UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA MATEMATICĂ ŞI INFORMATICĂ
DEPARTAMENTUL INFORMATICĂ

CEMÎRTAN CRISTIAN

Lucrarea individuală nr. 9

la disciplina ***Arhitectura Calculatoarelor și Limbaje de Asamblare***

Coordonator: Sturza Greta, lector universitar

Chişinău, 2021

Cuprins

Sarcină

Fie dat un șir simbolic. Să se elaboreze un program assembler ce găsește poziția ultimei virgule și să se înlocuiască cu primul simbol din șir diferit de spațiu liber.

Cod sursă

```
COMMENT *
      Lucrare individuala nr. 9, varianta complexa nr. 3
      Copyright Cemirtan Cristian 2021
      Grupa I 2101
*

INCLUDE stdlibc.inc

.MODEL small
.STACK

crlf EQU 0Dh, 0Ah
len EQU 20

.DATA
txt DB 'Introduceti sirul:', 0
fmt_ CATSTR <'%>, %len, <c%n', 0> ; '%20c%n'
fmt1 DB fmt_
fmt2 DB 'Sirul modificat:', crlf, '%.*s', 0
sir DB len DUP (?)
n DW ?

.CODE
; initializare
      .STARTUP
      mov es, dx

; afisare text
      lea si, txt
      call puts

; citire argument
      lea di, sir

      push OFFSET n OFFSET sir
      lea si, fmt1
      call scanf
      add sp, 2

; incarcam lungimea sirului
      mov cx, n

; cautam primul caracter in afara de spatiu
      cld
```

```
      mov al, ' '

      repe scasb
      jcxz iesire_err

      mov dl, [di - 1]

; gasim pozitia ultimei virgula
      std
      mov al, ','
      mov cx, n

      mov di, OFFSET sir - 1
      add di, cx

      repne scasb
      jcxz iesire_err

      mov [di + 1], dl

; afisare sirul modificat
      push OFFSET sir n
      lea si, fmt2
      call printf
      add sp, 2

; iesire cu succes
      .EXIT 0

iesire_err:
      .EXIT 1
END
```

Rezultat

```
E:\>tasm i9
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International

Assembling file:   i9.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  449k


E:\>tlink i9 stdlibc
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International

E:\>i9
Introduceti sirul:
  individ,9,complex
Sirul modificat:
  individ,9icomplex
E:\>
```

Figura 1. Rezultatul afișat la ecran.

Concluzie

Pe parcursul realizării a programului, am utilizat biblioteca actualizată `stdlibc`, care substituie pe cea precedentă – `stdio`. Biblioteca `stdlibc` introduce ajustări fine a procedurilor existente și implementări adiționale a procedurilor și a macro-definițiilor.

O funcționalitate recentă a procedurii `printf`, menționată în codul sursă, este precizia de afișare, notată cu specificatorul `'%.'`, urmat de un asterisc (`'*'`) sau un număr constant. În deosebire, asteriscul încarcă un argument adițional ca valoarea de precizie.

Precizia de afișare, utilizată cu șiruri ASCII, trunchiază copia șirului, dacă lungimea sa depășește valoarea de precizie.

Se anexează coduri sursă a bibliotecii `stdlibc`.

stdlibc_macro.inc

```
COMMENT *
      Lucrare individuala nr. 9, varianta complexa nr. 3
      Copyright Cemirtan Cristian 2021
      Grupa I 2101
*

INCLUDE stdlibc.inc

.MODEL small
.STACK

crlf EQU 0Dh, 0Ah
len EQU 20

.DATA
txt DB 'Introduceti sirul:', 0
fmt_ CATSTR <'%>, %len, <c%n', 0>
fmt1 DB fmt_
fmt2 DB 'Sirul modificat:', crlf, '%.*s', 0
sir DB len DUP (?)
n DW ?

.CODE
; initializare
      .STARTUP
      mov es, dx

; afisare text
      lea si, txt
      call puts

; citire argument
      lea di, sir

      push OFFSET n OFFSET sir
      lea si, fmt1
      call scanf
      add sp, 2

; incarcam lungimea sirului
      mov cx, n

; cautam primul caracter in afara de spatiu
      cld
      mov al, ' '

      repe scasb
      jcxz iesire_err

      mov dl, [di - 1]

; gasim pozitia ultimei virgula
```

```
      std
      mov al, ','
      mov cx, n

      mov di, OFFSET sir - 1
      add di, cx

      repne scasb
      jcxz iesire_err

      mov [di + 1], dl

; afisare sirul modificat
      push OFFSET sir n
      lea si, fmt2
      call printf
      add sp, 2

; iesire cu succes
      .EXIT 0

iesire_err:
      .EXIT 1
END
```

