

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 4](#Введение)

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

[1.1. Исходные данные к заданиям курсовой работы 5](#Исход_1_1)

[1.2. Анализ методических указаний, входных и выходных данных к заданиям курсовой работы 8](#А1_2)

[1.3. Выбор и обоснование необходимых библиотек и среды разработки 10](#А1_3)

[1.4. Выводы по 1 главе 10](#А1_4)

[2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ. 11](#Б2)

[2.1. Работа с наборами данных 11](#_Toc60177547)

[2.1.1. Построение алгоритма решения задания без графического интерфейса 11](#Б2_1)

[2.1.2. Исходный код реализации на языке программирования 12](#_Toc60177549)

[2.1.3. Примеры тестирования и отладки 13](#_Toc60177549)

[2.1.4. Скриншоты рехультатов работы 14](#Б2_1_4)

[2.2. Разработка экспертной системы 15](#_Toc60177550)

[2.2.1. Построение алгоритма решения задания с графическим интерфейсом 15](#Б2_2_1)

[2.2.2. Разработка программной реализации на языке программирования 18](#_Toc60177552)

[2.2.3. Тестрование и отладка 23](#_Toc60177552)

[2.2.4. Скриншоты результатов работы 24](#_Toc60177552)

[2.3. Разработка аналитической системы 24](#_Toc60177553)

[2.3.1. Построение алгоритма решения задания с графическим интерфейсом 24](#_Toc60177554)

[2.3.2. Исходный код реализации на языке программирования 25](#Б2_3_2)

[2.3.2.1. Проектирование стандартного функционала 36](#_Toc60177556)

[2.3.2.2. Проектирование расширенного функционала 36](#_Toc60177557)

[2.3.2.3. Индивидуальное задание реализации дополнительных функций 37](#_Toc60177557)

[2.3.3. Тестирование и отладка 38](#Б2_3_2)

[2.3.4. Скриншоты результатов работы 39](#Б2_3_2)

[2.4 Разработка аналитической системы 41](#_Toc60177558)

[2.4.1. Построение алгоритма решения задания с графическим интерфейсом 41](#_Toc60177559)

[2.4.2. Исходный код реализации языка программирования 42](#_Toc60177560)

[2.4.3. Тестирование и отладка 49](#_Toc60177561)

[2.4.4. Скриншоты результатов работы 49](#_Toc60177562)

[2.6 Выводы по 2 главе 54](#_Toc60177558)

[3 РАЗРАБОТКА ТРЕБОВАНИЙ К ТЕХНИЧЕСКИМ СРЕДСТВАМ РЕАЛИЗАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ 55](#_Toc60177563)

[Заключение 56](#_Toc60177564)

[Список использованной литературы 57](#_Toc60177565)

ВВЕДЕНИЕ

В настоящее время программное обеспечение играет большую роль во всех сферах деятельности. Владение навыками обработки данных и построения графических интерфейсов становится неотъемлемой частью подготовки специалистов в области информационных технологий.

В ходе курсовой работы идёт формирование практических умений по разработке программ для решения определённых задач. Программы разрабатываются на языке программирования Python. В ходе выполнения работы студенту необходимо продемонстрировать навыки реализации алгоритмов, обработку как текстовых, так и числовых данных, а также использование графического интерфейса GUI.

В данной курсовой работе необходимо решить 4 задания. 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 используется среда разработки 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.

Исходные данные задач:

* 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.

Задание №1 требует обработки текстового файла с последующим подсчетом количества повторений слов и их сортировкой, т.е. необходимо реализовать работу с наборами данных;

Формулировка задания:

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 экспертной системы с использованием GUI;

Формулировка задания:

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 представляет собой разработку аналитической системы, т.е. необходимо создать калькулятор с базовыми и инженерными функциями с использованием GUI;

Формулировка задания:

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, с использованием GUI.

Формулировка задания:

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 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 требования к функциональности программ, формату входных и выходных данных, а также примеры к 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.

Необходимо учесть следующие ключевые особенности:

Удаление знаков препинания, сортировка с использованием списка кортежей.

Задание №2: Банковская экспертная система (с GUI)

Необходимо чтобы программа могла выполнить пополнение, снятие, перевод средств, начисление процентов, отображение баланса.

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 базовыми и инженерными функциями (с GUI)

В стандартные функции входят 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.

Обязательно должны входит кнопки, дисплей, меню перехода, сообщения об ошибках.

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

Изначально количество дисков на каждом шпинделе задаётся цифрами ID.

Допустимые перемещения между шпинделями строго определены.

Необходимо определить минимальное количество итераций для переноса всех дисков на шпиндель №1.

В программе присутствует 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 и большому количеству встроенных инструментов.

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.

Умный поиск

Дополнительно планируется использовать следующие библиотеки Python:

tkinter — для создания графического интерфейса;

math — для реализации математических функций калькулятора;

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

copy – поверхностное и глубокое копирование

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 необходимые задачи. Также определены основные библиотеки, которые будут использоваться в ходе разработки. Все эти шаги позволили сформировать чёткое представление о задачах и способах их решения.

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 файла resourse\_1.txt, обрабатывает его, 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.

2.1.1. Построение алгоритма решения задания без графического интерфейса

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

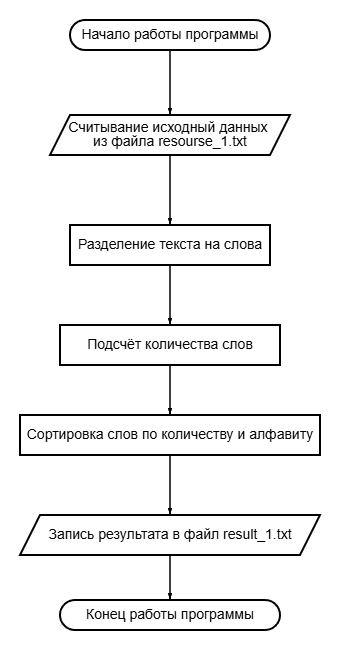


Рисунок 1. Блок схема алгоритма задания

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:

**with** open('resourse\_1.txt', 'r', encoding='utf-8') **as** file:

text = file.read()

# Удаляем знаки препинания с помощью replace

punctuation = ['.', ',', '!', '?', ':', ';', '-', '(', ')', '"', "'"]

**for** p **in** punctuation:

text = text.replace(p, '')

words = text.split()

words\_count = {}

**for** word **in** words:

# Приводим слова к нижнему регистру для учета регистра

word = word.lower()

words\_count[word] = words\_count.get(word, **0**) + **1**

**print**(words\_count)

sorted\_words = sorted(words\_count.items(), key=**lambda** item: (-item[**1**], item[**0**]))

**with** open('result\_1.txt', 'w', encoding='utf-8') **as** result\_file:

**for** word, count **in** sorted\_words:

line = f"{word} {count}**\n**"

result\_file.write(line)

**print**(f'Добавлена строка: {word} {count}')

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

your 1

Отладка производилась с помощью встроенного механизма отладки в PyCharm. Были протестированы сценарии с пустым файлом, отсутствующим файлом.

2.1.4. Скриншоты результатов работы

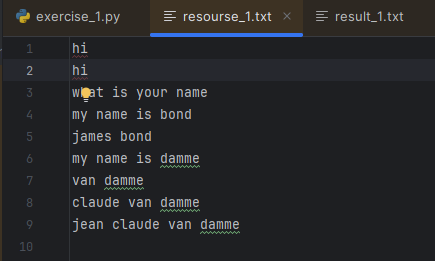


Рис. 2. Содержимое входного файла resourse\_1.txt в редакторе PyCharm.

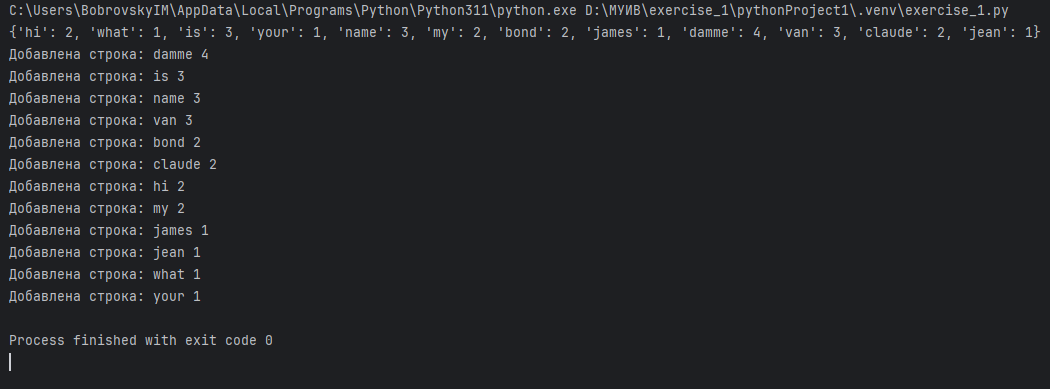


Рис. 3. Запуск программы и вывод результата в консоли.

