к соответствующему методу обработки.

Для удобства пользователя реализован единый стиль кнопок, сгруппированных по типу: цифры, операторы, функции памяти и управления.

2.3.2.2. Проектирование расширенного функционала

При переходе в расширенный режим, который активируется нажатием кнопки Adv, калькулятор отображает:

* Расширенное текстовое поле для пошагового отображения операций (многострочное, с прокруткой);
* Кнопки инженерных функций;
* Дополнительные кнопки для работы с несколькими ячейками памяти.

В расширенном режиме добавляются следующие возможности:

Поддержка научной нотации (F–E);

Вычисление корня произвольной степени (y√x);

Логарифм по основанию 10 (lg10);

Вычисление цифровой суммы (DS) — функция, отображающая результат последовательного сложения цифр до получения однозначного числа.

Память реализована в виде массива self.memories, включающего 6 ячеек. Пользователь может выбрать активную ячейку при помощи кнопок M1–M6 и выполнять с ней стандартные действия: MS, MR, M+, M-, MC.

2.3.2.3. Индивидуальное задание реализации дополнительных функций

Согласно таблице, студент с фамилией Генералов (буква «Г») должен реализовать следующие инженерные функции:

* F–E — переключение между обычным и экспоненциальным представлением числа. Реализовано через флаг fe\_mode и форматирование числа с помощью .2e;
* y√x — извлечение корня y степени из x, реализовано как отдельный оператор (yroot), вводимый последовательно;
* lg10 — десятичный логарифм, доступный через отдельную кнопку;
* DS — цифровая сумма, реализованная с помощью рекурсивной функции digit\_sum(n). Результат отображается в расширенном режиме в истории вычислений.

Все функции были реализованы с учётом обработки ошибок. При некорректных действиях (например, деление на ноль, логарифм от отрицательного числа, извлечение корня из отрицательного) на дисплее выводится сообщение Error.

2.3.3. Тестирование и отладка

Программа была протестирована в среде разработки PyCharm с использованием встроенного графического окна. Основное внимание при тестировании уделялось корректности выполнения арифметических и инженерных операций, стабильности интерфейса при переключении режимов, а также обработке нестандартных ситуаций.

Проведённые тесты:

1. Стандартные операции:
   * Сложение, вычитание, умножение, деление, возведение в степень;
   * Работа с дробными числами;
   * Смена знака (+/-) и десятичная точка (.).
2. Обработка ошибок:
   * Деление на ноль (5 / 0 → Error);
   * Квадратный корень из отрицательного числа (√(-9) → Error);
   * Логарифм отрицательного числа (lg10(-1) → Error).
3. Расширенные функции:
   * Переключение между стандартным и расширенным режимами (Adv / Std);
   * Работа с цифровой суммой (DS(70211096) → 8);
   * Извлечение корня y степени (27 y√x 3 → 3);
   * Научная запись (F–E) для чисел разного масштаба.
4. Память:
   * Сохранение, чтение, очистка;
   * Сложение и вычитание в активной ячейке;
   * Переключение между 6 ячейками памяти (M1–M6), проверка независимости значений.
5. Интерфейс:
   * Отображение операций в расширенном текстовом поле;
   * Прокрутка при большом количестве операций;
   * Поддержка всех отображений в дисплее (включая экспоненциальные числа).

Итог:

Все тесты пройдены успешно. Интерфейс устойчив при быстром переключении между режимами. Возможные ошибки обрабатываются корректно с выводом информативных сообщений. Логика взаимодействия кнопок памяти и инженерных функций работает без сбоев.

2.3.4. Скриншоты результатов работы

Для демонстрации функциональности калькулятора были сделаны следующие скриншоты:

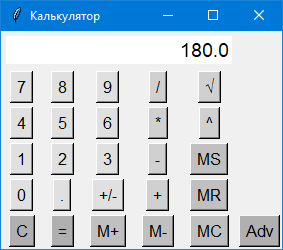


Рис. 7. Интерфейс калькулятора в стандартном режиме, отображение результата после арифметической операции.