stdlibc.inc

```
COMMENT *
      Antet pentru biblioteca stdlibc
      !Implementata partiala!
      Copyright Cemirtan Cristian 2021
*

INCLUDE stdlibc_macro.inc

EXTRN digitcount : PROC
EXTRN udigitcount : PROC
EXTRN wtos : PROC
EXTRN uwtos : PROC
EXTRN atoi : PROC
EXTRN strcpy : PROC
EXTRN strncpy : PROC
EXTRN memcpy : PROC
EXTRN memset : PROC
EXTRN strlen : PROC
EXTRN printf : PROC
EXTRN sprintf : PROC
EXTRN scanf : PROC
EXTRN puts : PROC
EXTRN gets : PROC
EXTRN getchar : PROC
```

stdlibc.asm

```
COMMENT *
      Biblioteca stdlibc
      pentru DOS i386
      !Implementata partiala!
      Copyright Cemirtan Cristian 2021
*

INCLUDE stdlibc_macro.inc

.MODEL tiny
.386
buffer_size EQU 255

; se incarca adresa parametrului
arg_sca MACRO r
      mov r, ss:[bx]
      add bx, 2
ENDM arg_sca

; se incarca in valoarea parametrului
; comportament nedorit daca r este di
; registrul di se distruge
arg_mov MACRO r
      mov di, ss:[bx]
      mov [di], r
      add bx, 2
ENDM arg_sca

.CODE
PUBLIC digitcount
PUBLIC udigitcount
PUBLIC itoa
PUBLIC uitoa
PUBLIC atoi
PUBLIC strcpy
PUBLIC strncpy
PUBLIC memcpy
PUBLIC memset
PUBLIC strlen
PUBLIC sprintf
PUBLIC printf
PUBLIC scanf
PUBLIC puts
PUBLIC gets
PUBLIC getchar

; nr. de cifre a unui numar cu semn
; intrare ax - numar
; iesire ax - nr. de cifre
digitcount PROC
      test ax, ax
      jns udigitcount
      neg ax
```

```
      jmp udigitcount
digitcount ENDP

; nr. de cifre a unui numar fara semn
; intrare ax - numar
; iesire ax - nr. de cifre
udigitcount PROC
      cmp ax, 10
      jae udigitcount$if_1

      mov ax, 1
      ret

udigitcount$if_1:
      cmp ax, 100
      jae udigitcount$if_2

      mov ax, 2
      ret

udigitcount$if_2:
      cmp ax, 1000
      jae udigitcount$if_3

      mov ax, 3
      ret

udigitcount$if_3:
      cmp ax, 10000
      jae udigitcount$finish

      mov ax, 4
      ret

udigitcount$finish:
      mov ax, 5
      ret
udigitcount ENDP

; intrare ax - numar, di - sir (lungimea de cel putin 7 octeti)
; iesire ax - nr. de caractere scrise, exclusiv 0
itoa PROC
      push ecx edx di

      test ax, ax
      jns itoa$res

      mov BYTE PTR [di], '-'
      neg ax
      inc di

; initializare
itoa$res:
      movzx edx, ax
      call digitcount
```

```
itoa$init:
      add di, ax
      mov eax, edx

      mov BYTE PTR [di], 0
      push di

      itoa$loop:
      ; salvare eax
            mov edx, eax

      ; impartire la 10
            imul eax, 0CCCDh
            shr eax, 19

      ; inmultirea la 10 si aflarea restului
            lea ecx, [eax + 4 * eax]
            lea ecx, [2 * ecx - '0']
            sub edx, ecx

            dec di
            mov [di], dl

            test eax, eax
            jnz itoa$loop

itoa$finish:
      pop ax di edx ecx
      sub ax, di
      ret
itoa ENDP

; intrare ax - numar, di - sir (lungimea de cel putin 7 octeti)
; iesire ax - nr. de caractere scrise, exclusiv 0
uitoa PROC
      push ecx edx di
      movzx edx, ax
      call udigitcount
      jmp itoa$init
uitoa ENDP

; intrare si - sir
; iesire ax - numar
atoi PROC
      push ebx edx si eax

; initializare
      xor eax, eax
      xor ebx, ebx
      xor edx, edx

; testare daca e negativ
      cmp BYTE PTR [si], '-'
      sete dl
      add si, dx
```

```
	atoi$loop:
		movzx ebx, BYTE PTR [si]

	; verificare daca este o cifra
		mov dh, bl
		xor dh, 30h

		cmp dh, 9
		jg atoi$finish

