! (literál karakter operátor): Az utána következő karaktert makró kifejtéskor közönséges karakterként kell kezelni. Pl.: a korábbi **errgen** makró

```
        errgen 103, <Expression !> 255>
```

hívásának hatása:

```
err103 db  'Error 103: Expression > 255'
```

de

```
        errgen 103, <Expression > 255>
```

hívásának hatása:

```
err103 db  'Error 103: Expression '
```


% (kifejezés operátor): Az utána lévő argumentum (kifejezés is lehet) értéke – és nem a szövege – lesz az aktuális paraméter. Pl.:

```
sym1    equ     100
sym2    equ     200
txt     equ     'Ez egy szöveg'

kif     macro   exp, val
        db      "&exp = &val"
        endm

        kif     <sym1+sym2>, %(sym1+sym2)
        kif     txt, %txt

        db      "sym1+sym2 = 300"
        db      "txt = 'Ez egy szöveg'"
```


Az alábbi példa a **%** használatán kívül a makró törzsön belüli makró hívást is bemutatja:

```
s       =       0
ErrMsg  MACRO   text
s       =       s+1
        Msg     %s,text
        ENDM

Msg     MACRO   sz,str
msg&sz  db      str
        ENDM
```



```
s       =       0
ErrMsg  MACRO   text
s       =       s+1
        Msg     %s,text
        ENDM
```

```
Msg     MACRO   sz,str
msg&sz  db      str
        ENDM
```

```
        ErrMsg 'syntax error'
```

makró hívás hatására bemásolásra kerül (**.LALL** hatására látszik a listán) az

```
s       =       s+1
        Msg     %s,'syntax error'
```

szöveg. **s** értéke itt **1**-re változik. Újabb makró hívás (**Msg**). A **%s** paraméter az **s** értékére (**1**) cserélődik, majd kifejtésre kerül ez a makró is, ebből kialakul:

```
msg1    db      'syntax error'
```



```
s       =       0
ErrMsg  MACRO   text
s       =       s+1
        Msg     %s,text
        ENDM
```

```
Msg     MACRO   sz,str
msg&sz  db      str
        ENDM
```

Egy újabb hívás és hatása:

```
        ErrMsg 'invalid operand'
msg2    db      'invalid operand'
```


;; (makró kommentár): A makró definíció megjegyzéseinek kezdetét jelzi. A **;;** utáni megjegyzés a makró kifejtés listájában nem jelenik meg.

LOCAL c1[,c2...]

c1, **c2**, **...** minden makró híváskor más, **??xxxx** alakú szimbólumra cserélődik, ahol **xxxx** a makró generátor által meghatározott hexadecimális szám. A **LOCAL** operátort közvetlenül a makró fej utáni sorba kell írni.

```
KOPOG   macro   n
        LOCAL   ujra
        mov     cx,n
ujra:   KOPP
        loop    ujra
        endm
```

Ha a programban többször hívnánk a **KOPOG** makrót, akkor a **LOCAL** operátor nélkül az **ujra** címke többször lenne definiálva.

Makró definíció belsejében lehet másik makró definíció is. A belső makró definíció csak a külső makró meghívása után jut érvényre, válik láthatóvá. Pl.:

```
shifts      macro       OPNAME ; makrót
                            ; definiáló makró
OPNAME&S    MACRO       OPERANDUS,N
            mov         cl, N
            OPNAME      OPERANDUS,cl
            ENDM
            endm
```



```
shifts      macro       OPNAME ; makrót
                            ; definiáló makró
OPNAME&S    MACRO       OPERANDUS,N
            mov         cl, N
            OPNAME      OPERANDUS,cl
            ENDM
            endm
```

Ha ezt a makrót felhívjuk pl.:

```
            shifts ROR
```

akkor a

```
RORS        MACRO   OPERANDUS,N
            mov     cl, N
            ROR     OPERANDUS,cl
            ENDM
```

makró definíció generálódik.


```
RORS        MACRO   OPERANDUS,N
            mov     cl, N
            ROR     OPERANDUS,cl
            ENDM
```

Mostantól meghívható a **RORS** makró is, pl.:

```
            RORS    AX, 5
```

aminek a hatása:

```
            mov     cl, 5
            ROR     AX,cl
```


Makró definíció belsejében meghívható az éppen definiálás alatt lévő makró is (a makró hívás ezáltal rekurzívvá válik).

```
PUSHALL macro   reg1,reg2,reg3,reg4,reg5
IFNB    <reg1>          ;; ha a paraméter nem üres
        push    reg1    ;; az első regiszter mentése
        PUSHALL reg2,reg3,reg4,reg5  ;; rekurzió
ENDIF
        ENDM
```

Most pl. a

```
        PUSHALL ax, bx, cx
```

makró hívás hatása:

```
        push    ax
        push    bx
        push    cx
```



```
PUSHALL macro   reg1,reg2,reg3,reg4,reg5
IFNB    <reg1>          ;; ha a paraméter nem üres
        push    reg1    ;; az első regiszter mentése
        PUSHALL reg2,reg3,reg4,reg5  ;; rekurzió
ENDIF
        ENDM

        PUSHALL ax, bx, cx
```

makró hívás hatása:

```
        push    ax
        PUSHALL bx, cx
```

az újabb hívás hatása:

```
        push    bx
        PUSHALL cx
```

az újabb hívás hatása:

```
        push    cx
        PUSHALL
```

ennek hatására nem generálódik semmi.

```
FL_CALLELJ     =   0
CALLELJ    macro ;; Eljárást beépítő és felhívó makró
       LOCAL   FIRST   ;; nem lenne fontos
IF     FL_CALLELJ      ;; a 2. hívástól igaz
       call    Elj     ;; elég felhívni az eljárást
       EXITM           ;; makró helyettesítés vége
ENDIF
FL_CALLELJ     =   1   ;; csak az első híváskor
       JMP     FIRST   ;; jut érvényre
Elj    proc            ;; eljárás deklaráció
       ...
       ret
Elj    endp
FIRST: call    Elj     ;; az eljárás felhívása
       endm
```


Az első **CALLELJ** hívás hatására az

```
FL_CALLELJ =   1
               JMP         ??0000
Elj            proc
               ...
               ret
Elj            endp
??0000:        call        Elj
```

utasítások generálódnak (**??0000** a **FIRST**-ből keletkezett).