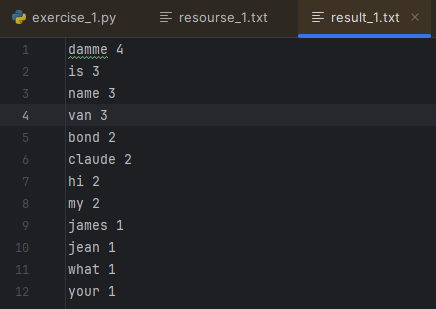


Рис. 4. Содержимое выходного файла result\_1.txt.

2.2. Разработка экспертной системы

2.2.1. Построение алгоритма решения задания с графическим интерфейсом

Данное задание заключается в создании 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. Максимально количество команд, которое может выполнить пользователь за одно нажатие кнопки Calculate - 20

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 происходит отображение программы. Справа поле ответа программы, слева кнопки “+” и “-”. При нажатии на которую, поле ввода программы либо добавляется, либо убирается. Максимальное количество полей 20.
2. Пользователь вводит команды в 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 правом 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.

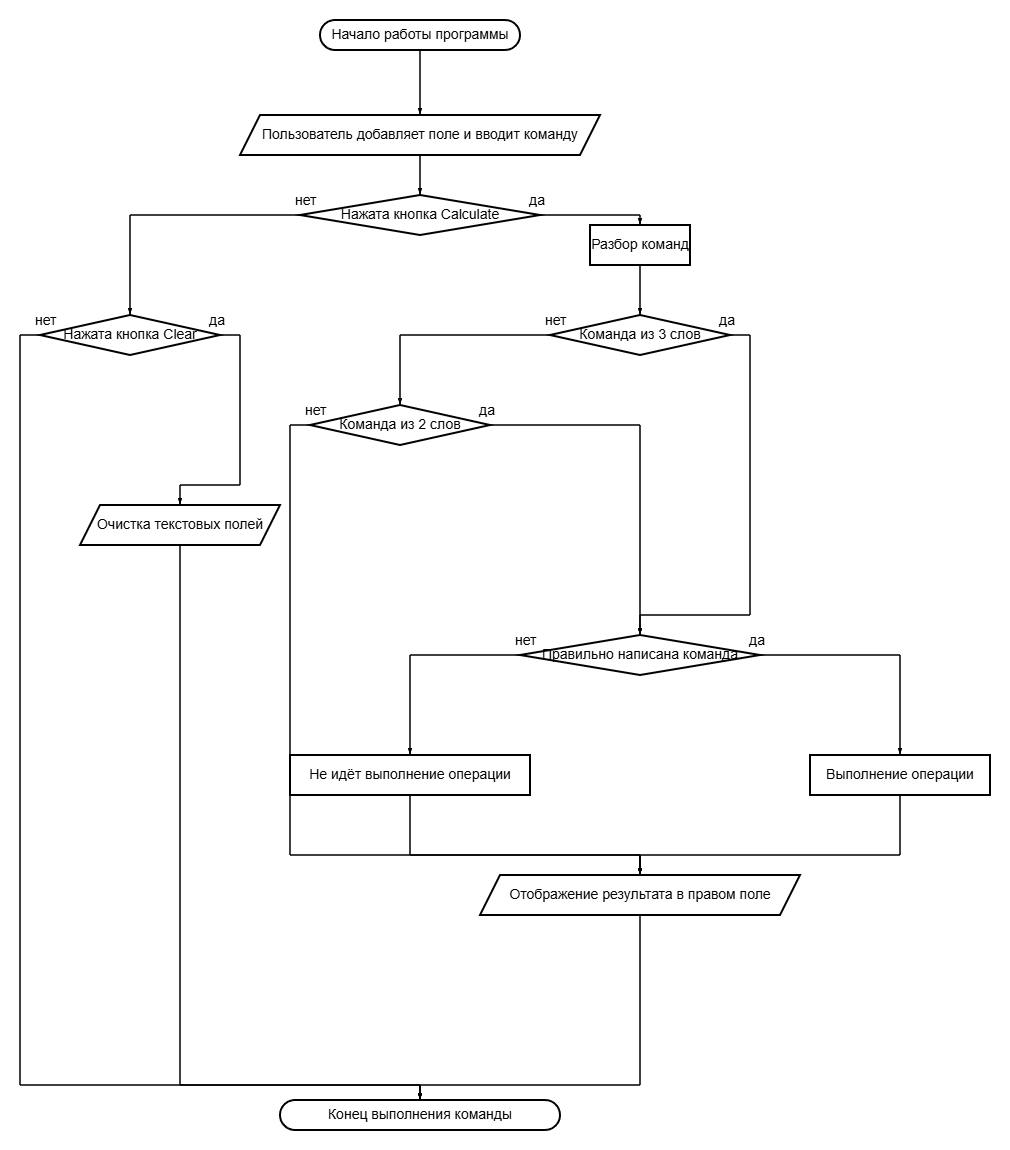


Рисунок 5. Блок схема алгоритма.

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:

**from** **os** **import** write

**from** **tkinter** **import** ttk

**import** **tkinter** **as** **tk**

**from** **django.utils.text** **import** normalize\_newlines

root = tk.Tk()

root.geometry('850x900')

max\_entry = **20**

cont\_entry = []

text\_area = None

scrollbar = None

**def** **error**(word):

entyon.config(state='normal')

entyon.delete(**0**, tk.END)

entyon.insert(**0**, f'Неизвестная команда: {word}')

entyon.config(state='readonly')

**print**(f'Неизвестная команда: {word}')

**def** **insert\_entry**():

entyon.config(state='normal')

entyon.delete(**0**, tk.END)

entyon.insert(**0**, 'Успешно код: 00')

entyon.config(state='readonly')

**print**('Успешно код: 00')

**def** **balance**(valueR):

**print**(valueR)

entyon.config(state='normal')

entyon.delete(**0**, tk.END)

**if** valueR == 'NO CLIENT':

entyon.insert(**0**, 'NO CLIENT')

**print**('NO CLIENT')

**else**:

entyon.insert(**0**, f'Баланс: {valueR}')

**print**(f'Ваш баланс: {valueR}')

entyon.config(state='readonly')

**def** **calculate**():

**global** text\_area, scrollbar

data\_dict = {}

**if** text\_area **is** **not** None:

text\_area.destroy()

scrollbar.destroy()

**with** open('resourse\_2.txt', 'r') **as** file:

**for** line **in** file:

pats = line.strip().split()

**if** len(pats) == **2**:

key = pats[**0**]

value = pats[**1**]

data\_dict[key] = value

file.close()

val = [entry.get() **for** entry **in** cont\_entry]

**for** i, values **in** enumerate(val, start=**1**):

**print**(f'Поле {i}: значение {values}')

words = values.split()

first\_word = words[**0**]

**if** len(words) == **3**:

second\_word = words[**1**]

third\_words = words[**2**]

**if** first\_word == 'DEPOSIT':

summa = int(third\_words)

**if** second\_word **in** data\_dict:

val\_sur = int(data\_dict[second\_word])

data\_dict[second\_word] = val\_sur + summa

insert\_entry()

**else**:

data\_dict[second\_word] = summa

insert\_entry()

**elif** first\_word == 'WITHDRAW':

summa = int(third\_words)

**if** second\_word **in** data\_dict:

val\_sur = int(data\_dict[second\_word])

data\_dict[second\_word] = val\_sur-summa

insert\_entry()

**else**:

data\_dict[second\_word] = **0** - summa

insert\_entry()

**else**:

error(first\_word)

**with** open('resourse\_2.txt', 'w') **as** file:

**for** key, value **in** data\_dict.items():

file.write(f'{key} {value}**\n**')

file.close()

**elif** len(words) == **2**:

second\_word = words[**1**]

**if** first\_word == 'BALANCE':

**if** second\_word **in** data\_dict:

balance(data\_dict[second\_word])

**else**:

balance('NO CLIENT')

**elif** first\_word == 'INCOME':

proc = int(second\_word)

**for** key, value **in** data\_dict.items():

**if** int(data\_dict[key]) >**0**:

balance\_n = int((int(data\_dict[key]) \*proc) // **100**)

data\_dict[key] = int(data\_dict[key]) + balance\_n

**else**:

data\_dict[key] = value

insert\_entry()