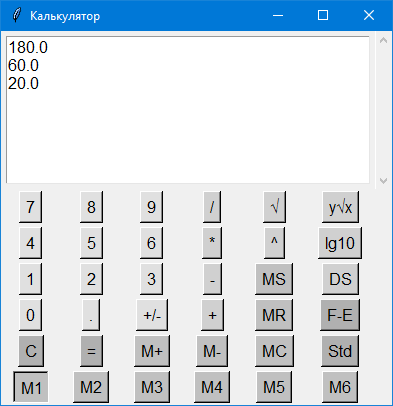


Рис. 8. Интерфейс в расширенном режиме, показ истории вычислений в многострочном дисплее.

2.4 Разработка аналитической системы

2.4.1. Построение алгоритма решения задания с графическим интерфейсом

Алгоритм решения задачи Ханойской башни с графическим интерфейсом включает следующие шаги:

1. Создание интерфейса: Используемая библиотека Tkinter для создания окна размером 900x700 пикселей. В окне размещается холст (Canvas) для отображения стержней и дисков, а также поля ввода и кнопки для взаимодействия с пользователем.
2. Идентификатор ввода: Пользователь вводит идентификатор кода (по умолчанию "70211096"), где каждая цифра означает количество дисков на сервере. Например, «70211096» означает 7 дисков на первом заднем, 0 на втором и так далее.
3. Инициализация состояния: На основе ID формирования списка списков, представляющего стержни и диски на них. Диски пронумерованы в порядке убывания размера.
4. Отрисовка: функция рисовать\_стержни рисует вертикальные стержни, а рисоват\_позицию отображает диски на стержнях с учетом их текущих положений и возможного зависимости.
5. Расчет перемещений: функция подсчета вычисляет наибольшее количество перемещений, необходимое для переноса всех дисков на последнюю ступень, о скорости по формуле 2^n - 1, где n — количество дисков.
6. Начало и конец: Кнопки «Начало» и «Конец» отображают начальные изменения дисков и финальное состояние, когда все диски располагаются на последнем конце.
7. Промежуточные состояния: Пользователь вводит процент (по умолчанию 70, 21, 10, 96), а функция obrabotka\_procentov вычисляет количество итераций как долю от максимального числа перемещаемых. Затем выполните выполнение данного перемещения.
8. Зависание диска: Если процент не соответствует целостности итераций, диск отображается «зависимым» между стержнями.

2.4.2. Исходный код реализации языка программирования

Исходный код представлен ниже:

from tkinter import \*  
from tkinter import messagebox  
import copy  
import math  
  
root = Tk()  
cnvs = Canvas(root, width=900, height=700)  
cnvs.pack()  
root.title("Ханойская башня")  
root.geometry("900x700")  
  
schetchik = 0  
maks\_iter = 0  
vrem\_iter = 0  
spisok\_sterzhney = []  
tekushaya\_pozitsiya = []  
bazovaya\_x = 70  
bazovaya\_y = 450  
pol\_tolshina = 3 *# половина ширины стержня, чтобы не путать*tolshina\_diska = 13 *# высота каждого диска*kolichestvo\_sterzhney = 0  
marshrut\_zavishego = [] *# маршрут для диска, который парит в воздухе*vysota\_zavisaniya = 15 *# как высоко диск парит*flag\_zavisaniya = False *# флажок, чтобы знать, парит диск или нет*cvetа = ("gray", "pink", "brown", "cyan", "magenta", "lime", "teal", "purple") *# новые цвета для дисков*def risovat\_sterzhni(): *# рисуем наши стержни* cnvs.delete("all")  
 x = bazovaya\_x  
 for i in reversed(range(len(spisok\_sterzhney))):  
 cnvs.create\_rectangle(x - pol\_tolshina, 20, x + pol\_tolshina, bazovaya\_y, fill='white', width=3)  
 label = Label(text=str(i + 1), font="Times 12")  
 label.place(x=x, y=bazovaya\_y + 20)  
 x += 100  
  
