Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

По домашнему заданию №1

Название : Обработка символьной информации

Дисциплина: Машинно-зависимые языки и основы компиляции

Студент	ИУ-42б		С.В. Астахов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			
		(Подпись, дата)	(И.О. Фамилия)

1 вариант

Москва, 2021

Задание

Дан текст, состоящий из слов, разделенных несколькими пробелами. Определить слова, начинающиеся с гласных букв и заканчивающиеся согласными.

Текст программы:

```
; Template for console application
	.586
	.MODEL  flat, stdcall
	OPTION CASEMAP:NONE
Include kernel32.inc
Include masm32.inc
IncludeLib kernel32.lib
IncludeLib masm32.lib
	.CONST
MsgExit  DB    "Press Enter to Exit",0AH,0DH,0
MsgEmpty  DB    0AH,0DH,"Result is empty",0AH,0DH,0
MsgResult  DB    0AH,0DH,"Result: ", 0
;MsgSeparator  DB    0AH,0DH,"==================",0AH,0DH,0
ReqStr DB 'Input string: ',13,10,0
;MsgGlas DB 'glasnay found',13,10,0
;MsgSogl DB 'soglasnay found',13,10,0
Space DB ' '
Endl db 0,0,0
MsgLn db 13,10,0
	.DATA
log DB ' '
glasStr DB 'AEYUIOJaeyuioj'
soglStr DB 'QWRTPSDFGHKLZXCVBNMqwrtpsdfgklzxcvbnm'
spaceStr DB ' '
;fl_glas byte 0
fl_space byte 0
;fl_inword byte 0
fl_transfer byte 0
fl_loop byte 0
	.DATA?
ESIarch dword 0
EDIarch dword 0
i1 dword 0
i2 dword 0
i3 dword 0
i4 dword 0
i5 dword 0
i6 dword 0
Cnt dword 10
glasN dword 10
inbuf    DB    100 DUP (?)
```

```
str1 DB 100 DUP (?)
str2 DB 100 DUP (?)
str3 DB 100 DUP (?)
str4 DB 100 DUP (?)
str5 DB 100 DUP (?)
str6 DB 100 DUP (?)
letter DB ' '
        .CODE
Start:
;
;    Add you statements
;    code 20 - space
        XOR    EAX,EAX

        Invoke StdOut,ADDR ReqStr
        Invoke StdIn,ADDR str1,LengthOf str1

        mov fl_transfer, 0
        mov i1, 0
        mov i2, 0
        cld
     glas_cycle:
        lea EDI, glasStr
        mov EBX, i1
        mov AL, str1[EBX]
        mov ECX, LengthOf glasStr
        repne scasb
          jne sogl
             ; glas branch
             cmp fl_space, 1
                ; turn on transfer
                mov fl_transfer, 1
             jmp merge1
          sogl:
             mov fl_transfer, 0
          merge1:

          mov fl_space, 0
          cmp fl_transfer,1
