

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА КОНСТРУЮВАННЯ ЕОА

ЗВІТ

з лабораторної роботи №3
по курсу «Основи мікропроцесорної техніки - 1»

Виконав:

студент гр. ДК-82

Сопіра Р. Я.

Перевірив:

доц. Корнєв В. П.

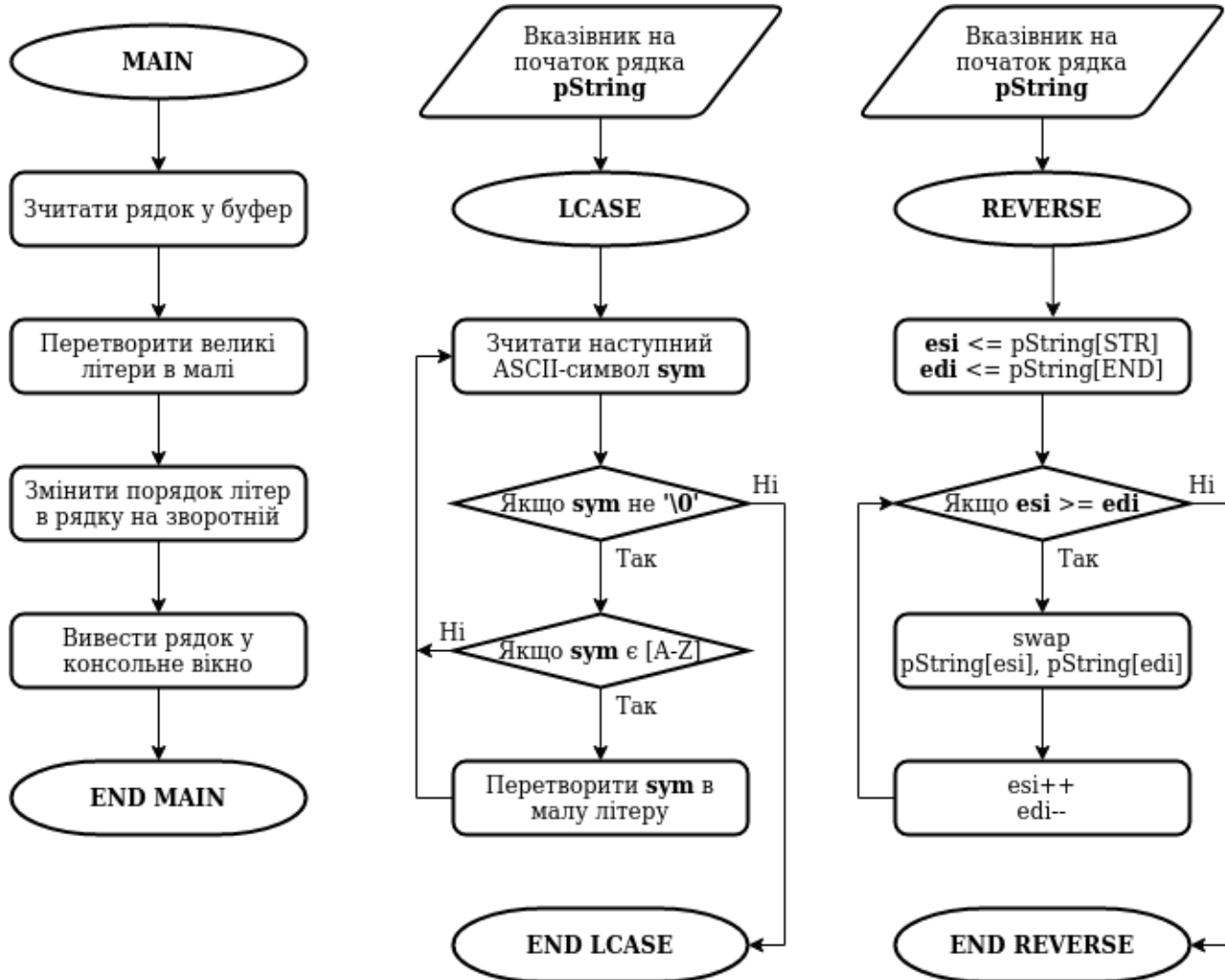
ст. в. Бондаренко Н. О.

Київ – 2020

Завдання

- 1) Ввести символний рядок із клавіатури
- 2) Замінити в отриманому рядку великі літери на малі
- 3) Вивести рядок у зворотньому порядку

Блок-схема алгоритму програми



Програма

```
INCLUDE Irvine32.inc

.data
buffer BYTE 16 DUP(?)           ; main buffer for strings

...

main PROC                       ; MAIN FUNC
                                ; str input
    mov edx, OFFSET buffer      ; str addr offset to edx
    mov ecx, (SIZEOF buffer) - 1 ; str buffer size to ecx
    call ReadString             ; reading string from stdin

    invoke Str_lcase, ADDR buffer ; str to lowercase
    invoke Str_reverse, ADDR buffer ; str reversing

                                ; str out
    mov edx, OFFSET buffer      ; str offset to edx
    call WriteString            ; str to stdout?
    call Crlf                   ; car return, line feed
    call WaitMsg                ; press any key

    exit
main ENDP
END main
```

Перші дві команди заносять у регістри даних (**edx**) і лічильника (**ecx**) адресу буфера для запису рядка і розмір ділянки доступної для запису відповідно. Далі викликається процедура *ReadString* із бібліотеки *Irvine32* параметрами якої і є дані, тільки що занесені в регістри **edx** та **ecx**.

