

Индивидуальное домашнее задание-3 (Гринченко Евгений, БПИ 236, вариант 15)

Небольшое предисловие перед началом - код программы я реализовывал сразу на оценку 8-9, поэтому не удалось соблюдать условия итеративности. Постараюсь отразить в этом отчёте выполняемость критериев на оценку 4-7.

Финальный вариант программы, которую я реализовал, чтобы претендовать на оценку 10, был разбит на 3 ассемблерных файла:

1. Файл data.asm

```
.include "macros.inc"

.equiv NAME_SIZE 256 # Размер буфера для имени файла
.equiv TEXT_SIZE 512 # Размер буфера для текста

.data

msg_yes:    .asciz "YES\n"
msg_no:     .asciz "NO\n"
input_answer: .space 3
test1:      .asciz "test1.txt"
test1_inv:   .asciz "test1_inv.txt"
test2:      .asciz "test2.txt"
test2_inv:   .asciz "test2_inv.txt"
test3:      .asciz "test3.txt"
test3_inv:   .asciz "test3_inv.txt"
result1:     .asciz "result1.txt"
result2:     .asciz "result2.txt"
result3:     .asciz "result3.txt"
file_name:   .space NAME_SIZE
buffer:      .space TEXT_SIZE
.align 3
```

2. Файл idz.asm

```
.text
.globl main
```

```

main:
    print("-----Начало программы-----\n")
    print("---Программа,которая на основе заданной ASCII-строки символов, решает вопрос, является ли
данная строка палиндромом---")
    print("Введите 0, если хотите запустить тесты, иначе любое другое число: ")
    print_console("Write your answer(0-test program,else - usual program): ")
    li a7,5
    ecall
    li t0, 0
    beq a0,t0, test_prog
    # Ввод пути до читаемого файла
    print("Input file path to read in console: ")
    print_console("Write your path of input-file: ")
    get(file_name, NAME_SIZE)

    # Открытие файла для чтения
    open(file_name, READ_ONLY)
    li    t0, -1
    beq   a0, t0, error_name
    mv    s0, a0  # Дескриптор исходного файла

    # Динамическая память для хранения содержимого
    li    a7, 9
    li    a0, TEXT_SIZE
    ecall
    mv    s3, a0  # Начало памяти
    mv    s5, a0  # Текущий адрес в памяти
    li    s6, 0   # Счетчик прочитанного текста

read_file:
    read_addr_reg(s0, s5, TEXT_SIZE)
    li    t1, -1
    beq   a0, t1, error_read
    beqz  a0, end_read
    add   s5, s5, a0
    add   s6, s6, a0
    j     read_file

end_read:
    # Закрытие исходного файла
    close(s0)

    # Запись данных в обратном порядке
    print("Input file path to write reversed content: ")
    print_console("Write your path of reversed input-file: ")
    get(file_name, NAME_SIZE)

```

```

# Открытие файла для записи
open(file_name, WRITE_ONLY)
mv    s1, a0

# Запись в обратном порядке
mv    t0, s6
addi  t0, t0, -1
reverse_write_loop:
    bltz  t0, reverse_write_done
    add   t1, s3, t0
    lbu   t2, 0(t1)
    li    a7, 64
    mv    a0, s1
    mv    a1, t1
    li    a2, 1
    ecall
    addi  t0, t0, -1
    j     reverse_write_loop
reverse_write_done:
    # Заккрытие файла
    close(s1)

# Проверка палиндрома (палиндром_проверка)
mv    t0, s3      # Начало исходного файла
add   t1, s3, s6   # Конец обратного файла
addi  t1, t1, -1

palindrome_check:
    blt  t1, t0, palindrome_yes # Если указатели пересеклись, палиндром

skip_start_spaces:
    lbu   t4, 0(t0)
    li    t6, 0x20      # ASCII код пробела
    beq   t4, t6, skip_start_space
    j     check_end_spaces
skip_start_space:
    addi  t0, t0, 1
    j     skip_start_spaces

check_end_spaces:
    lbu   t4, 0(t1)
    beq   t4, t6, skip_end_space
    j     compare_chars
skip_end_space:
    addi  t1, t1, -1