          jne no_copy
             copy_cycle:
                mov EBX, i1
                lea ESI, str1[EBX]
                mov EBX, i2
                lea EDI, str2[EBX]
                movsb
                inc i1
                inc i2
                mov EBX, i1
                mov fl_loop, 1
                mov AL, str1[EBX]
```

```
            cmp AL, 0
            jne no_endl_1
               mov fl_loop, 0
            no_endl_1:

            cmp AL, Space
            jne no_space_1
               mov fl_loop, 0
            no_space_1:

         cmp fl_loop, 1
      je copy_cycle
      jmp transfer_merge

  no_copy:
      skip_cycle:
         inc i1
         mov EBX, i1
         mov AL, str1[EBX]
         mov fl_loop, 1 ; skip fl

            cmp AL, 0
            jne no_endl_2
               mov fl_loop, 0
            no_endl_2:

            cmp AL, Space
            jne no_space_2
               mov fl_loop, 0
            no_space_2:

         cmp fl_loop, 1
      je skip_cycle
  transfer_merge:


mov EBX, i2
lea ESI, Space
lea EDI, str2[EBX]
movsb
inc i2


spaces_cycle:
  inc i1
  mov EBX, i1
  mov AL, str1[EBX]
  mov fl_loop, 1
    cmp AL, 0
    jne no_endl_3
        mov fl_loop, 0
```

```
            no_endl_3:

            cmp AL, Space
            je space_branch_1
                mov fl_loop, 0
            space_branch_1:
          cmp fl_loop, 1
         je spaces_cycle

         mov EBX, i1
         mov AL, str1[EBX]
         cmp AL,0
        jne glas_cycle

        ; empty check
         lea EDI, str2
         mov AL, Space
         mov ECX, i2
         repe scasb

     je empty_str2

        ;Invoke StdOut,ADDR MsgSeparator1
        ;Invoke StdOut,ADDR str2

        ; reverse
        mov ECX, i2
        sub ECX, 1
        lea edi, str3
        reverse:
           mov AL, str2[ECX]
           stosb
        loop reverse
        mov AL, str2[0]
        stosb

        ;Invoke StdIn,ADDR inbuf,LengthOf inbuf
        ;Invoke StdOut,ADDR MsgSeparator2
        ;Invoke StdOut,ADDR str3


;sogl on endl handler cycle

mov i3, 0
mov i4, 0
mov fl_transfer, 0

sogl_cycle:
        lea EDI, soglStr
        mov EBX, i3
        mov AL, str3[EBX]
        mov ECX, LengthOf soglStr
```