В даному випадку для запису безпосередньо символів будуть доступними чотирнадцять із шістнадцяти байтів рядка **buffer**, так як у п'ятнадцятий байт буде записаний символ нульового кінця рядка, а шістнадцятий просто не використовується.

Далі виконується “виклик” процедури *Str_lcase* за допомогою команди **invoke**, яка завантажує до стеку реєстри і параметри, що використовуються процедурою (прописані після директиви *USES* в об’яві самої процедури) і точку повернення із процедури.

Дана процедура використовується для перетворення великих ASCII-літер у малі. Власне перетворення ґрунтується на наступній особливості кодування ASCII: велика і мала літери відрізняються лише одним бітом (третім) у своєму коді — всі інші біти абсолютно ідентичні — і тому змінивши лише один біт числового значення можна перетворити велику літеру в малу й навпаки.

Таке перетворення можна легко виконати операцією логічного АБО, наприклад:

$$I=01001001 \text{ , } i=01101001$$

$$I \rightarrow i \Rightarrow 01001001_I \vee 00100000 = 01101001_i$$

Вихідний код *Str_lcase*:

```
Str_lcase PROC USES eax esi,          ; reworked Str_ucase from Irvine32
    pString:PTR BYTE

    mov esi, pString
L1:
    mov al, [esi]                     ; get char
    cmp al, 0                         ; end of string?
    je L3                             ; if yes then quit
    cmp al, 'A'                       ; below "A"?
    jb L2                             ; above "Z"?
    cmp al, 'Z'
    ja L2
    or BYTE PTR [esi], 00100000b      ; convert the char

L2:
    inc esi                           ; next char
    jmp L1

L3:
    ret
Str_lcase ENDP
```

Алгоритм роботи процедури досить простий: спочатку у регістр **esi** копіюється адреса, що вказує на початок рядка, наступною командою за цією адресою у **al** копіюється сам символ рядка, перевіряється спочатку на рівність з нулем (кінець рядка), а потім чи потрапляє даний символ до ряду великих літер.

Якщо символ потрапляє до цього ряду, то виконується вищезазначене перетворення, якщо ні — виконується перехід на мітку **L2**, де відбувається збільшення адреси на одиницю (**esi** вказує на наступний символ рядка) і перехід на початок, мітку **L1**.

Часткова таблиця ASCII кодів:

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
64	01000000	100	40	@	96	01100000	140	60	`
65	01000001	101	41	A	97	01100001	141	61	a
66	01000010	102	42	B	98	01100010	142	62	b
67	01000011	103	43	C	99	01100011	143	63	c
68	01000100	104	44	D	100	01100100	144	64	d
69	01000101	105	45	E	101	01100101	145	65	e
70	01000110	106	46	F	102	01100110	146	66	f
71	01000111	107	47	G	103	01100111	147	67	g
72	01001000	110	48	H	104	01101000	150	68	h
73	01001001	111	49	I	105	01101001	151	69	i
74	01001010	112	4A	J	106	01101010	152	6A	j
75	01001011	113	4B	K	107	01101011	153	6B	k
76	01001100	114	4C	L	108	01101100	154	6C	l
77	01001101	115	4D	M	109	01101101	155	6D	m
78	01001110	116	4E	N	110	01101110	156	6E	n
79	01001111	117	4F	O	111	01101111	157	6F	o
80	01010000	120	50	P	112	01110000	160	70	p
81	01010001	121	51	Q	113	01110001	161	71	q
82	01010010	122	52	R	114	01110010	162	72	r
83	01010011	123	53	S	115	01110011	163	73	s
84	01010100	124	54	T	116	01110100	164	74	t
85	01010101	125	55	U	117	01110101	165	75	u
86	01010110	126	56	V	118	01110110	166	76	v
87	01010111	127	57	W	119	01110111	167	77	w
88	01011000	130	58	X	120	01111000	170	78	x
89	01011001	131	59	Y	121	01111001	171	79	y
90	01011010	132	5A	Z	122	01111010	172	7A	z
91	01011011	133	5B	[123	01111011	173	7B	{
92	01011100	134	5C	\	124	01111100	174	7C	
93	01011101	135	5D]	125	01111101	175	7D	}
94	01011110	136	5E	^	126	01111110	176	7E	~
95	01011111	137	5F	_	127	01111111	177	7F	DEL

```

Disassembly      test.asm - x
40      mov     al, [edi]
41      mov     [esi], al
42      mov     [edi], bl
43
44      inc     esi                ; move pointers toward each other
45      dec     edi
46      jmp     swap_loop
47
48  finish:
49      ret
50
51  Str_reverse ENDP
52
53  ; #####
54  main PROC                ; MAIN FUNC
55                          ; str input
56      mov     edx, OFFSET buffer ; str addr offset to edx
57      mov     ecx, (SIZEOF buffer) - 1 ; str buffer size to ecx
58      call    ReadString        ; reading string from stdin
59
60      invoke Str_lcase, ADDR buffer ; str to lowercase < 1043ms elapsed
61      invoke Str_reverse, ADDR buffer ; str reversing
62
63      ; str out
64      mov     edx, OFFSET buffer ; str offset to edx
65      call    WriteString        ; str to stdout?
66      call    Crlf               ; car return, line feed
67      call    WaitMsg            ; press any key
68
69      exit
70  main ENDP
71  END main
72
73
74
75
76
77
78
79
80
81