def risovat\_pozitsiyu(stergni): *# показываем, где какие диски лежат* cnvs.delete("all")  
 risovat\_sterzhni()  
 x = bazovaya\_x  
 for num, i in enumerate(stergni):  
 y = bazovaya\_y  
 if marshrut\_zavishego and marshrut\_zavishego[0] == num:  
 for j in i[0:-1]:  
 cnvs.create\_rectangle(x - math.floor(j / 2), y, x + math.ceil(j / 2), y - tolshina\_diska, fill=cvetа[j % 8])  
 cnvs.create\_text(x, y - 5, text=(j), font="Times 8")  
 y -= tolshina\_diska  
 j = i[-1]  
 sdvig = 100 \* (marshrut\_zavishego[1] - marshrut\_zavishego[0]) / 2  
 cnvs.create\_rectangle(x - math.floor(j / 2) + sdvig, vysota\_zavisaniya, x + math.ceil(j / 2) + sdvig, vysota\_zavisaniya - tolshina\_diska, fill=cvetа[j % 8])  
 cnvs.create\_text(x + sdvig, vysota\_zavisaniya - 5, text=(j), font="Times 8")  
 else:  
 for j in i:  
 cnvs.create\_rectangle(x - math.floor(j / 2), y, x + math.ceil(j / 2), y - tolshina\_diska, fill=cvetа[j % 8])  
 cnvs.create\_text(x, y - 5, text=(j), font="Times 8")  
 y -= tolshina\_diska  
 x += 100  
  
def vvod\_id(): *# вводим ID и раскладываем диски по стержням* global spisok\_sterzhney  
 spisok\_sterzhney = []  
 try:  
 id\_str = str(pole\_vvoda\_1.get())  
 for i, j in enumerate(id\_str):  
 tmp = []  
 for i1 in reversed(range(int(j))):  
 tmp.append((len(id\_str) - i) \* 10 + i1 + 1)  
 spisok\_sterzhney.append(tmp)  
 risovat\_pozitsiyu(spisok\_sterzhney)  
 podschet()  
 except Exception:  
 messagebox.showerror("Error", "Ошибка при вводе ID")  
  
def podschet(): *# считаем, сколько всего перемещений нужно* global maks\_iter  
 maks\_iter = 0  
 l\_rod = 0  
 for i in reversed(range(3)):  
 l\_rod += len(spisok\_sterzhney[i])  
 maks\_iter += 2 \*\* l\_rod - 1  
 for i in range(3, len(spisok\_sterzhney)):  
 maks\_iter += (2 \*\* len(spisok\_sterzhney[i]) - 1)  
 l\_rod += len(spisok\_sterzhney[i])  
 maks\_iter += 2 \*\* l\_rod - 1  
  
def nachalo(): *# возвращаемся к началу* global marshrut\_zavishego  
 marshrut\_zavishego = []  
 risovat\_pozitsiyu(spisok\_sterzhney)  
 label1['text'] = "0"  
  
def konec(): *# показываем, как всё будет в конце* global marshrut\_zavishego  
 marshrut\_zavishego = []  
 end\_pos = []  
 last\_st = []  
 for i in spisok\_sterzhney:  
 last\_st += i  
 end\_pos.append([])  
 end\_pos[len(spisok\_sterzhney) - 1] = last\_st  
 risovat\_pozitsiyu(end\_pos)  
 label1['text'] = str(maks\_iter)  
  
def podschet\_hodov(num, st1, st2, st3, dlina, fix1, fix2, fix3): *# рекурсия для подсчета ходов* pw = 2 \*\* dlina  
 if pw < num:  
 num2 = num - pw  
 st2.append(st1[len(st1) - 1 - dlina])  
 st3 += st1[len(st1) - dlina:]  
 st1 = st1[:len(st1) - 1 - dlina]  
 st3, st2, st1 = podschet\_hodov(num2, st3, st2, st1, dlina - 1, fix3, fix2, fix1)  
 elif pw > num:  
 st1, st3, st2 = podschet\_hodov(num, st1, st3, st2, dlina - 1, fix1, fix3, fix2)  
 else:  
 st2.append(st1[len(st1) - dlina - 1])  
 st3 += st1[len(st1) - dlina:]  
 st1 = st1[:len(st1) - 1 - dlina]  
 if flag\_zavisaniya:  
 marshrut\_zavishego.append(fix2)  
 marshrut\_zavishego.append(fix1)  
 return (st1, st2, st3)  
  
