

АВС, ИД33, Вариант 19

Матвеева Ольга Романовна, БПИ239

Отчет

Задание:

Разработать программу, вычисляющую число вхождений различных отображаемых символов в заданной ASCII–строке. Вывод результатов организовать в файл (используя соответствующие преобразования чисел в строки).

Отображаемые символы:

В ASCII коды от 32 до 126 соответствуют отображаемым символам (символы, которые могут быть визуально представлены и распечатаны, в отличие от управляющих символов, которые не имеют видимого представления).

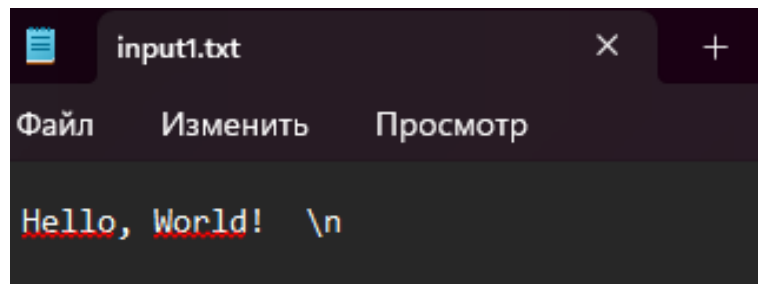
Используем следующий подход для разработки программы:

1. Прочитать строку из входного файла.
2. Создать массив для хранения частоты каждого отображаемого символа (от 32 до 126 в ASCII).
3. Пройти по строке и увеличить счетчик для каждого символа.
4. Записать результаты в выходной файл.

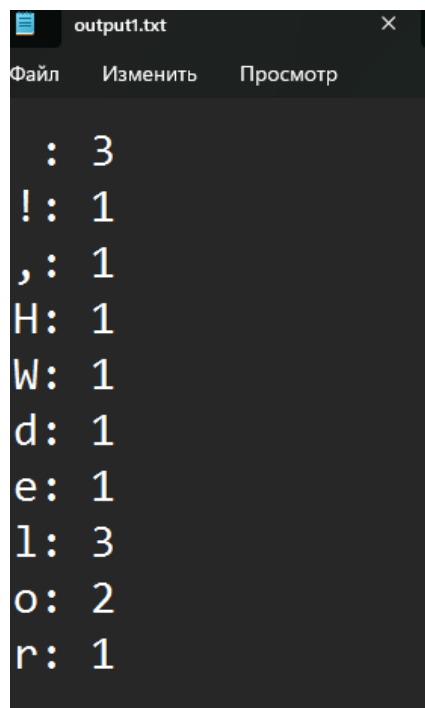
Данные записываются в следующем формате: "символ: число раз, которое он встречается в строке". Символы идут в том порядке, в котором их коды в ASCII.

Пользователь может ввести пути к входному и выходному файлам. Вывод в выходной файл производится автоматически, а вывод в консоль – по желанию пользователя (отвечает «Y» или «N» на соответствующий вопрос компьютерной программы).

Результаты тестовых прогонов кода:



A screenshot of a text editor window titled "input1.txt". The window has a menu bar with "Файл", "Изменить", and "Просмотр". The text content is "Hello, World! \n", where "Hello, World!" is underlined in red.



A screenshot of a text editor window titled "output1.txt". The window has a menu bar with "Файл", "Изменить", and "Просмотр". The text content shows character frequency statistics for the input text "Hello, World! \n".

Character	Frequency
:	3
!:	1
,:	1
H:	1
W:	1
d:	1
e:	1
l:	3
o:	2
r:	1

input2.txt

Файл Изменить Просмотр

```
\tJohnny Depp \n
```

output2.txt

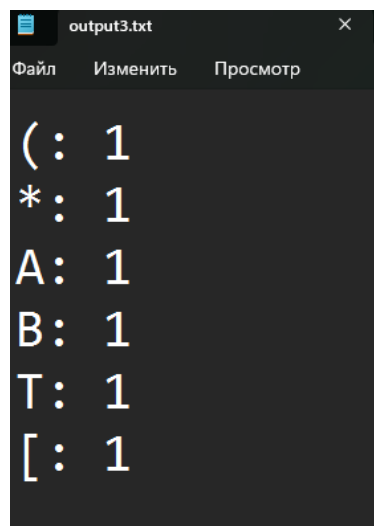
Файл Изменить Просмотр

```
: 2
D: 1
J: 1
e: 1
h: 1
n: 2
o: 1
p: 2
y: 1
```

input3.txt

Файл Изменить Просмотр

```
A\0B\0T[(*\t
```



Скриншоты библиотеки макросов:

Print an integer from a register

```
.macro print_int (%x)
    li a7, 1
    mv a0, %x
    ecall
.end_macro
```

Print a character from a register

```
.macro print_char(%x)
    li a7, 11
    mv a0, %x
    ecall
.end_macro
```

Push onto the stack

```
.macro push(%x)
    addi sp, sp, -4
    sw %x, (sp)
.end_macro
```

Move from the top of the stack to register x

```
.macro pop(%x)
    lw %x, (sp)
    addi sp, sp, 4
.end_macro
```

```
# Print a string
.macro print_string (%x)
    .data
    string: .asciz %x
    .align 2
    .text
    push (a0)
    li a7, 4
    la a0, string
    ecall
    pop (a0)
.end_macro

# New line
.macro new_line
    li a7, 11
    li a0, '\n'
    ecall
.end_macro
```

```
# Input file path
.macro get_path(%str, %len)
    la    a0 %str
    li    a1 %len
    li    a7 8
    ecall
    push(s0)
    push(s1)
    push(s2)
    li    s0 '\n'
    la    s1 %str
next:
    lb    s2 (s1)
    beq   s0 s2 replace
    addi  s1 s1 1
    b     next
replace:
    sb    zero (s1)
    pop(s2)
    pop(s1)
    pop(s0)
.end_macro
```