```

Registers
EAX = 00000000 EBX = 7FDFE000 ECX = 0000000F
EDX = 000D6000 ESI = 00000000 EDI = 00000000
EIP = 000D36CB ESP = 0038FB84 EBP = 0038FBBC
EFL = 00000202
OV = 0 UP = 0 EI = 1 PL = 0 ZR = 0 AC = 0 PE = 0
CY = 0

133 %
Registers Diagnostic Tools
Memory |
Address: 0:000D6000

0:000D6000	71	57	65	52	74	59	20	30	31	32	33	00	00	00	00	quantity 0123...
0:000D600E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0:000D601C	00	00	00	00	01	30	31	32	33	34	35	36	37	38	39 012345678
0:000D602A	19	41	42	43	44	45	46	20	20	20	20	20	20	20	20	SWAPCF
0:000D6038	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6046	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6054	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6062	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6070	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D607E	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D608C	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D609A	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60A8	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60B6	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60C4	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60D2	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60E0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60EE	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D60FC	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D610A	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6118	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6126	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6134	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6142	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0:000D6150	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

133 %

No issues found

Ln: 68 Ch: 1 MREQ GFL

The screenshot displays two panels from the Visual Studio Code IDE:

- Assembly Editor (Left Panel):** Shows the assembly code for the `Str_reverse` procedure. The code uses Irvine32.inc and implements a loop to reverse the string by swapping characters at the beginning and end of the buffer.

```

1  INCLUDE Irvine32.inc
2
3  .data
4  buffer BYTE 16 DUP(?)           ; main buffer for strings
5
6  .code
7  ; #####          #####
8  Str_reverse PROC USES eax esi,   ; reworked Str_ucase from Irvine32
9      pString:PTR BYTE
10
11      mov esi, pString
12  L1:
13      mov al, [esi]               ; get char
14      cmp al, 0                  ; end of string?
15      je L3                      ; if yes then quit
16      cmp al, 'A'                ; below "A"?
17      jnb L2                     ; above "Z"?
18      cmp al, 'Z'
19      ja L2
20      or BYTE PTR [esi], 0x0100000b ; convert the char
21
22  L2:
23      inc esi                    ; next char
24      jmp L1 L1 times elapsed
25
26
27  L3:
28      ret
29  Str_reverse ENDP
30
31 ; #####          #####
32 Str_reverse PROC USES esi edi,   ; str reverse function
33     pString:PTR BYTE
    
```

- Registers Window (Right Panel):** Displays the current state of CPU registers. Notable values include EAX=0, EBX=7EFDCE00, ECX=0000000F, EDI=00000000, ESP=0000166A, and EBP=003FBAB8. It also shows control flags like OV=0, UP=0, EI=1, PL=0, ZR=0, AC=0, PE=0, and CY=0.
- Memory Window (Bottom Right Panel):** Shows a memory dump starting at address 00400060. The first few bytes are 71, 57, 65, 52, 74, 59, 20, 30, 31, 32, 33, 34, 35, 37, 38, which correspond to the ASCII string "HELLO WORLD".

Наступний символ - 'W' - є великою літерою й тому перетворюється у 'w':

The screenshot shows a debugger window with assembly code for a function named `Str_lcase`. The code is written in x86 assembly and includes comments in English. The function is designed to convert a string to lowercase. It starts by including `Irvine32.inc` and defining a buffer. The main logic is in the `.code` section, where it moves the pointer to the string into `esi` and enters a loop labeled `L1`. In this loop, it checks if the current character is the null terminator (`0`). If not, it checks if it's an uppercase letter (`'A'` to `'Z'`). If it is, it converts it to lowercase by adding `0x00000020` (32). The loop continues until the null terminator is reached, at which point it returns. A `Str_reverse` function is also defined below.

```
1 INCLUDE Irvine32.inc
2
3 .data
4 buffer BYTE 16 DUP(?)           ; main buffer for strings
5
6 .code
7 ; #####
8 Str_lcase PROC USES eax esi,     ; reworked Str_ucase from Irvine32
9     pString:PTR BYTE
10
11     mov esi, pString
12 L1:
13     mov al, [esi]                ; get char
14     cmp al, 0                   ; end of string?
15     je L3                       ; if yes then quit
16     cmp al, 'A'                 ; below "A"?
17     jb L2                       ; above "Z"?
18     cmp al, 'Z'
19     ja L2
20     or BYTE PTR [esi], 00100000b ; convert the char < 1ms elapsed
21
22 L2:
23     inc esi                     ; next char
24     jmp L1
25
26 L3:
27     ret
28 Str_lcase ENDP
29
30 ; #####
31 Str_reverse PROC USES esi edi,   ; str reverse function
32     pString:PTR BYTE
33
```

The right-hand side of the image shows the debugger's registers and memory windows. The registers window displays the current state of the CPU registers: `EAX = 00000057`, `EBX = 7EFD0000`, `ECX = 0000000F`, `EDX = 000D6000`, `ESI = 000D6001`, `EDI = 00000000`, `EIP = 000D3686`, `ESP = 0038FBA0`, `EBP = 0038FBA8`, and `EFL = 00000293`. The memory window shows the address `0x000D6000` and its contents, which are mostly zeros.

