

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

*Звіти до комп'ютерних практикумів з кредитного модуля
«Програмування мовою Асемблер»*

Прийняв
доцент кафедри ТК
Лісовиченко О. І.
“ ... ” 2020 р.

Виконав
студент групи ІП-91
Кінчур В. В.

Київ 2020

Комп'ютерний практикум №4

Тема: масиви

Завдання

1. Написати програму знаходження суми елементів одновимірного масиву, елементи вводять користувач.
2. Написати програму пошуку максимального (або мінімального) елемента одновимірного масиву, елементи вводять користувач.
3. Написати програму сортування одновимірного масиву цілих чисел загального вигляду.
4. Написати програму пошуку координат всіх входжень заданого елемента в двовимірному масиві, елементи масиву та пошуковий вводять користувач

Текст программы (NASM):

```
global _start
```

```
%define UPPER_BOUND 256
```

```
%define BIT32_LEN 13 ; len(-2**31) + 1
```

```
%macro print_str 1+
```

```
    jmp %%output
```

```
%%string: db %1
```

```
%%output:
```

```
    mov ebx, 1
```

```
    mov eax, 4
```

```
    mov ecx, %%string
```

```
    mov edx, %%output-%%string
```

```
    int 80h
```

```
%endmacro
```

```
section .bss
```

```
arr    resd UPPER_BOUND ; buffer for one dimensional array
```

```
arrSize resd 1 ; size of one dimensional array
```

```
matrix resd UPPER_BOUND * UPPER_BOUND
```

```
rowSize resd 1
```

```
colSize resd 1
```

```
buffer resb BIT32_LEN
```

```
dummy  resb 1
```

```
err    resb 1          ; global variable to handle errors
```

```
section .text
```

```
_start:
```

```
;--- print messages to stdin
```

```
_print_intro:
```

```
    print_str 'Choose one of the following:', 0xA, 0x0
```

```
    print_str '1 - find sum of elements of array', 0xA, 0x0
```

```
    print_str '2 - find maximal element in array', 0xA, 0x0
```

```
    print_str '3 - sort array', 0xA, 0x0
```

```
    print_str '4 - find element in matrix', 0xA, 0x0
```

```
    print_str 'Type your choice: ', 0x0
```

```
;--- read answer option from stdin
```

```
_handle_input:
```

```
    call _read_32bit
```

```
    cmp byte [err], 1
```

```
    je .error
```

```
    cmp eax, 1
```

```
    jl .error
```

```
    cmp eax, 4
```

```
    jg .error
```

```
    cmp eax, 4
```

```
    je .matrix
```

```
.array:
```

```
    push eax          ; save eax
```

```
call _make_array
pop eax          ; restore eax
```

```
cmp eax, 1
je .sum
cmp eax, 2
je .max
cmp eax, 3
je .sort
```

.matrix:

```
call _make_matrix
call _find_in_matrix
jmp _exit
```

.sort:

```
print_str 'Sorted array:', 0xA, 0x0
call _sort_array
call _print_array
jmp _exit
```

.sum:

```
print_str 'Sum of elements of array: ', 0x0
call _find_sum
```

```
cmp byte [err], 1      ; check for error
jne .print_eax         ; all is OK
jmp _exit              ; overflow occurred
```

.max:

```
    print_str 'Maximal element of array: ', 0x0
    call _find_max
```

.print_eax:

```
    push eax
    call _print_num
    add esp, 4
```

```
    print_str 0xA
    jmp _exit
```

.error:

```
    print_str 'Invalid option! Try again: ', 0x0
    jmp _handle_input
```

;--- end program execution

_exit:

```
    mov ebx, 0
    mov eax, 1
    int 80h
```

;--- atoi(const char* str) -> int32 (eax)

;--- eax - output

;--- bl - current char

;--- dl - sign

_atoi:

```
    enter 0, 0
    xor eax, eax
```

```
xor ebx, ebx
xor edx, edx
mov ecx, [ebp+8]      ; pointer to first char in input
```

