Αναδρομικός αλγόριθμος

Ένας αναδρομικός αλγόριθμος

- επιλύει ένα πρόβλημα για κάποιες τιμές δεδομένων λύνοντας το ίδιο πρόβλημα για άλλες (σχετιζόμενες) τιμές δεδομένων. Είναι απαραίτητο βέβαια να λύνεται το πρόβλημα απευθείας για κάποιες συγκεκριμένες τιμές.

Αναδρομικός αλγόριθμος

Ένας αναδρομικός αλγόριθμος

- επιλύει ένα πρόβλημα για κάποιες τιμές δεδομένων λύνοντας το ίδιο πρόβλημα για άλλες (σχετιζόμενες) τιμές δεδομένων. Είναι απαραίτητο βέβαια να λύνεται το πρόβλημα απευθείας για κάποιες συγκεκριμένες τιμές.
- είναι συχνά πιο αποδοτικός (εύκολος, γρήγορος, απλός) από τον ισοδύναμο μη αναδρομικό.

Αναδρομικός αλγόριθμος

Ένας αναδρομικός αλγόριθμος

- επιλύει ένα πρόβλημα για κάποιες τιμές δεδομένων λύνοντας το ίδιο πρόβλημα για άλλες (σχετιζόμενες) τιμές δεδομένων. Είναι απαραίτητο βέβαια να λύνεται το πρόβλημα απευθείας για κάποιες συγκεκριμένες τιμές.
- είναι συχνά πιο αποδοτικός (εύκολος, γρήγορος, απλός) από τον ισοδύναμο μη αναδρομικό.
- υλοποιείται με υποπρόγραμμα που πρέπει να καλεί τον εαυτό του.

Αναδρομικός αλγόριθμος

Ένας αναδρομικός αλγόριθμος

- επιλύει ένα πρόβλημα για κάποιες τιμές δεδομένων λύνοντας το ίδιο πρόβλημα για άλλες (σχετιζόμενες) τιμές δεδομένων. Είναι απαραίτητο βέβαια να λύνεται το πρόβλημα απευθείας για κάποιες συγκεκριμένες τιμές.
- είναι συχνά πιο αποδοτικός (εύκολος, γρήγορος, απλός) από τον ισοδύναμο μη αναδρομικό.
- υλοποιείται με υποπρόγραμμα που πρέπει να καλεί τον εαυτό του.

Στη Fortran 95 τα υποπρογράμματα έχουν τη δυνατότητα να καλούν τον εαυτό τους, αν τροποποιηθούν κατάλληλα.

Παράδειγμα: παραγοντικό

Δύο ισοδύναμοι ορισμοί για το παραγοντικό ακέραιου μη αρνητικού αριθμού:

Μη αναδρομικός

$$n! = \begin{cases} 1 \times 2 \times \cdots \times (n-1) \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

Αναδρομικός

$$n! = \begin{cases} (n-1)! \times n\,, & n > 0\,, \\ 1\,, & n = 0\,. \end{cases}$$

Αναδρομική συνάρτηση παραγοντικού

Όχι απόλυτα σωστός κώδικας

Στις εντολές γράφουμε ακριβώς ό,τι μας λέει ο μαθηματικός ορισμός:

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Αναδρομική συνάρτηση παραγοντικού

Όχι απόλυτα σωστός κώδικας

Στις εντολές γράφουμε ακριβώς ό,τι μας λέει ο μαθηματικός ορισμός:

```
FUNCTION par(n)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: par

  IF (n >  0) par = par(n-1) * n
  IF (n == 0) par = 1
END FUNCTION par
```

Ο παραπάνω κώδικας δεν είναι ΑΠΟΛΥΤΑ ΣΩΣΤΟΣ.
Χρειάζεται τροποποίηση. Ας δούμε πρώτα πώς λειτουργεί.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με n=2.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).
- Το αποτέλεσμα της `par(0)` είναι `1`. Αυτό επιστρέφεται.