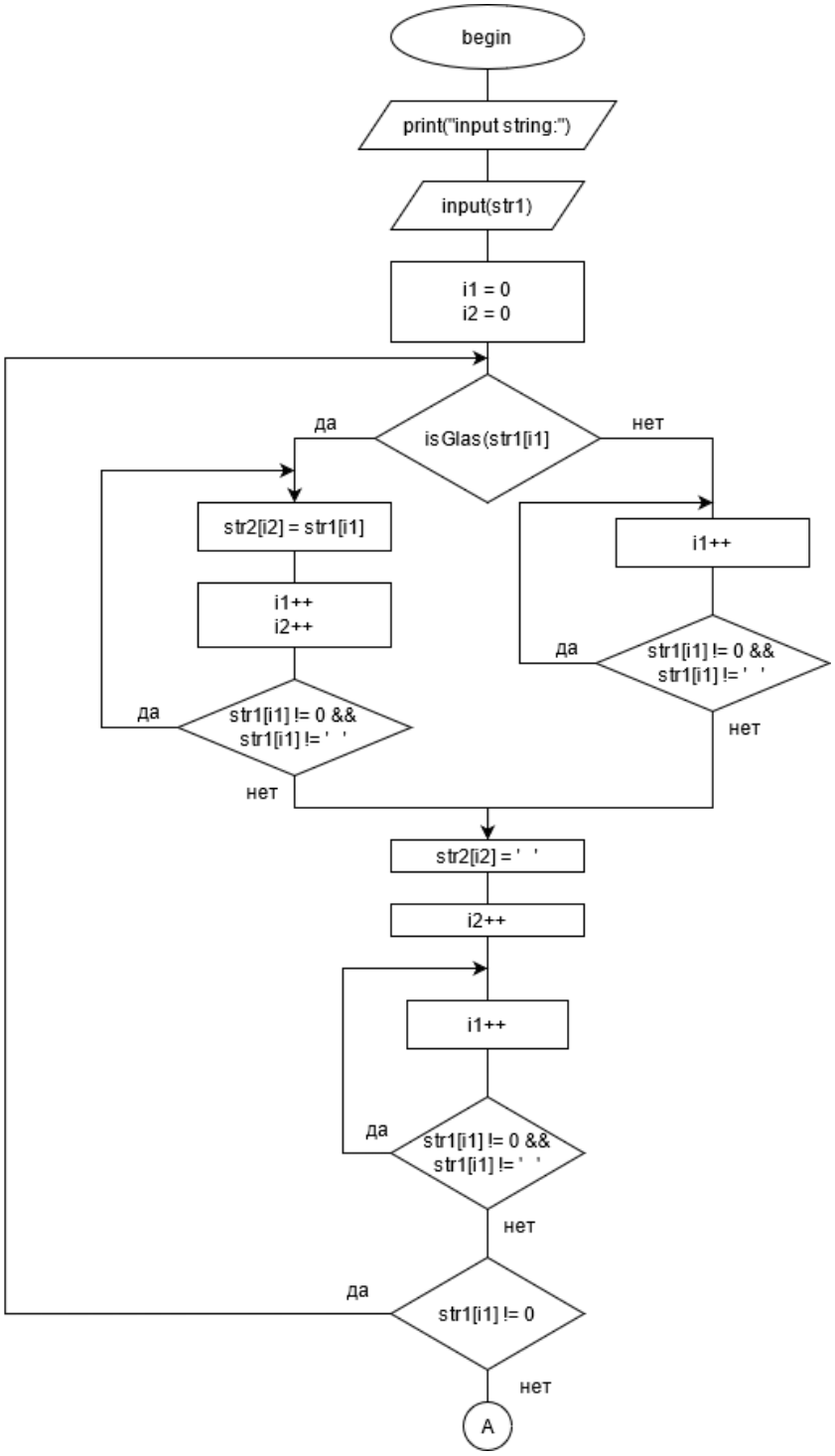
```
repne scasb
  je sogl2
     ; glas branch
     ; turn off transfer
     mov fl_transfer, 0
     jmp merge2
  sogl2:
     ;Invoke StdOut,ADDR MsgTraceSg
     mov fl_transfer, 1
  merge2:

  mov fl_space, 0

  cmp fl_transfer,1
  jne no_copy2
     copy_cycle2:
        mov EBX, i3
        lea ESI, str3[EBX]
        mov EBX, i4
        lea EDI, str4[EBX]
        movsb
        inc i3
        inc i4
        mov EBX, i3
        mov fl_loop, 1
        mov AL, str3[EBX]

           cmp AL, 0
           jne no_endl_4
              mov fl_loop, 0
           no_endl_4:

           cmp AL, Space
           jne no_space_3
              mov fl_loop, 0
           no_space_3:

        cmp fl_loop, 1
     je copy_cycle2
     jmp transfer_merge2

  no_copy2:
     skip_cycle2:
        inc i1
        mov EBX, i3
        mov AL, str3[EBX]

           cmp AL, 0
           jne no_endl_5
              mov fl_loop, 0
```

