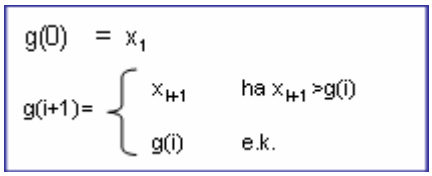
Feladat: Keressük meg egy egész számokból álló lista legnagyobb elemét!



g: a maximális érték az adott pontig

```
g :: [a] a ->a | Ord a

g [ ] y  = y
g [x:xs] y
        |x>y      = g xs x
                  = g xs y
myMax [x:xs] = g xs x

Start = myMax [1,3,2,7,4]
```

Saját jegyzeteim

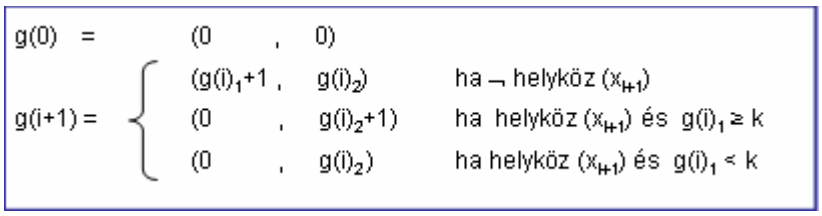
Feladat: Rendezzük a lista elemeit növekvő sorrendbe!

```
Insert :: a [a] -> [a] | Ord a
Insert e [ ] = [e]
Insert e [x:xs]
        |e<=x   = [e,x :xs]
                = [x:Insert e xs]

mysort x:xs] = foldr Insert [x] xs

Start = mysort [5,3,1]
```

Saját jegyzeteim

Feladat: Számoljuk meg, hány legalább k hosszúságú szó van a szövegben!

1. változat

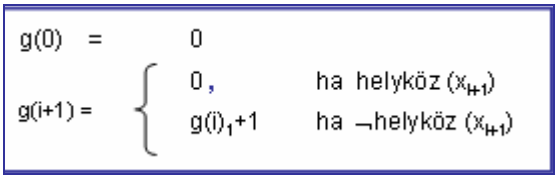


g2: az aktuális szó hossza az adott pontig
g3: a k-nál hosszabb szavak darabszáma az adott pontig

```
g [] g1 g2 k = g2

g [x:xs] g1 g2 k
        |x<>' '                  = g xs (g1+1)   g2       k
        |x==' ' && g1 >=k        = g xs  0       (g2+1)   k
        |x==' ' && g1 <k         = g xs  0       g2       k

count [x:xs] k = g ([x:xs]++[' ']) 0 0 k

Start = count ['a','R','a','R',' ','a','R',' ','a','R','a',' '] 3
```

Saját jegyzeteim

2. változat



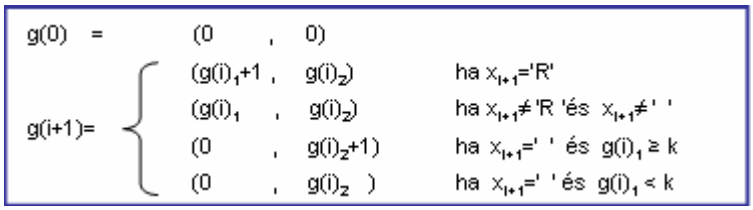
g: betűk száma az aktuális szóban az aktuális pontig

```
g x y
   |x==' ' = 0
          = (y+1)

count g [ ] y k = 0
count g [x:xs] y k
        |g x y == k    = 1 + count g xs (g x y) k
                       = count g xs (g x y) k


Start = count g ( ['a','R','a','R',' ','a','R',' ','a','R','a']  ++[' ']) 0 5
```

Saját jegyzeteim

Feladat: Számoljuk meg, hány olyan szó van a szövegben, amely legalább k db 'R' betűt tartalmaz!

1. változat

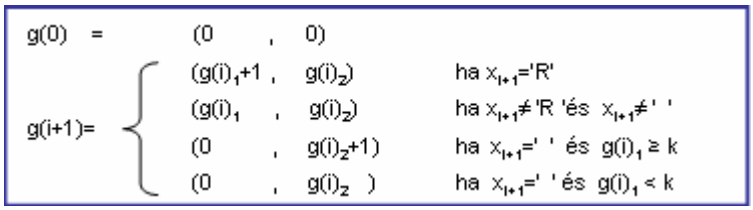


g1: az „R” betűk száma az aktuális szóban az aktuális pontig
g2: a legalább k db „R” betűt tartalmazó szavak darabszáma

```
g [] g1 g2 k = g2
g [x:xs] g1 g2 k
        |x=='R'                = g xs   (g1+1) g2     k
        |x<>' ' && x<>'R'      = g xs   g1     g2     k
        |x==' ' && g1>=k       = g xs   0      (g2+1) k
        |x==' ' && g1<k        = g xs   0      g2     k

RS [x:xs] k = g ([x:xs] ++ [' ']) 0 0 k

Start =  RS ['a','R','a','R',' ','a','R','R','R',' ','a','R','R','R'] 3
```

