



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Тестирование и верификация программного обеспечения»

Практическая работа №1

Студенты группы *ИКБО-50-23, Иващенко А.В.*
Галкин М. В

(подпись)

Преподаватель *Ильичев Г. П.*

(подпись)

Отчет представлен «__» _____ 202__ г.

Москва 2025 г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ СВОЕГО ПРОЕКТА

1. Введение

Поступил заказ для создания эмулятора Shell для языка оболочки UNIX-подобной операционной системы. Продукт должен реализовывать команды, содержать уже готовую конфигурацию и работать в режиме графического редактора.

2. Основания для разработки

Данный заказ поступил для проверки возможностей программистов в сфере разработки программ, на примере Эмулятора Shell. Проект должен продемонстрировать способности программистов и их компетентность в реализации сложных программ с использованием языка программирования Python. Для разработки предоставлены:

- Документация для работы в Python;
- Методические материалы для реализации заказа;
- ГОСТ 34.602-2020 для составления технического задания.

3. Назначение разработки

Целью разработки является создание простого Эмулятора Shell с конфигурацией в соответствии с заказом, а также определенными стандартными функциями для оболочки ОС. В итоге ожидается готовый безотказный продукт, способный продемонстрировать ожидаемый функционал.

4. Требования к программе

4.1 Функциональные требования

Эмулятор должен работать в режиме GUI и принимать образ виртуальной файловой системы в виде файла формата zip.

Должен быть создан конфигурационный файл в формате csv который будет содержать:

- Имя пользователя для показа в приглашении к вводу;
- Путь к архиву виртуальной файловой системы;
- Путь к старому архиву.

Эмулятор должен поддерживать команды:

- ls – отображение файлов и директорий внутри заданной;
- cd – перемещение по директориям;
- exit – выход из эмулятора;
- find – нахождение по шаблону файла/директории;
- chown – изменение владельца файла;
- tail – вывод последних строк текстового файла.

Запуск должен осуществляться через консоль (python main.py config.csv).

4.2 Требования к надежности

Эмулятор должен обрабатывать только команды, описанные в функциональных требованиях. Попытки ввести иные команды или случайный набор символов вне и внутри готовых должны сопровождаться сообщением об ошибке цвета, отличного от стандартного темного.

4.3 Условия эксплуатации

Операционная система Windows 10 или выше;

Минимальные требования персонального компьютера: современный процессор (любой), 256 Мб оперативной памяти, 1 Мб свободного места на

диске.

4.4 Требования к совместимости

Программа требует предустановки языка программирования Python, так как реализована на данном языке (.py).

5. Требования к интерфейсу

Эмулятор должен имитировать оболочку UNIX-подобной ОС, где основными цветами являются: розовый (фон терминала), темно-синий/близкий к черному (текст), красный (сообщения об ошибках) и белый/custom (внешняя составляющая терминала).

Текст должен быть разборчивым, достаточно большим. Формат вывода консоли: <пользователь>:~<текущая директория> “команда”. (С новой строки – результат выполнения команды, если предусмотрен).

6. Критерии приемки

Продукт считается соответствующим настоящему ТЗ и готовым к приемке, если:

- Успешно пройдены все тест-кейсы, составленные на основе функциональных требований, указанных в разделе 4.1;
- Интерфейс программы соответствует требованиям раздела 5;
- Программа запускается и функционирует на целевой операционной системе, указанной в п. 4.3.

7. Требования к документации

В состав поставки программного продукта должна входить следующая документация:

- Краткое руководство пользователя (в формате README.md).

8. Порядок контроля и приемки

Тестирование программы будет проводиться методом "черного ящика" на основе требований, изложенных в настоящем техническом задании.

Приемочные испытания включают в себя:

- Функциональное тестирование всех элементов интерфейса;
- Тестирование корректности вывода при выполнении команд;
- Тестирование удобства использования.

9. Этапы и сроки разработки

1. Проектирование архитектуры эмулятора – 1 дня;
2. Разработка кода программы и конфигурации – 4 дня;
3. Написание сопроводительной документации – 1 день;
4. Внутреннее тестирование – 1 день.

Общий срок разработки – 7 дней.

Руководство пользователя

1. Установка и запуск

1. Перейти по ссылке <https://github.com/Alexman454/TVS-2025-summer-IKBO-50-23-Umamusume2>;

2. Скачать как Zip архив файлы;
3. Распаковать архив и открыть в нем папку ПР1;
4. Скопировать путь и в консоли дойти до этой папки;
5. Запустить эмулятор в консоли:

– `python main.py config.csv`

ИЛИ

– `python3 main.py config.csv`

2. Эксплуатирование

У пользователя имеется поле ввода команды в низу интерфейса, поле для отображения результата ввода команд.

Пользователь может использовать 6 команд для работы с эмулятором:

ls <path> – Список файлов и директорий;

cd <path> – Смена директории;

exit – Выход из эмулятора;

tail [-n int] <path> – Создание файла;

chown <user> <path> – Вывод содержимого файла;

find <name> – Вывод содержимого файла.