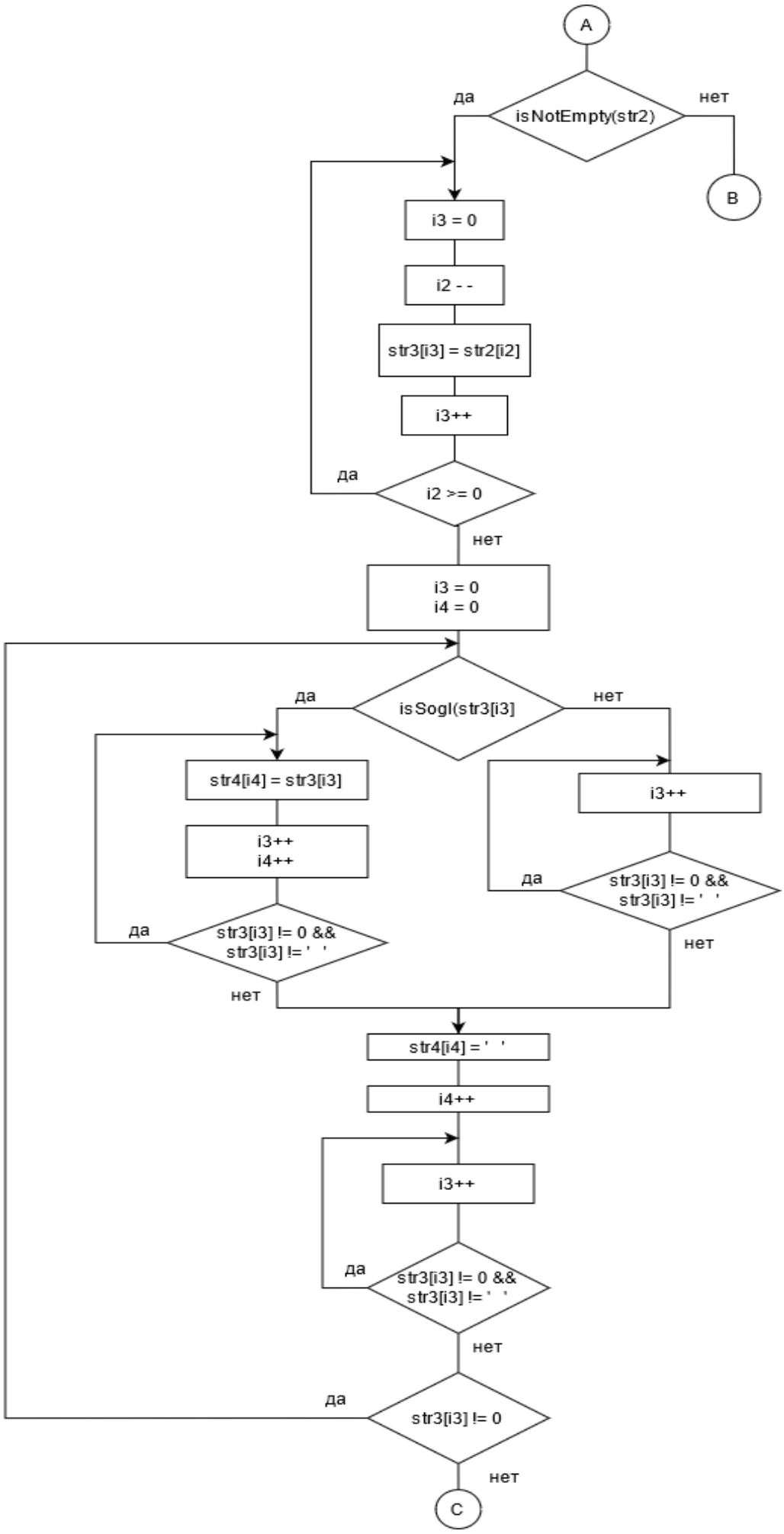
```
            no_endl_5:
            cmp AL, Space
            jne no_space_4
                mov fl_loop, 0
            no_space_4:

        cmp fl_loop, 1
      je skip_cycle2
   transfer_merge2:

  ;skip fl - on end to solve 1st elem problem
  mov fl_loop, 1

  mov EBX, i4
  lea ESI, Space
  lea EDI, str4[EBX]
  movsb
  inc i4


  spaces_cycle2:
    inc i3
    mov EBX, i3
    mov AL, str3[EBX]
    mov fl_loop, 1
      cmp AL, 0
      jne no_endl_6
          mov fl_loop, 0
      no_endl_6:

      cmp AL, Space
      je space_branch_2
          mov fl_loop, 0
      space_branch_2:
    cmp fl_loop, 1
  je spaces_cycle2

  mov EBX, i3
  mov AL, str3[EBX]
  cmp AL,0

jne sogl_cycle

;Invoke StdIn,ADDR inbuf,LengthOf inbuf
;Invoke StdOut,ADDR MsgSeparator3
;Invoke StdOut,ADDR str4

mov ECX, i4
sub ECX, 1
lea edi, str5
reverse2:
   mov AL, str4[ECX]
```

