



Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э.
Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»

КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

Домашняя работа

«ОБРАБОТКА ДВУХМЕРНЫХ МАССИВОВ ЦЕЛЫХ ЧИСЕЛ»

ДИСЦИПЛИНА: «Системное программирование»

Выполнил: студент гр. ИУК4-32Б

_____ (Моряков В.Ю.)
(Подпись)

Проверил:

_____ (Амеличева К.А.)
(Подпись)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга, 2023

Цель работы: выполнения домашней работы является практическое овладение навыками разработки программного кода на языке Ассемблер. Обработка массивов.

Задачи:

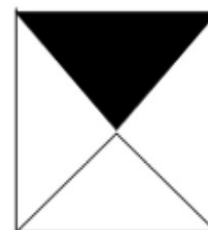
Основными задачами выполнения домашней работы являются изучение основных приемов обработки массивов: ввод-вывод, доступ к элементам массива, транспонирование, выполнение типовых операции.

Вариант 12

Вариант 12

Дана матрица.

- а) В каждой строке поместите нулевые элементы в конец строки.
- б) Проверить, равны ли поэлементно i -ая строка и i -ый столбец квадратной матрицы.
- в) Найдите максимальный элемент среди элементов матрицы, выделенных чёрным цветом (матрица квадратная).



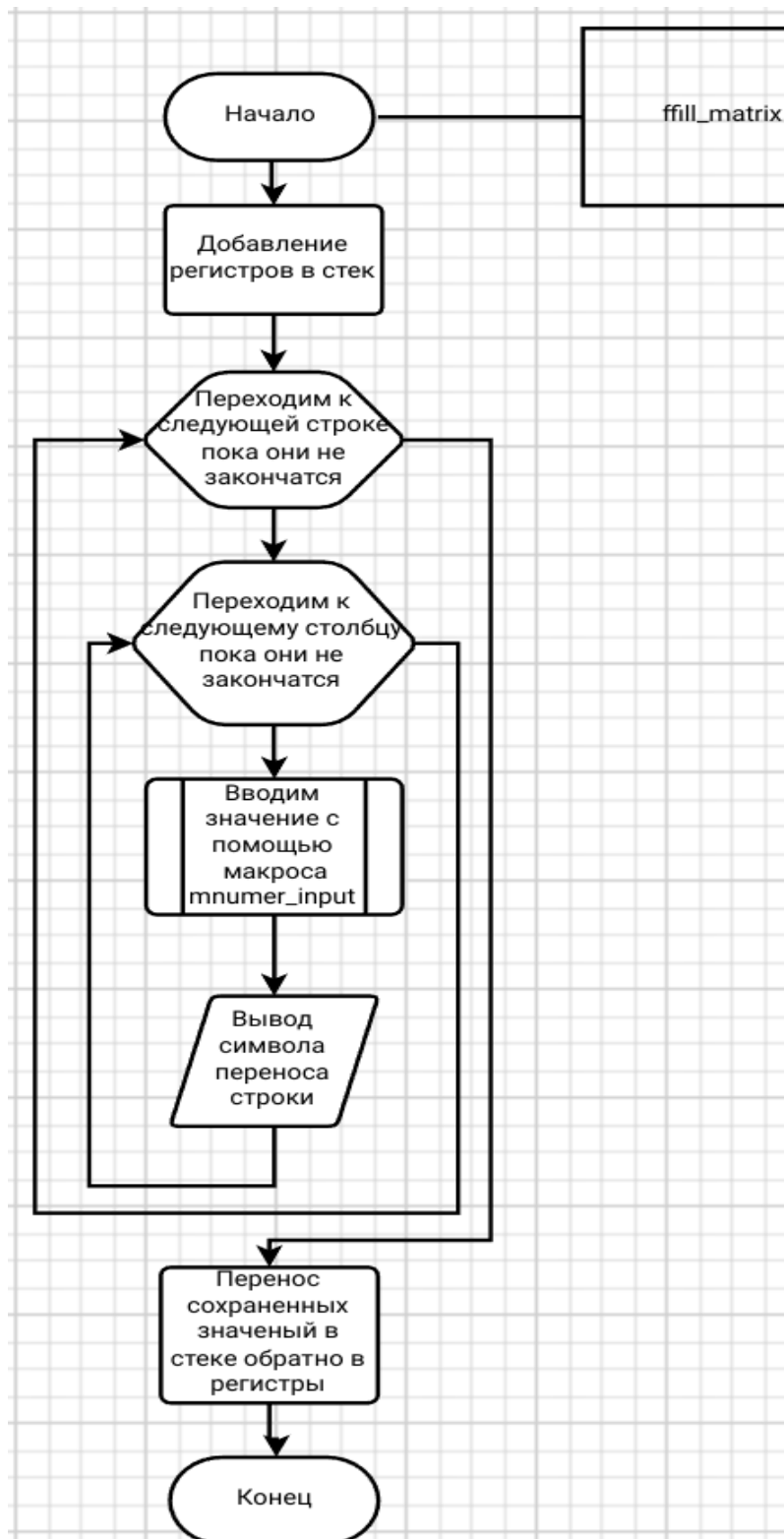
Постановка задачи

Работа предусматривает применение в основных приемов обработки массивов: создание массивов случайным образом с использованием датчика случайных чисел, ввод с клавиатуры, задание массивов по определенному закону, нахождение максимального и минимального элементов массива, перестановка строк и столбцов матрицы, сортировка строк и столбцов, с использованием алгоритма сортировки одномерного массива, перестановка блоков внутри матрицы, умножение матриц.