This screenshot shows the same assembly code as the previous image, but with a different execution state. The instruction pointer (EIP) is now `000D368A`, indicating that the program has executed the `jmp L1` instruction and jumped back to the start of the loop. The registers window shows updated values: `EAX = 00000057`, `EBX = 7EFD0000`, `ECX = 0000000F`, `EDX = 000D6000`, `ESI = 000D6002`, `EDI = 00000000`, `EIP = 000D368A`, `ESP = 0038FBA0`, `EBP = 0038FBA8`, and `EFL = 00000202`. The memory window remains the same, showing address `0x000D6000`.

Наступною “викликається” процедура *Str_reverse* для того, щоб виконати перестановку символів буферу в зворотній порядок.

Вихідний код *Str_reverse*:

```
Str_reverse PROC USES esi edi,      ; str reverse function
    pString:PTR BYTE

    mov     esi, pString             ; both esi & edi point to the start of
    mov     edi, pString
    dec     edi

find_end:                               ; finds the end index of a string
    inc     edi                     ; advance end pointer
    mov     al, [edi]               ; look for end of string
    cmp     al, 0                   ; if char is null term then
    jnz     find_end                ; false: move to next char
    dec     edi                     ; true: --edi to last char

swap_loop:
    cmp     esi, edi                 ; if start >= end, then we are finished
    jge     finish

    mov     bl, [esi]                ; swap characters
    mov     al, [edi]
    mov     [esi], al
    mov     [edi], bl

    inc     esi                     ; move pointers toward each other
    dec     edi
    jmp     swap_loop

finish:
    ret
Str reverse ENDP
```

Для того щоб вивести рядок у зворотньому порядку скористаємося наступним алгоритмом: визначимо регістри **esi**, **edi** як “вказівники” на початок і кінець рядка; перенесемо **edi** на останній символ простим циклом, що посимвольно шукає символ кінця рядка, тобто **0** і переходить на попередній символ.