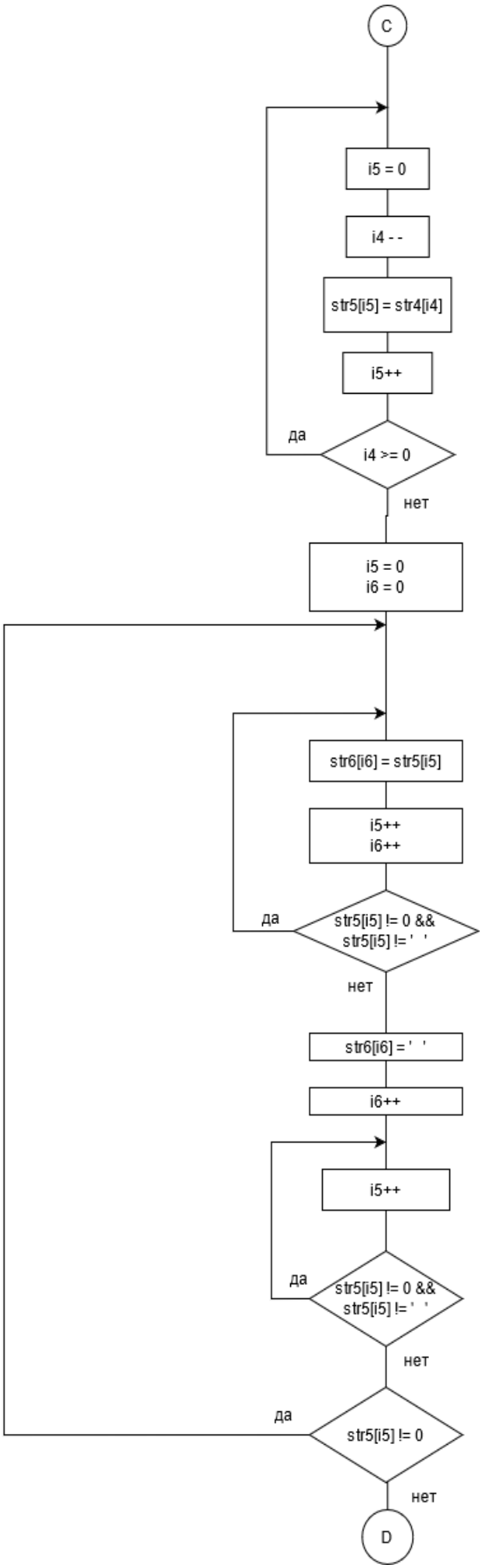
```
    stosb
 loop reverse2
 mov AL, str4[0]
 stosb

 ;Invoke StdIn,ADDR inbuf,LengthOf inbuf
 ;Invoke StdOut,ADDR MsgSeparator4
 ;Invoke StdOut,ADDR str5

 mov EAX, i4
 mov i5, EAX

; removing additional spaces cycle
mov i6, 0
mov i5, 0

big_spaces_cycle:
  lea EDI, glasStr
  mov EBX, i5
  mov AL, str5[EBX]
  ;mov ECX, LengthOf glasStr

    mov fl_space, 0

        copy_cycle3:
           mov EBX, i5
           lea ESI, str5[EBX]
           mov EBX, i6
           lea EDI, str6[EBX]
           movsb
           inc i5
           inc i6
           mov EBX, i5
           mov fl_loop, 1 ;cycle fl
           mov AL, str5[EBX]
              cmp AL, 0
              jne no_endl_7
                 mov fl_loop, 0
              no_endl_7:

              cmp AL, Space
              jne no_space_5
                 mov fl_loop, 0
              no_space_5:

           cmp fl_loop, 1
        je copy_cycle3


   mov EBX, i6
   lea ESI, Space
   lea EDI, str6[EBX]
```

```
    movsb
    inc i6

    spaces_cycle3:
      inc i5
      mov EBX, i5
      mov AL, str5[EBX]
      mov fl_loop, 1
        cmp AL, 0
        jne no_endl_8
            mov fl_loop, 0
        no_endl_8:

        cmp AL, Space
        je space_branch_3
            mov fl_loop, 0
        space_branch_3:
      cmp fl_loop, 1
    je spaces_cycle3

    mov EBX, i5
    mov AL, str5[EBX]
    cmp AL,0

  jne big_spaces_cycle

  ;==============================

    ; empty check
    lea EDI, str6
    mov AL, Space
    mov ECX, i6
    repe scasb
je empty_str6

   ;Invoke StdIn,ADDR inbuf,LengthOf inbuf
   ;Invoke StdOut,ADDR MsgSeparator5
   Invoke StdOut,ADDR MsgResult
   Invoke StdOut,ADDR str6
   Invoke StdOut,ADDR MsgLn
   ;Invoke StdOut,ADDR MsgSeparator
   jmp exit_point
empty_str6:
empty_str2:
   Invoke StdOut,ADDR MsgEmpty
exit_point:
    Invoke StdOut,ADDR MsgExit
    Invoke StdIn,ADDR inbuf,LengthOf inbuf

    Invoke ExitProcess,0
    End    Start
```