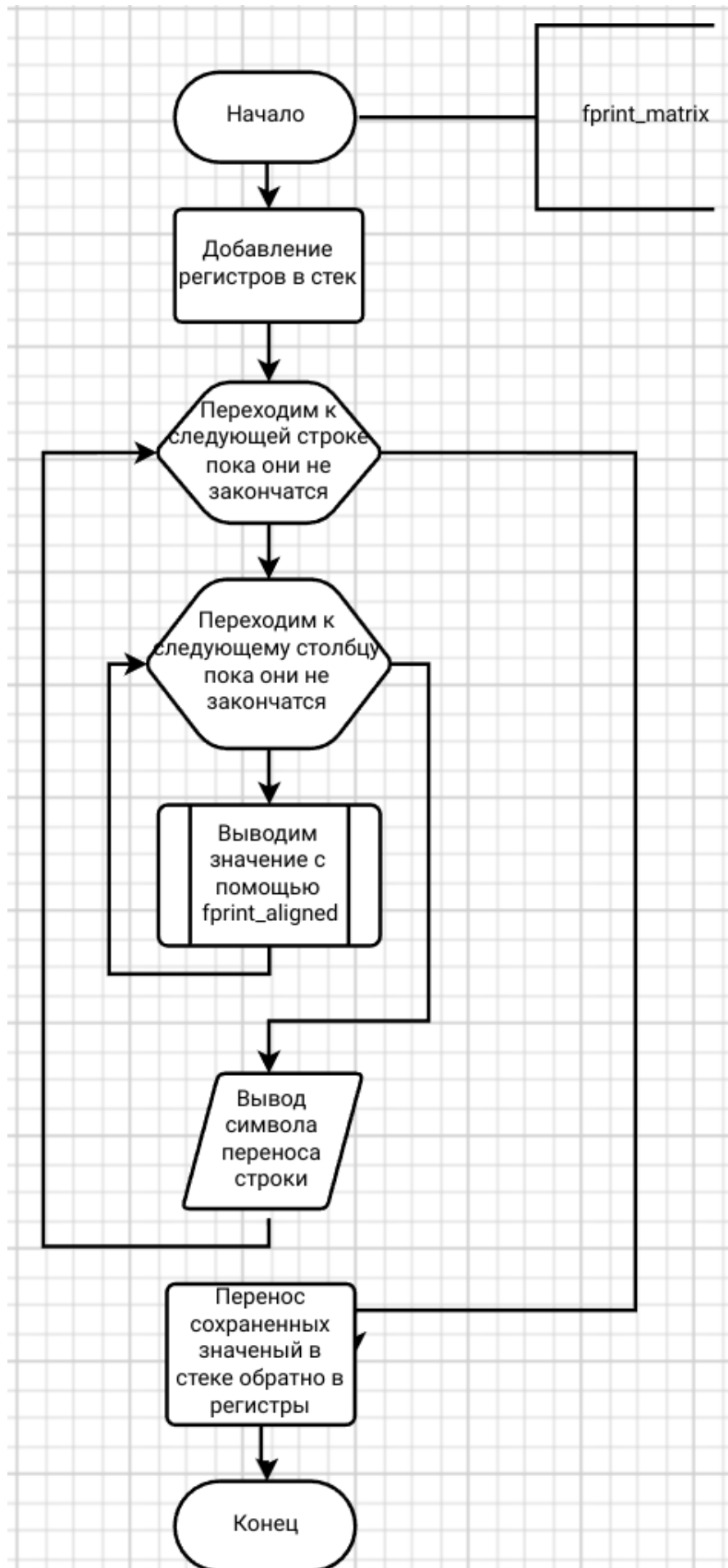
1. Ввести с клавиатуры и вывести на экран матрицу $m \times n$ (матрица не обязательно должна быть квадратная и может содержать нулевые и отрицательные элементы, если это предусмотрено условиями задания варианта);
2. Реализовать простейший интерфейс взаимодействия с пользователем для выполнения задания варианта до выбора команды «Выход»;
 - Транспонировать матрицу, результат вынести на экран;
 - Обработка элементов матрицы (задание а, б, в условия варианта), результат выполнения отобразить на экране;
 - Реализовать завершение выполнения программы.

Блок-схемы:

