ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εισαγωγή στον Προγραμματισμό Η/Υ για Χημικούς Μηχανικούς

Παρουσίαση Διαλέξεων: 4. Επανάληψη

Καθηγητής Δημήτρης Ματαράς

Δομή DO υπό συνθήκη:

a) Δομή DO WHILE

```
[όνομα δομής:]DO WHILE(συνθήκη)      !αρχική πρόταση
      ....
      εκφράσεις
      ....
END DO[όνομα δομής]                  !τελική πρόταση
```

- **[*όνομα δομής*]** : *προαιρετική* ονομασία της δομής
- ***συνθήκη*** : λογική μεταβλητή ή έκφραση με λογικό αποτέλεσμα
- ***εκφράσεις*** : εκφράσεις, προτάσεις, δομές…

```
a=0
DO WHILE (a<=5)
   a=a+1
   PRINT*,a

END DO
```

```
a=0
my:DO WHILE (a<=5)
   a=a+1
   PRINT*,a

END DO my
```

Δομή DO υπό συνθήκη:

b) Δομή DO UNTILL

```
[όνομα δομής:]DO                          !αρχική πρόταση
      ....
      εκφράσεις
  IF(συνθήκη)EXIT[όνομα δομής]            !πρόταση ελέγχου
END DO[όνομα δομής]                       !τελική πρόταση
```

- **[*όνομα δομής*]** : *προαιρετική* ονομασία της δομής
- ***συνθήκη*** : λογική μεταβλητή ή έκφραση με λογικό αποτέλεσμα
- ***εκφράσεις*** : εκφράσεις, προτάσεις, δομές…

```
a = 0
DO
   a = a + 1
   PRINT*, a
   IF(a > 5)EXIT
END DO
```

```
   a = 0
my:DO
   a = a + 1
   PRINT*,a
   IF(a > 5)EXIT my
END DO my
```

Λειτουργία δομών επανάληψης υπό συνθήκη

Στις δομές **DO** υπό συνθήκη:

a) DO WHILE: Για να ξεκινήσει η επανάληψη πρέπει να ισχύει αρχικά η *συνθήκη*

- Κάθε φορά πριν ξεκινήσει η επανάληψη ελέγχεται η *συνθήκη*
  - Αν ισχύει εκτελείται η επανάληψη
  - Αν δεν ισχύει ο έλεγχος μεταφέρεται ακριβώς μετά από το **END DO**

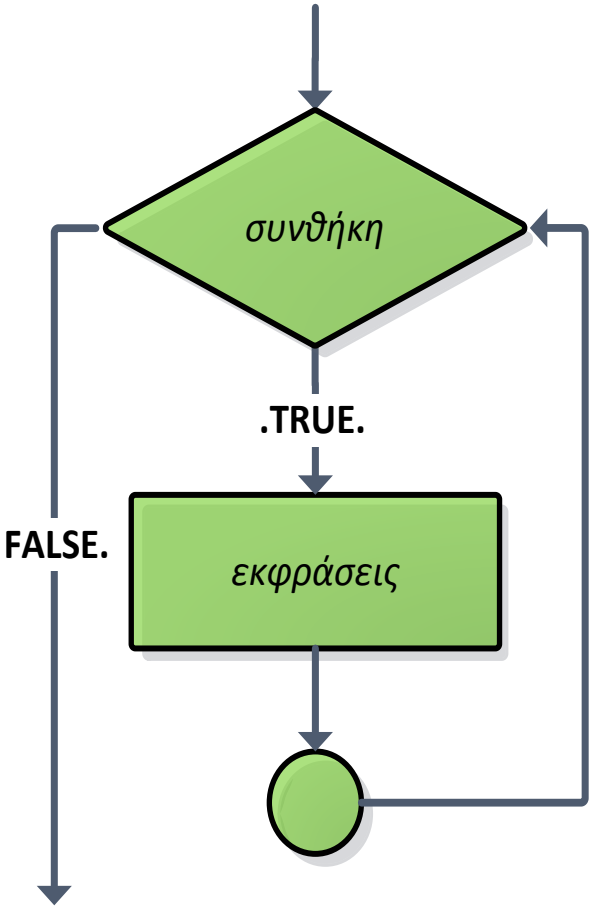
b) DO UNTILL: Η επανάληψη ξεκινάει ανεξαρτήτως του αν ισχύει η *συνθήκη*