```

```
j    check_end_spaces
```

```
compare_chars:
```

```
lbu   t2, 0(t0)
```

```
lbu   t3, 0(t1)
```

```
# Приведение к одному регистру (заглавные к строчным)
```

```
li    t6, 'A'
```

```
li    s7, 'Z'
```

```
blt   t2, t6, next_check
```

```
bgt   t2, s7, next_check
```

```
addi  t2, t2, 32
```

```
next_check:
```

```
blt   t3, t6, char_comp
```

```
bgt   t3, s7, char_comp
```

```
addi  t3, t3, 32
```

```
char_comp:
```

```
bne   t2, t3, palindrome_no
```

```
addi  t0, t0, 1
```

```
addi  t1, t1, -1
```

```
j     palindrome_check
```

```
palindrome_yes:
```

```
li    t0, 1
```

```
j     write_result
```

```
palindrome_no:
```

```
li    t0, 0
```

```
j     write_result
```

```
write_result:
```

```
print("Input file path to write: ")
```

```
print_console("Write your path of output-file: ")
```

```
get(file_name, NAME_SIZE)
```

```
write_result_continue:
```

```
# Открыть или создать файл для записи результата
```

```
open(file_name, WRITE_ONLY)
```

```
mv     s2, a0      # Сохранить дескриптор файла
```

```
beqz   t0, write_result_no
```

```
j     write_result_yes
```

```
write_result_yes:
```

```
li     a7, 64
```

```
mv     a0, s2
```

```
la     a1, msg_yes # msg_yes/msg_no зависит от результата
```

```
li    a2, 4
ecall
```

```
# Заккрытие файла
close(s2)
la t0, msg_yes
j choice
```

write_result_no:

```
# Записать результат
li    a7, 64
mv    a0, s2
la    a1, msg_no # msg_yes/msg_no зависит от результата
li    a2, 4
ecall
```

```
# Заккрытие файла
close(s2)
la t0, msg_no
j choice
```

choice:

```
print("Want to display the result on console? Write Y or N: ")
print_console("Write your answer: ")
get(input_answer, 3)
li s4, 'Y'
li s5, 'N'
lb t1, input_answer
beq t1, s4, output_yes
beq t1, s5, output_no
print("\nIncorrect input!")
exit
```

output_yes:

```
mv a0, t0
li a7, 4
ecall
print("The program has been successfully completed")
exit
```

output_no:

```
print("The program has been successfully completed")
exit
```

end_program:

```
exit
```

```

error_name:
    print("Incorrect file name\n")
    exit

error_read:
    print("Incorrect read operation\n")
    exit
test_prog:
    print("Запуск первого теста\n")
    print_console("Вывод первого теста: ")
    test(test1,test1_inv,result1,NAME_SIZE)
    print("Запуск второго теста\n")
    print_console("Вывод второго теста: ")
    test(test2,test2_inv,result2,NAME_SIZE)
    print("Запуск третьего теста\n")
    print_console("Вывод третьего теста: ")
    test(test3,test3_inv,result3,NAME_SIZE)
    print("Завершение тестов")
    exit

```

3. Библиотека макросов macros.inc

```

.macro get(%strbuf, %size)
    la a0 %strbuf
    li a1 %size
    li a7 8
    ecall
    push(s6)
    push(s7)
    push(s8)
    li s6 '\n'
    la s7 %strbuf
next:
    lb s8 (s7)
    beq s6 s8 replace
    addi s7 s7 1
    b next
replace:
    sb zero (s7)
    pop(s8)
    pop(s7)
    pop(s6)
.end_macro

