Πανεπιστήμιο Αιγαίου

Εισαγωγή στην Πληροφορική

Ενότητα 9: Διαδικασίες

Ανδρέας Παπασαλούρος

Τμήμα Μαθηματικών

Σάμος, Ιούνιος 2015

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη 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.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Τα προγράμματα που έχουμε μελετήσει ως τώρα αποτελούνται από μια ενιαία ακολουθία εντολών. Στην πράξη, μόνο πολύ μικρά (τετριμμένα) προγράμματα κατασκευάζονται με αυτό τον τρόπο. Σχεδόν κάθε πρόγραμμα αποτελείται από ένα κύριο πρόγραμμα, το οποίο περικλείεται από τις εντολές `program` και `end program`, και ένα σύνολο από *τμήματα προγραμμάτων* τα οποία ονομάζονται *υποπρογράμματα* και τα οποία συνενώνονται μεταξύ τους σχηματίζοντας το πλήρες πρόγραμμα. Τα υποπρογράμματα είναι τα θεμελιώδη *δομικά συστατικά* ενός προγράμματος.

Ένα υποπρόγραμμα είναι μια ακολουθία εντολών οι οποίες επιτελούν μια συγκεκριμένη, *καλώς ορισμένη* λειτουργία.
Το *κύριο πρόγραμμα καλεί* ένα υποπρόγραμμα με αποτέλεσμα ο *έλεγχος του προγράμματος* να μεταβιβάζεται στις εντολές του υποπρογράμματος. Το υποπρόγραμμα ανταλλάσσει δεδομένα με το κύριο πρόγραμμα από το οποίο καλείται με μηχανισμό ο οποίος περιγράφεται στη συνέχεια.
Στη Fortran ορίζονται δύο είδη υποπρογραμμάτων: *συναρτήσεις* (functions) και *υπορουτίνες*[1].

[1] Στην ορολογία της Fortran τόσο οι συναρτήσεις όσο και οι υπορουτίνες ονομάζονται *διαδικασίες*. Σε άλλες γλώσσες προγραμματισμού ο όρος *διαδικασία* αντιστοιχεί στις υπορουτίνες της Fortran

Μια συνάρτηση εκτελεί κάποιο υπολογισμό και *επιστρέφει* (return) κάποια τιμή ως αποτέλεσμα του υπολογισμού. Έχουμε ήδη χρησιμοποιήσει τις εγγενείς συναρτήσεις της Fortran , όπως οι COS και SQRT. Ένα παράδειγμα συνάρτησης δίνεται στη συνέχεια.

```
FUNCTION circle_area(r)
  IMPLICIT NONE
  ! Δήλωση του τύπου της τιμής της συνάρτησης
  REAL:: circle_area
  ! Δήλωση του τύπου της παραμέτρου της συνάρτησης
  REAL:: r
  ! Δήλωση τοπικών μεταβλητών και επώνυμων σταθερών
  REAL, PARAMETER :: pi = 3.14159

 ! Ορισμός της τιμής που επιστρέφει η συνάρτηση
  circle_area = pi * r ** 2
END FUNCTION circle_area
```

Το παραπάνω παράδειγμα αποτελεί τον *ορισμό* της συνάρτησης με όνομα `circle_area`. Παρατηρήστε ότι η δομή της συνάρτησης μοιάζει με δομή του κύριου προγράμματος.
Η συνάρτηση `circle_area` υπολογίζει το εμβαδό της επιφάνειας ενός κύκλου με ακτίνα `r`. Η εντολή

```
FUNCTION circle_area(r)
```

αποτελεί την *επικεφαλίδα* και σηματοδοτεί την αρχή του κώδικα της συνάρτησης. Η επικεφαλίδα περιλαμβάνει το όνομα της συνάρτησης καθώς και μια προαιρετική λίστα *παραμέτρων* για τις οποίες θα γίνει λόγος στη συνέχεια.

