ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΧΗΜΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΙΚΟ ΚΕΝΤΡΟ

Προγραμματισμός με Fortran 90/95

Compiler vs Interpreter (1/2)

Compiler (Μεταφραστής)
Interpreter (Διερμηνέας)

Source Code (Πηγαίος Κώδικας) → Αρχείο με τις εντολές σε μια γλώσσα προγραμματισμού

Machine Code (Κώδικας Μηχανής) → Εκτελέσιμο αρχείο με εντολές που καταλαβαίνει ο υπολογιστής

Source Code → Compiler → Machine Code → Output

Processing

Source Code → Interpreter → Output

Processing

Compiler vs Interpreter (2/2)

Compiler	Interpreter
Επεξεργάζεται ολόκληρο τον πηγαίο κώδικα με τη μία	Επεξεργάζεται γραμμή-προς-γραμμή (line-by-line) τον πηγαίο κώδικα
Αναφέρονται όλα τα σφάλματα αφού ελεγχθεί όλος ο πηγαίος κώδικας	Κάθε φορά αναφέρεται το πρώτο σφάλμα που θα προκύψει. Τα υπόλοιπα σφάλματα δεν ελέγχονται αν δεν διορθωθεί το προηγούμενο σφάλμα
Δημιουργεί κώδικα μηχανής (εκτελέσιμο αρχείο)	Δε δημιουργεί κώδικα μηχανής
Ο κώδικας μηχανής μπορεί να εκτελεστεί ανεξάρτητα με το αν είναι διαθέσιμος ο μεταφραστής	Για την εκτέλεση του πηγαίου κώδικα είναι απαραίτητος ο διερμηνέας
Δυσκολότερη ανάπτυξη του πηγαίου κώδικα (εντολές, συντακτικό, βελτιστοποίηση, εντοπισμός σφαλμάτων)	Ευκολότερη ανάπτυξη του πηγαίου κώδικα (εντολές, συντακτικό, βελτιστοποίηση, εντοπισμός σφαλμάτων)
Αποδοτικότερη εκτέλεση	Λιγότερο αποδοτική εκτέλεση
Παραδείγματα: C, C++, Fortran	**Παραδείγματα:** MATLAB, Python, Java

Fortran - Formula Translation

Αρχικά αναπτύχθηκε από την IBM τη δεκαετία του 1950 για επιστημονικούς υπολογισμούς και εφαρμογές μηχανικών.

Σήμερα χρησιμοποιείται για απαιτητικούς υπολογισμούς (υψηλή απαίτηση σε μνήμη και υπολογιστικό χρόνο) όπως στην υπολογιστική πρόγνωση καιρού, ρευστομηχανική, γεωφυσική, φυσική και χημεία.

Είναι μια δημοφιλής γλώσσα προγραμματισμού για υπολογιστές υψηλής απόδοσης (high performance computers) και χρησιμοποιείται σε προγράμματα (Linpack Benchmark) που συγκρίνουν και κατατάσσουν τους ταχύτερους υπερυπολογιστές στον κόσμο (TOP 500: https://www.top500.org/).

Πρότυπα (standards) της Fortran - https://wg5-fortran.org

Fortran IV (1966)

Fortran 77 (παλιό πρότυπο αλλά δημοφιλές)

Fortran 90 (σημαντική αναθεώρηση της Fortran 77)

Fortran 95 (μικρή αναθεώρηση της Fortran 90)

Fortran 2003

Fortran 2008

Fortran 2018

Fortran Compilers

GNU Fortran Compiler (gfortran), Flang, LFortran, FTN95,
G77, g2c, G95, PGI, Intel, NAG, Lahey Fortran, …

Από την Fortran 77 (old Fortran) στη Fortran 90 (modern Fortran)

Fortran 77	Fortran 90
Fixed source code	Free source code
Static memory allocation	Dynamic memory allocation
	Ασφαλέστερος και πιο Αξιόπιστος προγραμματισμός
	Παράλληλος προγραμματισμός

Υπολογιστικά Εργαλεία

Online Fortran Compiler
https://rextester.com/l/fortran_online_compiler
(βασίζεται στον compiler gfortran)

IDE (Integrated Development Environment)
Code::Blocks
http://www.codeblocks.org

Αρχή ενός κώδικα Fortran

program name

Κάθε κώδικας Fortran ξεκινάει με την εντολή program ακολουθούμενη από το όνομα name του κώδικα. Ο προγραμματιστής μπορεί να επιλέξει όποιο όνομα θέλει.