- Κάθε φορά εκτελείται η επανάληψη μέχρι την *πρόταση ελέγχου* και εξετάζεται η *συνθήκη*
  - Αν ισχύει ο έλεγχος μεταφέρεται ακριβώς μετά από το **END DO**
  - Αν δεν ισχύει εκτελείται η επανάληψη

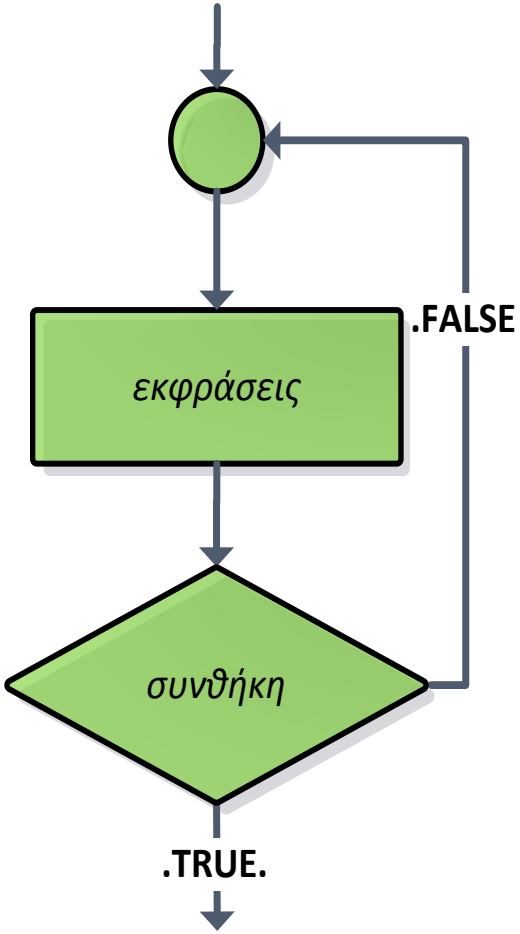
Είναι προφανές ότι, για το ίδιο πρόβλημα, η συνθήκη του DO WHILE πρέπει να είναι αντίθετη από εκείνη του DO UNTILL. Πχ. Αν το για DO WHILE η συνθήκη είναι **a>0** τότε είναι **a<=0** για το DO UNTILL.

Λειτουργία δομών επανάληψης υπό συνθήκη

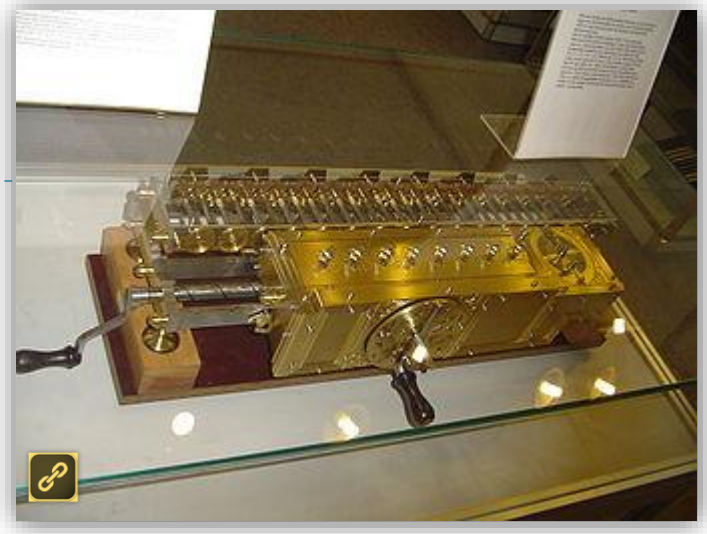
DO WHILE



DO UNTILL



Ένας υπολογισμός του π (Leibniz 1646-1716)



```
integer::i = 1, sign = -1
real(8)::pi, pid = 0,s = 0
   pi = 4 * atan(1.)
   do
       sign = -sign
   if(abs(pi - pid) < 1e-14)exit
       s   = s + sign / real(i)
       pid = 4 * s
       i   = i + 2
   enddo
   print*, pid, (i - 1) / 2
end
```