def vypolnit(): *# выполняем перемещения* global tekushaya\_pozitsiya, marshrut\_zavishego, vrem\_iter, iter  
 marshrut\_zavishego = []  
 tekushaya\_pozitsiya = copy.deepcopy(spisok\_sterzhney)  
 vrem\_iter = 0  
 l\_rod = 0  
 flag = False  
 ind1 = [2, 0, 1]  
 ind2 = [1, 2, 0]  
 for i in reversed(range(3)):  
 l\_rod += len(spisok\_sterzhney[i])  
 old\_vrem\_iter = vrem\_iter  
 vrem\_iter += 2 \*\* l\_rod - 1  
 if vrem\_iter >= iter:  
 st1 = tekushaya\_pozitsiya[i]  
 st2 = []  
 st3 = []  
 st1, st2, st3 = podschet\_hodov(iter - old\_vrem\_iter, st1, st2, st3, len(tekushaya\_pozitsiya[i]) - 1, i, ind1[i], ind2[i])  
 tekushaya\_pozitsiya[i] = st1  
 tekushaya\_pozitsiya[ind1[i]] += st2  
 tekushaya\_pozitsiya[ind2[i]] += st3  
 flag = True  
 break  
 else:  
 tekushaya\_pozitsiya[ind1[i]] += tekushaya\_pozitsiya[i]  
 tekushaya\_pozitsiya[i] = []  
 if not flag:  
 for i in range(3, len(spisok\_sterzhney)):  
 old\_vrem\_iter = vrem\_iter  
 vrem\_iter += (2 \*\* len(spisok\_sterzhney[i]) - 1)  
 if vrem\_iter >= iter:  
 st1 = tekushaya\_pozitsiya[i]  
 st2 = []  
 st3 = []  
 st1, st2, st3 = podschet\_hodov(iter - old\_vrem\_iter, st1, st2, st3, len(tekushaya\_pozitsiya[i]) - 1, i, i - 1, i - 2)  
 tekushaya\_pozitsiya[i] = st1  
 tekushaya\_pozitsiya[i - 1] += st2  
 tekushaya\_pozitsiya[i - 2] += st3  
 break  
 else:  
 tekushaya\_pozitsiya[i - 1] += tekushaya\_pozitsiya[i]  
 tekushaya\_pozitsiya[i] = []  
 l\_rod += len(spisok\_sterzhney[i])  
 old\_vrem\_iter = vrem\_iter  
 vrem\_iter += 2 \*\* l\_rod - 1  
 if vrem\_iter >= iter:  
 st1 = tekushaya\_pozitsiya[i - 1]  
 st2 = []  
 st3 = []  
 st1, st2, st3 = podschet\_hodov(iter - old\_vrem\_iter, st1, st2, st3, len(tekushaya\_pozitsiya[i - 1]) - 1, i - 1, i, i - 2)  
 tekushaya\_pozitsiya[i - 1] = st1  
 tekushaya\_pozitsiya[i] += st2  
 tekushaya\_pozitsiya[i - 2] += st3  
 break  
 else:  
 tekushaya\_pozitsiya[i] += tekushaya\_pozitsiya[i - 1]  
 tekushaya\_pozitsiya[i - 1] = []  
 risovat\_pozitsiyu(tekushaya\_pozitsiya)  
  
def obrabotka\_procentov(cs): *# обрабатываем проценты* global schetchik, iter, marshrut\_zavishego, flag\_zavisaniya, vrem\_iter  
 try:  
 vrem\_iter = int(cs) \* maks\_iter / 100  
 iter = int(vrem\_iter)  
 if vrem\_iter == iter:  
 flag\_zavisaniya = False  
 else:  
 iter += 1  
 flag\_zavisaniya = True  
 label1['text'] = str(vrem\_iter)  
 schetchik = 0  
 if 0 <= int(cs) <= 100:  
 vypolnit()  
 else:  
 messagebox.showerror("Error", "Неверный ввод процента")  
 except Exception:  
 messagebox.showerror("Error", "Неверный ввод процента")  
  