Saját jegyzeteim

2. változat



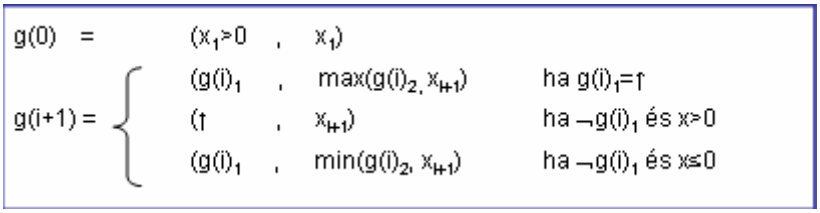
g1: az „R” betűk száma az aktuális szóban az aktuális pontig
g2: a legalább k db „R” betűt tartalmazó szavak darabszáma

```
g [] g1 g2 k
        | (g1>=k)    =(g2+1)
                      =g2
g [x:xs] g1 g2 k
        |x=='R'                = g xs   (g1+1) g2     k
        |x<>' ' && x<>'R'      = g xs   g1     g2     k
        |x==' ' && g1>=k       = g xs   0      (g2+1) k
        |x==' ' && g1<k        = g xs   0      g2     k

RS [x:xs] k = g [x:xs] 0 0 k

Start =    RS ['a','R','a','R',' ','a','R','R','R',' ','a','R','R','R'] 3
```

Saját jegyzeteim

Itt kezeljükaz utolsó elem megszámolását.

Feladat: Adott egy egész számokat tartalmazó lista. Ha a lista tartalmaz pozitív elemeket, akkor keressük meg a legnagyobbat, különben a legkisebbet!

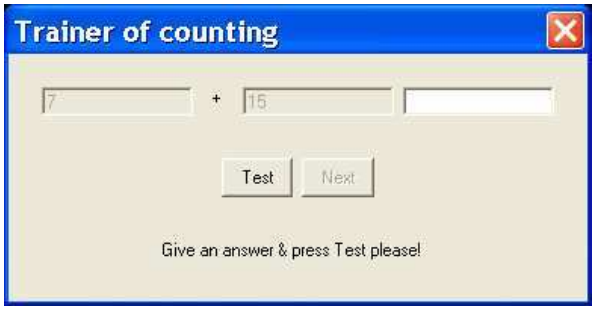


g1: Igaz (↑), ha maximumot kell számolni, hamis(↓), ha minimumot
g2: Ha g1 igaz, akkor ez a maximum, ha g1 hamis, akkor ez a minimum

```
g [ ] g1 g2 = g2
g [x:xs] g1 g2
        |g1                    = g xs   g1      (Max g2 x)
        |(not g1) && (x>0)     = g xs   True    x
        |(not g1) && (x<=0)    = g xs   g1      (Min g2 x)
        where
                Max a b = if (a<b) b a
                Min  a b = if (a<b) a b

MinMax [x:xs] = g xs (x>0) x

Start = MinMax [-1,8,-12,-3,-5]
```

Saját jegyzeteim

Feladat: Készítsünk el egy alkalmazást „összeadás gyakoroltatására.



Start

```
module count
import StdEnv,StdIO,Random

:: *World -> *World
Start world
  = startIO NDI Void initIO [] world
```

LocalSt

```
::
  = { op1        :: String
    , op2        :: String
    , result     :: String
    , msg        :: String
    , randseed   :: RandomSeed
    }
```

initIO

```
pst
  # randseed = nullRandomSeed
  # (randseed,pst)        = getNewRandomSeed(pst)
  # (id_dialog, pst)      = accPIO openId pst
  # (id_op1,pst)          = accPIO openId pst
  # (id_op2,pst)          = accPIO openId pst
  # (id_result,pst)       = accPIO openId pst
  # (id_msg,pst)          = accPIO openId pst
  # (id_next,pst)         = accPIO openId pst
  # (id_test,pst)         = accPIO openId pst
  # (_,pst)               = openDialog Void (dialog id_dialog id_op1 id_op2 id_result id_msg id_next
                            id_test randseed) pst
                        = pst
```