```

```
.macro print(%str)
    .data
    str: .asciz %str
    empt_str: .asciz ""
    .text
    la a1, empt_str
    li a7 59
    la a0 str
    ecall
.end_macro
```

```
.macro print_console(%str)
    .data
    str: .asciz %str
    .text
    push(a0)
    li a7 4
    la a0 str
    ecall
    pop(a0)
.end_macro
```

```
.eqv READ_ONLY 0 # Open for reading
.eqv WRITE_ONLY 1 # Open for writing
.macro open(%file_name, %opt)
    li a7 1024 # System call to open a file
    la a0 %file_name
    li a1 %opt # Open file for reading/writing/appending
    ecall
.end_macro
```

```
.macro close(%file_descriptor)
    li a7 57 # System call to close a file
    mv a0 %file_descriptor # Load file descriptor to a0
    ecall
.end_macro
```

```
# macros for reading information from file
.macro read_addr_reg(%file_descriptor, %reg, %size)
    li a7 63 # System call to read from file
    mv a0 %file_descriptor
    mv a1 %reg
    li a2 %size
```

```
ecall
```

```
.end_macro
```

```
.macro pop(%x)
```

```
    lw %x (sp)
```

```
    addi sp sp 4
```

```
.end_macro
```

```
.macro push(%x)
```

```
    addi sp sp -4
```

```
    sw %x (sp)
```

```
.end_macro
```

```
.macro exit
```

```
li a7 10
```

```
ecall
```

```
.end_macro
```

```
#макрос для тестирования
```

```
.macro test(%input,%input_inv,%output,%size)
```

```
test_main:
```

```
# Открытие файла для чтения
```

```
open(%input, READ_ONLY)
```

```
#li    t0, -1
```

```
#beq    a0, t0, error_name
```

```
mv     s0, a0  # Дескриптор исходного файла
```

```
# Динамическая память для хранения содержимого
```

```
li     a7, 9
```

```
li     a0, TEXT_SIZE
```

```
ecall
```

```
mv     s3, a0  # Начало памяти
```

```
mv     s5, a0  # Текущий адрес в памяти
```

```
li     s6, 0   # Счетчик прочитанного текста
```

```
read_file:
```

```
read_addr_reg(s0, s5, TEXT_SIZE)
```

```
li     t1, -1
```

```
beq     a0, t1, error_read
```

```
beqz    a0, end_read
```

```
add     s5, s5, a0
```

```
add     s6, s6, a0
```

```
j      read_file
```



```

end_read:
    # Заккрытие исходного файла
    close(s0)

    # Запись данных в обратном порядке
    #print("Input file path to write reversed content: ")
    #get(file_name, NAME_SIZE)

    # Открытие файла для записи
    open(%input_inv, WRITE_ONLY)
    mv    s1, a0

    # Запись в обратном порядке
    mv    t0, s6
    addi  t0, t0, -1
reverse_write_loop:
    bltz  t0, reverse_write_done
    add   t1, s3, t0
    lbu   t2, 0(t1)
    li    a7, 64
    mv    a0, s1
    mv    a1, t1
    li    a2, 1
    ecall
    addi  t0, t0, -1
    j     reverse_write_loop
reverse_write_done:
    # Заккрытие файла
    close(s1)

    # Проверка палиндрома (палиндром_проверка)
    mv    t0, s3      # Начало исходного файла
    add   t1, s3, s6   # Конец обратного файла
    addi  t1, t1, -1

palindrome_check:
    blt   t1, t0, palindrome_yes # Если указатели пересеклись, палиндром

skip_start_spaces:
    lbu    t4, 0(t0)
    li     t6, 0x20      # ASCII код пробела
    beq    t4, t6, skip_start_space
    j      check_end_spaces
skip_start_space:
    addi   t0, t0, 1
    j      skip_start_spaces

```

```

check_end_spaces:
    lbu    t4, 0(t1)
    beq    t4, t6, skip_end_space
    j      compare_chars
skip_end_space:
    addi   t1, t1, -1
    j      check_end_spaces

compare_chars:
    lbu    t2, 0(t0)
    lbu    t3, 0(t1)
    # Приведение к одному регистру (заглавные к строчным)
    li     t6, 'A'
    li     s7, 'Z'
    blt    t2, t6, next_check
    bgt    t2, s7, next_check
    addi   t2, t2, 32
next_check:
    blt    t3, t6, char_comp
    bgt    t3, s7, char_comp
    addi   t3, t3, 32
char_comp:
    bne    t2, t3, palindrome_no
    addi   t0, t0, 1
    addi   t1, t1, -1
    j      palindrome_check

palindrome_yes:
    li     t0, 1
    j      result

palindrome_no:
    li     t0, 0
    j      result

result:
    # Открыть или создать файл для записи результата
    open(%output, WRITE_ONLY)
    mv     s2, a0      # Сохранить дескриптор файла

    beqz   t0, write_result_no
    j      write_result_yes

write_result_yes:
    li     a7, 64