pole\_vvoda\_1 = Entry(font="Times 12")  
pole\_vvoda\_1.insert(0, "70211096")  
btn\_1 = Button(text="Ввод ID", command=vvod\_id)  
pole\_vvoda\_1.place(x=50, y=520, width=70)  
btn\_1.place(x=50, y=550, width=70)  
  
y\_knopki = 600  
btn\_2 = Button(text="Начало", command=nachalo)  
btn\_2.place(x=50, y=y\_knopki, width=70)  
  
btn\_3 = Button(text="Конец", command=konec)  
btn\_3.place(x=550, y=y\_knopki, width=70)  
  
pole\_vvoda\_2 = Entry(font="Times 12")  
pole\_vvoda\_2.insert(0, "70")  
btn\_4 = Button(text="П. 1", command=lambda: obrabotka\_procentov(pole\_vvoda\_2.get()))  
pole\_vvoda\_2.place(x=150, y=y\_knopki, width=70)  
btn\_4.place(x=150, y=y\_knopki + 50, width=70)  
  
pole\_vvoda\_3 = Entry(font="Times 12")  
pole\_vvoda\_3.insert(0, "21")  
btn\_5 = Button(text="П. 2", command=lambda: obrabotka\_procentov(pole\_vvoda\_3.get()))  
pole\_vvoda\_3.place(x=250, y=y\_knopki, width=70)  
btn\_5.place(x=250, y=y\_knopki + 50, width=70)  
  
pole\_vvoda\_4 = Entry(font="Times 12")  
pole\_vvoda\_4.insert(0, "10")  
btn\_6 = Button(text="П. 3", command=lambda: obrabotka\_procentov(pole\_vvoda\_4.get()))  
pole\_vvoda\_4.place(x=350, y=y\_knopki, width=70)  
btn\_6.place(x=350, y=y\_knopki + 50, width=70)  
  
pole\_vvoda\_5 = Entry(font="Times 12")  
pole\_vvoda\_5.insert(0, "96")  
btn\_7 = Button(text="П. 4", command=lambda: obrabotka\_procentov(pole\_vvoda\_5.get()))  
pole\_vvoda\_5.place(x=450, y=y\_knopki, width=70)  
btn\_7.place(x=450, y=y\_knopki + 50, width=70)  
  
label1 = Label(text="Количество итераций:\n", font="Arial 12")  
label1.place(x=150, y=y\_knopki - 100)  
  
label2 = Label(text="Количество итераций:\n", font="Arial 12")  
label2.place(x=150, y=y\_knopki - 50)  
  
root.mainloop()

2.4.3. Тестирование и отладка

Тестирование провода происходило следующим образом:

ID входа: Проверялись различные значения ID, включая "70211096". Убедились, что диски корректно восстанавливаются по стержням.

Максимальное число перемещений: Для ID "70211096" Функция подсчета правильно вычисляет общее количество ходов.

Кнопки "Начало" и "Конец": Начальное состояние отображает диски согласно ID, конечно — все диски на последнем месте.

Процентные значения: Ввод результатов 70, 21, 10, 96 показал правильное перемещение дисков. При нецелевых значениях диск зависит между стержнями.

Обработка ошибок: Некорректный ввод (буквы, отрицательные числа) вызывает ошибку в окне оператора.

Отладка применялась для исправления мелких ошибок в логике зависимости диска и обеспечение обновления холста.

2.4.4. Скриншоты результатов работы

Начальное состояние показано на рисунке 9.