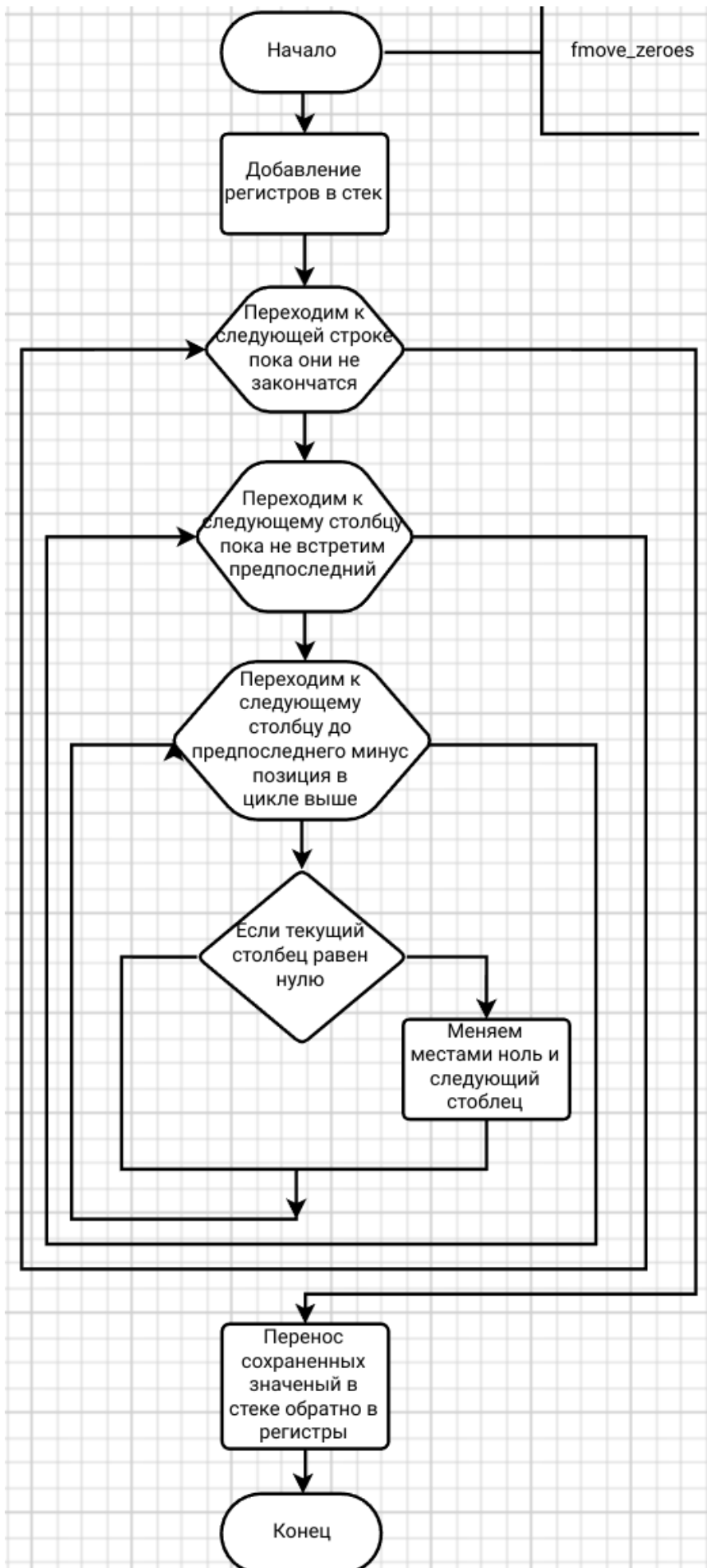
Ввод матрицы:



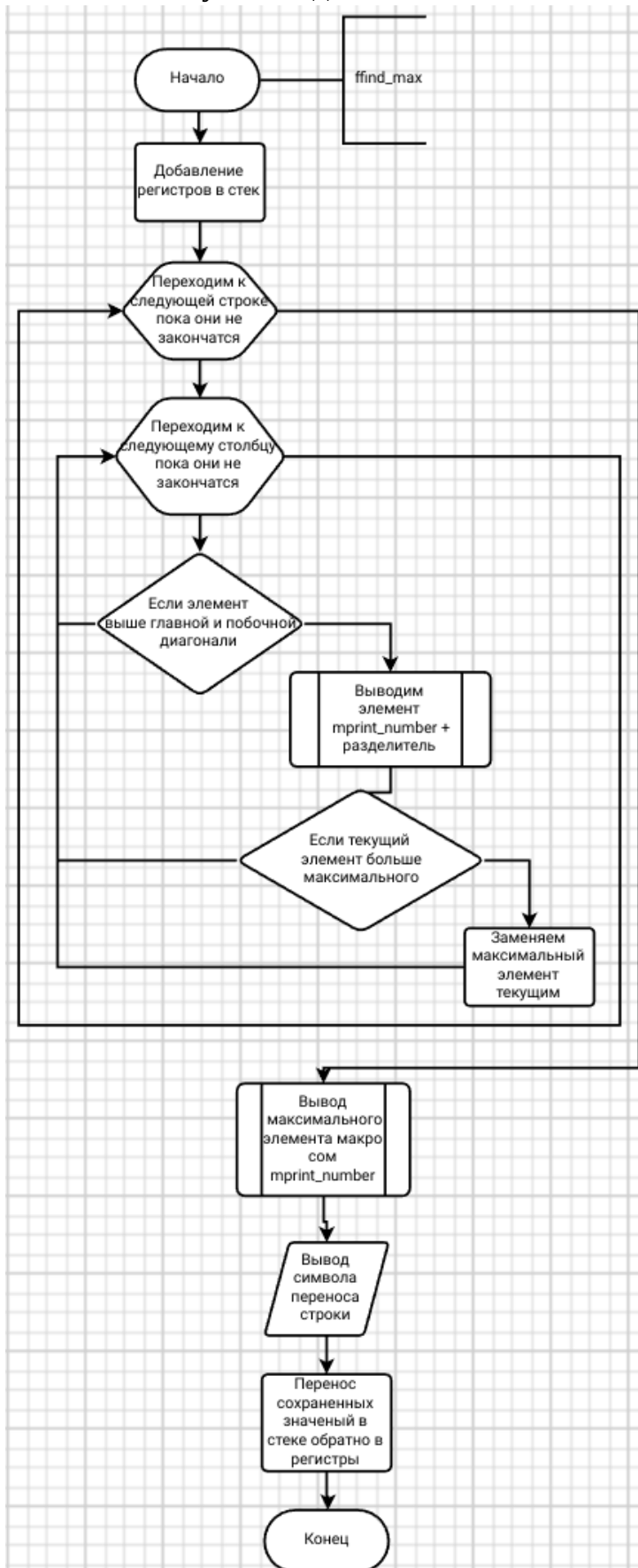
Вывод матрицы:



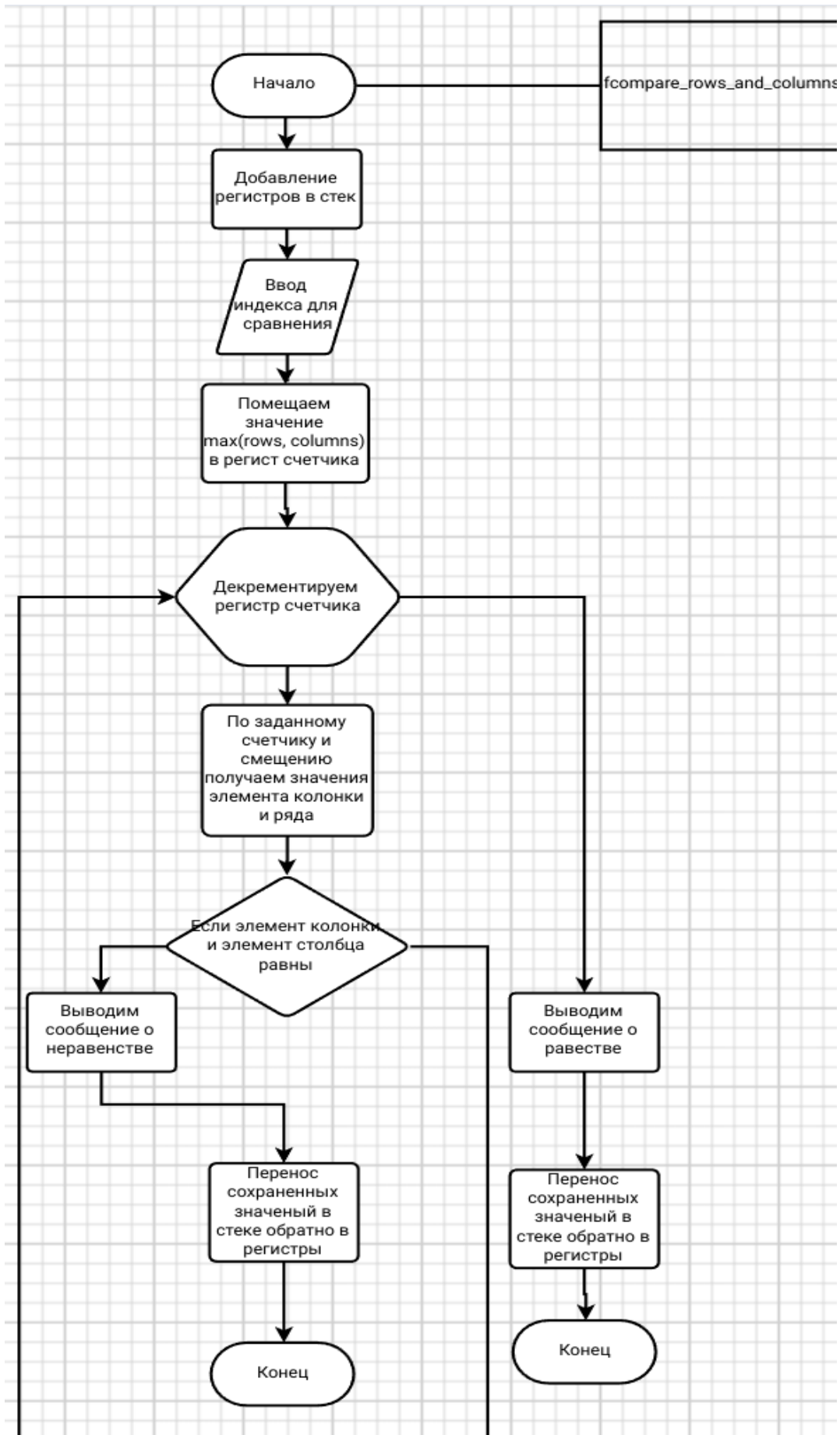
Перестановка нулей в конец:



Поиск максимума с заданной области:



Сравнение ряда и столбца:



Результаты работы программы:

Вывод матрицы после заполнения:

```
DOSBox 0.74-3, Cpu speed: 30...
2
0
0
20
01
20

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>1
-12 232 12 34
-122 0 999 -999
-2 -1 2 0
0 20 1 20

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>_
```

Перемещение нулей в конец:

```
DOSBox 0.74-3, Cpu speed: 30...

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>2

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>1
-12 232 12 34
-122 999 -999 0
-2 -1 2 0
20 1 20 0

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>
```


Сравнение ряда и колонки по заданному индексу:

```
DOSBox 0.74-3, Cpu speed: 30...
2. move zeros
3. compare row and column
4. find max
5. exit
>>1
-12 232 12 34
-122 999 -999 0
-2 -1 2 0
20 1 20 0

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>3
Input index>>1
Not equal in second task

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>
```