**with** open('resourse\_2.txt', 'w') **as** file:

**for** key, value **in** data\_dict.items():

file.write(f'{key} {value}**\n**')

file.close()

**else**:

error(first\_word)

**elif** len(words) == **1**:

**if** first\_word == 'BALANCE':

text\_area = tk.Text(root, wrap='word', height=**10**, width=**40**)

scrollbar = ttk.Scrollbar(root, command=text\_area.yview)

text\_area.grid(row=**4**, column=**5**, padx=**10**, pady=**10**)

scrollbar.grid(row=**4**, column=**6**, sticky='ns')

text\_area['yscrollcommand'] = scrollbar.set

**for** key, value **in** data\_dict.items():

text\_area.insert(tk.END, f'{key}: {value}**\n**')

**print**(f'{key}: {value}**\n**')

text\_area.config(state='disabled')

# Запрещаем фокусировку на текстовом поле

text\_area.bind("<Button-1>", **lambda** e: "break")

**else**:

error(first\_word)

**elif** len(words) == **4**:

second\_word = words[**1**]

third\_words = words[**2**]

four\_words = words[**3**]

**if** first\_word == 'TRANSFER':

summa = int(four\_words)

**if** second\_word **not** **in** data\_dict **and** third\_words **in** data\_dict:

data\_dict[second\_word] = **0** - summa

data\_dict[third\_words] = int(data\_dict[third\_words]) + summa

insert\_entry()

**elif** third\_words **not** **in** data\_dict **and** second\_word **in** data\_dict:

data\_dict[second\_word] = int(data\_dict[second\_word]) - summa

data\_dict[third\_words] = **0** + summa

insert\_entry()

**elif** second\_word **in** data\_dict **and** third\_words **in** data\_dict:

data\_dict[second\_word] = int(data\_dict[second\_word]) - summa

data\_dict[third\_words] = int(data\_dict[third\_words]) + summa

insert\_entry()

**else**:

data\_dict[second\_word] = **0** - summa

data\_dict[third\_words] = **0** + summa

insert\_entry()

**with** open('resourse\_2.txt', 'w') **as** file:

**for** key, value **in** data\_dict.items():

file.write(f'{key} {value}**\n**')

file.close()

**else**:

error(first\_word)

**def** **add\_entry**():

**if** len(cont\_entry)<max\_entry:

entry = ttk.Entry(root)

entry.grid(row=(len(cont\_entry)+**1**), column=**0**, padx=**10**, pady=**10**)

cont\_entry.append(entry)

**else**:

**print**('Максимальное количество полей')

**def** **delete\_entry**():

**if** len(cont\_entry)!= **0**:

entry = cont\_entry.pop()

entry.destroy()

**else**:

**print**('Больше нет полей')

**def** **clear\_val**():

**for** field **in** cont\_entry:

field.delete(**0**, tk.END)# Очищаем текст в каждом поле

entyon.config(state='normal')

entyon.delete(**0**, tk.END)

entyon.config(state='readonly')

cont\_entry.clear()

btnplus = ttk.Button(text='+', command=add\_entry)

btnminus = ttk.Button(text='-', command=delete\_entry)

btnclear = ttk.Button(text='Clear', command=clear\_val)

btncalc = ttk.Button(text='Calculate', command=**lambda** :calculate())

labels = ttk.Label(text='Ответ программы:')

entyon = ttk.Entry(state='readonly', width=**50**)

btnminus.grid(row=**0**, column=**1**, padx=**10**, pady=**10**)

btnplus.grid(row=**0**, column=**0**, padx=**10**, pady=**10**)

btnclear.grid(row=**0**, column=**3**,padx=**10**, pady=**10**)

btncalc.grid(row=**1**, column=**3**, padx=**10**, pady=**10**)

labels.grid(row=**0**, column=**4**, padx=**15**,pady=**10**)

entyon.grid(row=**0**, column=**5**, padx=**15**,pady=**10**)

root.mainloop()

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 Smirnov 500);

Снятие средств (WITHDRAW Ivanov 200);

Перевод средств между счетами (TRANSFER Ivanov Smirnov 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 успешной команде программа выводит сообщение «Успешно код: 00». При вводе некорректной команды программа выводит сообщение «Неизвестная команда (Команда)». Данные о пользователях хранятся в файле «resourse\_2.txt».

Отладка производилась с использованием встроенных инструментов PyCharm.

2.2.4. Скриншоты результатов работы

На рисунке 6 показан результат тестирования.

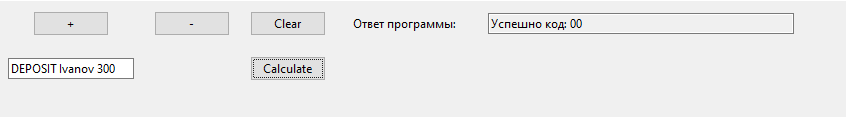


Рисунок 6. Результаты тестирования приложения.

На рисунке 7 показан результат ошибочной команды.

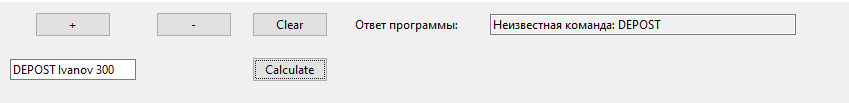


Рисунок 7. Результаты тестирования приложения

2.3. Разработка аналитической системы

2.3.1. Построение алгоритма решения задания с графическим интерфейсом

В данном задании 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 Бобровский и ID 70206239:

* Количество строк в цифровом дисплее составляет 8 (сумма всех цифр ID: 7+0+2+0+6+2+3+9 = 29 → 2+9 = 11 → 1+1 = 2).
* Количество ячеек памяти в расширенном режиме — 5 (сумма последних трёх цифр: 2+3+9 = 14 → 1+4 = 5).

Вид интерфейса и поведение калькулятора адаптируются под выбранный режим: стандартный или расширенный. Пользователь может переключаться между ними при помощи кнопкой Menu.

Основные этапы работы алгоритма:

1. Отображение текущего числа на цифровом дисплее;
2. Обработка ввода цифр, десятичной точки, изменения знака;
3. Реализация операций: +, -, \*, /, ^, √;
4. Поддержка дополнительных функций:
   * asin — вычисление арксинуса;
   * acos— вычисление арккосинуса;
   * atg— вычисление арктангенса;
   * log— вычисление логарифма по основанию;
   * n! — вычисление факториала числа;
5. Работа с памятью:
   * В стандартном режиме — одна память;
   * В расширенном — 5 отдельных ячеек;
6. В расширенном и обычном режиме присутствует текстовое поле с прокруткой для отображения всех операций;
7. Кнопка C очищает текущее состояние;
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.

**import** **tkinter** **as** **tk**

**from** **tkinter** **import** \*

**import** **math**

**import** **re**

**from** **django.contrib.sitemaps.views** **import** index

**from** **django.utils.autoreload** **import** ensure\_echo\_on

**from** **pyexpat.errors** **import** messages

root = tk.Tk()

root.geometry('400x500')

root.title('Калькулятор - Обычный режим')

id = None

buttons = []

buttons2 = []

memory = {}

value1 = None

value2 = None

**def** **menu**():

**global** id

**global** buttons

id = entry2.get()

**print**(id)

**if** buttons:

root.geometry('400x500')

root.title('Калькулятор - Обычный режим')

**for** button **in** buttons:

button.destroy()

buttons.clear()

**return**

**elif** id == None **or** id == '':

root.geometry('700x500')

root.title('Калькулятор - Расширенный режим')

btn1 = tk.Button(text='M+', command=**lambda** idx=**1**, val='M+': memory\_command(idx, val))

buttons.append(btn1)

btn1.grid(row=**4**, column=**6**, padx=**5**, pady=**5**)

btn2 = tk.Button(text='M-', command=**lambda** idx=**1**, val='M-': memory\_command(idx, val))

buttons.append(btn2)

btn2.grid(row=**4**, column=**7**, padx=**5**, pady=**5**)

btn3 = tk.Button(text='MC', command=**lambda** idx=**1**, val='MC': memory\_command(idx, val))

buttons.append(btn3)

btn3.grid(row=**4**, column=**8**, padx=**5**, pady=**5**)

btn4 = tk.Button(text='MR', command=**lambda** idx=**1**, val='MR': memory\_command(idx, val))

buttons.append(btn4)

btn4.grid(row=**4**, column=**9**, padx=**5**, pady=**5**)

btn5 = tk.Button(text='MS', command=**lambda** idx=**1**, val='MS': memory\_command(idx, val))

buttons.append(btn5)

btn5.grid(row=**4**, column=**10**, padx=**5**, pady=**5**)

asin\_btn = tk.Button(text='asin', command= asin)

buttons.append(asin\_btn)

asin\_btn.grid(row=**3**, column=**6**, padx=**5**, pady=**5**)

acos\_btn = tk.Button(text='acos', command=acos)

buttons.append(acos\_btn)

acos\_btn.grid(row=**3**, column=**7**, padx=**5**, pady=**5**)

atg\_btn = tk.Button(text='atg', command=atg)

buttons.append(acos\_btn)

atg\_btn.grid(row=**3**, column=**8**, padx=**5**, pady=**5**)

log\_btn = tk.Button(text='log', command=**lambda** :log())

buttons.append(log\_btn)

log\_btn.grid(row=**3**, column=**9**, padx=**5**, pady=**5**)

fc\_btn = tk.Button(text='n!', command=factor)

buttons.append(log\_btn)

fc\_btn.grid(row=**3**, column=**10**, padx=**5**, pady=**5**)