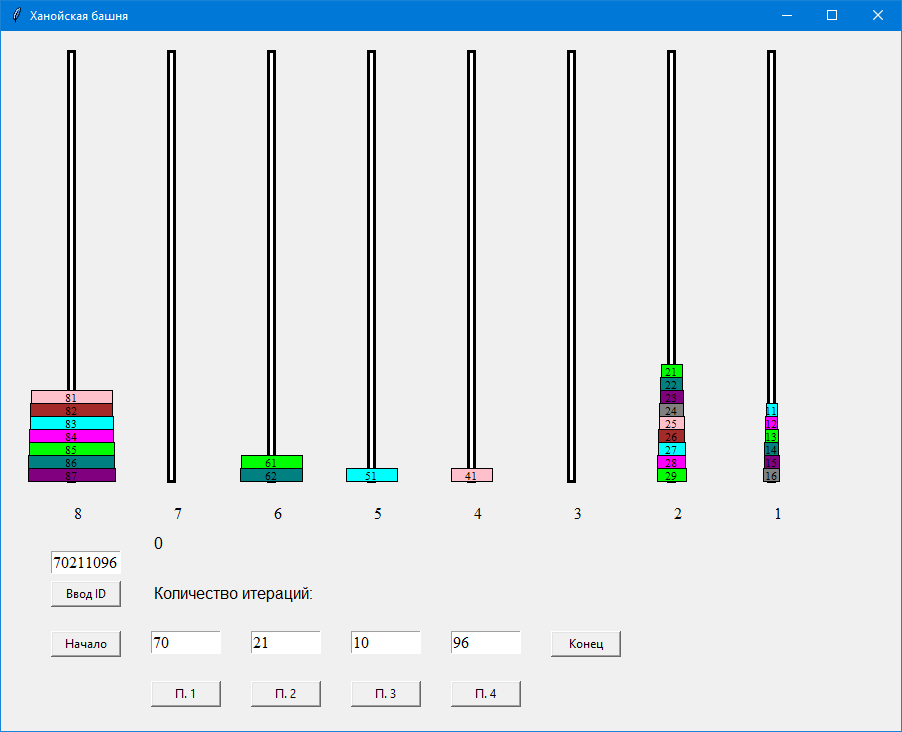


Рисунок 9. Начальное состояние дисков.

Окно с 8 стержнями, диски распределены согласно ID "70211096" (7 дисков на первом, 0 на втором и т.д.), цвета дисков — серый, розовый, коричневый и т.д.

Конечное состояние показано на рисунке 10.

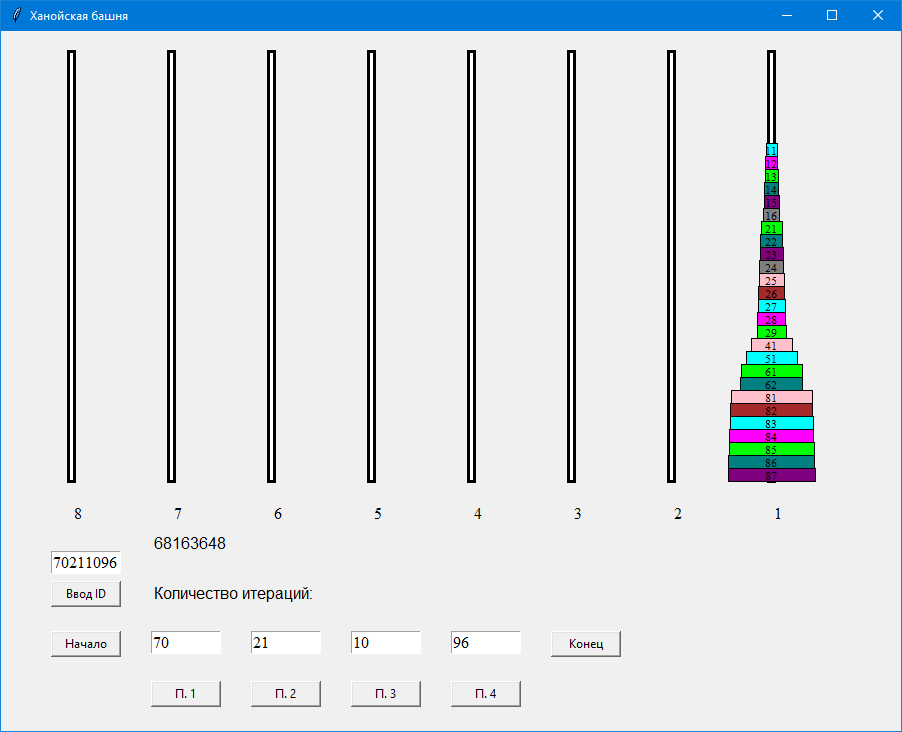


Рисунок 10. Конечное состояние дисков.