```

```

mv    a0, s2
la    a1, msg_yes # msg_yes/msg_no зависит от результата
li    a2, 4
ecall

# Закрытие файла
close(s2)
la t0, msg_yes
j output_yes

write_result_no:
# Записать результат
li    a7, 64
mv    a0, s2
la    a1, msg_no # msg_yes/msg_no зависит от результата
li    a2, 4
ecall

# Закрытие файла
close(s2)
la t0, msg_no
j output_yes

output_yes:
mv a0, t0
li a7, 4
ecall
print("The program has been successfully completed\n")
.end_macro

```

Описание программы

Цель программы — проверить, является ли ASCII-строка палиндромом. Дополнительно предусмотрена возможность работы в режиме тестирования с заранее подготовленными файлами. Вся программа построена с использованием макросов для упрощения взаимодействия с памятью, файлами и консолью.

Файл data.asm

Этот файл содержит:

1. Глобальные константы и параметры:

- `NAME_SIZE` — размер буфера для имени файла.
- `TEXT_SIZE` — размер буфера для текста.

2. Сегмент `.data`:

- **Сообщения и строки:**
 - `msg_yes` — выводится, если строка является палиндромом.
 - `msg_no` — выводится, если строка не является палиндромом.
 - **Буферы:**
 - `file_name` — для хранения имени файла.
 - `buffer` — для хранения текста.
 - **Тестовые данные:**
 - `test1`, `test1_inv`, `test2`, `test2_inv` и т.д. — имена входных, обратных и выходных файлов для тестов.
-

Файл `idz.asm`

Этот файл реализует логику программы:

1. Точка входа (`main`):

- Выводит начальное сообщение о запуске программы.
- Пользователь выбирает режим:
 - **Тестовый** (запуск тестов с заранее подготовленными файлами).
 - **Ручной** (ввод пути к файлам для проверки палиндрома).

2. Ручной режим:

- Пользователь вводит путь к файлу для чтения.
- Чтение содержимого файла в память.
- Создание обратной строки и запись её в новый файл.
- Проверка, является ли строка палиндромом:
 - Игнорируются пробелы.
 - Регистры приводятся к общему (заглавные к строчным).
- Результат (YES/NO) записывается в указанный файл и может быть выведен на консоль в зависимости от выбора пользователя.

3. Тестовый режим:

- Автоматически проверяет набор файлов (например, `test1.txt`, `test2.txt`, `test3.txt`).
- Для каждого файла создаётся обратная строка, проверяется палиндром и записывается результат.

4. Обработка ошибок:

- Проверяются корректность открытия и чтения файла.
 - В случае ошибок выводятся соответствующие сообщения.
-

Файл `macros.inc`

Этот файл содержит макросы для упрощения кода:

1. Работа с консолью:

- `print` — вывод строки в графическое диалоговое.
- `print_console` — вывод строки с использованием системного вызова.
- `get` — считывание ввода пользователя в буфер.

2. Работа с файлами:

- `open` — открытие файла в режиме чтения/записи.
- `close` — закрытие файла.
- `read_addr_reg` — чтение содержимого файла в указанный адрес памяти.

3. Управление памятью:

- `push / pop` — работа со стеком.

4. Выход из программы:

- `exit` — завершение выполнения.

5. Тестирование:

- `test` — макрос для автоматического тестирования файлов. Читает данные, создаёт обратную строку, проверяет палиндромность и записывает результат.
-

Алгоритм программы

Основные этапы:

1. Ввод данных:

- Чтение строки или пути к файлу с консоли.

2. Чтение содержимого файла:

- Загружает текст из файла в динамически выделенную память.

3. Создание обратной строки:

- Строка записывается в файл в обратном порядке.