	; * 10
		lea eax, [eax + 4 * eax]
		lea eax, [ebx + 2 * eax - '0']

		inc si
		jmp atoi$loop

atoi$finish:
; negatie daca minus
	test dl, dl
	jz atoi$finish_2
	neg eax

atoi$finish_2:
; ultimele 16 octeti a reg. eax sa fie recuperate
	mov dx, ax
	pop eax
	mov ax, dx

	pop si edx ebx
	ret
atoi ENDP

; intrare si - sursa, di - destinatie
; iesire ax - nr. de caractere copiate
strcpy PROC
	push bx dx

	xor bx, bx

	mov dl, [si]
	test dl, dl ; se verifica daca e terminator
	jz strcpy$finish

	strcpy$loop:
		mov [bx + di], dl

		inc bx
		mov dl, [bx + si]

		test dl, dl
		jnz strcpy$loop

strcpy$finish:
```

```
	mov BYTE PTR [bx + di], 0
	mov ax, bx
	pop dx bx
	ret
strcpy ENDP

; intrare ax - nr. max. de caractere de copiat, si - sursa, di - destinatie
; iesire ax - nr. de caractere copiate
strncpy PROC
	test ax, ax
	jnz strncpy$begin

	mov BYTE PTR [di], al ; se pune terminator
	ret

strncpy$begin:
	push bx dx

	xor bx, bx

	mov dl, [si]
	test dl, dl
	jz strncpy$finish

	strncpy$loop:
		mov [bx + di], dl

		inc bx
		mov dl, [bx + si]

		test dl, dl
		jz strncpy$finish

		cmp bx, ax
		jb strncpy$loop

strncpy$finish:
	mov BYTE PTR [bx + di], 0
	mov ax, bx
	pop dx bx
	ret
strncpy ENDP

; intrare ax - nr. de caractere de copiat, si - sursa, di - destinatie
; iesire nimic
memcpy PROC
	test ax, ax
	jz memcpy$ret

	push bx edx

	xor bx, bx

	test ax, 1 ; se verifica daca e par
	jz memcpy$word
```

```
	mov dl, [si] ; 1 octet
	mov [di], dl

	inc bx
	cmp bx, ax
	je memcpy$finish

memcpy$word:
; se aliniaza datele pentru a reduce nr. de cicluri
	test ax, 2 ; se verifica daca modulo cu 3 e 0
	jz memcpy$loop

	mov dx, [bx + si] ; 2 octeti
	mov [bx + di], dx

	add bx, 2
	cmp bx, ax
	je memcpy$finish

	memcpy$loop:
		mov edx, [bx + si] ; 4 octeti
		mov [bx + di], edx

		add bx, 4
		cmp bx, ax
		jne memcpy$loop

memcpy$finish:
	pop edx bx

memcpy$ret:
	ret
memcpy ENDP

; intrare ax - nr. de caractere de copiat, dl - caracter, di - destinatie
; iesire nimic
memset PROC
	test ax, ax
	jz memset$ret

	push bx edx

; populeaza toti octetii din edx cu dl
	mov dh, dl
	mov bx, dx
	shl edx, 16
	or dx, bx

	xor bx, bx

	test ax, 1
	jz memset$word

	mov [di], dl
```

```
	inc bx
	cmp bx, ax
	je memset$finish

memset$word:
	test ax, 2
	jz memset$loop

	mov [bx + di], dx

	add bx, 2
	cmp bx, ax
	je memset$finish

	memset$loop:
		mov [bx + di], edx

		add bx, 4
		cmp bx, ax
		jne memset$loop

memset$finish:
	pop edx bx

memset$ret:
	ret
memset ENDP

; intrare si - sursa
; iesire ax - lungimea sirului
strlen PROC
	xor ax, ax
	cmp [si], al
	jz strlen$finishz

	push bx
	xor bx, bx

	strlen$loop:
		inc bx
		cmp BYTE PTR [bx + si], 0
		jnz strlen$loop

strlen$finish:
	mov ax, bx
	pop bx
	ret

strlen$finishz:
	ret
strlen ENDP

; intrare si - sursa, di - destinatie, bx - varful stivei cu argumente
; iesire ax - nr. de caractere scrise, in afara de 0
```

```
sprintf_base:
      push cx dx si di bp

      mov bp, sp ; se face un cadru de stiva
      sub sp, 4 ; rezervez spatiu pentru o variabila temporara

      sprintf_base$loop:
            mov dl, [si]
            inc si

            test dl, dl
            jz sprintf_base$ret

            cmp dl, '%'
            je sprintf_base$fmt

      sprintf_base$putc:
            mov [di], dl
            inc di
            jmp sprintf_base$loop

            sprintf_base$fmt:
                  COMMENT *
                        Flaguri in ah:
                        0 - Este specificata manual precizia
                        1 - Fara semn
                        2 - Prefixare la numar fara semn (sau pozitiv daca
cu semn)
                        3 - Dimensiunea datei fortata la 1 octet
                        ... - rezervate
                  *

                  xor ax, ax

            sprintf_base$fmt_read:
                  mov dl, [si]
                  inc si

                  cmp dl, '%'
                  je sprintf_base$putc

                  cmp dl, 'c'
                  je sprintf_base$fmt_char

                  cmp dl, 'i'
                  je sprintf_base$fmt_decimal ; nu sunt implementate %x, %o

                  cmp dl, 'd'
                  je sprintf_base$fmt_decimal

                  cmp dl, 'u'
                  je sprintf_base$fmt_unsigned

                  cmp dl, 's'
                  je sprintf_base$fmt_string
```

```
      cmp dl, 'h'
      je sprintf_base$fmt_short

      cmp dl, '+'
      je sprintf_base$fmt_prefix_sign

      cmp dl, ' '
      je sprintf_base$fmt_prefix_space

      cmp dl, '.'
      je sprintf_base$fmt_precision

      cmp dl, 'n'
      jne sprintf_base$ret

sprintf_base$fmt_count:
      mov [bp - 2], si
      arg_sca si

      mov dx, ax
      mov ax, di
      sub ax, [bp + 2] ; reg. di initiala

      bt dx, 11
      jc sprintf_base$fmt_count_writebyte

      mov [si], ax
      mov si, [bp - 2]

      sprintf_base$fmt_count_writebyte:
            mov [si], al
            mov si, [bp - 2]

      jmp sprintf_base$loop

sprintf_base$fmt_char:
      arg_sca dl
      jmp sprintf_base$putc

sprintf_base$fmt_decimal:
      arg_sca dx

      bt ax, 11
      jnc sprintf_base$fmt_decimal_prflag

      xor dh, dh

sprintf_base$fmt_decimal_prflag:
      bt ax, 8
      jnc sprintf_base$fmt_decimal_prefix

      mov [bp - 2], ax
```

```
            ; daca ambele numere si precizia sunt 0, atunci nu se afiseaza
nimic
                  test dx, dx
                  setz ah

                  test cx, cx
                  setz al

                  cmp ax, 0101h
                  je sprintf_base$loop

                  mov ax, dx
                  call digitcount

                  sub cx, ax
                  jle sprintf_base$fmt_decimal_prefix

                  sprintf_base$fmt_decimal_precision_loop:
                        mov BYTE PTR [di], '0'
                        inc di
                        dec cx
                        jnz sprintf_base$fmt_decimal_precision_loop

                  mov ax, [bp - 2]

                  sprintf_base$fmt_decimal_prefix:
                        bt ax, 10
                        jnc sprintf_base$fmt_decimal_conv

                        test dx, dx
                        js sprintf_base$fmt_decimal_conv

                        mov al, [bp - 4]
                        mov [di], al
                        inc di

                  sprintf_base$fmt_decimal_conv:
                        bt ax, 9
                        jnc sprintf_base$fmt_decimal_conv_signed

                        mov ax, dx
                        call uitoa
                        add di, ax

                        jmp sprintf_base$loop

                  sprintf_base$fmt_decimal_conv_signed:
                        mov ax, dx
                        call itoa
                        add di, ax

                  jmp sprintf_base$loop