.loop:

```
mov bl, byte [ecx]    ; current char
cmp bl, 0              ; \0
je .done
cmp bl, 10             ; \n
je .done

cmp bl, 45             ; check for minus
je .check_sign
cmp bl, 48             ; less than '0'
jl .error
cmp bl, 57             ; greater than '9'
jg .error
```

.valid_digit:

```
sub bl, 48            ; get digit
imul eax, dword 10
jo .error
add eax, ebx
jo .error
jmp .new_iteration
```

.check_sign:

```
cmp ecx, [ebp+8]      ; check is it first char
jne .error
```

```
mov dl, 1
```

```
.new_iteration:
```

```
inc ecx
```

```
jmp .loop
```

```
.error:
```

```
mov byte [err], 1 ; an error occurred
```

```
jmp .exit ; jump to _error block
```

```
.done:
```

```
cmp dl, 0 ; check for sign
```

```
je .exit
```

```
neg eax
```

```
jo .error
```

```
.exit:
```

```
leave
```

```
ret
```

```
_sys_write:
```

```
mov ebx, 1
```

```
mov eax, 4
```

```
int 80h
```

```
ret
```

```
_sys_read:
```

```
mov ebx, 2
```

```
mov eax, 3
```



```
int 80h
```

```
ret
```

```
; --- print number to stdin
```

```
; --- input: stack top
```

```
_print_num:
```

```
enter 0, 0
```

```
mov edi, buffer
```

```
cmp dword [ebp+8], 0 ; check sign
```

```
jge .init
```

```
mov byte [edi], '-'
```

```
inc edi
```

```
neg dword [ebp+8] ; make input positive
```

```
.init:
```

```
mov eax, [ebp+8]
```

```
mov ebx, 10
```

```
.loop:
```

```
xor edx, edx
```

```
div ebx
```

```
add dl, 48 ; '0'
```

```
push edx
```

```
test eax, eax
```

```
jnz .loop
```

```
.stack:
```

```
    pop eax
    stosb
    cmp esp, ebp        ; check stack is empty
    jne .stack
```

.print:

```
    mov ecx, buffer
    mov edx, edi
    sub edx, buffer
    call _sys_write
```

.exit:

```
    leave
    ret
```

_read_32bit:

```
    mov byte [err], 0    ; nullify error
    mov ecx, BIT32_LEN   ; buffer length
```

.clear_buffer:

```
    mov byte [buffer+ecx], 0 ; fill with \0
    loop .clear_buffer
```

.read:

```
    mov ebx, 2
    mov ecx, buffer        ; input in buffer
    mov edx, BIT32_LEN
```

.flush_stdin:

```

mov eax, 3          ; sys_read
int 80h             ; call kernel
cmp byte [ecx+eax-1], 10  ; compare last char in stdin with \n
je .convert

```

```

mov edx, 1
mov ecx, dummy
jmp .flush_stdin

```

.convert:

```

push buffer          ; push str to convert
mov byte [err], 0    ; nullify error variable
call _atoi           ; if an error occurred,
add esp, 4            ; err = 1, else err = 0

```

.exit:

```

ret

```

;--- just read from stdin to eax number from [1, UPPER_BOUND]

_read_size:

```

call _read_32bit

cmp byte [err], 1    ; not a number
je .error
cmp eax, UPPER_BOUND
jg .error             ; size <= UPPER_BOUND
cmp eax, 1
jl .error             ; size > 0
ret

```

.error:

```
print_str 'Try again:', 0x20, 0x0  
jmp _read_size
```

; create array from stdin input

_make_array:

```
print_str 'Enter array size (integer from [1, 256]): ', 0x0  
call _read_size      ; eax = array size
```

.init:

```
mov [arrSize], eax  
print_str 'Please, fill array with 32bit signed integers: ', 0xA, 0x0
```

```
mov ecx, [arrSize]    ; initialize counter  
xor esi, esi
```