**else**:

root.geometry('700x500')

root.title('Калькулятор - Расширенный режим')

new\_id = id[-**3**:]

count\_memory = memory\_block(new\_id)

**print**(f'Количество ячеек - {count\_memory}')

cl = **6**

**for** i **in** range(count\_memory):

**if** cl == **12**:

cl = **6**

btn1 = tk.Button(text='M+', command= **lambda** idx = i, val = 'M+': memory\_command(idx, val))

buttons.append(btn1)

btn1.grid(row = i +**4**, column = cl, padx = **5**, pady = **5**)

btn2 = tk.Button(text='M-', command= **lambda** idx = i, val = 'M-': memory\_command(idx, val))

buttons.append(btn2)

btn2.grid(row=i + **4**, column=cl+**1**, padx=**5**, pady=**5**)

btn3 = tk.Button(text='MC', command= **lambda** idx = i, val = 'MC': memory\_command(idx, val))

buttons.append(btn3)

btn3.grid(row=i + **4**, column=cl+**2**, padx=**5**, pady=**5**)

btn4 = tk.Button(text='MR', command= **lambda** idx = i, val = 'MR': memory\_command(idx, val))

buttons.append(btn4)

btn4.grid(row=i + **4**, column=cl+**3**, padx=**5**, pady=**5**)

btn5 = tk.Button(text='MS', command= **lambda** idx = i, val = 'MS': memory\_command(idx, val))

buttons.append(btn5)

btn5.grid(row=i + **4**, column=cl+**4**, padx=**5**, pady=**5**)

asin\_btn = tk.Button(text='asin', command=asin)

buttons.append(asin\_btn)

asin\_btn.grid(row=**3**, column=**6**, padx=**5**, pady=**5**)

acos\_btn = tk.Button(text='acos', command=acos)

buttons.append(acos\_btn)

acos\_btn.grid(row=**3**, column=**7**, padx=**5**, pady=**5**)

atg\_btn = tk.Button(text='atg', command=atg)

buttons.append(acos\_btn)

atg\_btn.grid(row=**3**, column=**8**, padx=**5**, pady=**5**)

log\_btn = tk.Button(text='log', command=**lambda**: log())

buttons.append(log\_btn)

log\_btn.grid(row=**3**, column=**9**, padx=**5**, pady=**5**)

fc\_btn = tk.Button(text='n!', command=factor)

buttons.append(log\_btn)

fc\_btn.grid(row=**3**, column=**10**, padx=**5**, pady=**5**)

**def** **asin**():

value = entry.get()

**try**:

value = float(value)

**if** -**1** <= value <= **1**:

result = math.asin(value)

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f'asin({value}) = {result}')

**else**:

result = **0**

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f'asin({value}) = {result}')

**except** **Exception** **as** e:

add\_history(f'Error - {e}')

**def** **acos**():

value = entry.get()

**try**:

value = float(value)

**if** -**1** <= value <= **1**:

result = math.acos(value)

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f'acos({value}) = {result}')

**else**:

result = **0**

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f'acos({value}) = {result}')

**except** **Exception** **as** e:

add\_history(f'Error - {e}')

**def** **atg**():

value = entry.get()

**try**:

value = float(value)

result = math.atan(value)

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f'atg({value}) = {result}')

**except** **Exception** **as** e:

add\_history(f'Error - {e}')

**def** **log**():

**global** buttons2, value1,value2

**if** buttons2:

**for** button **in** buttons2:

button.destroy()

buttons2.clear()

text\_result.config(state='normal')

text\_result.delete(**1.0**, tk.END)

value1 = None

value2 = None

**else**:

text\_result.config(state='normal')

text\_result.delete(**1.0**, tk.END)

text\_result.config(state='disabled')

message = 'Введите, сначала число для подсчёта логарифма и нажмите кнопку добавить "Добавить". После чего введите второе число, но для основания и также нажмите соответствующую кнопку'

add\_history(message)

add\_history('Введите число для подсчета логарифма ...')

btn = tk.Button(text='Добавить', command=**lambda** :add\_log())

buttons2.append(btn)

btn.grid(row=**1**, column=**5**, padx=**5**, pady=**5**, columnspan =**2**)

**def** **add\_log**():

**global** value1,value2

**if** value1 **is** None **and** value2 **is** None:

value1 = float(entry.get())

add\_history(value1)

message = 'Введите основание для логарифма...'

add\_history(message)

entry.delete(**0**, tk.END)

entry.insert(**0**, '0')

**elif** value1 **is** **not** None **and** value2 **is** None:

value2=float(entry.get())

result = math.log(value1, value2)

add\_history(value2)

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

add\_history(f' log({value1}, {value2}) = {result}')

**def** **factor**():

**try**:

n = int(entry.get())

result = math.factorial(n)

add\_history(f'{n}! = {result}')

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

**except** **ValueError** **as** e:

add\_history(f'Error - {e}')

**print**(value1)

**print**(value2)

**def** **add\_digit**(digit):

value = entry.get()

**if** value == '0' **and** digit == '-':

entry.delete(**0**, tk.END)

entry.insert(**0**, digit)

**elif** value == '0' **and** digit == '.':

value = value + digit

entry.delete(**0**, tk.END)

entry.insert(**0**, value)

**elif** digit == '.' **and** digit **in** value:

entry.delete(**0**, tk.END)

entry.insert(**0**, value)

**elif** value == '-' **and** digit **in** '+/\*^':

new\_value = '0' + digit

entry.delete(**0**, tk.END)

entry.insert(**0**, new\_value)

**elif** value == '0' **and** digit **not** **in** '+/\*^':

entry.delete(**0**, tk.END)

entry.insert(**0**,digit)

**elif** value[-**1**] **in** '+-\*/':

**if** digit **in** '+-\*/':

value = value[:-**1**]

entry.delete(**0**, tk.END)

entry.insert(**0**, value + digit)

**else**:

new\_val = value + digit

entry.delete(**0**, tk.END)

entry.insert(**0**, new\_val)

**elif** value == '0' **and** digit **in** '+/\*':

new\_value = '0' + digit

entry.delete(**0**,tk.END)

entry.insert(**0**, new\_value)

**elif** value =='0' **and** digit =='0':

**return**

**else**:

new\_digit = entry.get()+ str(digit)

entry.delete(**0**,tk.END)

entry.insert(**0**,new\_digit)

**def** **clear**():

entry.delete(**0**,tk.END)

entry.insert(**0**, **0**)

**def** **entry\_get**():

value = entry.get()

calculate(value)

**def** **calculate**(val):

value = val

**print**(value)

match = re.search(r'(\d+)\s\*\^\s\*(\d+)', value)

**try**:

**if** match:

base = int(match.group(**1**))

exp = int(match.group(**2**))

**print**(f'znach-1 = {base} and znach-2 = {exp}')

res = math.pow(base, exp)

new\_value = re.sub(r'(\d+)\s\*\^\s\*(\d+)', str(res), value, count=**1**)

**print**(value)

**return** calculate(new\_value)

**else**:

result = eval(value)

add\_history(f'{entry.get()} = {result}')

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

**except** **Exception** **as** e:

add\_history(f'Error - {e}')

**def** **pow\_dig**(base,exp):

res = math.pow(base,exp)

**print**(res)

**return** int(res)

**def** **proc**():

value = entry.get()

match = re.search(r'(\d+)\s\*([+\-/\*])\s\*(\d+)', value)

match2 = re.search(r'(\d+)', value)

**if** match **or** match2:

**if** match:

result = (int(match.group(**1**)) \* int(match.group(**3**)))/**100**

res = str(match.group(**1**) + match.group(**2**) + str(result))

result = eval(res)