Παράμετροι

Συγκεκριμένα, στο παράδειγμα η μεταβλητή `r` χρησιμοποιείται για το πέρασμα της τιμής της ακτίνας για την οποία ζητείται ο υπολογισμός του εμβαδού. Τέτοιες μεταβλητές μέσω των οποίων ένα υποπρόγραμμα ανταλλάσσει δεδομένα με την μονάδα, πρόγραμμα ή άλλη διαδικασία, που την καλεί ονομάζονται *τυπικές παράμετροι* (formal ή dummy parameters). Ένα υποπρόγραμμα είναι δυνατόν να έχει μηδέν ή περισσότερες παραμέτρους. Οι παράμετροι δηλώνονται όπως και οι μεταβλητές. Η εντολή

```
REAL :: r
```

αποτελεί δήλωση παραμέτρου `r` ως τύπου `real`.

Επιστρεφόμενη τιμή συνάρτησης

Η τιμή που επιστρέφεται από τη συνάρτηση ορίζεται με την εντολή

```
circle_area = 3.14159 * r ** 2
```

η οποία είναι μια εντολή ανάθεσης σε μια μεταβλητή με όνομα *ίδιο με το όνομα της συνάρτησης*. Η μεταβλητή αυτή πρέπει να δηλωθεί στο τμήμα δηλώσεων της συνάρτησης ώστε να είναι γνωστός ο *τύπος* της τιμής της συνάρτησης (π.χ. `real`, `integer`, κ.λπ.). Στο παράδειγμα, η συνάρτηση επιστρέφει μια πραγματική τιμή όπως καθορίζεται από τη δήλωση

```
REAL :: circle_area
```

Τοπικές μεταβλητές

Στο τμήμα δηλώσεων μιας διαδικασίας είναι δυνατόν να δηλώνονται μεταβλητές οι οποίες χρησιμοποιούνται αποκλειστικά από τον κώδικα της διαδικασίας
Τέτοιες μεταβλητές ονομάζονται *τοπικές* γιατί η *εμβέλειά* τους περιορίζεται στον κώδικα της διαδικασίας στην οποία ορίζονται και δεν μπορούν να χρησιμοποιηθούν έξω από αυτόν.
Η οδηγία `implicit none` πρέπει να αναφέρεται σε κάθε διαδικασία ενός προγράμματος, ακόμη και αν έχει ήδη δηλωθεί στο κύριο πρόγραμμα.

Κλήση συνάρτησης

Μια συνάρτηση καλείται με τον ίδιο τρόπο με τις εγγενείς συναρτήσεις της Fortran. Έτσι, η παραπάνω συνάρτηση καλείται από το κύριο πρόγραμμα ή κάποια άλλη μονάδα του προγράμματος ως εξής

```
area = circle_area(2.5)
```

με αποτέλεσμα την ανάθεση της τιμής που επιστρέφει η συνάρτηση στη μεταβλητή `area`. Η τιμή `2.5` η οποία 'αντικαθιστά' την τυπική παράμετρο κατά την κλήση της συνάρτησης ονομάζεται *πραγματική παράμετρος*. Η σειρά και ο τύπος των πραγματικών παραμέτρων πρέπει να συμφωνεί με τη σειρά και τον τύπο των τυπικών παραμέτρων. Οι συγκεκριμένες παράμετροι είναι δυνατόν να είναι μεταβλητές, σταθερές ή παραστάσεις με τύπο ίδιο με τον τύπο της αντίστοιχης τυπικής παραμέτρου.

Στη γενική περίπτωση, η δομή μιας συνάρτησης είναι η ακόλουθη:

```
FUNCTION ονομα_συνάρτησης (παράμετρος, ...)
  Δηλώσεις μεταβλητών, παραμέτρων και επιστρεφόμενης τιμής
  εκτελέσιμες εντολές
END [FUNCTION [όνομα_συνάρτησης]]
```

Μέσα στο σώμα των εκτελέσιμων εντολών μιας συνάρτησης πρέπει να υπάρχει ανάθεση σε μεταβλητή με το όνομα της συνάρτησης όπως η

```
  circle_area = 3.14159 * r ** 2
```

για τη συνάρτηση `circle_area`.