            sprintf_base$fmt_string:
```

```
	mov dx, si
	arg_sca si

	bt ax, 8
	jc sprintf_base$fmt_string_len

	call strcpy
	jmp sprintf_base$fmt_string_finish

	sprintf_base$fmt_string_len:
		mov ax, cx
		call strncpy

	sprintf_base$fmt_string_finish:
		add di, ax
		mov si, dx

	jmp sprintf_base$loop

sprintf_base$fmt_unsigned:
	bts ax, 9
	jmp sprintf_base$fmt_decimal

sprintf_base$fmt_precision:
	bts ax, 8

	mov dl, [si]
	inc si

	cmp dl, '*'
	je sprintf_base$fmt_precision_arg

	xor dl, 30h
	cmp dl, 9
	ja sprintf_base$ret

	dec si

	mov dx, ax

	call atoi
	mov cx, ax

	call udigitcount
	add si, ax

	mov ax, dx
	jmp sprintf_base$fmt_read

sprintf_base$fmt_short:
	bts ax, 11
	jmp sprintf_base$fmt_read

sprintf_base$fmt_prefix_sign:
```

```
                bts ax, 10
                mov BYTE PTR [bp - 4], '+'
                jmp sprintf_base$fmt_read

            sprintf_base$fmt_prefix_space:
                bts ax, 10
                mov BYTE PTR [bp - 4], ' '
                jmp sprintf_base$fmt_read

            sprintf_base$fmt_precision_arg:
                arg_sca cx
                jmp sprintf_base$fmt_read

sprintf_base$ret:
      mov BYTE PTR [di], 0

      mov sp, bp ; se elibereaza cadrul
      pop bp

      mov ax, di
      pop di
      sub ax, di ; nr. de caractere scrise

      pop si dx cx
      ret

; intrare si - sursa, di - destinatie
; iesire ax - nr. de caractere citite
sprintf PROC
      push bx

; catre ultimul parametrul explicit dat pe stiva, inaintea copiei reg. ip
      mov bx, sp
      add bx, 4

      call sprintf_base

      pop bx
      ret
sprintf ENDP

; intrare si - sursa
; iesire ax - nr. de caractere citite
printf PROC
      push bx

      mov bx, sp
      add bx, 4

      push di bp

      mov bp, sp
      sub sp, buffer_size

      mov di, sp
```

```
	call sprintf_base

	push ax cx dx

; se afiseaza la ecran
	mov cx, ax
	mov ah, 40h
	mov bx, 1
	mov dx, di
	int 21h

	pop dx cx ax

	mov sp, bp

	pop bp di bx
	ret
printf ENDP

; intrare si - sursa, bx - varful stivei cu argumente
; iesire ax - nr. de argumente citite
scanf PROC
	push bx

	mov bx, sp
	add bx, 4

	push cx dx si di

	xor cx, cx

	push bp
	mov bp, sp ; cadru nou
	push cx ; variabila ce contine nr. de argumente citite

	scanf$loop:
		mov dl, [si]
		inc si

		test dl, dl
		jz scanf$ret

		cmp dl, '%'
		je scanf$fmt

		mov ax, scanf_lastchar

		isspace dl
		jnc scanf$consume

		scanf$consume_space:
		; se consuma spatiile goale
			isspace al
			jnc scanf$loop
```

```
            inc ah
            movzx di, ah
            add di, OFFSET scanf_buffer + 2

            mov al, [di]

            mov scanf_lastchar, ax
            jmp scanf$consume_space

      scanf$consume:
      ; se consuma caracterul daca coincide cu dl
      ; altfel iese din procedura
            cmp al, dl
            jne scanf$ret

            inc ah
            movzx di, ah
            add di, OFFSET scanf_buffer + 2

            mov al, [di]

            mov scanf_lastchar, ax
            jmp scanf$loop

      scanf$fmt:
            COMMENT *
                  Flaguri in ah:
                  0 - Rezultatul nu se pastreaza in adresa
argumentului
                  1 - Are loc argumentul %c
                  2 - Este specificat manual nr. maxim de caractere
                  3 - Dimensiunea datei fortata la 1 octet
                  4 - Fara semn
                  5 - Se citeste numar intreg