Шукаємо останній символ рядка:

```

18  cmp al, 'Z'          ; above "Z"?
19  ja  L2
20  or  BYTE PTR [esi], 00100000b ; convert the char
21
22  L2:
23  inc esi              ; next char
24  jmp L1
25
26  L3:
27  ret
28  Str_lcase ENDP
29
30  ; #####
31  Str_reverse PROC USES esi, edi ; str reverse function
32  pString:PTR BYTE
33
34  mov  esi, pString      ; both esi & edi point to the start of a string
35  mov  edi, pString
36  dec  edi               ; --edi < 1ms elapsed
37
38  find_end:
39  inc  edi               ; advance end pointer.
40  mov  al, [edi]         ; look for end of string.
41  cmp  al, 0             ; if char is null term then
42  jnz  find_end          ; false: move to next char
43  dec  edi               ; true: --edi to last char
44
45  swap_loop:
46  cmp  esi, edi          ; if start >= end, then we are finished.
47  jge  finish
48
49  mov  bl, [esi]         ; swap characters.
50  mov  al, [edi]
51  mov  [esi], al
52  mov  [edi], bl
53  inc  esi
54  dec  edi
55  jmp  swap_loop
56
57  finish:
58  ret
59  Str_reverse ENDP
60
61  ; #####
62  Str_upper PROC USES esi, edi ; str upper function
63  pString:PTR BYTE
64
65  mov  esi, pString
66  mov  edi, pString
67  dec  edi
68
69  find_end:
70  inc  edi
71  mov  al, [edi]
72  cmp  al, 0
73  jnz  find_end
74  dec  edi
75
76  swap_loop:
77  cmp  esi, edi
78  jge  finish
79
80  mov  bl, [esi]
81  mov  al, [edi]
82  mov  [esi], al
83  mov  [edi], bl
84  inc  esi
85  dec  edi
86  jmp  swap_loop
87
88  finish:
89  ret
90  Str_upper ENDP
91
92  ; #####
93  Str_toupper PROC USES esi, edi ; str toupper function
94  pString:PTR BYTE
95
96  mov  esi, pString
97  mov  edi, pString
98  dec  edi
99
100 find_end:
101 inc  edi
102 mov  al, [edi]
103 cmp  al, 0
104 jnz  find_end
105 dec  edi
106
107 swap_loop:
108 cmp  esi, edi
109 jge  finish
110
111 mov  bl, [esi]
112 mov  al, [edi]
113 mov  [esi], al
114 mov  [edi], bl
115 inc  esi
116 dec  edi
117 jmp  swap_loop
118
119 finish:
120 ret
121 Str_toupper ENDP
122
123 ; #####
124 Str_tolower PROC USES esi, edi ; str tolower function
125 pString:PTR BYTE
126
127 mov  esi, pString
128 mov  edi, pString
129 dec  edi
130
131 find_end:
132 inc  edi
133 mov  al, [edi]
134 cmp  al, 0
135 jnz  find_end
136 dec  edi
137
138 swap_loop:
139 cmp  esi, edi
140 jge  finish
141
142 mov  bl, [esi]
143 mov  al, [edi]
144 mov  [esi], al
145 mov  [edi], bl
146 inc  esi
147 dec  edi
148 jmp  swap_loop
149
150 finish:
151 ret
152 Str_tolower ENDP
153
154 ; #####
155 Str_strcmp PROC USES esi, edi ; str strcmp function
156 pString1:PTR BYTE
157 pString2:PTR BYTE
158
159 mov  esi, pString1
160 mov  edi, pString2
161 dec  edi
162
163 find_end1:
164 inc  edi
165 mov  al, [edi]
166 cmp  al, 0
167 jnz  find_end1
168 dec  edi
169
170 find_end2:
171 inc  edi
172 mov  al, [edi]
173 cmp  al, 0
174 jnz  find_end2
175 dec  edi
176
177 swap_loop:
178 cmp  esi, edi
179 jge  finish
180
181 mov  bl, [esi]
182 mov  al, [edi]
183 mov  [esi], al
184 mov  [edi], bl
185 inc  esi
186 dec  edi
187 jmp  swap_loop
188
189 finish:
190 ret
191 Str_strcmp ENDP
192
193 ; #####
194 Str_strncmp PROC USES esi, edi ; str strncmp function
195 pString1:PTR BYTE
196 pString2:PTR BYTE
197 count:DWORD
198
199 mov  esi, pString1
200 mov  edi, pString2
201 dec  edi
202
203 find_end1:
204 inc  edi
205 mov  al, [edi]
206 cmp  al, 0
207 jnz  find_end1
208 dec  edi
209
210 find_end2:
211 inc  edi
212 mov  al, [edi]
213 cmp  al, 0
214 jnz  find_end2
215 dec  edi
216
217 swap_loop:
218 cmp  esi, edi
219 jge  finish
220
221 mov  bl, [esi]
222 mov  al, [edi]
223 mov  [esi], al
224 mov  [edi], bl
225 inc  esi
226 dec  edi
227 jmp  swap_loop
228
229 finish:
230 ret
231 Str_strncmp ENDP
232
233 ; #####
234 Str_strcmpi PROC USES esi, edi ; str strcmpi function
235 pString1:PTR BYTE
236 pString2:PTR BYTE
237 count:DWORD
238
239 mov  esi, pString1
240 mov  edi, pString2
241 dec  edi
242
243 find_end1:
244 inc  edi
245 mov  al, [edi]
246 cmp  al, 0
247 jnz  find_end1
248 dec  edi
249
250 find_end2:
251 inc  edi
252 mov  al, [edi]
253 cmp  al, 0
254 jnz  find_end2
255 dec  edi
256
257 swap_loop:
258 cmp  esi, edi
259 jge  finish
260
261 mov  bl, [esi]
262 mov  al, [edi]
263 mov  [esi], al
264 mov  [edi], bl
265 inc  esi
266 dec  edi
267 jmp  swap_loop
268
269 finish:
270 ret
271 Str_strcmpi ENDP
272
273 ; #####
274 Str_strncmpi PROC USES esi, edi ; str strncmpi function
275 pString1:PTR BYTE
276 pString2:PTR BYTE
277 count:DWORD
278
279 mov  esi, pString1
280 mov  edi, pString2