ОПИСАНИЕ ОШИБОК ЭМУЛЯТОРА SHELL

1. Ошибка графического дизайна

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `init(self, terminal)` класса `Application` в файле `application.py`.
- Описание: Фон терминала имеет зеленый цвет вместо розового, как должно быть согласно техническому заданию.
- Способ обнаружения: При открытии консоли.

```
class Application:
```

```
def __init__(self, terminal):
    self.root = tk.Tk()
    self.root.title("Umamusume")
    self.root.geometry("500x600")
    self.root.resizable(False, False)
    self.root.configure(bg="#f4eff0")
    frame = tk.Frame(self.root)
    frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=(10, 0))
    self.output = tk.Text(frame, wrap=tk.WORD, bg="#f7d05", fg="#214050", font=("Courier New", 10), state=tk.DISABLED)
    self.output.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
    self.output.tag_configure("input", foreground="#214050")
    self.output.tag_configure("command", foreground="#214050")
    self.output.tag_configure("error", foreground="#c10619")
    self.input = tk.Entry(self.root, bg="#fdb5e0", fg="#214050", font=("Courier New", 10))
    self.input.pack(fill=tk.X, side=tk.BOTTOM, padx=10, pady=10)
    self.input.bind("<Return>", self.read)
    self.terminal = terminal
    self.terminal.link(self)
```

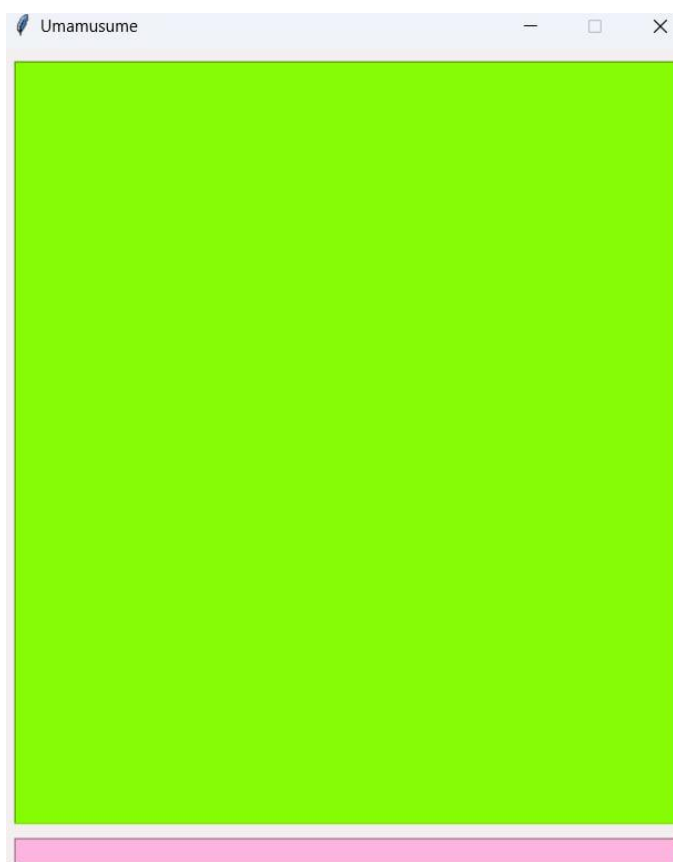


Рисунок 1-2 – Ошибка графического дизайна

2. Ошибка вывода

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `ls(self, args)` в файле `terminal.py`.
- Описание: Некорректный цвет текста в выводе результата исполнения команды.
- Способ обнаружения: При попытке использовать команду `ls`.

```
def ls(self, args):
    work_dir = self.path
    if len(args) > 0:
        work_dir = args[-1]
        work_dir = work_dir.strip('/')
        work_dir = work_dir.split('/')
        new_dir = self.path[:-1].split('/')
        if new_dir == [""]:
            new_dir = []
        for arg in work_dir:
            if arg == "..":
                if len(new_dir) > 0:
                    new_dir.pop()
            else:
                self.application.print("Некорректный путь к директории.", "error")
                work_dir = ""
        else:
            new_dir.append(arg)
        new_path = "/".join(new_dir) + "/"
        if new_path == "/":
            work_dir = ""
        for file in self.filesystem.namelist():
            if file.startswith(new_path):
                work_dir = new_path
        if work_dir is None:
            self.application.print("", "command")
    items = set()
    for item in self.filesystem.namelist():
        if item.startswith(work_dir):
            ls_name = item[len(work_dir):]
            if "/" in ls_name:
                ls_name = ls_name[:ls_name.index("/")]
            items.add(ls_name)
    self.application.print('\n'.join(sorted(filter(lambda x: len(x) > 0, items))), "error")
```