Πρέπει όμως να προσέξετε:

- να ξεκινάει με γράμμα κεφαλαίο ή μικρό

- επιτρέπετε η χρήση μόνο γραμμάτων (A – Z, a – z), ψηφίων (0 – 9) και της κάτω παύλας (underscore) _

- το μέγιστο επιτρεπόμενο μέγεθος είναι 31 χαρακτήρες

Τέλος ενός κώδικα Fortran

```
end
```

Κάθε κώδικας Fortran τερματίζεται με την εντολή **end**

Εναλλακτικά

```
end program
end program name
```

Εκτύπωση μηνυμάτων στην οθόνη του υπολογιστή

```
print *, 'my message'
```

Η παραπάνω εντολή εκτυπώνει στην οθόνη το μήνυμα που βρίσκεται μέσα στα εισαγωγικά
' '

Παράδειγμα

```
program My_first_code
print *, 'Hello World'
end
```

Όταν τρέξει ο παραπάνω κώδικας Fortran τότε στην οθόνη εμφανίζεται:

```
Hello World
```

Τι πρέπει να κάνουμε για να τρέξει ο κώδικας;

Compile, Build, Execute

Compile: Ο compiler θα ελέγξει αν ο κώδικάς μας έχει συντακτικά λάθη

Build: Ο compiler θα φτιάξει το εκτελέσιμο αρχείο

Execute: "Τρέξιμο" του εκτελέσιμου αρχείου

Σχόλια

Είναι σκόπιμο ο προγραμματιστής να εισάγει στον κώδικά του σχόλια (μη εκτελέσιμες εντολές) για να διευκολύνει την ανάγνωση του από τον χρήστη. Για να καταλάβει ο μεταγλωττιστής ότι μια γραμμή είναι σχόλιο και όχι εντολή πρέπει να ξεκινάει με το σύμβολο: !

Παράδειγμα

```
program My_second_code
! This is my second Fortran code
print *, 'Hello World'
end
```

Όταν τρέξει ο παραπάνω κώδικας τότε στην οθόνη εμφανίζεται:

```
Hello World
```

Ο μεταγλωττιστής δεν λαμβάνει υπόψη τη δεύτερη γραμμή του παραπάνω κώδικα γιατί ξεκινάει με !

Αριθμητικές Μεταβλητές (1)

Για να μπορέσουμε να κάνουμε αριθμητικές πράξεις με μεταβλητές πρέπει να ορίσουμε τον τύπο τους και το όνομά τους

Για να ορίσουμε μια μεταβλητή με όνομα *variable_name* στην οποία πρόκειται να αποθηκεύσουμε πραγματικούς (real) αριθμούς, γράφουμε:

```
real variable_name
```

Ομοίως για να ορίσουμε μια μεταβλητή με όνομα *variable_name* στην οποία πρόκειται να αποθηκεύσουμε ακέραιους (integer) αριθμούς, γράφουμε:

```
integer variable_name
```

Εναλλακτικά

```
real :: variable_name
integer :: variable_name
```

Αριθμητικές Μεταβλητές (2)

Σε κάθε κώδικα Fortran μετά την εντολή **program** πρέπει να μπαίνει η εντολή **IMPLICIT NONE**

Με την εντολή **IMPLICIT NONE** ο προγραμματιστής υποχρεώνεται να ορίσει κάθε μεταβλητή στον κώδικά του.

Αν επιλέξει να μην χρησιμοποιήσει την εντολή αυτή, τότε ο μεταγλωττιστής επιλέγει (έμμεσα) κάθε μεταβλητή που αρχίζει με τα γράμματα i, j, k, l, m, n να είναι integer και όλες τις άλλες real.

Ο έμμεσος τρόπος υποδήλωσης των μεταβλητών κρύβει σφάλματα (bugs) που δύσκολα "γιατρεύονται"!

Έμμεσος τρόπος υποδήλωσης μεταβλητών

Κάθε μεταβλητή που αρχίζει με τα γράμματα i, j, k, l, m, n είναι integer και όλες οι άλλες real.

Παράδειγμα

```
program code_3

i = 5.5
j = 3

print *, i
print *, j

end
```

Όταν τρέξει ο παραπάνω κώδικας τότε στην οθόνη εμφανίζεται:

```
5
3
```

Προσπαθήστε να τρέξετε τον παρακάτω κώδικα

```
program code_4
IMPLICIT NONE
```

```

i = 5.5
j = 3

print *, i
print *, j

end
```

Γενική μορφή ενός κώδικα Fortran

```
program name
IMPLICIT NONE
Δηλώσεις μεταβλητών
Εντολές

end
```

→ *π.χ. real, integer …*

→ *π.χ. αριθμητικές παραστάσεις, print, if, do …*

Παράδειγμα

```
program new_code
IMPLICIT NONE
real i, k
integer j

i = 5.5
j = 3
k = i + j

print *, i
print *, j
print *, k

end
```

Δηλώσεις μεταβλητών

Εντολές

Αριθμητικές παραστάσεις (1)