Saját jegyzeteim

Dialog

```
where
id_dialog id_op1 id_op2 id_result id_msg id_next id_test randseed
    #(x,randseed) = random randseed
    #(y,randseed) = random randseed
    # x_str       = toString(x/500)
    # y_str       = toString(y/500)
    = Dialog "Trainer of counting"
    { newLS =  { op1=x_str, op2=y_str, result="", msg=init_msg, randseed=randseed }
    , newDef =  LayoutControl
          (  EditControl x_str  displaywidth displayheight
             [ ControlId              id_op1
             , ControlSelectState  Unable
             ]
          :+: TextControl "  +  " []
           :+:  EditControl y_str  displaywidth displayheight
             [ ControlId              id_op2
             , ControlSelectState  Unable
             ]
              :+:  EditControl ""  displaywidth displayheight
             [ ControlId              id_result
             , ControlSelectState  Able
             ]
          ) [ControlPos (Center,zero)]
          :+:  LayoutControl
          (  ButtonControl "Test"
             [ ControlFunction     onTest
             , ControlTip      "   Checks the result."
             ,  ControlId           id_test
             ]
          :+:  ButtonControl       "Next"
             [ ControlFunction     onNext
             , ControlSelectState  Unable
             , ControlTip      "   Gives a new task."
             , ControlId           id_next
             ]
          )  [ControlPos (Center,zero)]
          :+:  LayoutControl
          (  TextControl init_msg
             [ ControlId           id_msg ]
          )   [ControlPos (Center,zero)]
    }
    [ WindowClose  (noLS closeProcess)
    , WindowId  id_dialog
    ]
 where
  displaywidth   = PixelWidth 100
  displayheight  = 1
  init_msg       = " Give an answer && press Test please! "
```

Saját jegyzeteim

onNext

```
onNext :: (LocalSt,PSt .l) -> (LocalSt,PSt .l)
onNext ({op1,op2,result,msg},pst)
  # good                = toInt(op1)+toInt(op2)==toInt(result)
  # (randseed,pst)      = getNewRandomSeed(pst)
  # (x,randseed)        = random randseed
  # (y,randseed)        = random randseed
  # x_str               = toString(x/500)
  # y_str               = toString(y/500)
  # op1                 = if good x_str op1
  # op2                 = if good y_str op2
  # result              = if good ""  result
  # msg                 = if (toInt(result)==0) "Give an answer && press Test please! " ""
  # io                  = disableControl id_next  pst.io
  # io                  = enableControl id_test  io
  # io                  = (setControlText id_test "Test" io)
  = ({op1=op1,op2=op2,result=result,msg=msg,randseed}
    ,appPIO (setControlTexts [(id_op1,op1)
                              ,(id_op2,op2)
                              ,(id_result,result)
                              ,(id_msg,msg)
                              ]) {pst & io= io} )
```

onTest

```
onTest :: (LocalSt,PSt .l) -> (LocalSt,PSt .l)
onTest ({op1,op2,result,msg,randseed},pst)
  # (wst,pst)   = accPIO (getParentWindow id_dialog) pst
  # result      = fromJust (snd (getControlText id_result (fromJust wst)))
  # good        = toInt(op1)+toInt(op2)==toInt(result)
  # msg         = if good  "Good!! Press Next to go on."
                       if (toInt(result)==0) "Give the answer first!" "Bad!! Try it again!"
  # io          = if good (enableControl id_next pst.io) (disableControl id_next pst.io)
  # io          = if good (disableControl id_test io) (enableControl id_test io)
  # io          = if good (setControlText id_test "Super !!" io) (setControlText id_test "Test"  io)

  = ({op1=op1,op2=op2,result=result,msg=msg,randseed=randseed}
    ,appPIO  (setControlTexts   [(id_op1,op1)
                                ,(id_op2,op2)
                                ,(id_result,result)
                                ,(id_msg,msg)
                                ]) {pst & io= io})
```

Saját jegyzeteim

Feladat: Nézzük meg, hogy egy karaktereket tartalmazó lista fordított sorrendben ugyanazt aeredményezi-e?

```
module Exc1_12
import StdEnv

Equal [] []       = True
Equal [a] [b]     = a==b
Equal [x,y] [u,v] = x==u && y == v
Equal [x:y] [u:v] = x==u && Equal y v

Reverse []      = []
Reverse [a] = [a]
Reverse [a,b] = [b,a]
Reverse [a:b] = Reverse b ++ [a]

Polindrome a = Equal a (Reverse a)

Start = Polindrome ['a','b','a']
```

Feladat: Készítsük el azt a függvényt, amely visszaadja a lista utolsó két elemét.

```
module Exc1_10
import StdEnv

LastTwo[a]       = []
LastTwo[a,b] = [a,b]
LastTwo[a:b] = LastTwo b

Start = LastTwo[1,2,3,4,5]
```

Feladat: Készítsük el azt a függvényt, amely kiszámolja egy egész szám számjegyeinek összegét.

```
module Exc1_3
import StdEnv
Isum :: Int -> Int
Isum a
        | a<10              = a
                            = ((a-(a/10)*10) + Isum (a/10))
Start = Isum 1234
```