4. Проверка палиндрома:

- Сравниваются символы с начала и конца строки.
- Пробелы игнорируются.
- Заглавные буквы приводятся к строчным.

5. Запись результата:

- Результат проверки записывается в файл.
- По запросу пользователя выводится на консоль.

Примеры использования

1. Тестовый режим:

- Программа автоматически обрабатывает три тестовых файла и сравнивает результаты.
- Проверяются строки:
 - `test1.txt` → обратный файл `test1_inv.txt` → результат `result1.txt`.

2. Ручной режим:

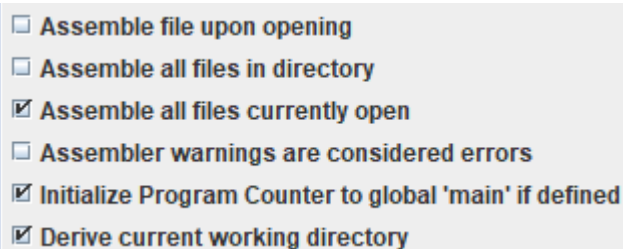
- Пользователь вводит путь к файлу (например, `input.txt`).
- Программа создаёт обратный файл (например, `reversed_input.txt`) и проверяет, является ли строка палиндромом.
- Выводит результат в выходной файл.

Примечания








- Программа учитывает пробелы и регистры при сравнении символов.
- Логика проверки встроена в отдельный блок `palindrome_check`.
- Тестирование упрощает отладку программы на заранее заданных примерах.
- Все критические операции (чтение, запись, проверка) обернуты в макросы для удобства повторного использования.

Примеры работы

Перед начало укажу, какие я ставил настройки в RARS

- 
- ☐ Assemble file upon opening
 - ☐ Assemble all files in directory
 - ☒ Assemble all files currently open
 - ☐ Assembler warnings are considered errors
 - ☒ Initialize Program Counter to global 'main' if defined
 - ☒ Derive current working directory

Вот так выглядит моя рабочая директория для выполнения кода программы

 idz.asm	02.12.2024 20:32	Assembler Source	5 KB
 macros.inc	02.12.2024 21:31	Include File	5 KB
 prompt.txt	02.12.2024 21:40	Текстовый докум...	2 KB
 test1.txt	02.12.2024 11:52	Текстовый докум...	9 KB
 test2.txt	02.12.2024 11:52	Текстовый докум...	1 KB
 test3.txt	02.12.2024 17:41	Текстовый докум...	1 KB
 data.asm	02.12.2024 20:32	Assembler Source	1 KB

`promt.txt` - файл, который я использую для ручного тестирования, там сейчас хранятся такие данные

[illegible]

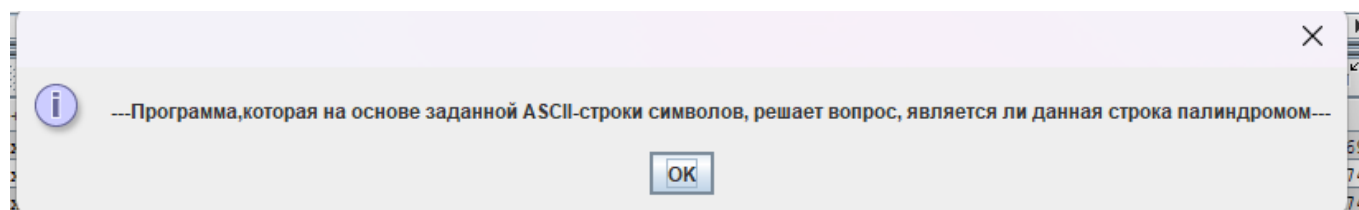
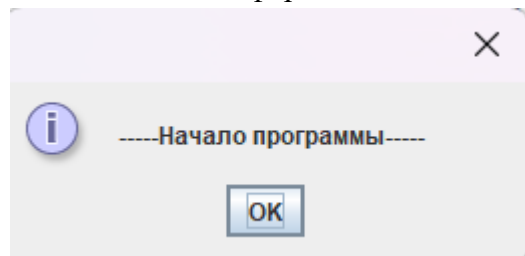
Запускаем программу

```
Go: execution completed successfully.

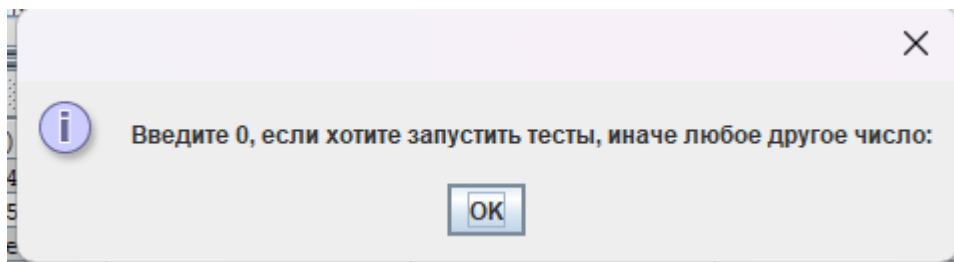
Assemble: assembling D:\HSE\idz3\macros.inc, D:\HSE\idz3\data.asm,
D:\HSE\idz3\idz.asm

Assemble: operation completed successfully.
```