281 dec  edi
282
283 find_end1:
284 inc  edi
285 mov  al, [edi]
286 cmp  al, 0
287 jnz  find_end1
288 dec  edi
289
290 find_end2:
291 inc  edi
292 mov  al, [edi]
293 cmp  al, 0
294 jnz  find_end2
295 dec  edi
296
297 swap_loop:
298 cmp  esi, edi
299 jge  finish
300
301 mov  bl, [esi]
302 mov  al, [edi]
303 mov  [esi], al
304 mov  [edi], bl
305 inc  esi
306 dec  edi
307 jmp  swap_loop
308
309 finish:
310 ret
311 Str_strncmpi ENDP
312
313 ; #####
314 Str_strcmpi2 PROC USES esi, edi ; str strcmpi2 function
315 pString1:PTR BYTE
316 pString2:PTR BYTE
317 count:DWORD
318
319 mov  esi, pString1
320 mov  edi, pString2
321 dec  edi
322
323 find_end1:
324 inc  edi
325 mov  al, [edi]
326 cmp  al, 0
327 jnz  find_end1
328 dec  edi
329
330 find_end2:
331 inc  edi
332 mov  al, [edi]
333 cmp  al, 0
334 jnz  find_end2
335 dec  edi
336
337 swap_loop:
338 cmp  esi, edi
339 jge  finish
340
341 mov  bl, [esi]
342 mov  al, [edi]
343 mov  [esi], al
344 mov  [edi], bl
345 inc  esi
346 dec  edi
347 jmp  swap_loop
348
349 finish:
350 ret
351 Str_strcmpi2 ENDP
352
353 ; #####
354 Str_strncmpi2 PROC USES esi, edi ; str strncmpi2 function
355 pString1:PTR BYTE
356 pString2:PTR BYTE
357 count:DWORD
358
359 mov  esi, pString1
360 mov  edi, pString2
361 dec  edi
362
363 find_end1:
364 inc  edi
365 mov  al, [edi]
366 cmp  al, 0
367 jnz  find_end1
368 dec  edi
369
370 find_end2:
371 inc  edi
372 mov  al, [edi]
373 cmp  al, 0
374 jnz  find_end2
375 dec  edi
376
377 swap_loop:
378 cmp  esi, edi
379 jge  finish
380
381 mov  bl, [esi]
382 mov  al, [edi]
383 mov  [esi], al
384 mov  [edi], bl
385 inc  esi
386 dec  edi
387 jmp  swap_loop
388
389 finish:
390 ret
391 Str_strncmpi2 ENDP
392
393 ; #####
394 Str_strcmp2 PROC USES esi, edi ; str strcmp2 function
395 pString1:PTR BYTE
396 pString2:PTR BYTE
397 count:DWORD
398
399 mov  esi, pString1
400 mov  edi, pString2
401 dec  edi
402
403 find_end1:
404 inc  edi
405 mov  al, [edi]
406 cmp  al, 0
407 jnz  find_end1
408 dec  edi
409
410 find_end2:
411 inc  edi
412 mov  al, [edi]
413 cmp  al, 0
414 jnz  find_end2
415 dec  edi
416
417 swap_loop:
418 cmp  esi, edi
419 jge  finish
420
421 mov  bl, [esi]
422 mov  al, [edi]
423 mov  [esi], al
424 mov  [edi], bl
425 inc  esi
426 dec  edi
427 jmp  swap_loop
428
429 finish:
430 ret
431 Str_strcmp2 ENDP
432
433 ; #####
434 Str_strncmp2 PROC USES esi, edi ; str strncmp2 function
435 pString1:PTR BYTE
436 pString2:PTR BYTE
437 count:DWORD
438
439 mov  esi, pString1
440 mov  edi, pString2
441 dec  edi
442
443 find_end1:
444 inc  edi
445 mov  al, [edi]
446 cmp  al, 0
447 jnz  find_end1
448 dec  edi
449
450 find_end2:
451 inc  edi
452 mov  al, [edi]
453 cmp  al, 0
454 jnz  find_end2
455 dec  edi
456
457 swap_loop:
458 cmp  esi, edi
459 jge  finish
460
461 mov  bl, [esi]
462 mov  al, [edi]
463 mov  [esi], al
464 mov  [edi], bl
465 inc  esi
466 dec  edi
467 jmp  swap_loop
468
469 finish:
470 ret
471 Str_strncmp2 ENDP
472
473 ; #####
474 Str_strcmp2i PROC USES esi, edi ; str strcmp2i function
475 pString1:PTR BYTE
476 pString2:PTR BYTE
477 count:DWORD
478
479 mov  esi, pString1
480 mov  edi, pString2
481 dec  edi
482
483 find_end1:
484 inc  edi
485 mov  al, [edi]
486 cmp  al, 0
487 jnz  find_end1
488 dec  edi
489
490 find_end2:
491 inc  edi
492 mov  al, [edi]
493 cmp  al, 0
494 jnz  find_end2
495 dec  edi
496
497 swap_loop:
498 cmp  esi, edi
499 jge  finish
500
501 mov  bl, [esi]
502 mov  al, [edi]
503 mov  [esi], al
504 mov  [
```

```

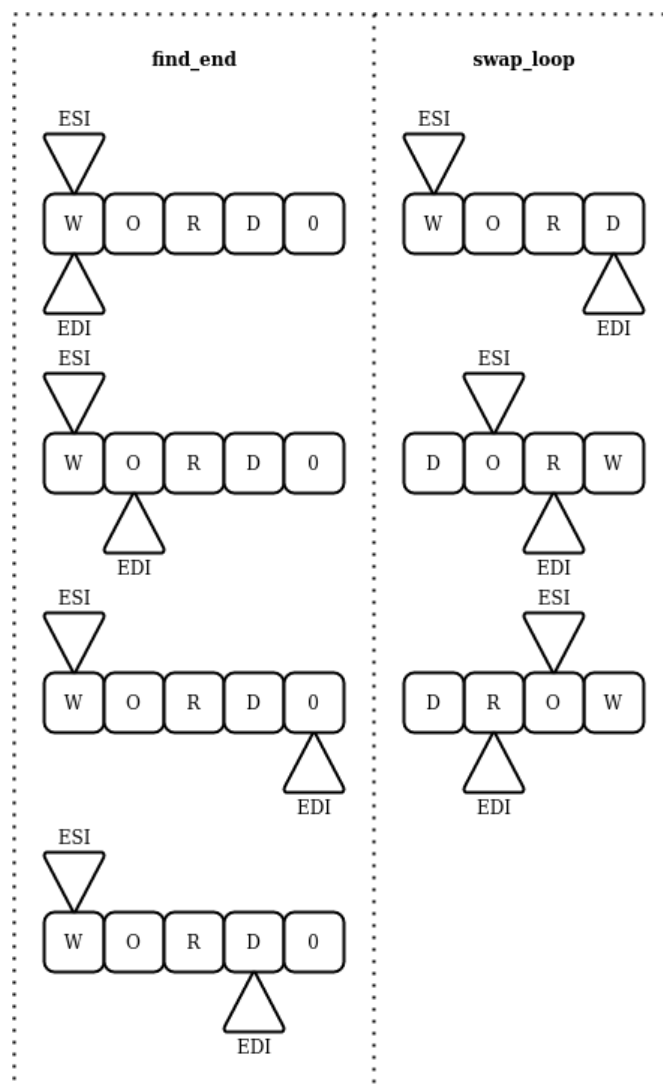
18      cmp al, 'Z'          ; above "Z"?
19      ja L2
20      or BYTE PTR [esi], 00100000b    ; convert the char
21
22 L2:
23     inc esi              ; next char
24     jmp L1
25
26 L3:
27     ret
28 Str_lcase ENDP
29
30 ; #####
31 Str_reverse PROC USES esi edi,       ; str reverse function
32     pString:PTR BYTE
33
34     mov esi, pString        ; both esi & edi point to the start of a string
35     mov edi, pString
36     dec edi
37 find_end:                    ; finds the end index of a string
38     inc edi                ; advance end pointer.
39     mov al, [edi]           ; look for end of string.
40     cmp al, 0               ; if char is null term then
41     jnz find_end            ; false: move to next char      5 ms elapsed
42     dec edi                 ; true: --edi to last char
43
44 swap_loop:
45     cmp esi, edi            ; if start >= end, then we are finished.
46     jge finish
47
48     mov bl, [esi]           ; swap characters.
49     mov al, [edi]
50     mov [esi], al