Η εντολή return

Η εντολή `return` τοποθετείται στον κώδικα ενός υποπρογράμματος και έχει ως αποτέλεσμα τον τερματισμό της εκτέλεσης του υποπρογράμματος και την συνέχιση της εκτέλεσης του κυρίου προγράμματος, από το σημείο στο οποίο έγινε η κλήση της διαδικασίας.

Παράδειγμα χρήσης της return είναι το ακόλουθο:

```
FUNCTION absolute(x)
IMPLICIT NONE
  REAL:: absolute, x

  IF (x >= 0) THEN
    absolute = x
    RETURN
  END IF
  absolute = -x
END FUNCTION absolute
```

Στην παραπάνω συνάρτηση υπολογισμού της μέσης τιμής, αν ο αριθμός είναι μη αρνητικός, μετά την ανάθεση absolute = x η συνάρτηση θα τερματιστεί και ο έλεγχος του προγράμματος θα επιστρέψει στο πρόγραμμα/υποπρόγραμμα το οποίο την κάλεσε χωρίς να εκτελεστεί η εντολή absolute = -x.

Υπορουτίνες

Οι υπορουτίνες διαφέρουν από τις συναρτήσεις στο ότι δεν επιστρέφουν μια συγκεκριμένη τιμή, αλλά ανταλλάσσουν δεδομένα, μέσω των παραμέτρων τους, με τις μονάδες που τις καλούν, όπως φαίνεται στο παράδειγμα που ακολουθεί.

```
SUBROUTINE swap (x, y)
  IMPLICIT NONE
  ! Δήλωση τυπικών παραμέτρων
  REAL :: x, y
  ! Δήλωση τοπικής μεταβλητής
  REAL :: temp

  ! Αντιμετάθεση των τιμών των μεταβλητών x και y
  temp = x
  x = y
  y = temp
END SUBROUTINE swap
```

Υπορουτίνες

Η υπορουτίνα swap ανταλλάσσει τις τιμές των μεταβλητών που δέχεται ως παραμέτρους. Η κλήση της από κάποια άλλη μονάδα προγράμματος γίνεται ως εξής:

```
REAL :: a, b
a = 1; b = 2
CALL swap(a,b)
```

Μετά την κλήση της swap με τον παραπάνω τρόπο, οι τιμές των μεταβλητών a, b εναλλάσσονται ώστε το a γίνεται 2 ενώ το b γίνεται 1.

Δομή μιας υπορουτίνας

Η γενική δομή μιας υπορουτίνας είναι η ακόλουθη:

```
SUBROUTINE ονομα_διαδικασίας (παράμετρος, ...)

  Δηλώσεις μεταβλητών και παραμέτρων

  Εκτελέσιμες εντολές

END [SUBROUTINE [όνομα_διαδικασίας]]
```

Η κλήση μιας διαδικασίας γίνεται με την εντολή `call`:

```
  CALL όνομα_υπορουτίνας(παράμετρος, ...)
```

Στη συνέχεια δίνεται ένα παράδειγμά μιας υπορουτίνας η οποία δέχεται ως παράμετρο το έτος γέννησης και το τρέχον έτος και τυπώνει την ηλικία ενός ατόμου.

```
SUBROUTINE printAge(year, now)
  INTEGER :: year, now, dif

  dif = now - year
  PRINT*, 'You are ', dif, ' years old'

END SUBROUTINE printAge
```

Η παραπάνω διαδικασία printAge έχει δύο *τυπικές παραμέτρους*, τα year και now. Η κλήση της παραπάνω διαδικασίας γίνεται με μια εντολή της μορφής:

```
CALL printAge(1985, 2007)
```

Παράμετροι εισόδου–εξόδου

Όπως αναφέρθηκε προηγουμένως, οι τυπικές παράμετροι χρησιμοποιούνται για την ανταλλαγή δεδομένων μεταξύ μιας διαδικασίας και του προγράμματος το οποίο την καλεί. Με αυτή την έννοια, μια παράμετρος μιας διαδικασίας είναι δυνατόν να χρησιμοποιηθεί για το πέρασμα δεδομένων *από* το πρόγραμμα που καλεί μια διαδικασία προς τη διαδικασία (παράμετρος εισόδου) ή, αντίστροφα, για την επιστροφή δεδομένων από τη διαδικασία προς το πρόγραμμα που την κάλεσε (παράμετρος εξόδου).