Возникает два информационных диалоговых окна

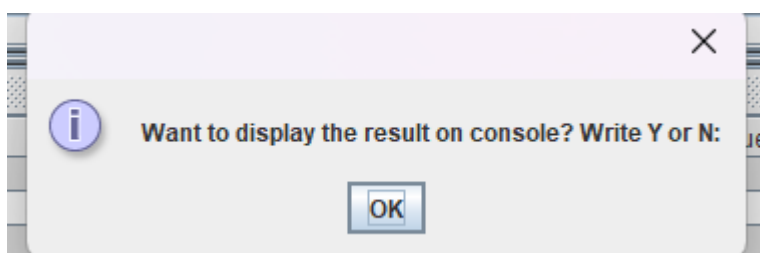
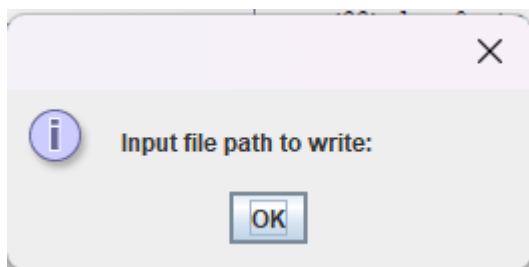
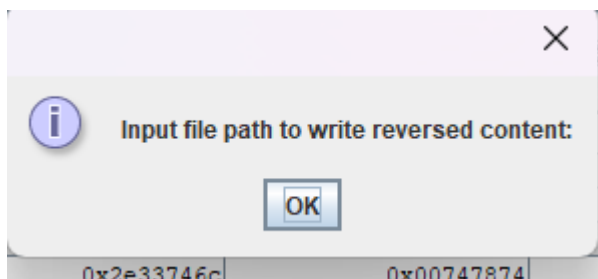
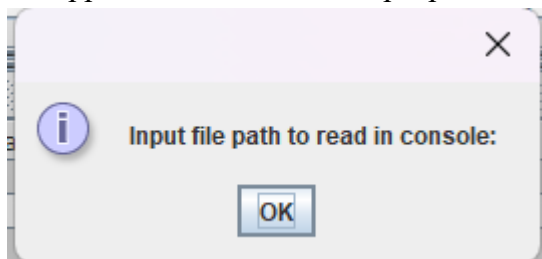


Далее появляется окно с информацией о режиме работы

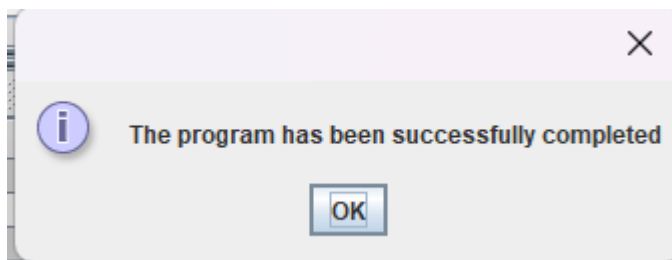


Далее предлагается вывести в консоль выбор пользователя

далее выпадают диалоговые окна для ввода имени файла с исходными данными, потом имя файла, куда сохранятся данные из исходного файла в обратном порядке для дальнейшего сравнения, далее нужно будет ввести имя выходного файла, а также ввести Y\N в зависимости от того, хотите вы выводить данные в консоль или нет, если не ввести Y\N, то выведется сообщения о некорректности данных и программа завершится.