Αναδρομική συνάρτηση παραγοντικού (2/2)

Πώς λειτουργεί

Αν γράψουμε `par(2)` ο compiler εκτελεί την par με `n=2`.

- Το αποτέλεσμα της `par(2)` είναι `par(1)*2`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=1`).
- Το αποτέλεσμα της `par(1)` είναι `par(0)*1`.
  Δεν μπορεί να το επιστρέψει: καλείται μια συνάρτηση (n par για `n=0`).
- Το αποτέλεσμα της `par(0)` είναι `1`. Αυτό επιστρέφεται.

Άρα

*par(2) → par(1)*2 → (par(0)*1)*2 → (1*1)*2 = 2!*

Γενικός ορισμός αναδρομικής συνάρτησης

Στον ορισμό (και τη δήλωση) της συνάρτησης πρέπει δηλώσουμε ότι είναι αναδρομική και να διαχωρίσουμε το όνομα με το οποίο γίνεται η κλήση της συνάρτησης από τη μεταβλητή του ονόματος της συνάρτησης:

```
RECURSIVE FUNCTION όνομα(παράμετροςΑ, παράμετροςΒ,...) &
RESULT (όνομα2)
    IMPLICIT NONE
    τύπος_παραμέτρου_Α, INTENT (xxx) :: παράμετροςΑ
    τύπος_παραμέτρου_Β, INTENT (yyy) :: παράμετροςΒ
    τύπος_επιστρεφόμενης_ποσότητας :: όνομα2

    τύπος_Α :: τοπική_μεταβλητή_Α, ...
    τύπος_Β :: τοπική_μεταβλητή_Β, ...

    ..... ! κώδικας
    όνομα2 = .....
END FUNCTION όνομα
```

Αναδρομική συνάρτηση παραγοντικού

Τροποποίηση του ορισμού

```
RECURSIVE FUNCTION par(n) RESULT(p)
  IMPLICIT NONE
  INTEGER, INTENT (in) :: n
  INTEGER :: p

  IF (n >  0) p = par(n-1) * n
  IF (n == 0) p = 1
END FUNCTION par
```

Ο παραπάνω κώδικας είναι ΣΩΣΤΟΣ.

Γενικός ορισμός αναδρομικής υπορουτίνας

Σε υπορουτίνα που καλεί τον εαυτό της χρειάζεται μόνο η προσθήκη της λέξης **RECURSIVE** πριν το **SUBROUTINE**:

```
RECURSIVE SUBROUTINE όνομα(παράμετροςA, παράμετροςB,...)
    IMPLICIT NONE
    τύπος_παραμέτρου_A, INTENT (xxx) :: παράμετροςA
    τύπος_παραμέτρου_B, INTENT (yyy) :: παράμετροςB

    τύπος_A :: τοπική_μεταβλητή_A, ...
    τύπος_B :: τοπική_μεταβλητή_B, ...

    ..... ! κώδικας
END SUBROUTINE όνομα
```

Παράδειγμα: πολυώνυμα Hermite (1/2)

Στη Μαθηματική Φυσική χρησιμοποιείται η οικογένεια πολυωνύμων Hermite:

$$\begin{aligned} H_0(x) &= 1 \,, \\ H_1(x) &= 2x \,, \\ H_2(x) &= 4x^2 - 2 \,, \\ H_3(x) &= 8x^3 - 12x \,, \\ H_4(x) &= 16x^4 - 48x^2 + 12 \,, \\ &\vdots \end{aligned}$$

Τα πολυώνυμα $H_n(x)$ ικανοποιούν την αναδρομική σχέση:

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \,, \qquad n \geq 2 \,.$$

Παράδειγμα: πολυώνυμα Hermite (2/2)