```

Коли **edi** на кінці рядка — починаємо цикл заміни літер місцями, який фактично буде еквівалентним наступному псевдокоду:

```
while (esi < edi):  
    swap(*esi, *edi);  
    esi++;  
    edi--;
```

Тобто поки не відбулося “перетинання” вказівників, виконується перестановка літер місцями і збільшення/зменшення відповідних вказівників на одиницю. Це проілюстровано нижче:

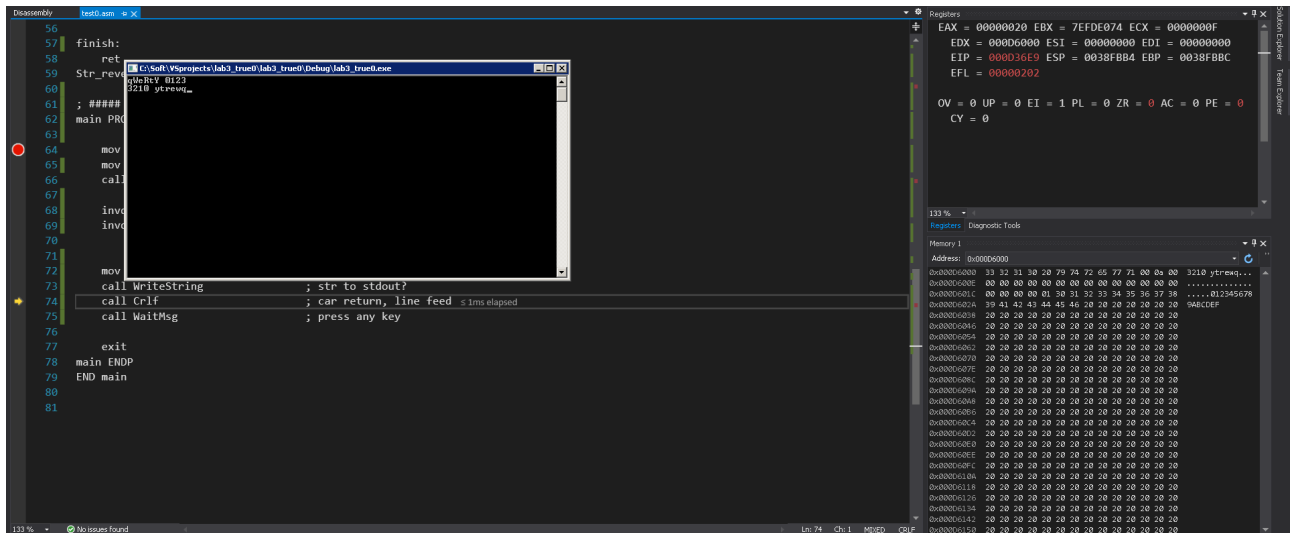


```
27 ret
28 Str_lcase ENDP
29
30 ; #####
31 Str_reverse PROC USES esi edi, ; str reverse function
32 pString:PTR BYTE
33
34 mov esi, pString ; both esi & edi point to the start of a string
35 mov edi, pString
36 dec edi
37 find_end: ; finds the end index of a string
38 inc edi ; advance end pointer.
39 mov al, [edi] ; look for end of string.
40 cmp al, 0 ; if char is null term then
41 jnz find_end ; false: move to next char
42 dec edi ; true: --edi to last char
43
44 swap_loop:
45 cmp esi, edi ; if start >= end, then we are finished.
46 jge finish
47
48 mov bl, [esi] ; swap characters.
49 mov al, [edi]
50 mov [esi], al < 1ms elapsed
51 mov [edi], bl
52
53 inc esi ; move pointers toward each other
54 dec edi
55 jmp swap_loop
56
57 finish:
58 ret
59 Str_reverse ENDP
```

```
27 ret
28 Str_lcase ENDP
29
30 ; #####
31 Str_reverse PROC USES esi edi, ; str reverse function
32 pString:PTR BYTE
33
34 mov esi, pString ; both esi & edi point to the start of a string
35 mov edi, pString
36 dec edi
37 find_end: ; finds the end index of a string
38 inc edi ; advance end pointer.
39 mov al, [edi] ; look for end of string.
40 cmp al, 0 ; if char is null term then
41 jnz find_end ; false: move to next char
42 dec edi ; true: --edi to last char
43
44 swap_loop:
45 cmp esi, edi ; if start >= end, then we are finished.
46 jge finish
47
48 mov bl, [esi] ; swap characters.
49 mov al, [edi]
50 mov [esi], al
51 mov [edi], bl
52
53 inc esi ; move pointers toward each other < 1ms elapsed
54 dec edi
55 jmp swap_loop
56
57 finish:
58 ret
59 Str_reverse ENDP
```