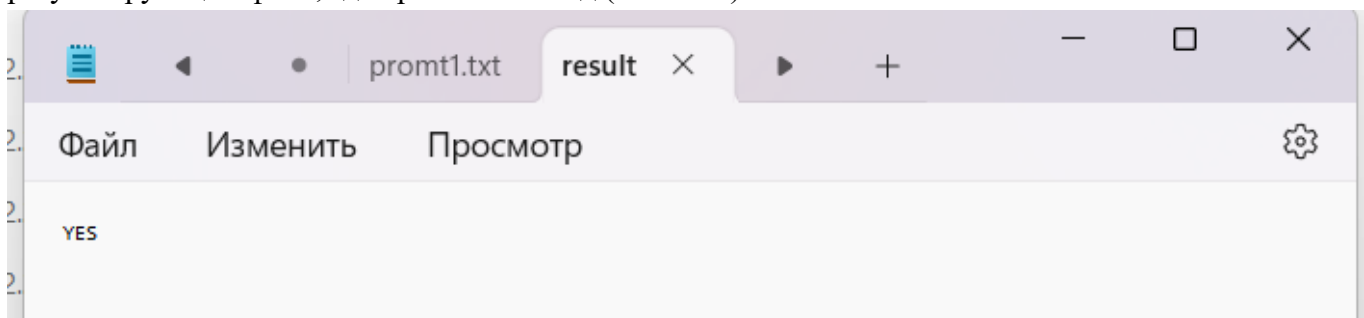


По окончании будет такое графическое окно



```
Write your answer(0-test program,else - usual program): 1
Write your path of input-file: promt.txt
Write your path of reversed input-file: promt1.txt
Write your path of output-file: result.txt
Write your answer: Y
YES
```

Заметим, что в нашей директории создались файлы с исходными данными в обратном порядке и результирующий файл, где хранится вывод (YES/NO)

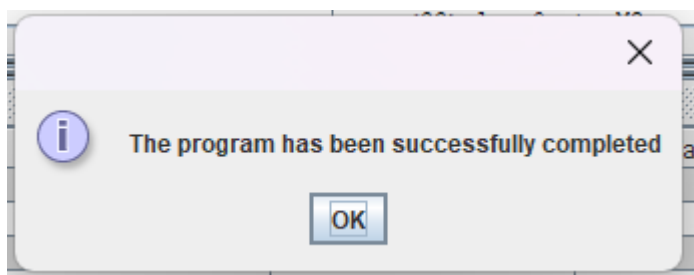
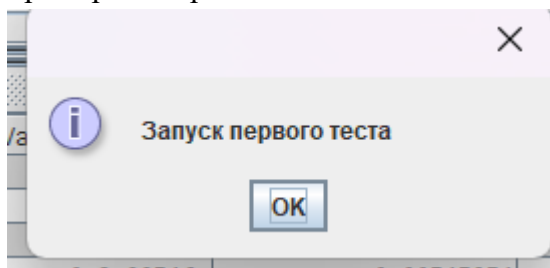