**elif** match2:

result = int(match2.group(**0**))/**100**

**else**:

result = None

entry.delete(**0**,tk.END)

entry.insert(**0**,result)

**def** **pow\_dig2**():

value = entry.get()

result = int(value)\*\***2**

add\_history(f'{value} ^ 2 = {result}')

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

**def** **result\_sqrt**():

**try**:

value = entry.get()

result = math.sqrt(int(value))

add\_history(f'{result}')

entry.delete(**0**, tk.END)

entry.insert(**0**, result)

**except** **Exception** **as** e:

add\_history(f'Ошибка {e}')

entry.delete(**0**, tk.END)

**def** **sum\_rec**(val):

**if** int(val) <= **10**:

**if** int(val) == **1**:

**return** **2**

**else**:

**return** val

**else**:

new\_val = sum(int(digit) **for** digit **in** str(val))

**return** sum\_rec(new\_val)

**def** **memory\_block**(val):

**if** int(val) <= **9**:

**if** int(val) == **1**:

**return** **2**

**else**:

**return** val

**else**:

new\_val = sum(int(digit) **for** digit **in** str(val))

**return** memory\_block(new\_val)

**def** **memory\_command**(index, digit):

**global** memory

value = entry.get()

**if** digit == 'M+':

value = value.replace('-', '')

memory[index] = value

**elif** digit == 'M-':

**if** value[**0**] == '-':

memory[index] = value

**else**:

value = '-' + value

memory[index] = value

**elif** digit == 'MC':

memory.pop(index)

**elif** digit == 'MR':

val = memory[index]

**if** value[-**1**] **in** '+-\*/':

value = value + val

entry.delete(**0**, tk.END)

entry.insert(**0**,value)

**else**:

entry.delete(**0**, tk.END)

entry.insert(**0**, val)

**elif** digit == 'MS':

**pass**

**print**(f'Кнопка {digit}, Ячейка памяти {index}')

**def** **rec**():

**global** id

digit = entry2.get()

id = digit

res = sum\_rec(digit)

**print**(f'Количество строк - {res}')

text\_result.config(height=res)

**def** **add\_history**(operation):

text\_result.config(state='normal')

text\_result.insert(tk.END,f'{operation}**\n**')

text\_result.config(state='disabled')

expression = tk.StringVar(value="0")

text\_result = tk.Text(root, height=**11**, width=**33**,state='disabled', wrap='word')

entry = tk.Entry(root, justify=tk.RIGHT, textvariable=expression)

tk.Label(text='Введите ID студента').grid(row=**8**, column=**0**, padx=**5**, pady=**5**)

entry2 = tk.Entry(root)

tk.Button(text='C', command=clear, width=**5**).grid(row=**2**,column=**3**,padx=**5**,pady=**5**, sticky = 'wens')

tk.Button(text='Menu', command=menu).grid(row=**2**,column=**0**,padx=**5**,pady=**5**, sticky = 'wens')

tk.Button(text='<>', command=rec).grid(row=**2**,column=**1**,padx=**5**,pady=**5**, sticky = 'wens')

tk.Button(text='.', command= **lambda** :add\_digit('.')).grid(row=**3**,column=**2**,padx=**5**,pady=**5**, sticky = 'wens')

tk.Button(text='=', command=entry\_get).grid(row=**7**, column=**1**,padx=**5**,pady=**5**,sticky ='wens', columnspan=**2**)

tk.Button(text="+", command = **lambda** :add\_digit('+')).grid(row=**4**, column=**3**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="^", command = **lambda** :add\_digit('^')).grid(row=**3**, column=**3**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="%", command = **lambda** :proc()).grid(row=**3**, column=**5**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="^2", command = **lambda** :pow\_dig2()).grid(row=**5**, column=**5**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="#", command = **lambda** :result\_sqrt()).grid(row=**4**, column=**5**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="-", command = **lambda** :add\_digit('-')).grid(row=**5**, column=**3**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="\*", command = **lambda** :add\_digit('\*')).grid(row=**6**, column=**3**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text="/", command = **lambda** :add\_digit('/')).grid(row=**7**, column=**3**, padx=**5**, pady=**5**, sticky='wens')

tk.Button(text='1', command=**lambda**: add\_digit('1')).grid(row=**4**, column=**0**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='2', command=**lambda**: add\_digit('2')).grid(row=**4**, column=**1**,sticky ='wens',padx=**5**,pady=**5**)

tk.Button(text='3', command=**lambda**: add\_digit('3')).grid(row=**4**, column=**2**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='4', command=**lambda**: add\_digit('4')).grid(row=**5**, column=**0**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='5', command=**lambda**: add\_digit('5')).grid(row=**5**, column=**1**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='6', command=**lambda**: add\_digit('6')).grid(row=**5**, column=**2**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='7', command=**lambda**: add\_digit('7')).grid(row=**6**, column=**0**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='8', command=**lambda**: add\_digit('8')).grid(row=**6**, column=**1**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='9', command=**lambda**: add\_digit('9')).grid(row=**6**, column=**2**,padx=**5**,pady=**5**,sticky ='wens')

tk.Button(text='0', command=**lambda**: add\_digit('0')).grid(row=**7**, column=**0**,padx=**5**,pady=**5**,sticky ='wens')

text\_result.grid(row=**0**, column=**0**,padx=**5**,pady=**5**, sticky = 'wens',columnspan=**3**)

entry.grid(row = **1**, column=**0**, padx=**5**,pady=**5**, sticky='wens', columnspan= **3**)

entry2.grid(row=**9**, column=**0**,padx = **5**, pady=**5**, columnspan=**2**)

root.mainloop()

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 (#);
* Ввод знака +-/\*;
* Сброс данных (C);
* Вывод результата (=).

Дисплей в этом режиме реализован в виде поля Text, в котором отображается результат вычисления. Управление происходит при помощи кнопок, каждая из которых привязана к соответствующему методу обработки. Введённые символы отображаются в поле Entry.

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 Menu, калькулятор отображает:

* Кнопки инженерных функций;
* Дополнительные кнопки для работы с несколькими ячейками памяти.

В расширенном режиме добавляются следующие возможности:

1. Вычисление арксинуса;
2. Вычисление арккосинуса;
3. Вычисление арктангенса;
4. Вычисление логарифма по основанию;
5. Вычисление факториала числа.

Память реализована в виде массива «memory», включающего 5 ячеек. Пользователь может выполнять стандартные действия: MS, MR, M+, M-, MC на каждой ячейке памяти.

ID студента вносится внизу в поле Entry. После ввода необходимо нажать на кнопку «< >», для подсчета и применения необходимых изменений.

2.3.2.3. Индивидуальное задание реализации дополнительных функций

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

* asin — Вычисление арксинуса;
* acos — Вычисление арккосинуса;
* atg — Вычисление арктангенса;
* log — логарифм по основанию, доступный через отдельную кнопку;
* n! — Вычисление факториала от числа.

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 об соответствующей ошибке. Результат всех операций выводится на дисплей. В случае операции с логарифмом, текстовое поле, где отображается результат, очищается и выводятся сообщения подсказки, по которым пользователь выполняет операцию.

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 и вычисление корня;
2. Обработка ошибок:
   * При делении на ноль выводится соответствующее сообщение;
   * При возведении корня из отрицательного числа выводится соответствующее сообщение;
   * При решении логарифма из отрицательного числа выводится соответствующее сообщение.
   * Выход за диапазон арккосинуса и арксинуса выводится в ответе 0
3. Расширенные функции:
   * Переключение между стандартным и расширенным режимами (Menu);
   * Работа с арксинусом, арккосинусом, арктангенсом;
   * Вычисление логарифма по основанию и факториала числа.
   * Обработка ID студента которая активируется кнопкой < >
4. Память:
   * Сохранение, чтение, очистка, сохранение со знаком + и -;
5. Интерфейс:
   * Отображение операций в текстовом поле;
   * Прокрутка при большом количестве операций;
   * Поддержка всех отображений в дисплее (включая экспоненциальные числа).

2.3.4. Скриншоты результатов работы

На рисунке 7 отображен калькулятор в обычном режиме.

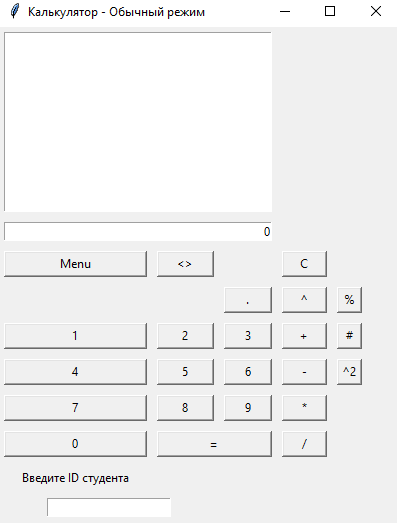


Рис. 7. Интерфейс калькулятора в обычном режиме.