Бачимо, що перший й останній символи помінялись місцями.

Далі, після виконання процедури *Str_reverse*, для того щоб вивести в консольне вікно отриманий рядок скористаємося процедурою *WriteString* з *Irvine32*, попередньо завантаживши в регістр **edx** зміщення, що вказує на перший символ рядка.



The screenshot shows a debugger window with assembly code on the left and registers/memory on the right. The assembly code is as follows:

```
56 finish:
57 ret
58 Str_rev
59 ; #####
60 ; #####
61 ; #####
62 main PROC
63
64 mov     eax, 0
65 mov     ebx, 0
66 call    Str_rev
67
68 inv     eax, 0
69 inv     ebx, 0
70
71 mov     ecx, 0
72
73 call    WriteString      ; str to stdout?
74 call    CrLf             ; car return, line feed <ims elapsed
75 call    WaitMsg          ; press any key
76
77 exit
78 main ENDP
79 END main
80
81
```

The registers window on the right shows the following values:

Register	Value
EAX	00000020
EBX	7EFD074
ECX	0000000F
EDX	000D6000
ESI	00000000
EDI	00000000
EIP	000D36E9
ESP	0038FB84
EBP	0038FBBC
EFL	00000202

The memory window shows the address 00000000 and the value 00000000.

Отримуємо наш перетворений рядок, виведений у зворотньому порядку.

Висновок

У процесі виконання даної лабораторної роботи було створено програму, що реалізує прості операції введення/виведення за допомогою процедур описаних в бібліотеці *Irvine32* та прості операції перетворення символьних рядків, а саме заміну певних символів на інші та перестановку символів у рядку місцями.

В результаті виконання вдалося ознайомитися з базовими процедурами наявними у бібліотеці, методами адресації даних, командами порівняння та переходів.

Вихідний код програм доступний за посиланням:

[\[===\]](#)

Повний вихідний код програми:

```
INCLUDE Irvine32.inc
```

```
.data
```

```
buffer BYTE 16 DUP(?)           ; main buffer for strings
```

```
.code
```

```
Str_lcase PROC USES eax esi,      ; reworked Str_ucase from Irvine32  
    pString:PTR BYTE
```

```
    mov esi, pString
```

```
L1:
```

```
    mov al, [esi]                ; get char
```

```
    cmp al, 0                    ; end of string?
```

```
    je L3                       ; if yes then quit
```

```
    cmp al, 'A'                  ; below "A"?
```

```
    jb L2
```

```
    cmp al, 'Z'                  ; above "Z"?
```

```
    ja L2
```

```
    or BYTE PTR [esi], 00100000b ; convert the char
```

```
L2:
```

```
    inc esi                      ; next char
```

```
    jmp L1
```

```
L3:
```

```
    ret
```

```
Str_lcase ENDP
```


Str_reverse PROC USES esi edi, ; str reverse function

pString:PTR BYTE

mov esi, pString ; both esi & edi point to the start of a string

mov edi, pString

dec edi

find_end: ; finds the end index of a string

inc edi ; advance end pointer.

mov al, [edi] ; look for end of string.

cmp al, 0 ; if char is null term then

jnz find_end ; false: move to next char

dec edi ; true: --edi to last char

swap_loop:

cmp esi, edi ; if start >= end, then we are finished.

jge finish

mov bl, [esi] ; swap characters.

mov al, [edi]

mov [esi], al

mov [edi], bl

inc esi ; move pointers toward each other

dec edi

jmp swap_loop

finish:

ret

Str_reverse ENDP

```

main PROC                                ; MAIN FUNC

                                ; str input

    mov edx, OFFSET buffer              ; str addr offset to edx
    mov ecx, (SIZEOF buffer) - 1; str buffer size to ecx
    call ReadString                     ; reading string from stdin


    invoke Str_lcase, ADDR buffer  ; str to lowercase
    invoke Str_reverse, ADDR buffer ; str reversing


                                ; str out

    mov edx, OFFSET buffer              ; str offset to edx
    call WriteString                    ; str to stdout?
    call Crlf                          ; car return, line feed
    call WaitMsg                       ; press any key


    exit
main ENDP
END main

```