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

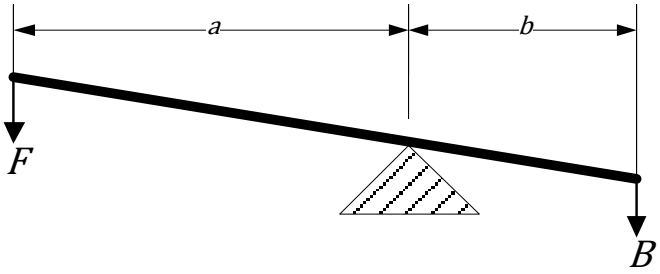
```
3.14159274101258          71895645
Press any key to continue . . .
```

Ένας υπολογισμός του π ξανά (Leibniz)

```
integer::n = 0
real(8)::pi, pid = 0, s = 0
   pi = 4 * atan(1.d0)
   do
      if(abs(pi - pid) < 1e-14)exit
      s = s + ((-1)**n / (2*n+1.))
      pid = 4 * s
      n = n + 1
   enddo
   print*, pid, n
end
```

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

ΠΑΡΑΔΕΙΓΜΑ: Η εξίσωση του μοχλού είναι $F \times a = B \times b$. Να φτιάξετε πρόγραμμα που υπολογίζει το μήκος του μοχλού που απαιτείται για να ανυψώσει βάρος που δίνεται από τον χρήστη με δύναμη που είναι μόνο 25 Kg, αν το υπομόχλιο βρίσκεται σε απόσταση 0.5 μέτρων από το βάρος.




```
program lever !Εξετάσεις Φεβ. 2011 Υ&Α άσκηση 4
    implicit none
!δηλώσεις:
    real::f, a = 0, ba, b = 0.5

!αρχή:
    call execute_command_line('chcp 1253')
    print*, 'βάρος?'; read*, ba

    do
      a = a + 1    !αυξάνουμε διαδοχικά το μήκος του μοχλού
      f = ba * b / a !υπολογίζουμε τη δύναμη που απαιτείται
      print*, 'με μήκος μοχλού', a + b,'m'
      print*, 'απαιτείται δύναμη', f, 'Kg'
    if(f <= 25)exit; enddo

end program lever
```

Δομή DO με αρίθμηση:

```
[όνομα δομής:]DO αριθμητής=τιμή1,τιμή2[,βήμα] !αρχική πρόταση
      ....
      εκφράσεις
      ....
END DO[όνομα δομής]                            !τελική πρόταση
```

- **[*όνομα δομής*]** : *προαιρετική* ονομασία της δομής
- ***αριθμητής*** : *ακέραια* μεταβλητή.
  - Στο τέλος κάθε επανάληψης: ***αριθμητής=αριθμητής+βήμα***
- ***τιμή$_1$*** : η αρχική τιμή του ***αριθμητή***
- ***τιμή$_2$*** : η τελική τιμή του ***αριθμητή***
- **[*,βήμα*]** : το ακέραιο βήμα αύξησης ή μείωσης του ***αριθμητή***.
  - *Υποχρεωτικό μόνον όταν:* ***βήμα≠1***

```
a = 0
DO i = 1, 5
   a = a + 1; PRINT*, a
END DO
```

```
a = 0
myfirst:DO i = 1 , 5
   a = a + 1; PRINT*, a
END DO myfirst
```

Πως λειτουργεί η δομή DO με αρίθμηση:

- ***Υπολογισμός αριθμού επαναλήψεων* N :** **N =MAX(INT((*τιμή*$_2$ - *τιμή*$_1$ + *βήμα*)/*βήμα*) ,0)**