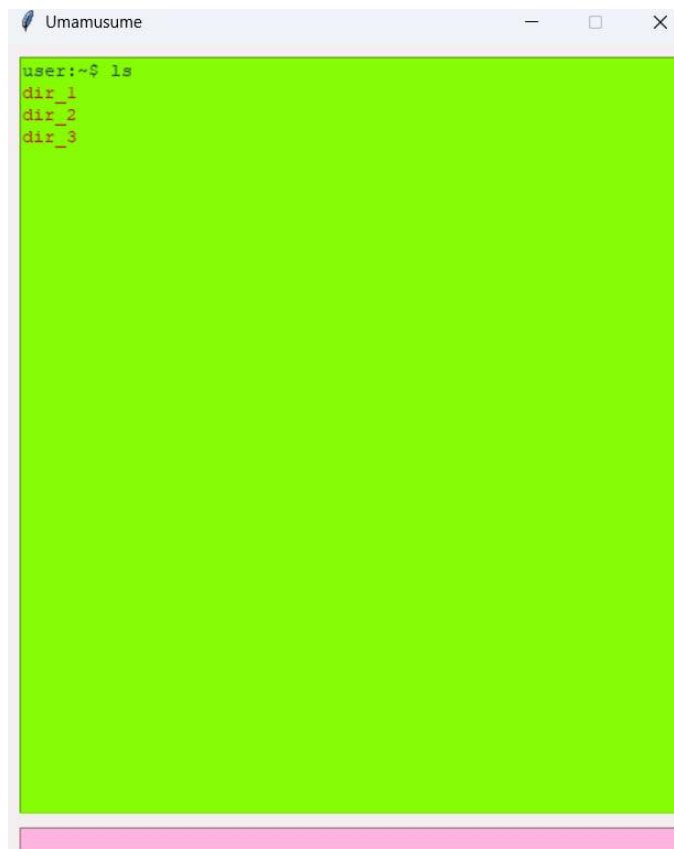


Рисунок 3-4 – Ошибка вывода

3. Ошибка вывода

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `cd(self, args)` в файле `terminal.py`.
- Описание: Некорректный цвет текста в выводе результата исполнения команды.
- Способ обнаружения: При попытке использовать команду `"cd .."` в корневой папке.

```
def cd(self, args):  
    if len(args) == 0:  
        return ""  
    directory = args[-1]  
    directory = directory.strip('/')  
    directory = directory.split('/')  
    new_dir = self.path[:-1].split('/')  
    if new_dir == [""]:  
        new_dir = []  
    for arg in directory:  
        if arg == "..":  
            if len(new_dir) > 0:  
                new_dir.pop()  
            else:  
                self.application.print("Ошибка. Некорректный путь к директории.", "command")  
                return  
        else:  
            new_dir.append(arg)
```



Рисунок 5-6 – Ошибка вывода

4. Ошибка вывода

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `chown(self, args)` в файле `terminal.py`.
- Описание: Некорректный текст в сообщении после выполнения команды.
- Способ обнаружения: При попытке использовать команду `chown`.

```
def chown(self, args):  
    if len(args) < 2:  
        return "Необходимо указать нового владельца и имя файла."  
  
    new_owner = args[0]  
    file_name = args[1]  
    file_path = self.path + file_name  
    temp_zip_path = self.fs_path + '.tmp'  
    self.application.print(f"Владелец(пробел) файла(пробель) '{file_name}' (пробел) изменён(пробел) на(пробел) '{new_owner}'.", "command")
```