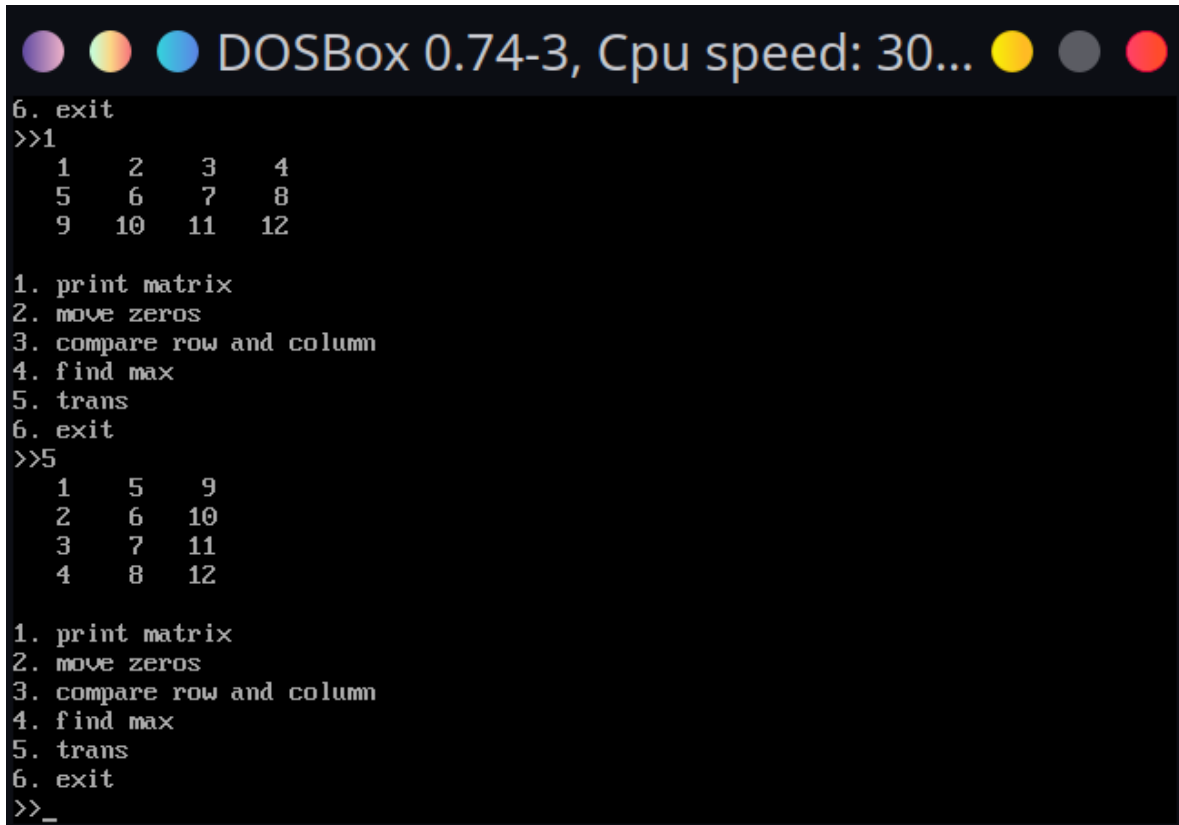
Нахождение максимального элемента в выделенной области:

```
DOSBox 0.74-3, Cpu speed: 30...
3. compare row and column
4. find max
5. exit
>>3
Input index>>1
Not equal in second task

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>4
232 12

Maximal value: 232
1. print matrix
2. move zeros
3. compare row and column
4. find max
5. exit
>>_
```

И чуть не забыл про транспонирование матрицы:



```
DOSBox 0.74-3, Cpu speed: 30...
6. exit
>>1
  1   2   3   4
  5   6   7   8
  9  10  11  12

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. trans
6. exit
>>5
  1   5   9
  2   6  10
  3   7  11
  4   8  12

1. print matrix
2. move zeros
3. compare row and column
4. find max
5. trans
6. exit
>>_
```

Вывод:

В ходе выполнения домашней работы были изучены операции над матрицами на языке программирования tasm(turbo assembler).

Листинг программы:

```
.model small
.386
.stack 100h

.data
max_rows equ 100
max_columns equ 100

; rows dw ?
; columns dw ?

; matrix dw max_rows * max_columns dup(?)

rows dw 4
columns dw 3

matrix dw 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
; matrix dw max_rows * max_columns dup(2)
res_matrix dw max_rows * max_columns dup(0)

message db 'Enter a number of any length: $'

msg_menu db '1. print matrix', 0, 10
db '2. move zeros', 0, 10
db '3. compare row and column', 0, 10
db '4. find max', 0, 10
db '5. trans', 0, 10
db '6. exit', 0, 10
db '>>', '$'

msg_not_equal_second db 'Not equal in second task', 13, 10, '$'
msg_equal_second db 'Equal in second task', 13, 10, '$'
msg_maximal db 'Maximal value:', '$'
msg_input_index db 'Input index>>', '$'
msg_input_rows db 'Input rows>>', '$'
msg_input_columns db 'Input columns>>', '$'

matrix_maximal dw 0

index dw 0
offset equ 2
tdgt equ add dx, 30h

buffer_size equ 255

input_buffer db 8 dup('$')
result_buffer dw 0
number_buffer dw 0
result dw 0

bfr_transpose dw ?
bfr_fill_matrix dw ?
bfr_output_aligned dw 0

adr_matrix dw ?
.code

mnumber_input MACRO var:req
local exit, convert, greater, break, isDigit, skip
push ax
push bx
push dx
push cx
push si

mov ah, 0ah
mov dx, offset input_buffer
int 21h

xor ax, ax
mov si, 2

cmp input_buffer[2], '-'
jne convert

inc si

convert:
mov al, input_buffer[si]

cmp ax, '0'
jge greater
jmp break

greater:
cmp ax, '9'
jle isDigit
jmp break

isDigit:
and ax, 0fh
push ax

mov ax, result_buffer
```

```

        mov bl, 10
        mul bl
        mov result_buffer, ax

        pop ax
        add result_buffer, ax
        inc si
        jmp convert
break:
        cmp ax, 0dh
        ; jne far ptr input_errorL

skip:
        mov ax, result_buffer
        mov var, ax

        cmp input_buffer[2], '-'
        jne exit
        neg var

exit:
        mov result_buffer, 0

        pop si
        pop cx
        pop dx
        pop bx
        pop ax
        ENDM

input_errorL:
        call fexit

        mnumber_output macro num:req
        local exit, skip, pushing, outputLoop
        push ax
        push bx
        push dx
        push cx

        mov ax, num
        cmp ax, 0
        jnl skip
        mov dx, '-'
        call fprint_char_by_addr
        neg ax

skip:
        mov bx, 10
        xor cx, cx

pushing:
        xor dx, dx
        div bx
        push dx
        inc cx
        test ax, ax
        jnz pushing

        mov ah, 02h

outputLoop:
        pop dx
        add dx, '0'
        int 21h
        loop outputLoop

exit:
        pop cx
        pop dx
        pop bx
        pop ax
        endm

mput_msg macro msg
        mov dx, offset msg
        endm

fexit proc
        mov ax, 4c00h
        int 21h
        fexit endp

fprint_ln proc
        push ax
        push dx

        mov dl, 10
        mov ah, 02h
        int 21h

        mov dl, 13
        mov ah, 02h
        int 21h

        pop dx