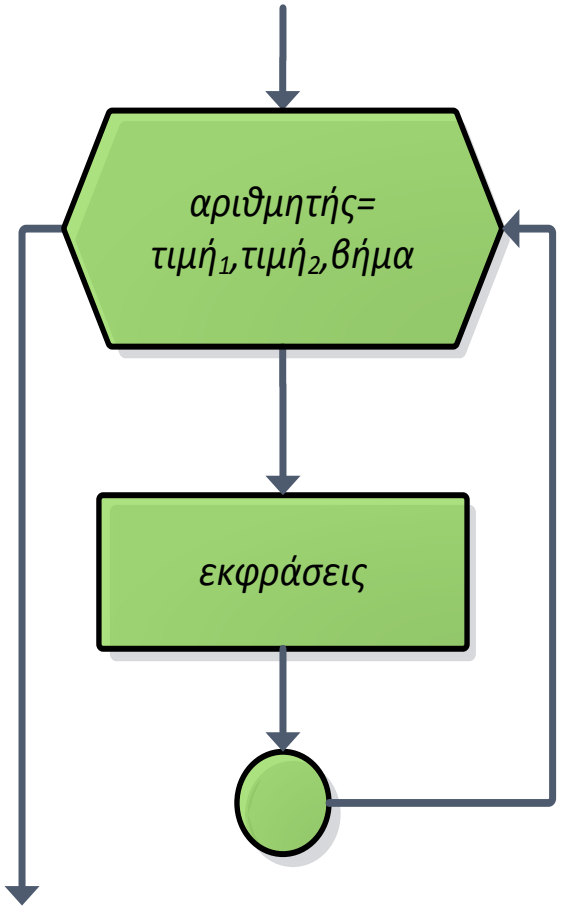
- N=0, αν τιμή$_1$ > τιμή$_2$ και βήμα >0, ***π.χ*** **DO i=5 , 1**
- ή N=0, αν τιμή$_1$ < τιμή$_2$ και βήμα <0, ***π.χ*** **DO i=1 , 5 , -1**

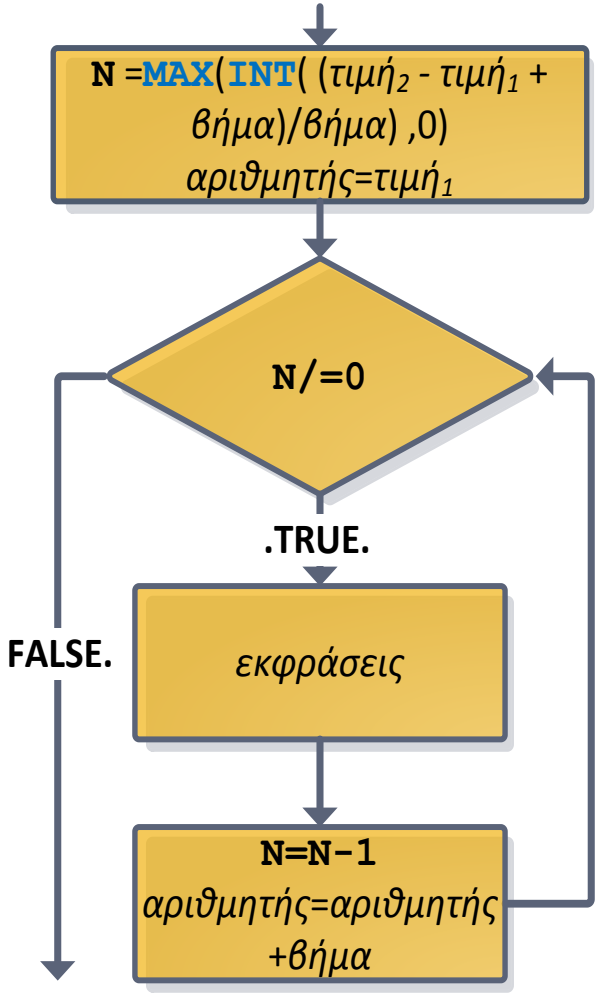
- ***Μια επανάληψη εκτελείται όσο ισχύει η συνθήκη :*** ***αριθμητής * βήμα ≤ τιμή$_2$ * βήμα***

Στο **DO** με αρίθμηση: υπολογίζεται ο αριθμός των επαναλήψεων
N=MAX (INT ((*τιμή*$_2$ − *τιμή*$_1$ + *βήμα*) / *βήμα*) , 0)

1. Στην αρχή της πρώτης επανάληψης: ***αριθμητής=τιμή$_1$***
2. Στο τέλος κάθε επανάληψης: **N=N−1** και ***αριθμητής=αριθμητής+βήμα***
3. Στην αρχή της επόμενης επανάληψης ελέγχεται η συνθήκη: **N/=0**
   a. Αν είναι αληθής γίνεται η επανάληψη
   b. Αν είναι ψευδής ο έλεγχος μεταφέρεται ακριβώς μετά από το **END DO**
4. Επομένως μετά το τέλος της επανάληψης: ***αριθμητής=τιμή$_2$+βήμα***

Πως λειτουργεί η δομή DO με αρίθμηση:





Αθροίσματα

των 10 πρώτων φυσικών αριθμών

```
S = 0
DO i = 0, 9
  S = S + i
ENDDO
```

```
S = 0; i = 0
DO
  i = i + 1
  IF(i > 9)EXIT
  S = S + i
ENDDO
```

```
S = 0; i = 1
DO WHILE(i<10)
  S = S + i
  i = i + 1
ENDDO
```

```
S = 0; i = 1
DO
  S = S + i
  i = i + 1
IF(i == 10)EXIT; ENDDO
```

Γινόμενα

το παραγοντικό του 9

```
P = 1; n = 9
DO i = 1, n
  P = P * i
ENDDO
```

```
P = 1; i = 0; n = 9
DO
  i = i + 1
  IF(i == n + 1)EXIT
  P = P * i
ENDDO
```

```
P = 1; i = 1; n = 9
DO WHILE
  P = P * i
  i = i + 1
ENDDO
```

```
P = 1; i = 1; n = 9
DO
  P = P * i
  i = i + 1
IF(i > n)EXIT; ENDDO
```

Η ενδιάμεση συνθήκη CYCLE

```
[όνομα δομής:]DO !συνήθως με αρίθμηση-αρχική πρόταση
      ....
      εκφράσεις1                    ! έκταση1
      ....
   IF(συνθήκη)CYCLE[όνομα δομής]     !πρόταση ελέγχου
      ....
      εκφράσεις2                    ! έκταση2
      ....
END DO[όνομα δομής]                          !τελική πρόταση
```

Στις δομες **DO** υπό συνθήκη με **CYCLE**:

- Αν ισχύει η συνθήκη εκτελείται μόνο η έκταση1
- Αν δεν ισχύει εκτελούνται όλα

Η ενδιάμεση συνθήκη EXIT

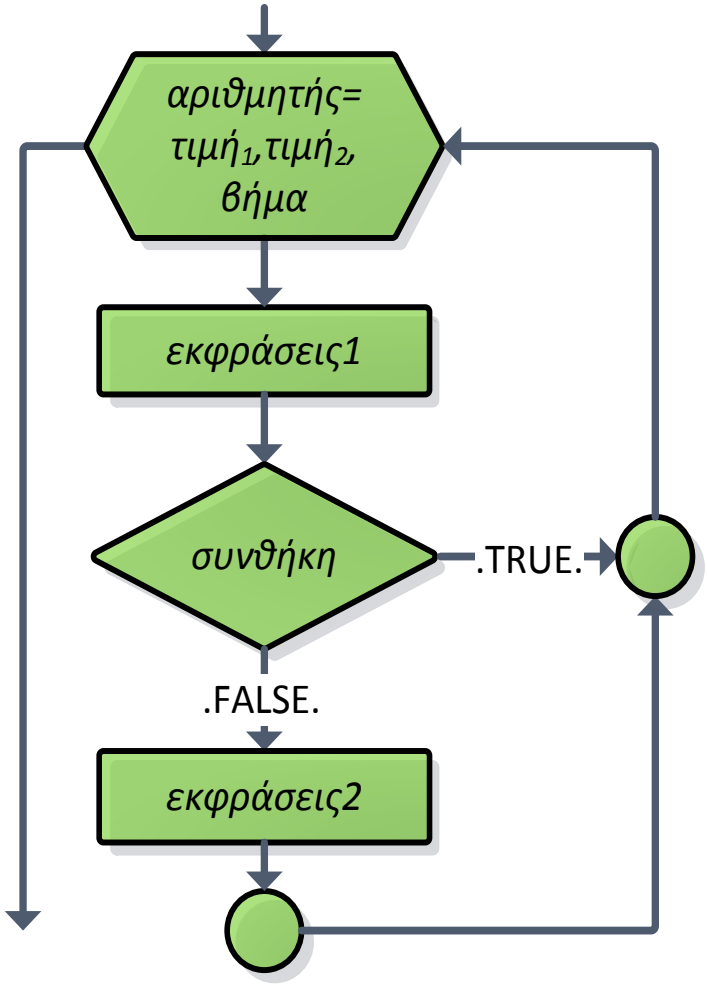
```
[όνομα δομής:]DO                                    !αρχική πρόταση
      ....
      εκφράσεις1                                    ! έκταση1
      ....
   IF(συνθήκη)EXIT[όνομα δομής]                     !πρόταση ελέγχου
      ....
      εκφράσεις2                                    ! έκταση2
      ....
END DO[όνομα δομής]                                 !τελική πρόταση
```

Στις δομες **DO** υπό συνθήκη με **EXIT**:

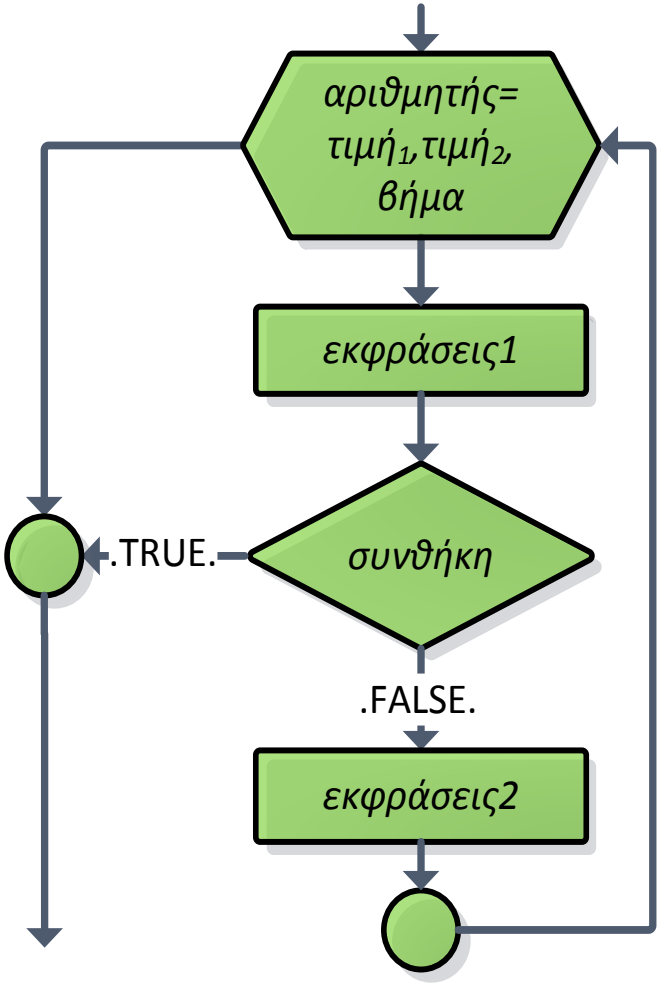
- Αν ισχύσει η συνθήκη ο έλεγχος μεταφέρεται ακριβώς μετά από το **END DO**
- Όσο δεν ισχύει εκτελούνται όλα

Πως λειτουργούν

CYCLE



EXIT



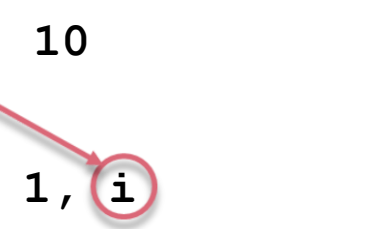
Η προπαίδεια

```
integer::a, i, j
 do i = 2, 10
  do j = 1, 10
    a = j * i
    print '($,i5)', a
  enddo
   print*
 enddo
end
```

```
    2    4    6    8   10   12   14   16   18   20
    3    6    9   12   15   18   21   24   27   30
    4    8   12   16   20   24   28   32   36   40
    5   10   15   20   25   30   35   40   45   50
    6   12   18   24   30   36   42   48   54   60
    7   14   21   28   35   42   49   56   63   70
    8   16   24   32   40   48   56   64   72   80
    9   18   27   36   45   54   63   72   81   90
   10   20   30   40   50   60   70   80   90  100
```

Παραγοντικό των αριθμών 1-10

```
integer::a, i, j
  do i = 1, 10
     a = 1
    do j = 1, i
       a = a * j
       print '($,i8)', a
    enddo
     print*
  enddo
end
```



Παραγοντικό των αριθμών 1-10

```
1
1       2
1       2       6
1       2       6       24
1       2       6       24      120
1       2       6       24      120     720
1       2       6       24      120     720     5040
1       2       6       24      120     720     5040    40320
1       2       6       24      120     720     5040    40320   362880
1       2       6       24      120     720     5040    40320   362880  3628800
```