Схема алгоритма:




A
да
isNotEmpty(str2)
нет
B
i3 = 0
i2 - -
str3[i3] = str2[i2]
i3++
да
i2 >= 0
нет
i3 = 0
i4 = 0
да
isSogl(str3[i3]
нет
str4[i4] = str3[i3]
i3++
i4++
да
str3[i3] != 0 &&
str3[i3] != ' '
нет
i3++
да
str3[i3] != 0 &&
str3[i3] != ' '
нет
str4[i4] = ' '
i4++
i3++
да
str3[i3] != 0 &&
str3[i3] != ' '
нет
да
str3[i3] != 0
нет
C


C
i5 = 0
i4 - -
str5[i5] = str4[i4]
i5++
да
i4 >= 0
нет
i5 = 0
i6 = 0
str6[i6] = str5[i5]
i5++
i6++
да
str5[i5] != 0 &&
str5[i5] != ' '
нет
str6[i6] = ' '
i6++
i5++
да
str5[i5] != 0 &&
str5[i5] != ' '
нет
да
str5[i5] != 0
нет
D

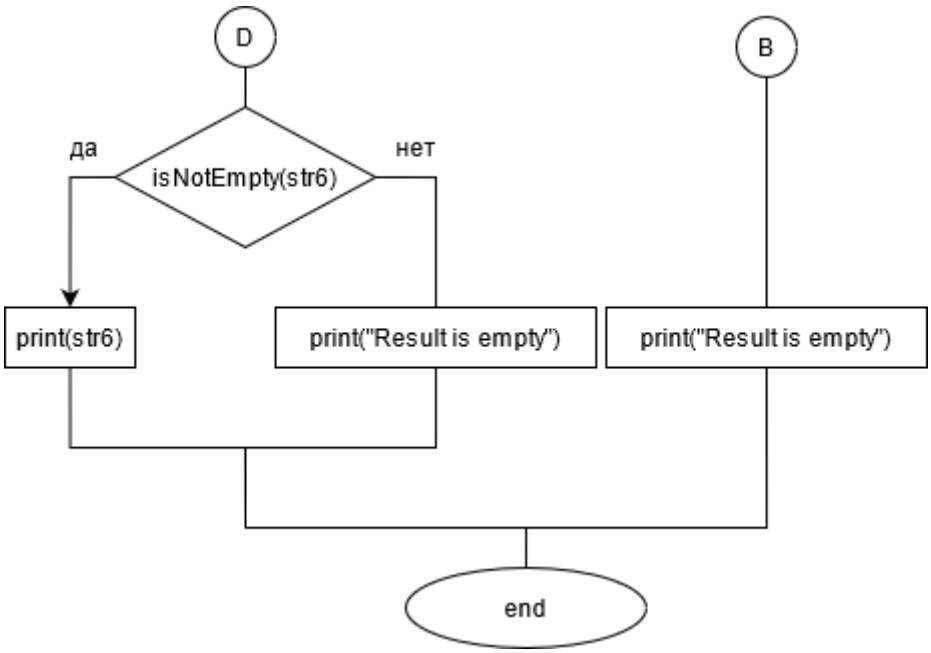


Рисунок 1 — схема алгоритма

Тестирование программы:

Таблица 1 — результаты тестирования

Исходные данные	Ожидаемый результат	Полученный результат
aek ai marat japan	Result: aek japan	Result: aek japan
(нажатие enter)	Result is empty	Result is empty
90797098797	Result is empty	Result is empty
aaaa ooo kkkk nn	Result is empty	Result is empty
mn kk alp eax pp	Result: alp eax	Result: alp eax

Контрольные вопросы

1. Дайте определение символьной строки.

Строка — упорядоченная последовательность символов. В случае ассемблера MASM – размер 1 символа — 1 байт.

2. Назовите основные команды обработки цепочек?

- Movs – пересылка строки
- cmps – сравнение строк
- scas – сканирование цепочки
- stos – сохранение элемента из регистра AL в цепочку

- lods – сохранение элемента из цепочки в регистр AL
- rep, repe, repne – перфиксы повторения

3. Какие операции выполняют строковые команды MOVS? Какие особенности характерны для этих команд?

Команды movs пересылают строки из источника в приемник, по 1 элементу. Источник индексируется с помощью регистра ESI, приемник — EDI. Направление изменения регистров определяется флагом DF. Для пересылки нескольких элементов необходимо использовать префикс rep и записать число пересылаемых элементов регистр ECX.