```

```

        pop ax
        ret
fprint_ln endp

; as an argument mov dx, [matrix + 2]
fprint_char_by_addr proc
    push ax
    push dx

    mov ah, 02h
    int 21h

    pop dx
    pop ax
    ret
fprint_char_by_addr endp

fprint_align_item proc
    push ax
    push bx
    push cx

    push dx
    mov ax, dx
    push ax
    mov bx, 10d
    cwd
    idiv bx
    pop ax
    cmp ax, dx
    pop dx
    je if_single_digit_number

    push dx
    mov ax, dx
    push ax
    mov bx, 100d
    cwd
    idiv bx
    pop ax
    cmp ax, dx
    pop dx
    je if_two_digit_number

    push dx
    mov ax, dx
    push ax
    mov bx, 1000d
    cwd
    idiv bx
    pop ax
    cmp ax, dx
    pop dx
    je if_three_digit_number

print_align_item_end:
    mov number_output bfr_output_aligned
    pop cx
    pop bx
    pop ax
    ret

if_single_digit_number:
    mov bfr_output_aligned, dx

    mov dx, ''
    call fprint_char_by_addr

    mov dx, bfr_output_aligned
    cmp dx, 0
    jl if_two_digit_number

    mov dx, ''
    call fprint_char_by_addr

    mov dx, ''
    call fprint_char_by_addr

    jmp print_align_item_end

if_two_digit_number:
    mov bfr_output_aligned, dx

    mov dx, ''
    call fprint_char_by_addr

    mov dx, bfr_output_aligned
    cmp dx, 0
    jl if_three_digit_number

    mov dx, ''
    call fprint_char_by_addr

    jmp print_align_item_end

if_three_digit_number:
    mov bfr_output_aligned, dx

```

```

        mov dx, bfr_output_aligned
        cmp dx, 0
        jl print_align_item_end

        mov dx, ''
        call fprint_char_by_addr
        jmp print_align_item_end
fprint_align_item endp

fprint_matrix proc
push ax
push cx
push bx
push dx
push si
push di

mov cx, 0; rows counter
mov bx, rows

print_matrix_iloop:
    cmp cx, bx
    je print_matrix_iloop_end

    xor ax, ax
    mov ax, cx

    push ax
    push bx

    mov ax, offst
    mov bx, columns
    mul bx
    mov dx, ax

    pop bx
    pop ax

    mul dx
    mov si, ax

    mov di, ax

    push ax
    push bx
    mov ax, offst
    mov bx, columns
    mul bx
    add di, ax
    pop bx
    pop ax

    xor ax, ax

print_matrix_jloop:
    cmp si, di
    je print_matrix_jloop_end

    mov dx, [matrix + si]
    call fprint_align_item

    mov dx, ''
    call fprint_char_by_addr

    add si, offst
    jmp print_matrix_jloop

print_matrix_jloop_end:
    inc cx

    mov dx, 10
    call fprint_char_by_addr

    jmp print_matrix_iloop

print_matrix_iloop_end:
    pop ax
    pop cx
    pop bx
    pop dx
    pop si
    pop di
    ret

fprint_matrix endp

fmove_zeros proc
push ax
push cx
push bx
push dx
push si
push di

mov ax, 0