                  ... - rezervate
            *

            xor ax, ax
            inc WORD PTR [bp - 2]

      scanf$fmt_read:
            mov dl, [si]
            inc si

            cmp dl, 'c'
            je scanf$fmt_char

            cmp dl, 'i'
            je scanf$fmt_decimal ; nu sunt implementate %x, %o

            cmp dl, 'd'
            je scanf$fmt_decimal

            cmp dl, 'u'
```

```
        je scanf$fmt_unsigned

        cmp dl, 's'
        je scanf$fmt_string

        cmp dl, '*'
        je scanf$fmt_ignore

        cmp dl, 'h'
        je scanf$fmt_short

        cmp dl, 'n'
        je scanf$fmt_count

        xor dl, 30h
        cmp dl, 9
        jbe scanf$fmt_maxlen

        dec WORD PTR [bp - 2]
        jmp scanf$ret

scanf$fmt_char:
        bts ax, 9
        call stdin_tokenizer

        add cl, al
        adc ch, 0

        bt ax, 8
        jc scanf$loop

        push si

        mov dx, di

        arg_sca si

        mov di, si
        mov si, dx

        xor ah, ah
        call memcpy

        pop si

        jmp scanf$loop

scanf$fmt_decimal:
        dec WORD PTR [bp - 2]

        bts ax, 13
        call stdin_tokenizer
```

```
        test al, al
        jz scanf$ret

        cmp BYTE PTR [di], '-'
        jne scanf$fmt_decimal_cnv

        mov dl, [di + 1]
        xor dl, 30h
        cmp dl, 9
        ja scanf$ret

scanf$fmt_decimal_cnv:
        inc WORD PTR [bp - 2]

        add cl, al
        adc ch, 0

        mov dx, si
        mov si, di
        mov di, ax

        call atoi

        mov si, dx

        bt di, 8
        jc scanf$loop

        bt di, 11
        jc scanf$fmt_decimal_shortarg

        arg_mov ax
        jmp scanf$loop

scanf$fmt_decimal_shortarg:
        arg_mov al
        jmp scanf$loop

scanf$fmt_string:
        call stdin_tokenizer

        add cl, al
        adc ch, 0

        bt ax, 8
        jc scanf$loop

        mov dx, si

        mov si, di
        arg_sca di

        call strcpy
```

```
                mov si, dx

                jmp scanf$loop

        scanf$fmt_maxlen:
                bts ax, 10
                dec si

                mov dh, ah

                call atoi
                mov di, ax

                call udigitcount
                add si, ax

                mov ax, di
                mov ah, dh

                jmp scanf$fmt_read

        scanf$fmt_ignore:
                bts ax, 8
                jmp scanf$fmt_read

        scanf$fmt_short:
                bts ax, 11
                jmp scanf$fmt_read

        scanf$fmt_unsigned:
                bts ax, 12
                jmp scanf$fmt_decimal

        scanf$fmt_count:
                bt ax, 11
                jc scanf$fmt_count_shortarg

                arg_mov cx
                jmp scanf$loop

        scanf$fmt_count_shortarg:
                arg_mov cl
                jmp scanf$loop

scanf$ret:
        mov ax, [bp - 2]

        mov sp, bp
        pop bp di si dx cx bx
        ret

; ax - masca + lungimea sirului cerut
; al - caractere citite, di - inceputul sirului
stdin_tokenizer:
```

```
	push bx dx cx

	lea di, scanf_buffer + 2
	mov cx, scanf_lastchar

	test cl, cl
	jz stdin_tokenizer$read

	movzx bx, ch
	add di, bx

	mov [di], cl
	jmp stdin_tokenizer$trim

stdin_tokenizer$read:
	sub di, 2
	mov cx, ax

	mov ah, 0Ah
	mov dx, di
	int 21h

	mov dl, ah
	mov ah, 2
	int 21h