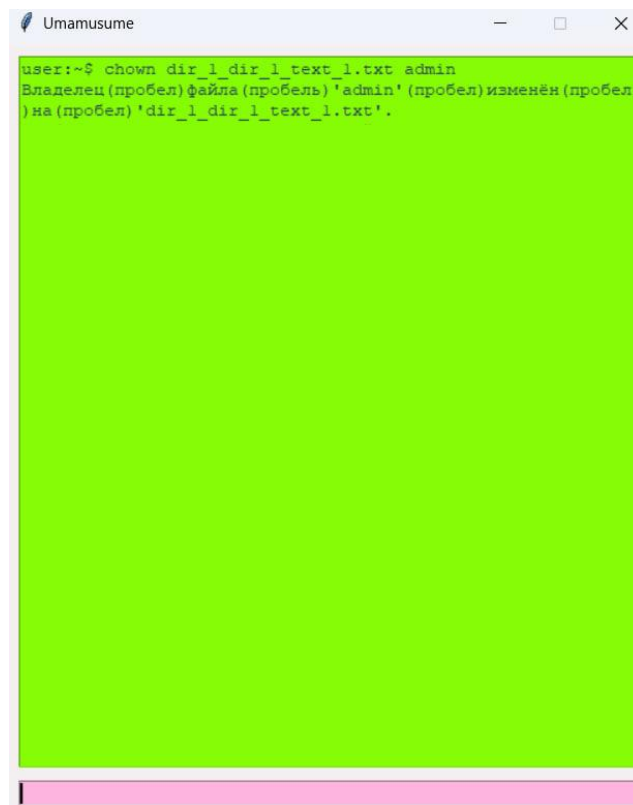
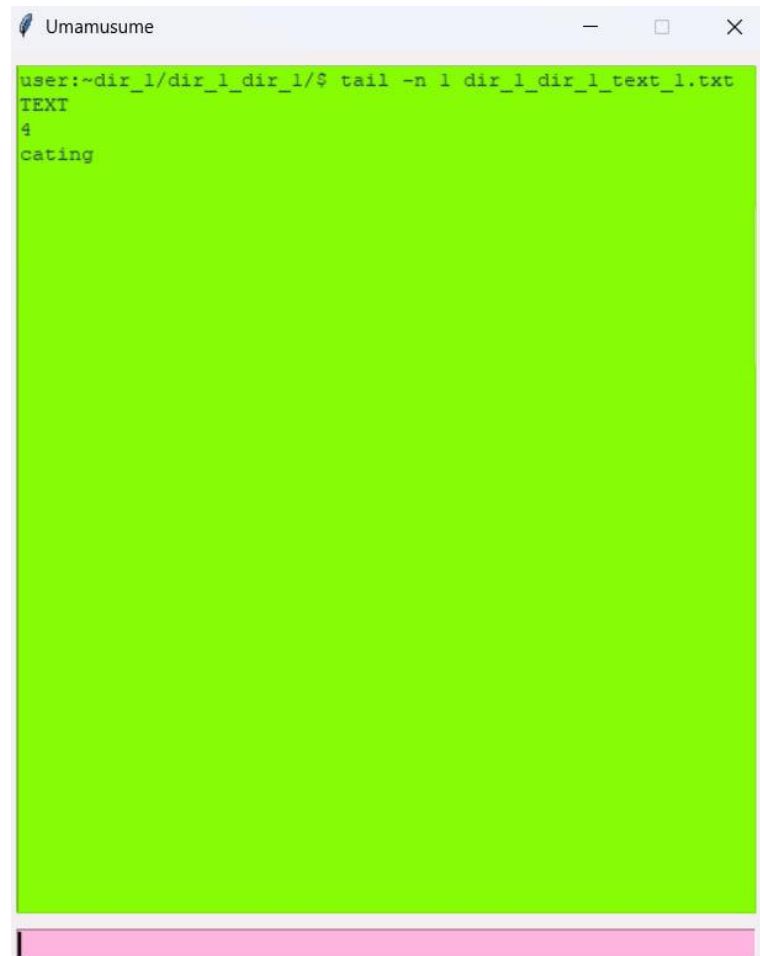


Рисунок 7-8 – Ошибка вывода

5. Ошибка вывода

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `tail(self, params)` в файле `terminal.py`.
- Описание: при попытке вывести содержимое одной строки командой `tail` выводится на больше, чем нужно строк.
- Способ обнаружения: При попытке использовать команду `tail` с условием `-n 1`.

```
def tail(self, params):
    if len(params) == 0:
        return "Неправильное название файла"
    file = params[-1]
    num_lines = 10
    if len(params) > 1 and params[0] == "-n":
        try:
            num_lines = int(params[-2])+2
        except ValueError:
            return "Некорректное количество строк."
```



```
Umamusume
user:~dir_1/dir_1_dir_1/$ tail -n 1 dir_1_dir_1_text_1.txt
TEXT
4
cating
```