```

```

move_zeros_iloop:
    cmp ax, rows
    je move_zeros_iloop_end

    mov cx, 0
move_zeros_jloop:
    push ax
    mov ax, rows
    dec ax
    cmp cx, ax
    pop ax
    je move_zeros_jloop_end

    push ax
    ; mov si, columns * offst * ax
    push ax
    push bx
    mov ax, offst
    mov bx, columns
    mul bx
    mov si, ax
    pop bx
    pop ax
    mul si
    mov si, ax

    ; mov di, columns * offst * ax + columns * offst
    mov di, ax
    push ax
    push bx
    mov ax, offst
    mov bx, columns
    mul bx
    add di, ax
    pop bx
    pop ax
    sub di, offst

    pop ax
move_zeros_kloop:
    cmp si, di
    je move_zeros_kloop_end

    mov dx, [matrix + si]
    cmp dx, 0
    je swap

    add si, offst
    jmp move_zeros_kloop

swap:
    ; tdtg
    ; call fprintf_char_by_addr
    push ax
    mov ax, [matrix + si + offst]
    mov [matrix + si], ax
    mov [matrix + si + offst], 0
    pop ax
    add si, offst
    jmp move_zeros_kloop

move_zeros_kloop_end:
    inc cx
    jmp move_zeros_jloop

move_zeros_jloop_end:
    inc ax
    jmp move_zeros_iloop

move_zeros_iloop_end:
    pop ax
    pop cx
    pop bx
    pop dx
    pop si
    pop di
    ret

fmove_zeros endp

; dx argument required
fprintf_message proc
    mov ah, 09h
    int 21h
    ret
fprintf_message endp

fcompare_rows_columns proc
    push ax
    push bx
    push cx
    push dx

    ; danger bug is possible

```

```

xor ax, ax
input_msg msg_input_index
call fprint_message
call finput
xor ah, ah
sub al, 30h
cmp ax, rows
jge fcompare_rows_columns_exit
mov index, ax
call fprint_ln

mov ax, index
push ax
mov ax, columns
mov bx, offst
mul bx
mov bx, ax
pop ax
mul bx
mov si, ax

push ax
push bx
mov ax, offst
mov bx, columns
mul bx
sub ax, offst
add si, ax
pop bx
pop ax
mov di, columns
dec di
jmp compare_rows_columns_loop

fcompare_rows_columns_exit:
call fprint_message
pop dx
pop cx
pop bx
pop ax
ret

compare_rows_columns_loop:
mov ax, index
push ax
mov ax, offst
mov bx, columns
mul bx
mov bx, ax
pop ax
mul bx
sub ax, offst
cmp si, ax
je compare_rows_columns_loop_end

cmp di, -1
je compare_rows_columns_loop_end

; column item
push ax
push bx
push cx

; mov dx, [matrix + 2 * columns * offst + index * offst]
mov dx, [matrix]
mov ax, di

push ax
mov ax, offst
mov bx, columns
mul bx
mov bx, ax
pop ax

mul bx
push ax
mov ax, index
mov bx, offst
mul bx
mov bx, ax
pop ax
add ax, bx

mov bx, ax
mov dx, [bx]

pop cx
pop bx
pop ax

; push dx
; tdt
; call fprint_char_by_addr
; pop dx

; row item
mov bx, [matrix + si]
; push dx

```



```

; mov dx, bx
; tdt
; call fprint_char_by_addr
; pop dx

cmp bx, dx
jne compare_rows_columns_not_equal

sub si, offst
dec di
jmp compare_rows_columns_loop

compare_rows_columns_loop_end:
input_msg msg_equal_second
jmp fcompare_rows_columns_exit

compare_rows_columns_not_equal:
input_msg msg_not_equal_second
jmp fcompare_rows_columns_exit

fcompare_rows_columns endp

ffind_maximal proc
push ax
push cx
push bx
push dx
push si
push di

mov cx, 0; rows counter
mov bx, rows

find_maximal_iloop:
cmp cx, bx
je find_maximal_iloop_end

xor ax, ax

mov ax, cx

push ax
push bx
mov ax, rows
mov bx, offst
mul bx
mov dx, ax
pop bx
pop ax

mul dx
mov si, ax

mov di, ax

push ax
push bx
mov ax, offst
mov bx, columns
mul bx
add di, ax
pop bx
pop ax

xor ax, ax

find_maximal_jloop:
cmp si, di
je find_maximal_jloop_end

; blows makes ax: 0123
push ax
push bx
mov ax, si
mov bx, offst
cwd
div bx
mov bx, columns
div bx
; mov dx, ax
pop bx
pop ax

cmp dx, cx
jg find_maximal_grt_mn_dgnl

find_maximal_jloop_ret:
add si, offst
jmp find_maximal_jloop

find_maximal_jloop_end:
inc cx

call fprint_ln

jmp find_maximal_iloop

```

```

find_maximal_loop_end:
    jmp ffind_maximal_exit

find_maximal_set_new:
    mov [matrix_maximal], dx
    jmp find_maximal_jloop_ret

find_maximal_grt_sd_dgnl:
    mov dx, [matrix + si]
    mnumber_output dx
    mov dx,
    call fprint_char_by_addr

    mov dx, [matrix + si]
    cmp dx, matrix_maximal
    jg find_maximal_set_new
    jmp find_maximal_jloop_ret

find_maximal_grt_mn_dgnl:
    push ax
    push dx
    ; mov dx, [matrix + si]
    ; call fprint_char_by_addr
    ; call fprint_ln
    mov ax, columns
    sub ax, cx
    dec ax
    cmp dx, ax
    pop ax
    pop dx
    jl find_maximal_grt_sd_dgnl
    jmp find_maximal_jloop_ret

ffind_maximal_exit:
    call fprint_ln
    mput_msg msg_maximal
    call fprint_message
    mov dx, matrix_maximal
    mnumber_output dx
    pop ax
    pop cx
    pop bx
    pop dx
    pop si
    pop di
    ret

ffind_maximal endp

finput proc
    mov ah, 1
    int 21h
    ret
finput endp

fshow_menu proc
    mput_msg msg_menu
    call fprint_message

    call finput

    cmp al, '1'
    je option1
    cmp al, '2'
    je option2
    cmp al, '3'
    je option3
    cmp al, '4'
    je option4
    cmp al, '5'
    je option5
    cmp al, '6'
    je fshow_menu_exit

    jmp fshow_menu

option1:
    call fprint_ln
    call fprint_matrix
    call fprint_ln
    jmp fshow_menu

option2:
    call fprint_ln
    call fmove_zeros
    call fprint_ln
    jmp fshow_menu

option3:
    call fprint_ln
    call fcompare_rows_columns
    call fprint_ln
    jmp fshow_menu

option4:
    call fprint_ln
    call ffind_maximal
    call fprint_ln