Open a file

```
.macro open(%file, %o)
    li    a7, 1024
    la    a0, %file
    li    a1, %o
    ecall
.end_macro
```

Read a file

```
.macro read(%desc, %str, %len)
    li    a7, 63
    mv    a0, %desc
    la    a1, %str
    li    a2, %len
    ecall
.end_macro
```

Close a file

```
.macro close(%desc)
    li    a7, 57
    mv    a0, %desc
    ecall
.end_macro
```



```
# Write information to a file
.macro write(%desc, %str, %len)
    li    a7, 64
    mv     a0, %desc
    la     a1, %str
    li     a2, %len
    ecall
.end_macro

# Allocate memory
.macro allocate(%len)
    li a7, 9
    li a0, %len
    ecall
.end_macro

# Exit program
.macro exit
    li a7, 10
    ecall
.end_macro
```

Скриншоты основной программы:

```
.include "my_macrolib.s"

.eqv INPUT_BUFFER_SIZE 512
.eqv OUTPUT_BUFFER_SIZE 512
.eqv ASCII_START 32
.eqv ASCII_END 126
.eqv ASCII_RANGE 95  #(ASCII_END - ASCII_START + 1)

.data
input_buffer: .space INPUT_BUFFER_SIZE  # Buffer for input string
output_buffer: .space OUTPUT_BUFFER_SIZE  # Buffer for output result
frequency: .space 380  # 95 * 4, Frequency array for characters from 32 to 126
output_msg: .asciz "%c: %d\n"

.text
main:
     # Read string from input file
    print_string("Enter the path to the input file: ")
    get_path(input_buffer, INPUT_BUFFER_SIZE)  # Read file name
    open(input_buffer, 0)  # Open file for reading
    li t5, -1
    beq a0, t5, file_open_error  # Check for successful file opening

     # Initialize frequency array
    la s0, frequency  # Address of frequency array
    li s1, 0  # Index for frequency array
```

```

init_frequency:
    li t5, ASCII_RANGE
    bge s1, t5, read_line      # If initialization is complete, go to reading line
    sw zero, 0(s0)             # Initialize frequency to zero
    addi s0, s0, 4             # Increment address by 4 bytes
    addi s1, s1, 1             # Increment index
    j init_frequency           # Repeat loop

read_line:
    allocate(INPUT_BUFFER_SIZE) # Allocate memory for buffer
    mv a1, a0                  # Save file descriptor
    read(a1, input_buffer, INPUT_BUFFER_SIZE) # Read string
    li t5, -1
    beq a0, t5, file_read_error # Check for successful read
    la s2, input_buffer         # Index for input string

next_char:
    lb t0, (s2)                # Load next character
    beq t0, zero, write_results # If end of string, go to writing results
    li t5, ASCII_START
    blt t0, t5, next_char      # Ignore non-printable characters
    li t5, ASCII_END
    bgt t0, t5, next_char      # Ignore non-printable characters

    la t1, frequency
    # Increase frequency of the corresponding character
    addi t1, t0, -ASCII_START  # Shift index for frequency array
    slli t1, t1, 2             # Multiply by 4

```

```

lw t2, (t1)           # Get frequency of current character
addi t2, t2, 1         # Increment counter
sw t2, (t1)           # Save back to array
addi s2, s2, 1         # Move to next character
j next_char           # Return to character reading

```

write_results:

```

print_string("Enter the path to the output file: ")
get_path(input_buffer, INPUT_BUFFER_SIZE) # Read file name for writing
open(input_buffer, 1)                     # Open file for writing
li t5, -1
beq a0, t5, file_open_error               # Check for successful file opening

```

Prompt to output results to console

```

print_string("Do you want to print results to console? (Y/N): ")
get_path(input_buffer, INPUT_BUFFER_SIZE) # Read user response
li t6, 'Y'
la t5, input_buffer
lb t4, (t5)                               # Read first character of response

```

```

li s3, ASCII_START # Start of ASCII range

```

write_loop:

```

li t5, ASCII_END # End of ASCII range
bgt s3, t5, close_file # If reached end of range, close file
addi t1, s3, -ASCII_START # Index for frequency array (character - 32)
slli t1, t1, 2 # Multiply by 4 (size of one element)
la t0, frequency # Load address of frequency array into t0

```



```

lw t2, 0(t0)
bne t2, zero, write_line
addi s3, s3, 1
j write_loop

```

```

# Get frequency of current character
# If frequency is non-zero, write
# Move to next character
# Repeat loop

```

write_line:

```

# Write string to output file
write(a0, output_buffer, 8)
la t5, output_buffer
addi t5, t5, 8
addi s3, s3, 1
beq t4, t6, print_to_console
j write_loop

```

```

# Write 8 bytes (character + frequency)
# Load address of output_buffer into t5
# Increment address by 8 bytes
# Move to next character
# If response is 'Y', go to print to console
# Go back to loop if response is 'N'

```

print_to_console:

```

# Print character and its frequency to console
print_char(s3)
print_string(": ")
print_int(t2)
new_line
j write_loop

```

```

# Print character to console

# Print its frequency to console
# Print a new line
# Return to start of loop

```

close_file:

```

close(a0)
exit

```

```

# Close output file

```

```
# Error handling
file_open_error:
    print_string("Error opening file\n")
    exit

file_read_error:
    print_string("Error reading file\n")
    exit
```