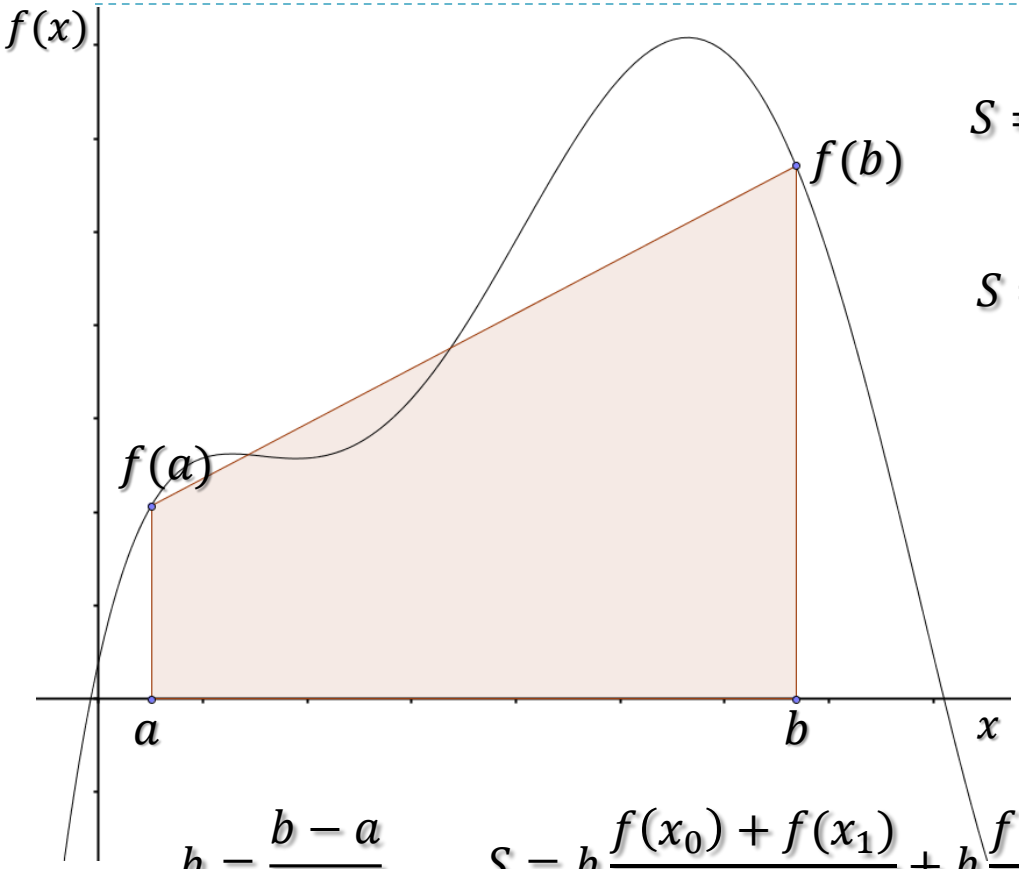
ΠΑΡΑΔΕΙΓΜΑ: Να φτιάξετε πρόγραμμα που ζητά επαναληπτικά από τον χρήστη την τιμή μιας ακεραίας μεταβλητής μέχρις ότου το άθροισμα των τιμών να γίνει μεγαλύτερο ή ίσο του 100 και τυπώνει το πλήθος των τιμών, τον μέσο όρο, την ελάχιστη τιμή και την μέγιστη τιμή. Χρησιμοποιήστε επανάληψη τύπου: *όσο…επανάλαβε*.

```
program sumto100 !Εξετάσεις Φεβ. 2011 Υ&Α άσκηση 1
    implicit none
    integer::x, s, n, min, max
    call execute_command_line('chcp 1253')
    print*,'δώσε ένα αριθμό'; read*,x
    s = x; min = x; max = x; n = 1
    do; print*,'δώσε ένα αριθμό'; read*,x
        if (s + x <= 100)then   !για να μην υπερβούμε το 100
            s = s + x           !άθροισμα
            n = n + 1           !πλήθος
            if(x > max) max = x !μέγιστος
            if(x < min) min = x !ελάχιστος
        else; exit
        endif
    enddo
    print*,'ο μέσος όρος των',n,'αριθμών είναι:',real(s)/n
    print*,'ο ελάχιστος είναι:',min
    print*,'ο μέγιστος είναι :',max
end program sumto100
```