Παράμετροι εισόδου–εξόδου

Μια παράμετρος σε μια διαδικασία, είτε συνάρτηση είτε υπορουτίνα, είναι ταυτόχρονα παράμετρος εισόδου και εξόδου. Για παράδειγμα, οι παράμετροι της διαδικασίας `swap` που παρουσιάστηκε προηγουμένως χρησιμοποιούνται τόσο για τη μεταβίβαση δεδομένων στη `swap`, όσο και για τη λήψη δεδομένων από αυτή. Αυτό το χαρακτηριστικό του μηχανισμού περάσματος παραμέτρων της Fortran είναι δυνατόν να οδηγήσει σε ανεπιθύμητες *παρενέργειες* κατά την κλήση μιας διαδικασίας.

Έτσι, η κλήση μιας διαδικασίας είναι δυνατόν να προκαλέσει μια ανεπιθύμητη αλλαγή στην τιμή μιας παραμέτρου της, οδηγώντας σε σφάλμα που είναι δύσκολο να ανιχνευτεί σε ένα πρόγραμμα.
Η Fortran παρέχει ένα μηχανισμό για την *ρητή* δήλωση μιας τυπικής παραμέτρου ως παραμέτρου εισόδου ή εξόδου. Αυτό γίνεται με χρήση της λέξης `intent` κατά τη δήλωση τύπου μιας τυπικής παραμέτρου. Η `intent` μπορεί να έχει τις εξής μορφές:

Η λέξη intent

`intent(in)`	Η τυπική παράμετρος χρησιμοποιείται ως παράμετρος *εισόδου*
`intent(out)`	Η τυπική παράμετρος χρησιμοποιείται ως παράμετρος *εξόδου*
`intent(inout)`	Η τυπική παράμετρος χρησιμοποιείται ως παράμετρος *εισόδου και εξόδου*

Η λέξη intent – Παράδειγμα

Το παρακάτω παράδειγμα δείχνει τη χρήση της intent:

```
SUBROUTINE mysub(inputArg, outputArg)
  REAL, INTENT(IN) :: inputArg
  REAL, INTENT(OUT):: outputArg

  outputArg = 2. * inputArg
  inputArg = 1.5 ! Λανθασμένη εντολή
END SUBROUTINE mysub
```

Σύμφωνα με τα παραπάνω, η υπορουτίνα swap είναι ισοδύναμη με την ακόλουθη:

```
SUBROUTINE swap2 (x, y)
  IMPLICIT NONE
  ! Δήλωση τυπικών παραμέτρων
  REAL, INTENT(INOUT):: x, y
  ! Δήλωση τοπικής μεταβλητής
  REAL :: temp
  temp = x
  x = y
  y = temp
END SUBROUTINE swap2
```

Δομή προγράμματος

Ένα *εκτελέσιμο* πρόγραμμα, δηλ. ένα πρόγραμμα το οποίο μπορεί ο χρήστης να 'τρέξει' σε έναν υπολογιστή, διαθέτει τουλάχιστον το *κύριο πρόγραμμα* και ένα σύνολο από υποπρογράμματα, συναρτήσεις ή υπορουτίνες. Ένα πρόγραμμα αποτελείται από τα εξής:
Το κύριο πρόγραμμα (main program), το οποίο όπως έχει ήδη αναφερθεί ορίζεται ως εξής:

```
PROGRAM Όνομα_προγράμματος
Εντολές...
END PROGRAM Όνομα_προγράμματος
```

Εξωτερικά (external) υποπρογράμματα

Τα εξωτερικά υποπρογράμματα ορίζονται έξω από το σώμα ενός προγράμματος, δηλαδή έξω από τον κώδικα μεταξύ των εντολών `program` και `end program`. Ένα εξωτερικό υποπρόγραμμα είναι δυνατόν να ορίζεται στο ίδιο ή σε διαφορετικό αρχείο πηγαίου κώδικα με το κύριο πρόγραμμα.