Αναδρομική συνάρτηση που υπολογίζει τα $H_n(x)$:

```
RECURSIVE FUNCTION hermite(n,x) RESULT (h)
  IMPLICIT NONE
  INTEGER,          INTENT (in) :: n
  DOUBLE PRECISION, INTENT (in) :: x
  DOUBLE PRECISION              :: h

  IF (n==0) h = 1.0d0
  IF (n==1) h = 2.0d0 * x
  IF (n>=2) h = 2.0d0 * &
    (x * hermite(n-1,x) - (n-1) * hermite(n-2,x))
END FUNCTION hermite
```

Υποπρογράμματα κατά στοιχείο (ELEMENTAL)

Στις ενσωματωμένες συναρτήσεις το όρισμα μπορεί να είναι μία τιμή αλλά επιτρέπεται να είναι και διάνυσμα:

```
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
a = 3.2d0
y = SQRT(x)
b = SQRT(a)
```

Επιθυμούμε το ίδιο να μπορεί να γίνει και σε δικά μας υποπρογράμματα που δέχονται απλές μεταβλητές. Γι' αυτό συμπληρώνουμε τη δήλωση με το **ELEMENTAL** πριν τη λέξη **FUNCTION** ή **SUBROUTINE**.

Παράδειγμα υπορουτίνας ELEMENTAL (1/2)

Ορισμός

```
ELEMENTAL SUBROUTINE swap(a,b)
   IMPLICIT NONE
   DOUBLE PRECISION, INTENT (inout) :: a,b

   DOUBLE PRECISION                 :: c

   c = a
   a = b
   b = c
END SUBROUTINE swap
```

Παράδειγμα υπορουτίνας ELEMENTAL (2/2)

Δήλωση και κλήση

```
INTERFACE
   ELEMENTAL SUBROUTINE swap(a,b)
      IMPLICIT NONE
      DOUBLE PRECISION, INTENT (inout) :: a,b
   END SUBROUTINE swap
END INTERFACE
DOUBLE PRECISION :: x,y, a(100), b(100)
x = 2.6d0
y = 1.3d0
a = 3.2d0
b = 5.3d0
CALL swap(x,y)
CALL swap(a,b)
```

Υπορουτίνα παραγωγής τυχαίων αριθμών (1/2)

Η υπορουτίνα **RANDOM_NUMBER** δέχεται μια πραγματική μεταβλητή. Κάθε φορά που καλείται αποδίδει στο όρισμα τυχαία τιμή στο διάστημα $[0, 1)$.
Η υπορουτίνα είναι **ELEMENTAL** άρα δέχεται και διάνυσμα πραγματικών στους οποίους αποδίδει τυχαίες τιμές.

Υπορουτίνα παραγωγής τυχαίων αριθμών (1/2)

Η υπορουτίνα **RANDOM_NUMBER** δέχεται μια πραγματική μεταβλητή. Κάθε φορά που καλείται αποδίδει στο όρισμα τυχαία τιμή στο διάστημα $[0, 1)$.
Η υπορουτίνα είναι **ELEMENTAL** άρα δέχεται και διάνυσμα πραγματικών στους οποίους αποδίδει τυχαίες τιμές.

Παράδειγμα

```
DOUBLE PRECISION :: x, y, z(100)
CALL RANDOM_NUMBER(x)  ! x τυχαίος
CALL RANDOM_NUMBER(y)  ! y άλλος τυχαίος
CALL RANDOM_NUMBER(z)  ! z άλλοι 100 τυχαίοι
```

Υπορουτίνα παραγωγής τυχαίων αριθμών (2/2)

Αλλαγή διαστήματος

Αν r είναι τυχαίος πραγματικός αριθμός στο $[0, 1)$, τότε κάνουμε τη μετατροπή $x = \kappa r + \lambda$ και επιλέγουμε τους συντελεστές κ, λ ώστε η ποσότητα x να βρίσκεται εντός των επιθυμητών ορίων.
Εύκολα επαληθεύεται ότι ο $x = (b - a)r + a$ ικανοποιεί τις σχέσεις $a \leq x < b$, άρα ο x είναι τυχαίος πραγματικός στο διάστημα $[a, b)$.