Рисунок 9-10 – Ошибка вывода

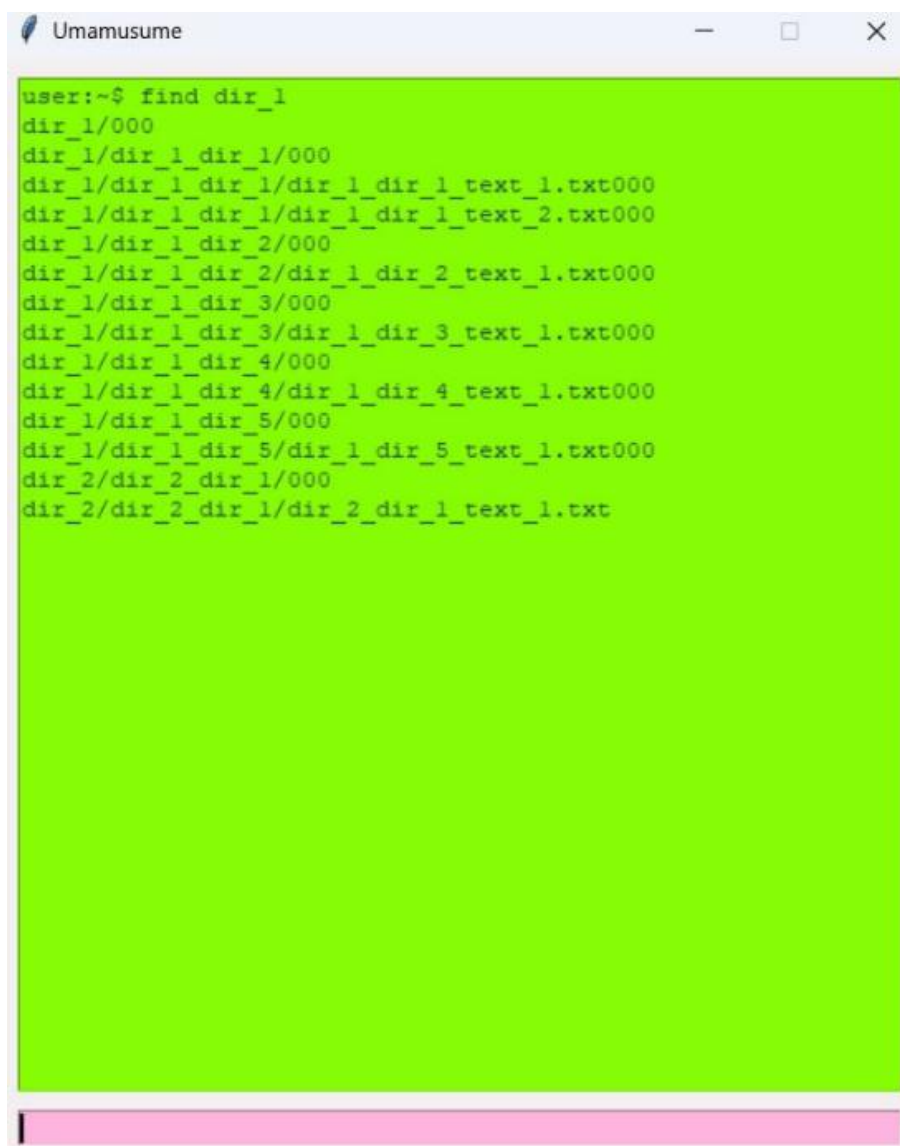
6. Ошибка вывода

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `find(self, args)` в файле `terminal.py`.
- Описание: Выводятся лишние символы (000 в конце).
- Способ обнаружения: При попытке использовать команду `find`.

```

def find(self, args):
    if len(args) < 1:
        return ""
    search_term = args[0]
    results = []
    for item in self.filesystem.namelist():
        if search_term in item:
            results.append(item)
    if results:
        self.application.print('000\n'.join(sorted(results)), "command")
        return ""
    else:
        return ""

```



```

user:~$ find dir_1
dir_1/000
dir_1/dir_1_dir_1/000
dir_1/dir_1_dir_1/dir_1_dir_1_text_1.txt000
dir_1/dir_1_dir_1/dir_1_dir_1_text_2.txt000
dir_1/dir_1_dir_2/000
dir_1/dir_1_dir_2/dir_1_dir_2_text_1.txt000
dir_1/dir_1_dir_3/000
dir_1/dir_1_dir_3/dir_1_dir_3_text_1.txt000
dir_1/dir_1_dir_4/000
dir_1/dir_1_dir_4/dir_1_dir_4_text_1.txt000
dir_1/dir_1_dir_5/000
dir_1/dir_1_dir_5/dir_1_dir_5_text_1.txt000
dir_2/dir_2_dir_1/000
dir_2/dir_2_dir_1/dir_2_dir_1_text_1.txt

```

Рисунок 11-12 – Ошибка вывода

ТЕХНИЧЕСКОЕ ЗАДАНИЕ СТОРОННЕГО ПРОЕКТА

Введение

Данный документ представляет собой техническое задание (ТЗ) на разработку консольного приложения «MatrixCalculator». Приложение предназначено для выполнения базовых и продвинутых математических операций с матрицами. Область применения включает учебный процесс (для студентов и преподавателей математических специальностей), инженерные расчеты и прототипирование алгоритмов линейной алгебры.

Основания для разработки

Разработка инициирована в связи с отсутствием простых, кроссплатформенных и не требующих графического интерфейса инструментов для работы с матрицами.

Назначение разработки

Приложение предназначено для выполнения базовых и продвинутых математических операций с матрицами. Область применения включает учебный процесс (для студентов и преподавателей математических специальностей), инженерные расчеты и прототипирование алгоритмов линейной алгебры.

Требования к программе

Функциональные требования

Приложение должно предоставлять пользовательское меню для выбора операций. Программа должна последовательно запрашивать у пользователя все необходимые данные.

Перечень функций:

1. Ввод матриц: Возможность ввода матриц с клавиатуры с указанием размерности.
2. Сложение и вычитание матриц: Операции выполняются над матрицами одинаковой размерности.
3. Умножение матрицы на число.
4. Перемножение матриц: С контролем согласованности размерностей (число столбцов первой матрицы должно равняться числу строк второй).
5. Транспонирование матрицы.
6. Вычисление определителя (детерминанта) матрицы: Для квадратных матриц.
7. Вычисление обратной матрицы: Для квадратных невырожденных матриц (с определителем $\neq 0$).
8. Возведение матрицы в степень: Для натуральных степеней и квадратных матриц.

9. Выход из программы.