Εξωτερικά υποπρογράμματα

```
PROGRAM external_test

  !Δήλωση του τύπου εξωτερικής συνάρτησης
  REAL :: afunction
  ...
END PROGRAM external_test

SUBROUTINE something
 ...
END subourine something

FUNCTION afunction
REAL :: afunction
...
END FUNCTION afunction
```

Εσωτερικά (internal) υποπρογράμματα

Τα εσωτερικά υποπρογράμματα ορίζονται *μέσα* στο σώμα ενός κυρίου προγράμματος, χρησιμοποιώντας την λέξη `contains`. Η λέξη κλειδί `contains`

Η δομή ενός προγράμματος με εξωτερικά υποπρογράμματα είναι η ακόλουθη:

Εσωτερικά (internal) υποπρογράμματα

```
PROGRAM internal
 ! ...

CONTAINS
 ! Ορισμός εσωτερικών διαδικασιών και συναρτήσεων

  SUBROUTINE something
   ! ...
  END SUBROUTINE something

  FUNCTION afunction
    REAL :: afunction
    ! ...
  END FUNCTION afunction

END PROGRAM internal
```

Υποπρογράμματα μονάδων (module procedures)

Μια *μονάδα* (module) είναι μια *ενότητα* προγράμματος η οποία μπορεί να περιέχει δηλώσεις μεταβλητών και υποπρογραμμάτων τα οποία είναι δυνατόν να χρησιμοποιούνται από άλλα υποπρογράμματα του ίδιου προγράμματος, ή ακόμη και από διαφορετικά προγράμματα. Μια μονάδα περιέχει σχετιζόμενες διαδικασίες, π.χ. μαθηματικές συναρτήσεις, οι οποίες πρόκειται να χρησιμοποιηθούν από ένα ή περισσότερα προγράμματα.

Η μορφή μιας μονάδας είναι η ακόλουθη:

Υποπρογράμματα μονάδων (module procedures)

```
MODULE όνομα_μονάδας
  !Δηλώσεις...
   CONTAINS
  !Δηλώσεις διαδικασιών
END MODULE όνομα_μονάδας
```

Παράδειγμα (module procedure)

Στη συνέχεια δίνεται ο ορισμός της συνάρτησης circle_area μέσα σε μια μονάδα με όνομα geometry_module:

```
MODULE geometry_module
  ! Τμήμα δηλώσεων μονάδας
  REAL, PARAMETER :: pi = 3.14159

  ! Τμήμα υποπρογραμμάτων μονάδας

CONTAINS

 FUNCTION circle_area(r)
   IMPLICIT NONE
   REAL :: circle_area
   REAL :: r

   circle_area = pi * r ** 2
```

Υποπρογράμματα μονάδων (module procedures)

Ένα κύριο πρόγραμμα ή υποπρόγραμμα χρησιμοποιεί τις διαδικασίες μιας μονάδας μέσω μιας δήλωσης use, όπως φαίνεται στο παρακάτω πρόγραμμα, το οποίο καλεί τη συνάρτηση circle_area της μονάδας geometry_module.

