```
    LEN     EQU    80
DOSSEG
.MODEL SMALL

.STACK 100h

.DATA
        Prompt1 DB  "Input a string:  ", 0
        Prompt2 DB  "Do another?  ", 0
        String  DB  (LEN + 1) dup(?)
.CODE
        EXTRN getStr:NEAR, putStr:NEAR, newLine:NEAR
;------------------------------------------------------------
Start:
        mov   ax, @DATA
        mov   ds, ax
Begin:
        mov   ax, OFFSET Prompt1
        push  ax
        call  putStr
        call  newLine
        call  newLine

        mov   ax, OFFSET String
        push  ax
        call  getStr
        call  newLine
        call  newLine

    mov   ax, OFFSET String
    push  ax
    call  putStr
```

```
        call  newLine
        call  newLine

        mov   ax, OFFSET Prompt2
        push  ax
        call  PutStr

        mov   ax, OFFSET String
        push  ax
        call  getStr

        call  newLine
        call  newLine

        mov   bx, OFFSET String
        cmp   BYTE PTR [bx], 'y'
        je    Begin

                mov   ax, 4c00h
                int   21h
END Start
```

MACROS:
Look at code in prog5_4.asm (above)

```
        mov     ax, OFFSET Prompt1
        push    ax
        call    putStr
```

is repreated (with variations) several times in the program

MACROS allow us to avoid this kind of mindless repetition

Broadly speaking:
MACROS: simply substitute one piece of text with another

syntax:

```
macroName       MACRO ListOfReplaceableParameters

        ...
        macro code
        ...

ENDM
```

MACRO is a reserved word
ENDM end of macro definition

To invoke a macro named Macro1, just use:

```
        Macro1  listofArguments
```

MASM replaces this line by a copy of the macro code. Each occurence of a replaceable parameter replaced by the corresponding argument

Consider simple MACRO definition

```
inpStr  MACRO   Buffer

        push    ax
        mov     ax, OFFSET Buffer
        push    ax
        call    getStr
        pop     ax
ENDM
```

Makes the call to getStr easier

To invoke this macro:

```
        inpStr  String
```

MASM replaces this invocation by

```
        push    ax
        mov     ax, OFFSET String
```

```
push    ax
call    getStr
pop     ax
```

Here's another macro for calling putStr

```
outStr    MACRO Buffer

          push  ax
          mov   ax, OFFSET Prompt1
          push  ax
          call  putStr
          pop   ax

ENDM
```

Let's rewrite program prog5_4.asm using these macros:

```
    LEN    EQU    80
DOSSEG
.MODEL SMALL

.STACK 100h

.DATA
          Prompt1 DB  "Input a string:  ", 0
          Prompt2 DB  "Do another?  ", 0
          String  DB  (LEN + 1) dup(?)
.CODE
          EXTRN getStr:NEAR, putStr:NEAR, newLine:NEAR
;------------------------------------------------------------
Start:
          mov   ax, @DATA
          mov   ds, ax
Begin:
```

```
        outStr     Prompt1
        call  newLine
        call  newLine

        inpStr     String
        call  newLine
        call  newLine

        outStr     String
    call  newLine
    call  newLine

        outStr     Prompt2

        inpStr     String

    call  newLine
    call  newLine

    mov   bx, OFFSET String
    cmp   BYTE PTR [bx], 'y'
    je    Begin

        mov   ax, 4c00h
        int   21h
END Start
```

Well we know how to define a macro
we know how to invoke a macro

Where do we put macro definitions?

We usually gather related sets of macros and put them in an include file.

Here’s how I would (partially) solve your assignment

```
TITLE AS6.asm

INCLUDE        as6.mac      ; Contains the macro
                            ; definition of strLeng

LEN    EQU    80

DOSSEG
.MODEL SMALL

.STACK 100h

.DATA
          Prompt1 DB  "Input a string:  ", 0
          Prompt2 DB  "Do another?  ", 0
          String  DB  (LEN + 1) dup(?)
.CODE
     EXTRN getStr:NEAR, putStr:NEAR, newLine:NEAR, DE
COUT:NEAR
;-------------------------------------------------------------
Start:
          mov   ax, @DATA
          mov   ds, ax
Begin:
          mov   ax, OFFSET Prompt1 ; shd replace with macro
          push  ax                 ; invocation
          call  putStr
          call  newLine
          call  newLine

          mov   ax, OFFSET String  ; should also replace with
```

```
        push  ax                ; macro invocation
        call  getStr
        call  newLine
        call  newLine

        strLeng String          ; Invoking macro strLeng

        mov     dx, ax          ; outputing string's  length
        CALL    DECOUT
        call    newLine

        mov   ax, 4c00h ; exiting
        int   21h
END Start
```

The file as6.mac would look like

```
;-------------------------------------------------------------
COMMENT    #   MACRO HEADER
NAME:          strLeng        FILE:as6.mac
PURPOSE:       finds length of asciiz string
CALLED AS:      strLeng Buffer
                 where Buffer is a storage area for the
                 string. Buffer must be in the segment pointed
                 to by ds
REMARKS:       Buffer must be long enough to hold input string
             #   and null-terminator

strLeng MACRO String

          code...

ENDM
;-------------------------------------------------------------
```

Debugging code with macros is more difficult.
Look out.