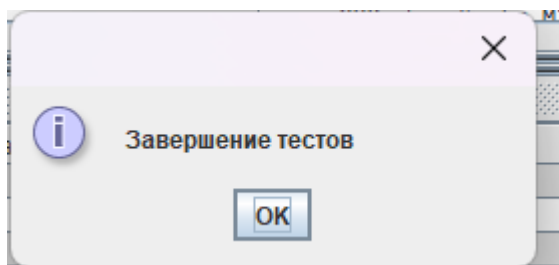
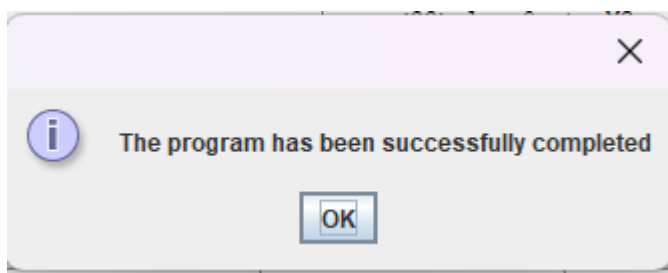
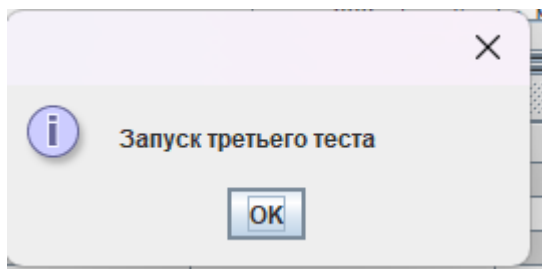
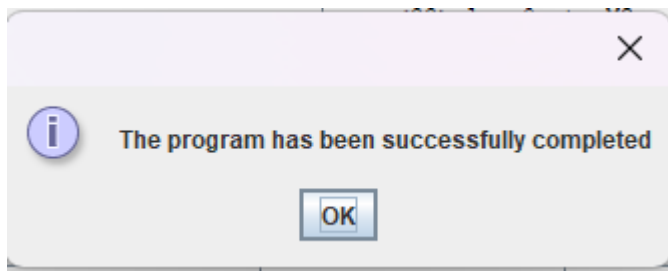
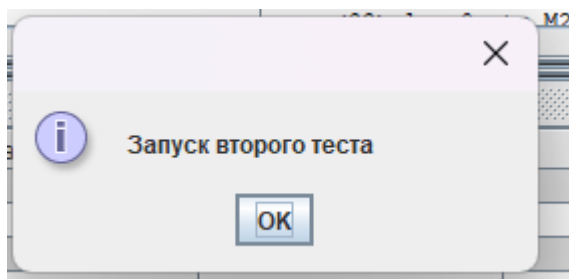
пример для N:

```
Write your answer(0-test program,else - usual program): 1
Write your path of input-file: promt.txt
Write your path of reversed input-file: promt1.txt
Write your path of output-file: result.txt
Write your answer: N

-- program is finished running (0) --
```

пример тестирования





При выполнении тестов, результат автоматически выводится в консоль.

```
Write your answer(0-test program,else - usual program): 0  
Вывод первого теста: YES  
Вывод второго теста: YES  
Вывод третьего теста: YES
```

данные берутся из этих файлов

test1.txt	02.12.2024 22:09	Текстовый докум...	13 КБ
test2.txt	02.12.2024 22:09	Текстовый докум...	1 КБ
test3.txt	02.12.2024 17:41	Текстовый докум...	1 КБ

тестовая программа создаёт такие файлы в нашей рабочей директории

test1_inv.txt	02.12.2024 22:06	Текстовый докум...	9 КБ
result1.txt	02.12.2024 22:06	Текстовый докум...	1 КБ
test2_inv.txt	02.12.2024 22:06	Текстовый докум...	1 КБ
result2.txt	02.12.2024 22:06	Текстовый докум...	1 КБ
test3_inv.txt	02.12.2024 22:06	Текстовый докум...	1 КБ
result3.txt	02.12.2024 22:06	Текстовый докум...	1 КБ

ещё пример тестового прогона с изменённым вторым файлом

```
Write your answer(0-test program,else - usual program): 0
Вывод первого теста: YES
Вывод второго теста: NO
Вывод третьего теста: YES
-- program is finished running (0) --
```

Разработанный код на ассемблере RISC-V реализует автоматизированную обработку файлов, включая чтение, запись и преобразование текста, а также проверку строк на палиндром.

Программа построена на макросах, которые упрощают повторное использование кода, обеспечивая модульность и читаемость. Реализованные функции включают управление файлами, динамическое распределение памяти, обработку строк и проверку символов. Такой подход делает код гибким, масштабируемым и удобным для дальнейшего расширения.

Всем добра!