Требования к надежности

Отказоустойчивость: Приложение не должно завершаться с ошибкой при вводе пользователем нечисловых значений, неверных размерностей матриц или при попытке выполнения недопустимой операции (например, деление на ноль при вычислении обратной матрицы). В таких случаях должно выводиться понятное сообщение об ошибке с предложением повторить ввод или выбрать другую операцию.

Восстановление после сбоев: после возникновения ошибки приложение должно возвращать пользователя в главное меню или на предыдущий шаг ввода.

Условия эксплуатации

Среда выполнения: Интерпретатор Python версии 3.8 или выше.

Аппаратные требования: Любой компьютер, способный запускать интерпретатор Python. Особых требований к объему оперативной памяти или процессору не предъявляется.

Требования к совместимости

Приложение должно быть кроссплатформенным и работать в любой операционной системе, где установлен корректный интерпретатор Python (Windows, Linux, macOS).

Взаимодействие с другими системами не предусмотрено.

Требования к интерфейсу

Интерфейс — консольный (командная строка). Взаимодействие осуществляется через последовательный вывод меню и запросов на ввод данных.

Текстовое описание интерфейса:

1. При запуске отображается главное меню с нумерованным списком операций.
2. Пользователь вводит цифру, соответствующую операции.
3. Для операций, требующих ввода матриц, программа запрашивает:

Количество строк и столбцов.

Поэлементный ввод значений матрицы (по строкам).

4. После ввода всех необходимых данных программа выводит на экран исходные матрицы (если применимо) и результат операции в читаемом, форматированном виде (например, каждую строку матрицы с отступами).

5. После вывода результата программа возвращает пользователя в главное меню.

Критерии приемки

- Успешно выполняются 100% позитивных тест-кейсов (корректные данные) для всех заявленных операций. Результаты должны совпадать с расчетами, выполненными вручную.

- Успешно обрабатываются 100% негативных тест-кейсов (некорректные данные: буквы вместо цифр, неверные размерности и т.д.) с выводом адекватных сообщений об ошибках без завершения работы программы.

- Программа запускается и выполняет все заявленные функции на трех указанных ОС: Windows 10, Ubuntu 22.04, macOS Ventura.

Требования к документации

Обязательная документация

- Запрос пользователя с описанием требуемой системы
- Техническое описание программы
- Техническое задание
- Исходный код с соответствующими комментариями

Порядок контроля и приемки

Приемка будет проводиться путем тестирования методом «черного ящика» на соответствие критериям приемки (п.6). Разработчик предоставляет исполняемый `.ру` файл и документацию. Заказчик проводит тестирование по заранее подготовленным тестовым сценариям:

Тестирование включает в себя следующие группы тест-кейсов:

1. Позитивное тестирование (корректные данные и операции):

Сложение матриц.

Действия: Выбрать операцию сложения. Ввести две матрицы размерности 2x2 с целыми числами.

Ожидаемый результат: Программа выводит корректную сумму матриц.

Вычитание матриц.

Действия: Выбрать операцию вычитания. Ввести две матрицы размерности 2x2.

Ожидаемый результат: Программа выводит корректную разность матриц.

Умножение матрицы на число.

Действия: Выбрать операцию умножения на число. Ввести матрицу и числовой скаляр.

Ожидаемый результат: Программа выводит матрицу, каждый элемент которой умножен на скаляр.

Умножение матриц (согласованные размерности).

Действия: Выбрать операцию умножения матриц. Ввести матрицу А размерности 2×3 и матрицу В размерности 3×2 .

Ожидаемый результат: Программа выводит корректную матрицу-произведение размерности 2×2 .

Транспонирование матрицы.

Действия: Выбрать операцию транспонирования. Ввести прямоугольную матрицу (например, 3×2).

Ожидаемый результат: Программа выводит корректно транспонированную матрицу (2×3).

Вычисление определителя.

Действия: Выбрать операцию вычисления определителя. Ввести квадратную матрицу (например, 3×3) с известным определителем, не равным нулю.

Ожидаемый результат: Программа выводит корректное числовое значение определителя.

Вычисление обратной матрицы.

Действия: Выбрать операцию нахождения обратной матрицы. Ввести квадратную матрицу (2×2 или 3×3) с известным определителем, не равным нулю.

Ожидаемый результат: Программа выводит корректную обратную матрицу. Проверка: умножение исходной матрицы на полученную обратную дает единичную матрицу (проверка вручную или выбором операции умножения в программе).

Возведение матрицы в степень.

Действия: Выбрать операцию возведения в степень. Ввести квадратную матрицу и натуральную степень (например, 2 или 3).

Ожидаемый результат: Программа выводит корректно возведенную в степень матрицу.

2. Негативное тестирование (обработка ошибок):

Ввод нечисловых значений.

Действия: На любом шаге ввода (размерность, элемент матрицы) ввести символы, отличные от цифр (например, abc, 1.2.3).

Ожидаемый результат: Программа выводит понятное сообщение об ошибке и повторяет запрос, не завершаясь аварийно.

Операции с матрицами несовместимой размерности.