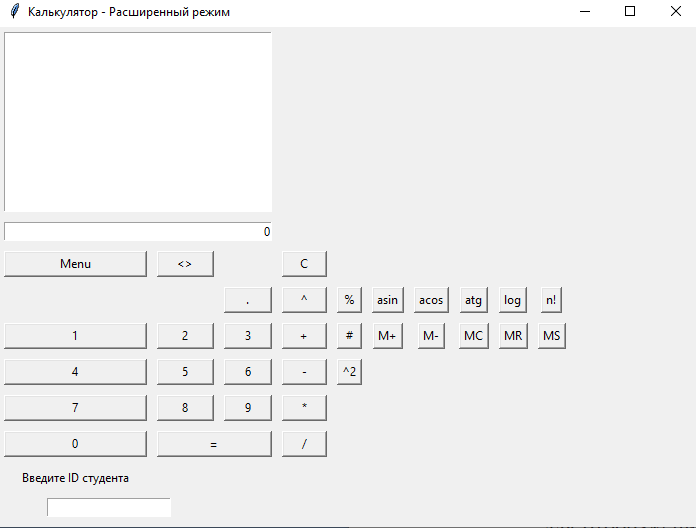


Рис. 8. Интерфейс в расширенном режиме с пустым полем ID.

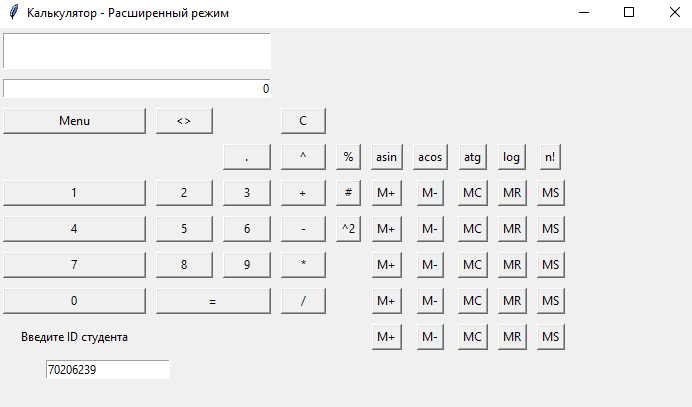


Рисунок 9. Интерфейс в расширенном режиме с введенным полем ID.

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 900x900 пикселей. В окне размещается холст (Canvas) для отображения стержней и дисков, а также поля ввода и кнопки для взаимодействия с пользователем.
2. Пользователь вводит идентификатор кода, где каждая цифра означает количество дисков на стержне. Например, «70206239» означает 7 дисков на первом стержне, 0 на втором и так далее.
3. На основе ID формирования списка списков, представляющего стержни и диски на них. Диски пронумерованы в порядке убывания размера.
4. Функция draw\_spl() рисует вертикальные стержни, а draw\_disk() отображает диски на стержнях с учетом их текущих положений и возможного зависимости.
5. 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. Кнопки «Начало» и «Конец» отображают начальные изменения дисков и финальное состояние, когда все диски располагаются на последнем конце.
7. Пользователь вводит процент (по умолчанию 70, 20, 62, 39), а функция obrabotka\_procentov() вычисляет количество итераций как долю от максимального числа перемещаемых. Затем выполните выполнение данного перемещения.
8. Если процент не соответствует целостности итераций, диск отображается «зависимым» между стержнями.

2.4.2. Исходный код реализации языка программирования

Исходный код представлен ниже:

**import** **tkinter** **as** **tk**

**from** **tkinter** **import** ttk, Canvas, messagebox

**from** **tkinter.ttk** **import** Style

**import** **math**

**import** **copy**

**from** **tkinter** **import** \*

root = tk.Tk()

root.geometry('900x900')

**class** **Hanoi**:

**def** **\_\_init\_\_**(self, master):

self.master = master

self.canvas = tk.Canvas(master, width=**900**, height=**600**)

self.canvas.pack()

self.splinders = [[],[],[],[],[],[],[],[]]

self.x\_base = **50**

self.y\_base = **450**

self.height\_disk = **15**

self.zav = []

self.id\_str = '0'

self.colors = ("gray", "pink", "brown", "cyan", "magenta", "lime", "teal", "purple")

self.height\_zav = **50**

self.digit = '0'

self.max\_iter = **0**

self.flag\_zav = False

self.schetchik = **0**

self.maks\_iter = **0**

self.vrem\_iter = **0**

self.tekushaya\_pozitsiya = []

self.iter = **0**

self.draw\_spl()

self.panel()

**def** **draw\_spl**(self):

**print**('Вызван метод draw\_sp')

self.canvas.delete('all')

self.canvas.create\_rectangle(**40**, **450**, **760**, **480**, width=**3**, outline="#783636")

x = self.x\_base

y0 = self.y\_base

**for** i **in** reversed(range(len(self.splinders))):

self.canvas.create\_rectangle(x - **2**, **100**, x + **2**, y0, fill='white', width=**3**, outline='#783636')

label = tk.Label(text=str(i + **1**), font="Times 12")

label.place(x=x-**5**, y=y0+**3**)

x += **100**

**print**('Метод draw\_spl полностью выполнен')

**def** **draw\_disk**(self, stergni):

**print**('Вызван метод draw\_disk')

self.canvas.delete("all")

self.draw\_spl()

x = self.x\_base

**for** num, i **in** enumerate(stergni):

y = self.y\_base

**if** self.zav **and** self.zav[**0**] == num:

**for** j **in** i[**0**:-**1**]:

self.canvas.create\_rectangle(x - math.floor(j / **2**), y, x + math.ceil(j / **2**), y - self.height\_disk,

fill=self.colors[j % **8**])

self.canvas.create\_text(x, y - **5**, text=(j), font="Times 8")

y -= self.height\_disk

j = i[-**1**]

sdvig = **100** \* (self.zav[**1**] - self.zav[**0**]) / **2**

self.canvas.create\_rectangle(x - math.floor(j / **2**) + sdvig, self.height\_zav, x + math.ceil(j / **2**) + sdvig,self.height\_zav - self.height\_disk, fill=self.colors[j % **8**])

self.canvas.create\_text(x + sdvig, self.height\_zav - **5**, text=(j), font="Times 8")

**else**:

**for** j **in** i:

self.canvas.create\_rectangle(x - math.floor(j / **2**), y, x + math.ceil(j / **2**), y - self.height\_disk,

fill=self.colors[j % **8**])

self.canvas.create\_text(x, y - **5**, text=(j), font="Times 8")

y -= self.height\_disk

x += **100**

**def** **spl\_val**(self):

self.splinders = [[],[],[],[],[],[],[],[]]

**def** **get\_id**(self):

self.id\_str = str(self.entry.get())

self.spl\_val()

**print**(self.id\_str)

**if** len(self.id\_str) > **8**:

messagebox.showinfo("Error",f'Количество цифр в id превышает 9!')

**return**

**else**:

**for** i, j **in** enumerate(self.id\_str):

**print**(j)

**for** i1 **in** reversed(range(int(j))):

self.splinders[i].append((len(self.id\_str) - i) \* **10** + i1 + **1**)

**print**(self.splinders)

self.draw\_disk(self.splinders)

self.count\_one()

**def** **get\_btn**(self, btn):

**print**('Вызван метод get\_btn')

**if** btn == '1':

value = int(self.entry\_p\_1.get())

**elif** btn == '2':

value = int(self.entry\_p\_2.get())

**elif** btn == '3':

value = int(self.entry\_p\_3.get())

**else**:

value = int(self.entry\_p\_4.get())

self.obrabotka\_procentov(value)