.fill:

```
push ecx              ; save registers  
push esi
```

```
print_str 'array['  
call _print_num       ; print idx (esi)  
print_str ']' = '
```

```
call _read_32bit      ; eax = number  
pop esi              ; restore esi
```

```
cmp byte [err], 1     ; check for error
```

```
je .error
```

```
pop ecx
```

```
mov [arr+esi*4], eax      ; arr[i] = number
```

```
inc esi
```

```
loop .fill
```

```
.exit:
```

```
ret
```

```
.error:
```

```
print_str 'An error occurred. Try again!', 0xA, 0x0
```

```
pop ecx
```

```
jmp .fill
```

```
;--- place sum of array to eax
```

```
_find_sum:
```

```
mov byte [err], 0        ; nullify error
```

```
xor eax, eax
```

```
xor esi, esi
```

```
mov ecx, [arrSize]
```

```
.loop:
```

```
add eax, [arr+esi*4]
```

```
jo .overflow             ; check for overflow
```

```
inc esi
```

```
loop .loop
```

```
ret
```

.overflow:

```
print_str 0xA, 'An overflow error occurred!', 0xA, 0x0
mov byte [err], 1      ; an error occurred
ret
```

; place maximal element of array to eax

_find_max:

```
mov eax, [arr]
mov ecx, [arrSize]
xor esi, esi
```

.loop:

```
mov edx, [arr+esi*4]
cmp edx, eax
jle .next
mov eax, edx
```

.next:

```
inc esi
loop .loop
ret
```

; sort array (by reference)

_sort_array:

```
cmp dword [arrSize], 1    ; check for 1 element
jle .exit
```

```
mov ecx, [arrSize]        ; initialize outer loop
dec ecx
```

.outer:

```
    mov edx, ecx          ; initialize inner loop
    xor esi, esi
```

.inner:

```
    mov eax, [arr+esi*4]
    cmp eax, [arr+esi*4+4] ; arr[j] > arr[j+1]
    jl .no_swap
    xchg eax, [arr+esi*4+4] ; swap
    mov [arr+esi*4], eax
```

.no_swap:

```
    inc esi
    dec edx
    jnz .inner

    loop .outer          ; while (--ecx)
```

.exit:

```
    ret
```

; print array to stdin

_print_array:

```
    print_str '[ ', 0x0

    mov ecx, [arrSize]
    xor esi, esi
```

.loop:

push ecx ; save ecx

push dword [arr+esi*4]

call _print_num

add esp, 4

print_str 0x20 ; print whitespace

pop ecx ; restore ecx

inc esi

loop .loop

.exit:

print_str ']', 0xA, 0x0

ret

; create matrix from stdin input

_make_matrix:

.read_sizes:

print_str 'Enter number of rows (integer from [1, 256]): ', 0x0

call _read_size

mov [rowSize], eax

print_str 'Enter number of columns (integer from [1, 256]): ', 0x0

call _read_size

mov [colSize], eax

.init:


```
xor ebx, ebx          ; i
```

```
.outer:
```

```
xor esi, esi          ; j
```

```
.inner:
```

```
push esi
```

```
push ebx
```

```
print_str 'matrix[' , 0x0
```

```
call _print_num        ; TOS = ebx
```

```
print_str '][' , 0x0
```

```
mov eax, [esp]
```

```
xchg eax, [esp+4]
```

```
mov [esp], eax          ; swap values in stack
```

```
call _print_num        ; TOS = esi
```

```
print_str ']' = '      ; matrix[i][j] =
```

```
call _read_32bit        ; eax = int(input)
```

```
pop esi
```

```
cmp byte [err], 1       ; check for error
```

```
je .error
```

```
pop ebx
```

```
mov edx, [rowSize]
```

```
imul edx, ebx
```

```
add edx, esi
mov [matrix+edx*4], eax    ; matrix[i][j] = num
```

```
inc esi
cmp esi, [colSize]
jl .inner
```

```
inc ebx
cmp ebx, [rowSize]
jl .outer
```

.exit:

```
ret
```

.error:

```
print_str 'An error occurred. Try again!', 0xA, 0x0
pop ebx
jmp .inner
```