Действия:

а) Выбрать сложение/вычитание. Ввести матрицы разной размерности (например, 2×2 и 3×3).

б) Выбрать умножение матриц. Ввести матрицы, где число столбцов первой не равно числу строк второй (например, 2×2 и 3×3).

Ожидаемый результат: Программа выводит понятное сообщение об ошибке (например, "Ошибка: для сложения матрицы должны быть одного размера") и возвращает в меню или шаг ввода.

Операции, требующие квадратную матрицу, с неквадратной.

Действия: Выбрать операцию нахождения определителя/обратной матрицы/возведения в степень. Ввести прямоугольную матрицу (например, 2×3).

Ожидаемый результат: Программа выводит понятное сообщение об ошибке (например, "Ошибка: матрица должна быть квадратной").

Вычисление обратной матрицы для вырожденной матрицы (определитель = 0).

Действия: Выбрать операцию нахождения обратной матрицы. Ввести квадратную матрицу с нулевым определителем.

Ожидаемый результат: Программа выводит понятное сообщение об ошибке (например, "Ошибка: определитель матрицы равен нулю, обратной матрицы не существует").

ТК-Н-05: Ввод недопустимой размерности.

Действия: При запросе размерности ввести число меньше или равное

нулю (например, 0, -5).

Ожидаемый результат: Программа выводит сообщение об ошибке (например, "Размерность матрицы должна быть положительным числом") и повторяет запрос.

ОПИСАНИЕ ДОПУЩЕННЫХ ОШИБОК В ПО "MatrixCalculator"

1. Ошибка валидации размера матрицы при вводе

- Тип ошибки: Логическая/Интерфейсная
- Местоположение: Функция `input_matrix()` в основном файле
- Описание: Закомментирована проверка соответствия количества введенных элементов заданной размерности матрицы
- Способ обнаружения: При вводе строки с количеством элементов, не соответствующим заявленной размерности (например, для матрицы 2x2 ввести 3 элемента в строке)

```
def input_matrix():  
    """Ввод матрицы с клавиатуры"""  
    try:  
        rows = int(input("Введите количество строк: "))  
        cols = int(input("Введите количество столбцов: "))  
  
        print(f"Введите элементы матрицы {rows}x{cols} построчно (через пробел):")  
        matrix = []  
  
        for i in range(rows):  
            try:  
                row = list(map(float, input(f"Строка {i + 1}: ").split()))  
                # if len(row) != cols:  
                #     print(f"Ошибка: должно быть {cols} элементов в строке")  
                #     raise ValueError  
                matrix.append(row)  
            except ValueError:  
                return None  
        return matrix  
    except ValueError:  
        print("Ошибка: введите целые числа для размерности")  
        return None
```

```
=====
Калькулятор матриц
=====
1. Ввод матрицы
2. Сложение матриц
3. Вычитание матриц
4. Умножение матрицы на число
5. Умножение матриц
6. Транспонирование матрицы
7. Вычисление определителя
8. Вычисление обратной матрицы
9. Возведение матрицы в степень
0. Выход
=====
Выберите операцию (0-9): 1
Введите количество строк: 2
Введите количество столбцов: 2
Введите элементы матрицы 2x2 построчно (через пробел):
Строка 1: 1 2 3
Строка 2: 3 2 1

Матрица:
1.00  2.00  3.00
3.00  2.00  1.00
```

Рисунок 13-14 – Ошибка валидации размера матрицы при вводе

2. Ошибка валидации операций с матрицами

- Тип ошибки: Логическая
- Местоположение: Функции `matrix_addition()`, `matrix_multiplication()`
- Описание: Закомментированы проверки совместимости размерностей матриц для операций сложения и умножения
- Способ обнаружения: Попытка сложить матрицы разной размерности или умножить матрицы с несовместимыми размерами

```
def matrix_addition():
    """Сложение матриц"""
    print("Введите первую матрицу:")
    m1 = input_matrix()
    if m1 is None: return

    print("Введите вторую матрицу:")
    m2 = input_matrix()
    if m2 is None: return

    # if len(m1) != len(m2) or (len(m1[0]) != len(m2[0])):
    #     print("Ошибка: матрицы должны быть одинаковой размерности")
    #     return
    result = m.add(m1, m2)

    m.print_matrix(result, "Результат сложения")

def matrix_multiplication():
    """Умножение матриц"""
    print("Введите первую матрицу:")
    m1 = input_matrix()
    if m1 is None: return

    print("Введите вторую матрицу:")
    m2 = input_matrix()
    if m2 is None: return

    # if len(m1[0]) != len(m2):
    #     print("Ошибка: число столбцов первой матрицы должно равняться числу строк второй")
    #     return

    result = m.multiply(m1, m2)
    m.print_matrix(result, "Результат умножения матриц")
```

```
=====
Калькулятор матриц
=====
1. Ввод матрицы
2. Сложение матриц
3. Вычитание матриц
4. Умножение матрицы на число
5. Умножение матриц
6. Транспонирование матрицы
7. Вычисление определителя
8. Вычисление обратной матрицы
9. Возведение матрицы в степень
0. Выход
=====
Выберите операцию (0-9): 2
Введите первую матрицу:
Введите количество строк: 2
Введите количество столбцов: 2
Введите элементы матрицы 2x2 построчно (через пробел):
Строка 1: 1 2
Строка 2: 2 1
Введите вторую матрицу:
Введите количество строк: 3
Введите количество столбцов: 3
Введите элементы матрицы 3x3 построчно (через пробел):
Строка 1: 1 2 3
Строка 2: 2 3 1
Строка 3: 3 1 2

Результат сложения:
2.00  4.00
4.00  4.00
```