**def** **panel**(self):

btn\_style = ttk.Style()

btn\_style.configure('TButton', width=**20**, height=**50**, background='#783636')

self.entry = tk.Entry(self.master)

self.entry.place(x=**400**, y=**700**)

self.entry\_p\_1 = ttk.Entry(self.master, width=**5**)

self.entry\_p\_1.place(x=**380**,y=**550**)

self.entry\_p\_1.insert(**0**,'70')

self.entry\_p\_2 = ttk.Entry(self.master, width=**5**)

self.entry\_p\_2.place(x=**430**, y=**550**)

self.entry\_p\_2.insert(**0**,'20')

self.entry\_p\_3 = ttk.Entry(self.master, width=**5**)

self.entry\_p\_3.place(x=**480**,y=**550**)

self.entry\_p\_3.insert(**0**,'62')

self.entry\_p\_4 = ttk.Entry(self.master, width=**5**)

self.entry\_p\_4.place(x=**530**, y=**550**)

self.entry\_p\_4.insert(**0**,'39')

btn\_1 = ttk.Button(text='П.1', command=**lambda**: self.get\_btn('1'), width=**5**)

btn\_1.place(x=**380**, y=**600**)

btn\_2 = ttk.Button(text='П.2', command=**lambda**: self.get\_btn('2'), width=**5**)

btn\_2.place(x=**430**, y=**600**)

btn\_3 = ttk.Button(text='П.3', command=**lambda**: self.get\_btn('3'), width=**5**)

btn\_3.place(x=**480**, y=**600**)

btn\_4 = ttk.Button(text='П.4', command=**lambda**: self.get\_btn('4'), width=**5**)

btn\_4.place(x=**530**, y=**600**)

btn\_save = ttk.Button(text='Save', style='TButton', command=self.get\_id)

btn\_save.place(x=**400**, y=**750**)

btn\_start = ttk.Button(text='Start', style='TButton', command=self.start)

btn\_start.place(x=**150**, y=**550**)

btn\_end = ttk.Button(text='End', style='TButton', command=self.end)

btn\_end.place(x=**650**, y=**550**)

self.label = ttk.Label(self.master,text=f'Итераций: {self.digit}')

self.label.place(x=**400**, y=**500**)

**def** **update\_label**(self):

**try**:

**print**(f'Label exists: {self.label}')

self.label.config(text=f'Итераций: {self.digit}')

**except** **AttributeError** **as** e:

**print**("AttributeError caught:", e)

**def** **start**(self):

self.zav= []

self.draw\_disk(self.splinders)

self.digit = '0'

self.update\_label()

**def** **end**(self): # показываем, как всё будет в конце

self.zav = []

end\_pos = []

last\_st = []

**for** i **in** self.splinders:

last\_st += i

end\_pos.append([])

end\_pos[len(self.splinders) - **1**] = last\_st

self.draw\_disk(end\_pos)

self.vrem\_iter = str(self.max\_iter)

self.digit = self.vrem\_iter

self.update\_label()

**def** **count\_one**(self):

**print**("Вызван метод count\_one")

self.max\_iter = **0**

l\_rod = **0**

**for** i **in** reversed(range(**3**)):

l\_rod += len(self.splinders[i])

self.max\_iter += **2** \*\* l\_rod - **1**

**for** i **in** range(**3**, len(self.splinders)):

self.max\_iter += (**2** \*\* len(self.splinders[i]) - **1**)

l\_rod += len(self.splinders[i])

self.max\_iter += **2** \*\* l\_rod - **1**

**print**('Метод count\_move полностью выполнен')

**def** **count\_move**(self, num, pl1, pl2, pl3, dlina, fix1, fix2, fix3): # рекурсия для подсчета ходов

pw = **2** \*\* dlina

# Условие выхода из рекурсии

**if** dlina < **0**:

**return** (pl1, pl2, pl3) # или возвращайте что-то иное, если dlina меньше 0

**if** (pw < num):

num2 = num - pw

pl2.append(pl1[len(pl1) - **1** - dlina])

pl3 += pl1[len(pl1) - dlina:]

pl1 = pl1[:len(pl1) - **1** - dlina]

**return** self.count\_move(num2, pl3, pl2, pl1, dlina - **1**, fix3, fix2, fix1)

**elif** pw > num:

**return** self.count\_move(num, pl1, pl3, pl2, dlina - **1**, fix1, fix3, fix2)

**else**: # pw == num

pl2.append(pl1[len(pl1) - dlina - **1**])

pl3 += pl1[len(pl1) - dlina:]

pl1 = pl1[:len(pl1) - **1** - dlina]

**if** self.flag\_zav:

self.zav.append(fix2)

self.zav.append(fix1)

self.update\_label()

**return** (pl1, pl2, pl3)

**def** **run**(self): # выполняем перемещения

self.zav = []

self.tekushaya\_pozitsiya = copy.deepcopy(self.splinders)

self.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(self.splinders[i])

old\_vrem\_iter = self.vrem\_iter

self.vrem\_iter += **2** \*\* l\_rod - **1**

**if** self.vrem\_iter >= self.iter:

st1 = self.tekushaya\_pozitsiya[i]

st2 = []

st3 = []

st1, st2, st3 = self.count\_move(self.iter - old\_vrem\_iter, st1, st2, st3, len(self.tekushaya\_pozitsiya[i]) - **1**, i,ind1[i], ind2[i])

self.tekushaya\_pozitsiya[i] = st1

self.tekushaya\_pozitsiya[ind1[i]] += st2

self.tekushaya\_pozitsiya[ind2[i]] += st3

flag = True

**break**

**else**:

self.tekushaya\_pozitsiya[ind1[i]] += self.tekushaya\_pozitsiya[i]

self.tekushaya\_pozitsiya[i] = []

**if** **not** flag:

**for** i **in** range(**3**, len(self.splinders)):

old\_vrem\_iter = self.vrem\_iter

self.vrem\_iter += (**2** \*\* len(self.splinders[i]) - **1**)

**if** self.vrem\_iter >= self.iter:

st1 = self.tekushaya\_pozitsiya[i]

st2 = []

st3 = []

st1, st2, st3 = self.count\_move(self.iter - old\_vrem\_iter, st1, st2, st3, len(self.tekushaya\_pozitsiya[i]) - **1**,i, i - **1**, i - **2**)

self.tekushaya\_pozitsiya[i] = st1

self.tekushaya\_pozitsiya[i - **1**] += st2

self.tekushaya\_pozitsiya[i - **2**] += st3

**break**

**else**:

self.tekushaya\_pozitsiya[i - **1**] += self.tekushaya\_pozitsiya[i]

self.tekushaya\_pozitsiya[i] = []

l\_rod += len(self.splinders[i])

old\_vrem\_iter = self.vrem\_iter

self.vrem\_iter += **2** \*\* l\_rod - **1**

**if** self.vrem\_iter >= self.iter:

st1 = self.tekushaya\_pozitsiya[i - **1**]

st2 = []

st3 = []

st1, st2, st3 = self.count\_move(self.iter - old\_vrem\_iter, st1, st2, st3,

len(self.tekushaya\_pozitsiya[i - **1**]) - **1**, i - **1**, i, i - **2**)

self.tekushaya\_pozitsiya[i - **1**] = st1

self.tekushaya\_pozitsiya[i] += st2

self.tekushaya\_pozitsiya[i - **2**] += st3

**break**

**else**:

self.tekushaya\_pozitsiya[i] += self.tekushaya\_pozitsiya[i - **1**]

self.tekushaya\_pozitsiya[i - **1**] = []

self.draw\_disk(self.tekushaya\_pozitsiya)

**def** **obrabotka\_procentov**(self,cs): # обрабатываем проценты

self.vrem\_iter = int(cs) \* self.max\_iter / **100**

self.iter = int(self.vrem\_iter)

**if** self.vrem\_iter == self.iter:

self.flag\_zav = False

**else**:

self.iter += **1**

self.flag\_zav = True

self.digit = self.vrem\_iter

self.update\_label()

self.schetchik = **0**

**if** **0** <= int(cs) <= **100**:

self.run()

**else**:

messagebox.showerror("Error", "Неверный ввод процента")

app = Hanoi(root)

root.mainloop()

2.4.3. Тестирование и отладка

В процессе тестирования были проверены множество вариантов сбоя программы. Проверялись различные ID. При вводе ID больше 8 символов, программы выводит соответствующую ошибку. Диски расставляются на стержнях верно. Программа верно подсчитывается максимальное число перемещений. Кнопки «Начало» и «Конец» выполняются верно. Диски расстанавливаются либо в исходном положении, либо на последнем стержне в собранном состоянии. Программа верно расставляется на последнем стержне диски, в зависимости от их диаметра. В полях ввода процентных значениях, при некорректном вводе программа не выполняется.

2.4.4. Скриншоты результатов работы

На рисунке 10 показано изначальное состояние программы.