4. Какие операции выполняют строковые команды CMPS, SCAS? Какие особенности характерны для этих команд?

Команды CMPS сравнивают элементы цепочек выставляя флаги, как будто элементы одной строки вычитаются из элементов второй и увеличивает/уменьшает адреса, записанные в регистрах ESI и EDI в зависимости от значения флага DF. Команды SCAS сравнивают аналогичным образом значение в регистре AL и элемент строки приемника по адресу, хранящемуся в регистре EDI.

5. Как обеспечить циклическую обработку строк?

Префикс rep будет выполнять команду указанную после него, уменьшая значение ECX, пока оно не достигнет 0.

Repe/repne будет выполнять команду указанную после него, уменьшая значение ECX, пока оно не достигнет 0 или пока в цепочках не встретятся различные/одинаковые элементы.

6. Какова роль флага DF во флаговом регистре при выполнении команд обработки строк?

Флаг DF определяет будут ли команды обработки строк уменьшать или увеличивать значения ESI, EDI при каждом вызове. DF = 0 – увеличение адресов, DF = 1 – уменьшение. Команда std – выставить DF = 1, cld – выставить DF = 0.

7. Какие макрокоманды используются в среде RADASM для ввода и вывода строк?

Invoke StdIn, ADDR str, LengthOf str – ввод строки str

Invoke StdOut, ADDR str – вывод строки str

8. Как правильно выбрать тестовые данные для проверки алгоритма обработки строки?

Тестовые данные должны учитывать все возможные выходные результаты работы алгоритма(в том числе отсутствие искомых последовательностей или их повторение), а так же все возможные форматы ввода — пустая строка, символы разных алфавитов с разделителями(пробелы, запятые и т.д.) и без, цифры, спецсимволы.

Вывод: в ходе данной работы были получены базовые навыки обработки символьных последовательностей с помощью языка ассемблера и основные и некоторые семантические ошибки, которые легко допустить при написании программ обработки строк.