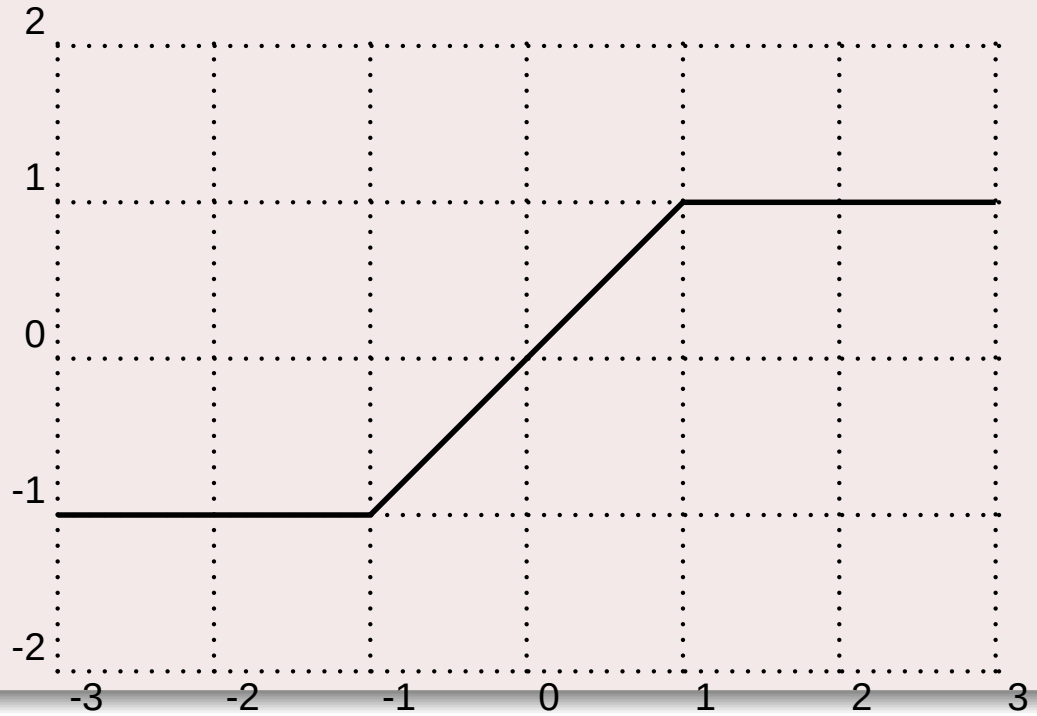
```
PROGRAM geometry_program
USE geometry_module
IMPLICIT NONE
  REAL :: a
  a = 1.0

  PRINT*, 'Το εμβαδόν είναι:', circle_area(a)
END PROGRAM geometry_program
```

Η δήλωση use πρέπει να προηγείται όλων των άλλων εντολών μέσα στο πρόγραμμα ή υποπρόγραμμα.

Παράδειγμα

Να γραφεί συνάρτηση `f(x)` στη Fortran η οποία υλοποιεί την μαθηματική συνάρτηση που παριστάνει το παρακάτω διάγραμμα:



Λύση

Η συνάρτηση f παίρνει τις παρακάτω τιμές:

$$f(x) = \begin{cases} -1 & \text{αν } x \leq -1 \\ x & \text{αν } x > -1 \text{ και } x < 1 \\ 1 & \text{αν } x \geq 1 \end{cases}$$

Λύση (συν.)

Με βάση τα παραπάνω, ο κώδικας ο οποίος δίνει την αντίστοιχη τιμή στην συνάρτηση f είναι ο ακόλουθος:

```
IF (x < -1) THEN
  f = -1
ELSE IF (x < 1) THEN
  f = x
ELSE
  f = 1
END IF
```

Λύση (συν.)

Η συνάρτηση έχει μια παράμετρο, x, τύπου real και επιστρέφει τιμή επίσης τύπου real. Ο πλήρης κώδικας της f δίνεται στη συνέχεια:

```
FUNCTION f(x)
IMPLICIT NONE
  REAL:: f,x

  IF (x <= -1) THEN
    f = -1
  ELSE IF (x < 1) THEN
    f = x
  ELSE
    f = 1
  END IF
END FUNCTION f
```

Λύση (συν.)