Все диски переработаны на последний этап, общее число итераций включено в метку.

Промежуточное состояние (70%), как и зависание показаны на рисунке 11.

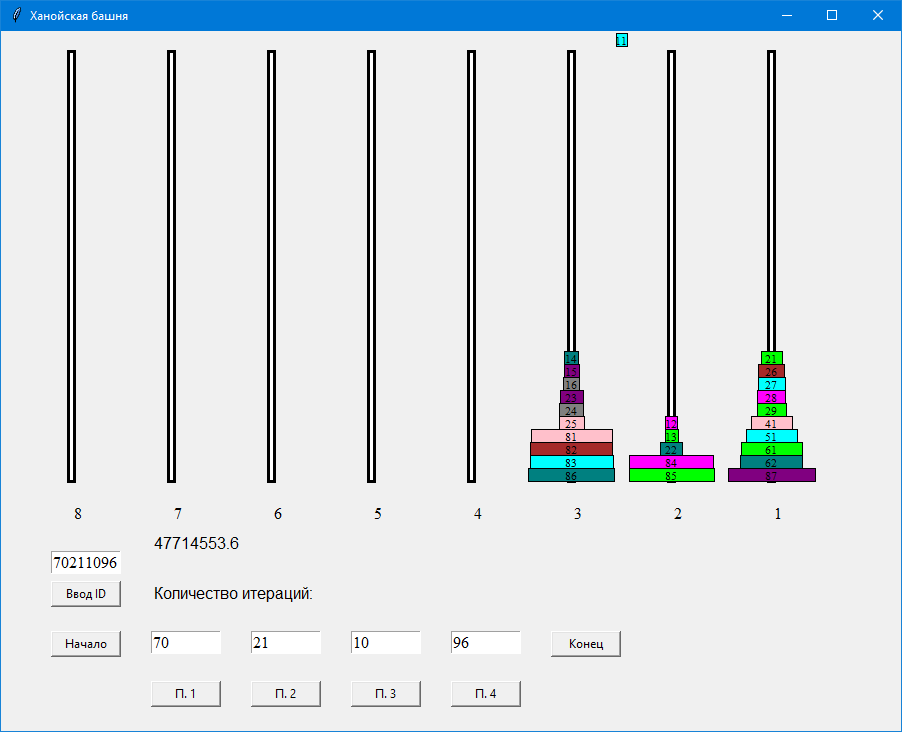


Рисунок 11. Промежуточное состояние и зависание диска.

Диски частично перенесены, один диск может быть зависимым между стержнями.

2.6. Выводы по 2 главе

В результате выполнения второго раздела курсовой работы были разработаны и реализованы четыре прикладных программных продукта на языке Python с использованием среды разработки PyCharm. Каждая программа решает конкретную задачу и демонстрирует применение различных подходов в программировании — от консольной обработки текстов до построения полноценных графических интерфейсов.

Первое задание показало, как можно работать с текстовыми файлами, производить статистический анализ и сортировку данных. Второе задание позволило реализовать экспертную систему с обработкой пользовательских команд и управлением состоянием данных. Третье задание дало возможность спроектировать расширяемый инженерный калькулятор с учётом индивидуальных параметров, включая расчёты по ID и специфические математические функции. Четвёртое задание продемонстрировало навыки визуализации сложной логики — модифицированной задачи о Ханойских башнях — и работу с графикой.