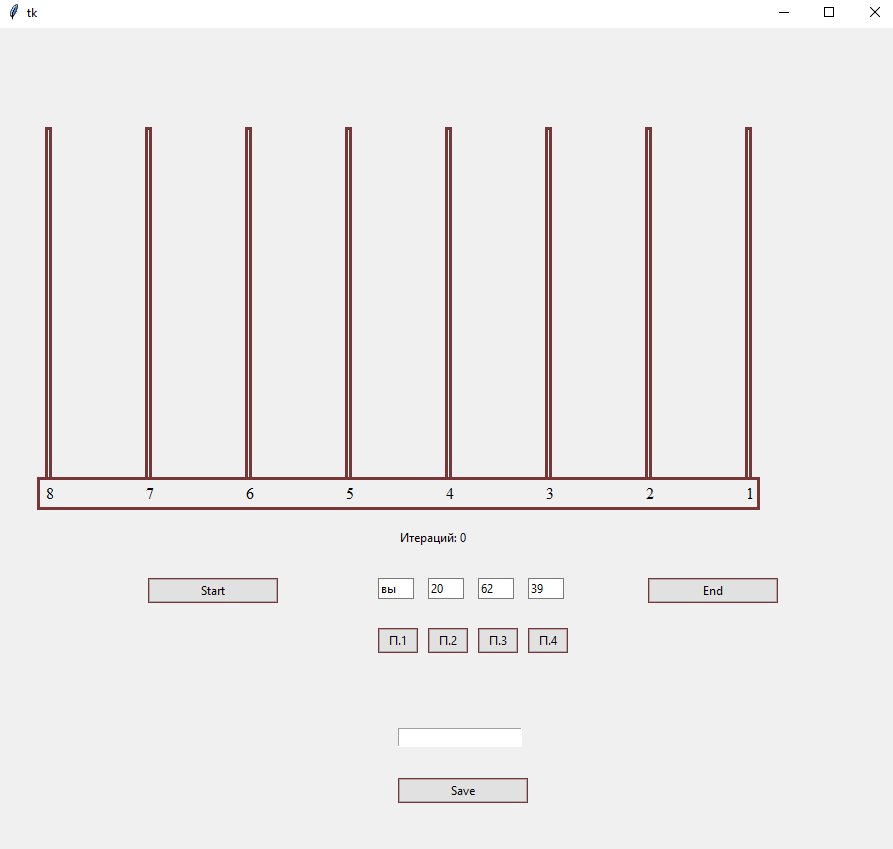


Рисунок 10. Изначальное состояние программы.



Рисунок 11. Начальное состояние дисков.

Окно с 8 стержнями, диски распределены согласно ID "70206239" (7 дисков на первом, 0 на втором и т.д.), цвета дисков случайные.

На рисунке 12 показано конечное состояние дисков.

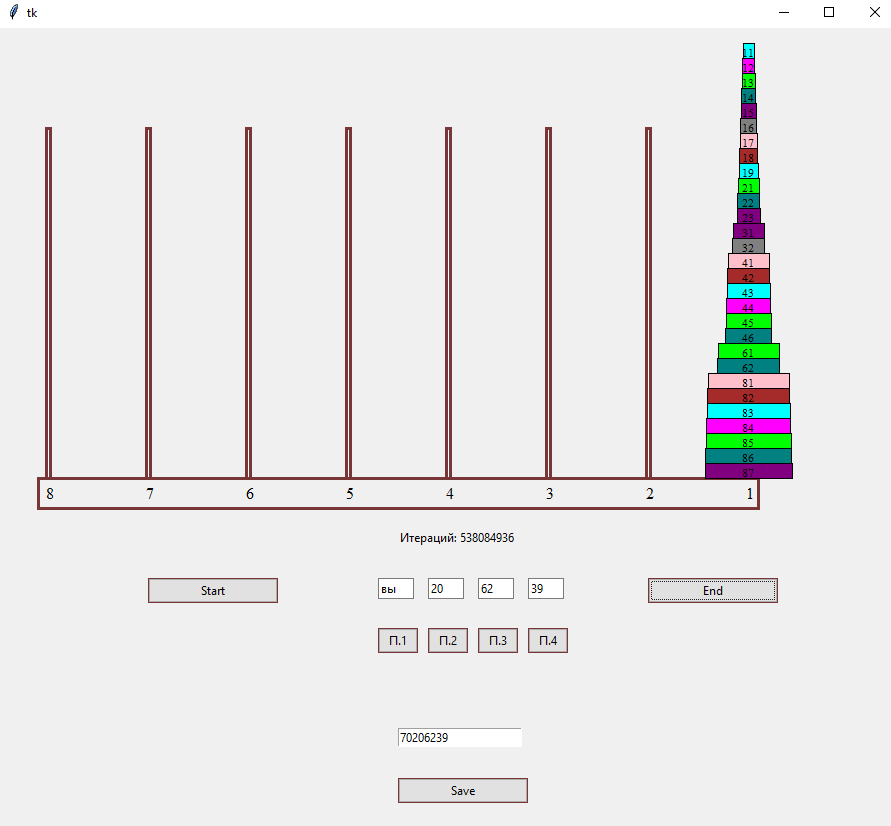


Рисунок 12. Конечное состояние дисков.

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

Промежуточное состояние (70%), как и зависание показаны на рисунке 13.

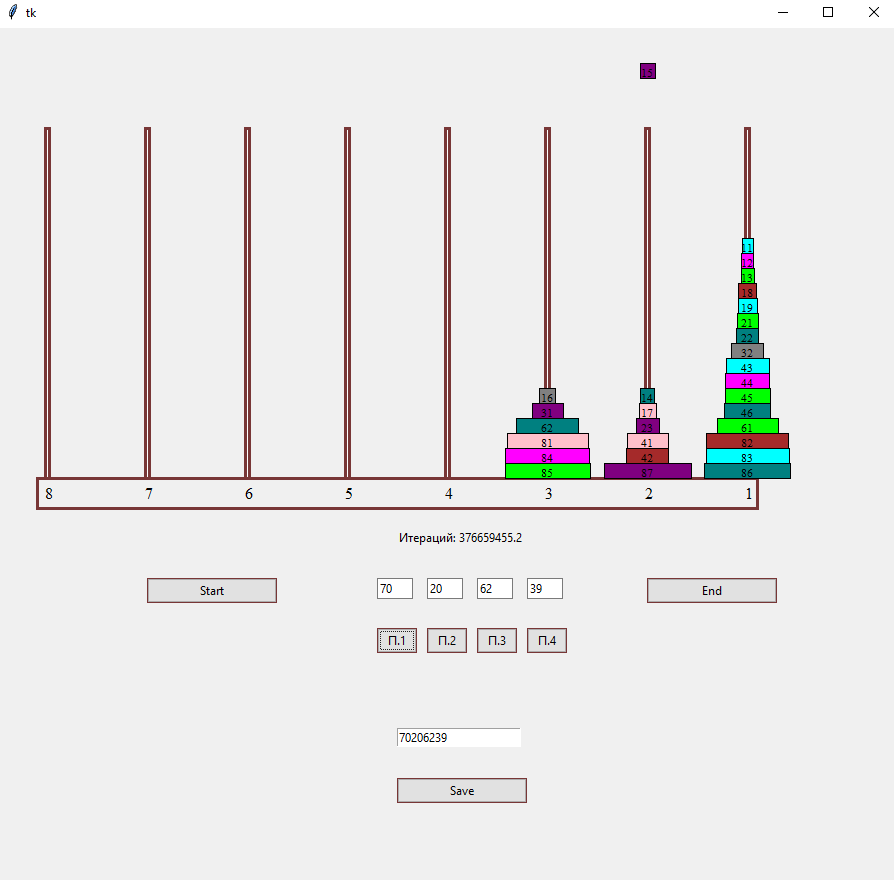


Рисунок 13. Промежуточное состояние и зависание диска.

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

2.6. Выводы по 2 главе

В процессе 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.

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

2. 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.

3. Файлы. Работа с файлами [Электронный ресурс]. – Режим доступа: <https://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html> - Загл. с экрана.

4. Руководство по Tkinter – METANIT [Электронный русурс] - <https://metanit.com/python/tkinter>- Загл. с экрана.

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

6. Ханойская башня - Википедия, свободная энциклопедия [Электронный ресурс]. – 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.

7. Словари (dict) и работа с ними. Методы словарей [Электронный ресурс]. – Режим доступа: <https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html> - Загл. с экрана.

8. Руководство по языку программирования Python – METANIT [Электронный ресурс] - <https://metanit.com/python/tutorial/> - Загл. с экрана.\

9. GUI-приложения с помощью Python-Tkinter [Электронный ресурс] – Режим доступа: <https://habr.com/ru/companies/otus/articles/903526/> - Загл. с экрана.

10. Tkinter – Интерфейс Python для Tcl/Tk [Электронный ресурс] – Режим доступа: <https://docs.python.org/3/library/tkinter.html> - Загл. с экрана.