_find_in_matrix:

```
%define toFind [ebp-4]
```

```
enter 4,0
```

.read_number:

```
print_str 'Type number to find (32bit signed integer): ', 0x0
call _read_32bit      ; eax = int(input)
cmp byte [err], 1
```

```
je .read_number  
mov toFind, eax      ; save to local variable
```

```
.init:  
  
print_str 'Suitable indices:', 0xA, 0x0  
xor ebx, ebx
```

```
.outer:  
  
xor esi, esi
```

```
.inner:  
  
mov eax, [rowSize]  
imul ebx  
add eax, esi  
  
mov eax, [matrix+eax*4]  
cmp eax, toFind  
je .print
```

```
.next:  
  
inc esi  
cmp esi, [colSize]  
jl .inner  
  
inc ebx  
cmp ebx, [rowSize]  
jl .outer
```

```
.exit:
```

leave

ret

.print:

push esi

push ebx

call _print_num

print_str 0x20

mov eax, [esp]

xchg eax, [esp+4]

mov [esp], eax

call _print_num

print_str 0xA

pop esi

pop ebx

jmp .next

Введені та отримані результати

Тестування програми №1

```
Choose one of the following:
1 - find sum of elements of array
2 - find maximal element in array
3 - sort array
4 - find element in matrix
Type your choice: 1
Enter array size (integer from [1, 256]): 10
Please, fill array with 32bit signed integers:
array[1] = 1
array[2] = 2
array[3] = 3
array[4] = 4
array[5] = 5
array[6] = 6
array[7] = 7
array[8] = 8
array[9] = 9
array[10] = -1
Sum of elements of array: 44
```

Тестування програми №2

```
Choose one of the following:
1 - find sum of elements of array
2 - find maximal element in array
3 - sort array
4 - find element in matrix
Type your choice: 2
Enter array size (integer from [1, 256]): 10
Please, fill array with 32bit signed integers:
array[1] = 1
array[2] = -100
array[3] = 12
array[4] = abcd
An error occured. Try again!
array[4] = -121
array[5] = 23457
array[6] = 3534535
array[7] = 3243232323
An error occured. Try again!
array[7] = 121
array[8] = -1
array[9] = 0
array[10] = 12
Maximal element of array: 3534535
```

Тестування програми №3

```
Choose one of the following:
1 - find sum of elements of array
2 - find maximal element in array
3 - sort array
4 - find element in matrix
Type your choice: 3
Enter array size (integer from [1, 256]): 10
Please, fill array with 32bit signed integers:
array[1] = 1
array[2] = -100
array[3] = 0
array[4] = 3123
array[5] = -12
array[6] = 34667
array[7] = -323
array[8] = 6546
array[9] = --1
An error occured. Try again!
array[9] = 12
array[10] = 1-
An error occured. Try again!
array[10] = 1.1
An error occured. Try again!
array[10] = 23
Sorted array:
[ -323 -100 -12 0 1 12 23 3123 6546 34667 ]
```

Тестування програми №4

```
Choose one of the following:
1 - find sum of elements of array
2 - find maximal element in array
3 - sort array
4 - find element in matrix
Type your choice: 4
Enter number of rows (integer from [1, 256]): 3
Enter number of columns (integer from [1, 256]): 3
matrix[1][1] = 1
matrix[1][2] = 2
matrix[1][3] = 3
matrix[2][1] = 4
matrix[2][2] = 5
matrix[2][3] = 6
matrix[3][1] = 7
matrix[3][2] = 8
matrix[3][3] = 1
Type number to find (32bit signed integer): 1
Suitable indices:
1 1
3 3
```

Схема функціонування програми

Див. додаток.

Висновок

1. Написав програму знаходження суми елементів одновимірного масиву, елементи вводять користувач.
2. Написав програму пошуку максимального (або мінімального) елемента одновимірного масиву, елементи вводять користувач.
3. Написав програму сортування одновимірного масиву цілих чисел загального вигляду.
4. Написав програму пошуку координат всіх входжень заданого елемента в двовимірному масиві, елементи масиву та пошуковий вводять користувач.