Все приложения были протестированы и успешно прошли проверку на корректность работы, устойчивость к ошибкам и соответствие методическим требованиям. Использование объектно-ориентированного подхода, модульной структуры кода и стандартных библиотек Python сделало проекты надёжными и удобными для расширения.

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

Для корректной работы разработанных программных продуктов необходимы минимальные технические характеристики, которые позволяют обеспечить стабильное выполнение кода, отображение графических интерфейсов и взаимодействие с пользователем.

Рекомендуемые технические средства:

1. Операционная система: Windows 10 или выше, macOS, либо любая актуальная сборка Linux;
2. Процессор: двухъядерный (x86 или x64) с тактовой частотой не ниже 2 ГГц;
3. Оперативная память: от 4 ГБ (рекомендуется 8 ГБ для комфортной работы с несколькими окнами и интерфейсами);
4. Место на диске: не менее 500 МБ для установки Python, библиотеки и среды разработки;
5. Экран: разрешение от 1280×720, рекомендуется 1366×768 или выше (особенно для корректного отображения визуализации);
6. Программное обеспечение:
   * Язык программирования Python версии 3.9 или выше;
   * Среда разработки PyCharm Community Edition (или аналогичная IDE с поддержкой Python);
   * Установленные стандартные библиотеки Python (tkinter, math, collections, и др.).

Дополнительных сторонних библиотек установка не требует, все задачи реализованы с использованием стандартного функционала, что упрощает переносимость и поддержку приложений.

ЗАКЛЮЧЕНИЕ

Курсовая работа позволила закрепить теоретические знания и на практике реализовать прикладные задачи различного уровня сложности. Были разработаны программы, демонстрирующие навыки работы с файлами, анализом текстовых данных, графическим интерфейсом, обработкой пользовательского ввода и визуализацией алгоритмов.

Каждое задание было выполнено в соответствии с методическими указаниями и индивидуальными параметрами (фамилия и ID). Это позволило не только продемонстрировать владение инструментами разработки, но и проявить творческий подход в реализации интерфейсов и логики работы программ.

Полученные в ходе работы навыки могут быть использованы при решении реальных инженерных, аналитических и образовательных задач. Разработанные решения универсальны, масштабируемы и могут быть адаптированы под другие исходные данные.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Доусон М. Программируем на Python / М.Доусон; пер. с англ. В.Порицкий. – С-П.:Питер, - 2019. – 416 стр.

2. МакГрат М. Python. Программирование для начинающих / М. МакГрат; пер. с англ. М. Райтман. – М.: Эксмо, 2015. – 178 стр.

3. PEP 8 - руководство по написанию кода на Python [Электронный ресурс]. – Режим доступа: https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html, свободный. – Загл. с экрана.

4. Гамма Э. Паттерны объектно-ориентированного проектирования / Э. Гам-ма, Р. Хелм, Дж. Ральф, Дж. Влиссидес / пер. с англ. А.А. Слинкин. – С-П.: Питер, - 2020 – 448 стр.

5. Копец Д. Классические задачи Computer Science на языке Python / Д. Ко-пец; пер. с англ. Е.Л. Сандицкая. – М.: Прогресс книга, - 2020. – 256 стр.

6. Хайнеман Дж. Алгоритмы. Справочник с примерами на C, C++, Java и Py-thon / Дж. Хайнеман, Г. Поллис, Ст. Селков / пер. с англ. И.В. Красиков. -М.: Вильямс, 2017. – 432 стр.

7. GUI Help/Tkinter book - Викиучебник [Электронный ресурс]. – Режим до-ступа: https://ru.wikibooks.org/wiki/GUI\_Help/Tkinter\_book, свободный. – Загл. с экрана.

8. Ханойская башня - Википедия, свободная энциклопедия [Электронный ре-сурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BD%D0%BE%D0%B9%D1%81%D0%BA%D0%B0%D1%8F\_%D0%B1%D0%B0%D1%88%D0%BD%D1%8F. – Загл. с экрана.

9. Кент Б. – экстремальное программирование. Разработка через тестирование / пер. с англ. П. Анджан. – С-П.: Питер, 2018. – 224 стр.