ΠΑΡΑΔΕΙΓΜΑ: Να φτιάξετε πρόγραμμα που ζητά από τον χρήστη επαναληπτικά δύο ακέραιους αριθμούς και υπολογίζει και τυπώνει το γινόμενο τους χρησιμοποιώντας τον αλγόριθμο του Ρωσικού πολλαπλασιασμού. Ο αλγόριθμος λειτουργεί ως εξής: Για δυο δεδομένους ακέραιους αριθμούς ο πρώτος πολλαπλασιάζεται επί 2 και ο δεύτερος διαιρείται δια 2 επαναληπτικά. Σε κάθε βήμα αθροίζεται ο πρώτος αριθμός μόνο εφόσον ο δεύτερος είναι περιττός. Η επανάληψη σταματά όταν ο δεύτερος αριθμός γίνει 1. Η τιμή του αθροίσματος μας δίνει το γινόμενο των 2 αριθμών.

```
program russian !Εξετάσεις Φεβ. 2011 Υ&Α άσκηση 2
   implicit none
   integer::first, second, s
   call execute_command_line('chcp 1253')
  do
     print*,'ΔΩΣΕ ΔΥΟ ΑΚΕΡΑΙΟΥΣ ΑΡΙΘΜΟΥΣ'
     read*,first, second
     s = 0
    do while(second > 0)
       if(mod(second,2) /= 0)s = s + first
       first  = first  * 2
       second = second / 2
    enddo
     print*,'ΤΟ ΓΙΝΟΜΕΝΟ ΕΙΝΑΙ:', s
  enddo
end program russian
```

Η μέθοδος του τραπεζίου



$$S = \int_a^b \left[f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \right] dx$$

$$S = (b - a)\frac{f(a) + f(b)}{2}$$

$$h = \frac{b - a}{n} \qquad S = h\frac{f(x_0) + f(x_1)}{2} + h\frac{f(x_1) + f(x_2)}{2} + \cdots + h\frac{f(x_{n-1}) + f(x_n)}{2}$$

$$S = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n) \right]$$

Η μέθοδος του τραπεζίου

αριθμητικός υπολογισμός του ορισμένου ολοκληρώματος μιας συνάρτησης

```
program trapezoid          !έστω για τη συνάρτηση sin(x)
   implicit none
   real(8) ::a = 0, b = 4 * atan(1._8)
   integer ::n = 1000
   real(8) ::h, s, area
   integer ::i

   h = (b - a) / real(n,8)
   s = (sin(a) + sin(b))/2

   do i = 1, n-1
      s = s + sin(a + i * h)
   end do

   area = h * s

   print *,'for n=', n
   print *,'the integral of the function is:', area

end program trapezoid
```

$$\frac{f(x_0)+f(x_n)}{2}$$

$$\sum_{i=1}^{n-1} f(x_i)$$

$$S = h\left[\frac{f(x_0)+f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right]$$

Η μέθοδος του τραπεζίου

δυο στιγμιότυπα

```
 for n=        1000
 the integral of the function is:   1.9999983550656624

Process returned 0 (0x0)   execution time : 0.039 s
```

```
 for n=   100000000
 the integral of the function is:   2.0000000000003761

Process returned 0 (0x0)   execution time : 6.397 s
```

Βρίσκω μια ρίζα της εξίσωσης $y = x^3 - x^2 - 2$ χωρίς μαθηματικά!

```
program greedy
    implicit none

!   δηλώσεις:
    real::xa = 1., xb = 2.!όρια
    real::step = 1e-7      !βήμα
    real::x, y

!   αρχή:
    x = xa - step
    do while (x < xb)
        x = x + step          !ανεξάρτητη μεταβλητή
        y = x**3 - x**2 - 2 !εξαρτημένη μεταβλητή
        if(abs(y)<1e-6)exit !κριτήριο
    enddo
    print *, 'y=', y, 'at x=', x

end program greedy
```

```
 y=  -6.46507317E-07 at x=   1.69562066

Process returned 0 (0x0)   execution time : 0.080 s
```

Πρώτοι αριθμοί

```
program primes
   implicit none
!  δηλώσεις:
   integer:: n, p
!  αρχή:
   endless: do
      print *, 'give an integer greater than 2'
       read *, n; if(n <= 2) exit; p = 2
        in: do
         test: if (mod(n,p)==0) then
          print *, n, 'is NOT a prime =', p, '     x', n/p
          print *; exit in
         elseif (p * p > n)      then test
          print *, n, 'is a prime number'
          print *; exit in
         endif test
          p = p + 1
        enddo in
   enddo endless
end program primes
```

Σημείωμα Αναφοράς

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημείωμα Αδειοδότησης