```

```

        jmp fshow_menu
option5:
        call fprint_ln
        call ftranspose_matrix
        call fprint_ln
        jmp fshow_menu
fshow_menu_exit:
        ret
fshow_menu endp
ffill_matrix proc
push ax
push cx
push bx
push dx
push si
push di

mov cx, 0; rows counter
mov bx, rows

        jmp skip
fill_matrix_iloop_end:
        pop ax
        pop cx
        pop bx
        pop dx
        pop si
        pop di
        ret

skip:
fill_matrix_iloop:
        cmp cx, bx
        je fill_matrix_iloop_end

        xor ax, ax

        mov ax, cx

        push ax
        push bx

        mov ax, offst
        mov bx, columns
        mul bx
        mov dx, ax

        pop bx
        pop ax

        mul dx
        mov si, ax

        mov di, ax

        push ax
        push bx
        mov ax, offst
        mov bx, columns
        mul bx
        add di, ax
        pop bx
        pop ax

        xor ax, ax
fill_matrix_jloop:
        cmp si, di
        je fill_matrix_jloop_end

        mnumber_input bfr_fill_matrix
        push ax
        mov ax, bfr_fill_matrix
        mov [matrix + si], ax
        pop ax

        mov dx, 10
        call fprint_char_by_addr

        ; mov dx, 10
        ; call fprint_char_by_addr

        add si, offst
        jmp fill_matrix_jloop
fill_matrix_jloop_end:
        inc cx

        mov dx, 10
        call fprint_char_by_addr
        jmp fill_matrix_iloop

```

```

ffill_matrix endp

mTransposeMatrix macro matrix, row, col, resMatrix
local rowLoop, colLoop
push ax; Сохранение регистров, используемых в макросе, в стек
push bx
push cx
push di
push si
push dx
xor di, di; Обнуляем смещение по строкам
mov cx, row

rowLoop:
; Внешний цикл, проходящий по строкам
push cx
xor si, si; Обнуляем смещение по столбцам
mov cx, col

colLoop:
; Внутренний цикл, проходящий по столбцам
mov ax, col
mul di; Устанавливаем смещение по строкам
add ax, si; Устанавливаем смещение по столбцам
mov bx, ax
mov ax, matrix[bx]
push ax; Заносим текущий элемент в стек
mov ax, row
mul si; Устанавливаем смещение по строкам
add ax, di; Устанавливаем смещение по столбцам
; (смещения по строкам и столбцам меняются
; местами по сравнению с оригинальной матрицей)
mov bx, ax
pop ax
mov resMatrix[bx], ax; Заносим в новую матрицу элемент
; сохранённый в стеке
add si, 2; Переходим к следующему элементу
; (размером в слово)
loop colLoop
add di, 2; Переходим к следующей строке
pop cx
loop rowLoop
pop dx; Перенос сохранённых значений обратно в регистры
pop si
pop di
pop cx
pop bx
pop ax
endm mTransposeMatrix

fprint_transpode_matrix proc
push ax
push cx
push bx
push dx
push si
push di

mov cx, 0; rows counter
mov bx, rows

print_transpode_matrix_iloop:
cmp cx, bx
je print_transpode_matrix_iloop_end

xor ax, ax

mov ax, cx

push ax
push bx

mov ax, offst
mov bx, columns
mul bx
mov dx, ax

pop bx
pop ax

mul dx
mov si, ax

mov di, ax

push ax
push bx
mov ax, offst
mov bx, columns
mul bx
add di, ax
pop bx
pop ax

xor ax, ax

print_transpode_matrix_jloop:

```

```

        cmp si, di
        je print_transpose_matrix_jloop_end
        mov dx, [res_matrix + si]
        call fprint_align_item
        mov dx, ''
        call fprint_char_by_addr
        add si, offst
        jmp print_transpose_matrix_jloop
print_transpose_matrix_jloop_end:
        inc cx
        mov dx, 10
        call fprint_char_by_addr
        jmp print_transpose_matrix_iloop
print_transpose_matrix_iloop_end:
        pop ax
        pop cx
        pop bx
        pop dx
        pop si
        pop di
        ret
fprint_transpose_matrix endp

ftranspose_matrix proc
mTransposeMatrix matrix, rows, columns, res_matrix
        mov     ax, rows
        mov     bx, columns
        mov     columns, ax
        mov     rows, bx

        mov ax, res_matrix
        mov matrix, ax
        call fprint_transpose_matrix

        mov ax, rows
        mov bx, columns
        mov columns, ax
        mov rows, bx

        ret
ftranspose_matrix endp

start:
        mov ax, @data
        mov ds, ax

        mput_msg msg_input_rows
        call fprint_message
        mnumber_input rows
        call fprint_ln

        mput_msg msg_input_columns
        call fprint_message
        mnumber_input columns
        call fprint_ln
        call ffill_matrix
        call fshow_menu
        call fexit
        end start

```