	movzx bx, BYTE PTR [di + 1]
	add di, 2
	mov BYTE PTR [bx + di], 0

	mov ax, cx
	mov cl, [di]

	test cl, cl
	jz stdin_tokenizer$read

stdin_tokenizer$trim:
	xor bx, bx

	bt ax, 9
	jc stdin_tokenizer$char

	isspace cl
	jnc stdin_tokenizer$string_len

	stdin_tokenizer$trim_loop:
		inc di
		mov cl, [di]

		isspace cl
		jc stdin_tokenizer$trim_loop

		test cl, cl
		jz stdin_tokenizer$read
```

```
stdin_tokenizer$string_len:
      bt ax, 13
      jc stdin_tokenizer$decimal_signcheck

      bt ax, 10
      jnc stdin_tokenizer$string_loop

      test al, al
      jz stdin_tokenizer$finish

      stdin_tokenizer$string_len_loop:
            inc bx
            mov cl, [bx + di]

            isspace cl
            setc dl

            test cl, cl
            setz dh

            test dx, dx
            jnz stdin_tokenizer$finish

            dec al
            jz stdin_tokenizer$finish

      jmp stdin_tokenizer$string_len_loop

stdin_tokenizer$string_loop:
      inc bx
      mov cl, [bx + di]

      isspace cl
      jc stdin_tokenizer$finish

      test cl, cl
      jnz stdin_tokenizer$string_loop

stdin_tokenizer$finish:
      mov BYTE PTR [bx + di], 0
      mov al, bl

      add bx, di
      sub bx, OFFSET scanf_buffer + 2

      mov ch, bl
      mov scanf_lastchar, cx

      pop cx dx bx
      ret

stdin_tokenizer$char:
      bt ax, 10
      jc stdin_tokenizer$char_len
```

```
	inc bx
	mov cl, [bx + di]
	jmp stdin_tokenizer$finish

stdin_tokenizer$char_len:
	test al, al
	jz stdin_tokenizer$finish

	stdin_tokenizer$char_len_loop:
		inc bx
		mov cl, [bx + di]

		test cl, cl
		setz dl

		dec al
		setz dh

		test dx, dx
		jnz stdin_tokenizer$finish

	jmp stdin_tokenizer$char_len_loop

stdin_tokenizer$decimal_signcheck:
	cmp cl, '-'
	jne stdin_tokenizer$decimal_check

	inc bx
	mov cl, [di + 1]

stdin_tokenizer$decimal_check:
	xor cl, 30h

	cmp cl, 9
	jbe stdin_tokenizer$decimal_len

	xor cl, 30h
	jmp stdin_tokenizer$finish

stdin_tokenizer$decimal_len:
	bt ax, 10
	jnc stdin_tokenizer$decimal_loop

	xor cl, 30h

	test al, al
	jz stdin_tokenizer$finish

	stdin_tokenizer$decimal_len_loop:
		inc bx
		mov cl, [bx + di]

		isspace cl
		setc dl
```

```
            test cl, cl
            setz dh

            test dx, dx
            jnz stdin_tokenizer$finish

            xor cl, 30h
            cmp cl, 9
            seta dl

            dec al
            setz ah

            test dx, dx
            jz stdin_tokenizer$decimal_len_loop

            xor cl, 30h

      jmp stdin_tokenizer$finish

      stdin_tokenizer$decimal_loop:
            inc bx
            mov cl, [bx + di]

            isspace cl
            setc dl

            test cl, cl
            setz dh

            test dx, dx
            jnz stdin_tokenizer$finish

            xor cl, 30h
            cmp cl, 9
            jbe stdin_tokenizer$decimal_loop

            xor cl, 30h
            jmp stdin_tokenizer$finish
scanf ENDP

puts PROC
      push cx si
      lea cx, printf
      lea si, s_fmt
      jmp put_base
puts ENDP

gets PROC
      push cx si
      lea cx, scanf
      lea si, s_fmt
      jmp put_base
gets ENDP
```

```
getchar PROC
      push cx si
      lea cx, scanf
      lea si, c_fmt
      jmp put_base
getchar ENDP

put_base:
      call cx
      pop si cx
      ret

s_fmt DB '%s', 0Dh, 0Ah, 0
c_fmt DB '%c', 0Dh, 0Ah, 0
scanf_lastchar DW 0
scanf_buffer DB buffer_size, buffer_size + 1 DUP (?)
END
```