Ένα *πρόγραμμα* το οποίο καλεί την παραπάνω συνάρτηση δίνεται στη συνέχεια

```
PROGRAM FunctionUser
IMPLICIT NONE
  REAL :: f ! Δήλωση του τύπου της συνάρτησης
  REAL :: y

  PRINT *, 'Δώστε τιμή: ', READ *, y
  PRINT *, 'Η τιμή της συνάρτησης είναι :', f(y)
END PROGRAM
```

Λύση (συν.)

Στο παραπάνω παράδειγμα `y` είναι η *πραγματική* παράμετρος κλήσης της συνάρτησης ενώ `x` είναι η *τυπική παράμετρος* της `f`. Προσέξτε ότι η ανάγνωση της τιμής `y` όσο και η εκτύπωση του αποτελέσματος γίνεται μέσα στο κύριο πρόγραμμα, `FunctionUser`. Η συνάρτηση `f` είναι υπεύθυνη *μόνο* για τον υπολογισμό της τιμής της για κάθε τιμή του ορίσματός της και *όχι* για την ανάγνωση δεδομένων και την εκτύπωση των αποτελεσμάτων.

Παράδειγμα

Να γραφεί *συνάρτηση* `myFrac(n)`το οποίο υπολογίζει το ακόλουθο συνεχές κλάσμα προσεγγιστικά χρησιμοποιώντας n όρους:

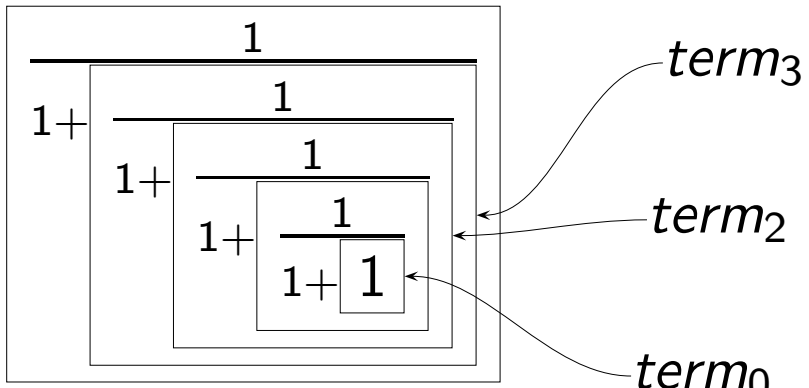
$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{\ddots\,\cfrac{\ddots}{1+\cfrac{1}{1+\frac{1}{2}}}}}}$$

Λύση

Ως παράδειγμα παίρνουμε μια συγκεκριμένη μικρή τιμή για το n, έστω $n = 4$. Τότε το παραπάνω κλάσμα γίνεται:

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{2}}}}$$

Παρατηρούμε ότι το κλάσμα ακολουθεί μια δομή η οποία φαίνεται στο παρακάτω:



Λύση (συν.)

Η παραπάνω έκφραση μπορεί να δημιουργηθεί επαναληπτικά από μια έκφραση *term*, τέτοια ώστε

Λύση (συν.)

Η παραπάνω έκφραση μπορεί να δημιουργηθεί επαναληπτικά από μια έκφραση *term*, τέτοια ώστε

$$term_i = \frac{1}{1 + term_{i-1}}, i > 0$$

και

Λύση (συν.)

Η παραπάνω έκφραση μπορεί να δημιουργηθεί επαναληπτικά από μια έκφραση *term*, τέτοια ώστε

$$term_i = \frac{1}{1 + term_{i-1}}, i > 0$$

και

$$term_0 = 1$$

Λύση (συν.)

Με βάση τα παραπάνω, ο κώδικας για τον σχηματισμό του κλάσματος με n όρους (κλάσματα) είναι ο ακόλουθος:

```
term = 1.
DO i = 1, n
  term = 1. / (1. + term)
END DO
```

Λύση (συν.)

```
FUNCTION myFrac(n)
IMPLICIT NONE
  REAL(8) :: myFrac,term
  INTEGER, INTENT(IN) :: n
  INTEGER :: i

  term = 1.
  DO i = 1, n
    term = 1. / (1. + term)
  END DO
  myFrac = term
END FUNCTION myFrac
```