Рисунок 15-17 – Ошибка валидации операций с матрицами

3. Ошибка обработки вырожденных матриц

- Тип ошибки: Логическая
- Местоположение: Функция `matrix_inverse()`
- Описание: Закомментирована проверка определителя на равенство нулю перед вычислением обратной матрицы
- Способ обнаружения: Попытка вычислить обратную матрицу для вырожденной матрицы (с определителем = 0)

```
def matrix_inverse():
    """Вычисление обратной матрицы"""
    print("Введите квадратную матрицу:")
    matrix = input_matrix()
    if matrix is None: return

    if len(matrix) != len(matrix[0]):
        print("Ошибка: матрица должна быть квадратной")
        return

    # d = m.det(matrix)
    # if abs(d) < 1e-10:
    #     print("Ошибка: определитель равен нулю, обратной матрицы не существует")
    #     return
```



```
=====
Калькулятор матриц
=====
1. Ввод матрицы
2. Сложение матриц
3. Вычитание матриц
4. Умножение матрицы на число
5. Умножение матриц
6. Транспонирование матрицы
7. Вычисление определителя
8. Вычисление обратной матрицы
9. Возведение матрицы в степень
0. Выход
=====
Выберите операцию (0-9): 9
Введите квадратную матрицу:
Введите количество строк: 2
Введите количество столбцов: 2
Введите элементы матрицы 2x2 построчно (через пробел):
Строка 1: 1 2
Строка 2: 2 3
Введите степень (натуральное число): -1

Матрица в степени -1:
1.00  2.00
2.00  3.00
```

Рисунок 20 -21 – Ошибка валидации степени матрицы

5. Неполная обработка ошибок

- Тип ошибки: Логическая
- Местоположение: Функция `matrix_inverse()` в модуле `matrix.py`
- Описание: Закомментирована проверка на вырожденность матрицы

в алгоритме обращения

- Способ обнаружения: Попытка обратить вырожденную матрицу

```
def matrix_inverse(matrix):
    """Вычисление обратной матрицы методом Гаусса-Жордана"""
    n = len(matrix)

    # Проверка на квадратность
    if n != len(matrix[0]):
        raise ValueError("Матрица должна быть квадратной")

    # Создаем расширенную матрицу [A|I]
    augmented = []
    for i in range(n):
        row = matrix[i][:] # Копируем строку исходной матрицы
        # Добавляем единичную матрицу
        row.extend([1 if j == i else 0 for j in range(n)])
        augmented.append(row)

    # Прямой ход метода Гаусса
    for i in range(n):
        # Ищем максимальный элемент в столбце
        max_row = i
        for j in range(i + 1, n):
            if abs(augmented[j][i]) > abs(augmented[max_row][i]):
                max_row = j

        # Меняем строки местами
        augmented[i], augmented[max_row] = augmented[max_row], augmented[i]

        # Проверяем, что диагональный элемент не нулевой
        # if abs(augmented[i][i]) < 1e-10:
        #     raise ValueError("Матрица вырожденная, обратной не существует")
```

```
=====
Калькулятор матриц
=====
1. Ввод матрицы
2. Сложение матриц
3. Вычитание матриц
4. Умножение матрицы на число
5. Умножение матриц
6. Транспонирование матрицы
7. Вычисление определителя
8. Вычисление обратной матрицы
9. Возведение матрицы в степень
0. Выход
=====
Выберите операцию (0-9): 6
Введите матрицу:
Введите количество строк: 2
Введите количество столбцов: 2
Введите элементы матрицы 2x2 построчно (через пробел):
Строка 1: 1 1
Строка 2: 1 1

Транспонированная матрица:
1.00  1.00
1.00  1.00
```

Рисунок 22-23 – Неполная обработка ошибок

ЗАКЛЮЧЕНИЕ

В ходе практической работы мы прописали техническое задание с дополнительной документацией, создали программу на её основе с внесением в неё ошибок и протестировали чужую работу, обменявшись программами и документацией к ним. Составленное ТЗ чужой команды выполнено в удобном формате, понятным простым пользователям. Сама программа тоже работает в крайне удобном формате через консоль. Проверив работу программы, мы выявили ошибки в программе и задокументировали их. В конечном итоге мы получили практический опыт тестирования программы методом «Черного ящика».