A további **CALLELJ** hívások esetén csak egyetlen utasítás, a

```
call      Elj
```

utasítás generálódik.

A megoldás előnye, hogy az eljárás akkor és csak akkor része a programnak, ha a program tartalmazza az eljárás felhívását is, és mégsem kell törődjünk azzal, hogy hozzá kell-e szerkesztenünk a programhoz vagy se.

Megváltoztathatunk egy makró definíciót azáltal, hogy újra definiáljuk.

Makró definíción belül előfordulhat másik makró definíció.

E két lehetőség kombinációjából adódik, hogy a makró definíción belül megadhatunk ugyanarra a makró névre egy másik definíciót, ezáltal készíthető olyan makró, amely „átdefiniálja” önmagát.

Az önmagát átdefiniáló makrók esetében a belső és külső definíciót lezáró **ENDM** utasítások között egyetlen utasítás sem szerepelhet – még kommentár sem!

Önmagát „átdefiniáló” makró (az előző feladat másik megoldása):

```
CALLELJ2   macro           ; külső makró definíció
           jmp     FIRST
Elj2       proc            ; eljárás deklaráció
           ...
           ret
Elj2       endp
FIRST:     call    Elj2    ; eljárás hívás
CALLELJ2   MACRO           ; belső makró definíció
           call    Elj2    ; eljárás hívás
           ENDM        ; belső makró definíció vége
           endm        ; külső makró definíció vége
```


CALLELJ2 első hívásakor a kifejtés eredménye:

```
           jmp     FIRST
Elj2       proc            ; eljárás deklaráció
           ...
           ret
Elj2       endp
FIRST:     call    Elj2    ; eljárás hívás
CALLELJ2   MACRO           ; belső makró definíció
           call    Elj2    ; eljárás hívás
           ENDM        ; belső makró definíció vége
```

A kifejtés **CALLELJ2** újabb definícióját tartalmazza, ez felülírja az eredeti definíciót, és a továbbiak során ez a definíció érvényes. Ez alapján a későbbi **CALLELJ2** hívások esetén

```
call    Elj2
```

a kifejtés eredménye.

Megjegyezzük, hogy most is szerencsésebb lett volna a **FIRST** címkét lokálissá tenni. Igaz, hogy csak egyszer generálódik, de így a **CALLELJ2** makró használójának tudnia kell, hogy a **FIRST** címke már „foglalt”!

Ha egy **M_név** makró definíciójára nincs szükség a továbbiak során, akkor a

```
PURGE    M_név
```

pszeudo utasítással kitörölhetjük.

Blokk ismétlés

Nemcsak a blokk definíciójának kezdetét jelölik ki, hanem a kifejtést (hívást) is előírják. A program más részéről nem is hívhatók.

Blokk ismétlés **kifejezés**-szer:

```
REPT    kifejezés
...     ; ez a rész ismétlődik
ENDM
```


A korábban ismertetett kopogást így is megoldhattuk volna:

A korábbi megoldás:

```
KOPOG   macro   n
        LOCAL   ujra
        mov     cx,n
ujra:   KOPP
        loop    ujra
        endm
```

```
        REPT    N
        KOPP
        ENDM
```

Ha pl. **N=3**, akkor ennek a hatására a

```
        KOPP
        KOPP
        KOPP
```

makró hívások generálódnak.

Megjegyzés: Most **N** nem lehet változó – fordítási időben ismert kell legyen az értéke!

Blokk ismétlés argumentum lista szerint:

```
IRP     par, <arg1[,arg2...]>
... ;   ez a rész többször bemásolásra
... ;   kerül úgy, hogy par rendre
... ;   fölveszi az arg1,arg2... értéket
ENDM

IRP     x, <1,2,3>
db      x
ENDM

db      1
db      2
db      3
```


Blokk ismétlés string alapján:

```
IRPC    par,string
...     ; ez a rész kerül többször bemásolásra úgy,
...     ; hogy par rendre fölveszi
...     ; a string karaktereit
ENDM
```

Ezt a **string**-et nem kell idézőjelek közé tenni (újabb ismétlés jelentene). Ha a **string**-en belül pl. szóköz vagy **,** is előfordul, akkor **<>** jelek közé kell tenni.

Az előző feladatot így is megoldhattuk volna:

```
IRPC    x,123
db      x
ENDM
```

Másik példa:

```
IRPC    x,ABCDEFGHIJKLMNOPQRSTUVWXYZ
db      '&x'          ;; nagy betűk
db      '&x'+20h      ;; kis betűk
ENDM
```

Hatása:

```
db      'A'
db      'A'+20h             a kódja
. . .
db      'Z'
db      'Z'+20h             z kódja
```

Fontos az `&` jel, nélküle `'x'`-ben `x` nem paraméter, hanem string lenne!