Για να δώσουμε τιμή σε μια μεταβλητή χρησιμοποιούμε το σύμβολο **=**

variable_name **=** *τιμή*

ή

variable_name **=** *αριθμητική παράσταση*

Το σύμβολο **=** στη είναι στην ουσία **εντολή**

`j = 3` σημαίνει: δώσε στην αριθμητική μεταβλητή `j` την τιμή 3

ενώ

`j = j + 1` σημαίνει: δώσε στην αριθμητική μεταβλητή `j` την τιμή που είχε η μεταβλητή `j` και πρόσθεσε την τιμή 1

Αριθμητικές παραστάσεις (2)

Οι αριθμητικοί τελεστές που χρησιμοποιούνται στις αριθμητικές παραστάσεις είναι:

+ Πρόσθεση
- Αφαίρεση
* Πολλαπλασιασμός
/ Διαίρεση
** Ύψωση σε δύναμη

Η προτεραιότητα των τελεστών είναι:

** Υψηλή
* και / Μεσαία
+ και - Χαμηλή

Οι παρενθέσεις ( ) αλλάζουν την προκαθορισμένη σειρά (προτεραιότητα των τελεστών) που γίνονται οι πράξεις

Πρώτα γίνονται οι πράξεις μέσα στις παρενθέσεις και μετά οι υπόλοιπες

Μεταξύ τελεστών με την ίδια προτεραιότητα οι πράξεις γίνονται από αριστερά προς τα δεξιά

Πράξεις μεταξύ μεταβλητών ή αριθμών

Οι μεταβλητές και οι αριθμοί που χρησιμοποιούνται στις αριθμητικές παραστάσεις μπορεί να είναι ή μόνο real ή μόνο integer ή συνδυασμός και των δυο.

Αυτό που πρέπει να έχουμε υπόψη μας είναι:

Πράξεις μεταξύ real μεταβλητών ή αριθμών έχουν real αποτέλεσμα

Πράξεις μεταξύ integer μεταβλητών ή αριθμών έχουν integer αποτέλεσμα

Πράξεις μεταξύ real και integer μεταβλητών ή αριθμών έχουν real αποτέλεσμα

Παράδειγμα

Έστω ότι θέλουμε να υπολογίσουμε το αποτέλεσμα της διαίρεσης του 5 με το 2 και να το αποθηκεύσουμε σε μια μεταβλητή **a**

Η τιμή που θα πάρει τελικά η μεταβλητή **a** εξαρτάται από το πώς θα την δηλώσουμε στην αρχή του κώδικα και από το πώς θα γράψουμε την διαίρεση.

	real a	**integer a**
	Τιμή	
a = 5./2.	2.5	2
a = 5/2	2.	2
a = 5./2	2.5	2
a = 5/2.	2.5	2

Εσωτερικές συναρτήσεις (intrinsic functions)

Συνάρτηση	Επεξήγηση	Δήλωση του x	Αποτέλεσμα
real(x)	Μετατροπή του x σε REAL	INTEGER	REAL
int(x)	Μετατροπή του x σε INTEGER	REAL	INTEGER
abs(x)	Απόλυτη τιμή του x	INTEGER	INTEGER
		REAL	REAL
sqrt(x)	Τετραγωνική ρίζα του x	REAL	REAL
sin(x)	Ημίτονο του x	REAL	REAL
cos(x)	Συνημίτονο του x	REAL	REAL
tan(x)	Εφαπτομένη του x	REAL	REAL
exp(x)	e^x	REAL	REAL
log(x)	ln(x)	REAL	REAL
log10(x)	log(x)	REAL	REAL

Εισαγωγή δεδομένων από το πληκτρολόγιο

```
read *, variable_name
```

Η χρήση της εντολής **`read *,`** υποχρεώνει το χρήστη να εισάγει κάποιο νούμερο ή κείμενο από το πληκτρολόγιο, το οποίο και αποθηκεύεται στην μεταβλητή *variable_name*

Παράδειγμα

```
program test
IMPLICIT NONE
real x

print *, 'Give a real number'
read *, x

print *, x

end
```

Όταν τρέξει ο παραπάνω κώδικας τότε στην οθόνη εμφανίζεται:

```
Give a real number
```

και η εκτέλεση του σταματά μέχρι ο χρήστης να εισάγει από το πληκτρολόγιο έναν αριθμό και να πατήσει το ENTER.

Στη συνέχεια εκτυπώνεται ο αριθμός που πληκτρολόγησε ο χρήστης.

Έλεγχος της ροής του κώδικα (1)

Δομή 1

```
if (συνθήκη) then
   .
   εντολές
   .
endif
```

Αν η συνθήκη μέσα στην παρένθεση ικανοποιείται, τότε εκτελούνται οι εντολές που βρίσκονται μετά το **then** και πριν το **endif**,
αλλιώς η εκτέλεση του κώδικα συνεχίζεται με τις εντολές που βρίσκονται μετά το **endif**

Παράδειγμα

```
program blockif
IMPLICIT NONE
real x

x = -4.8

if (x<0) then

  print *, 'Negative'

endif

end
```

Τελεστές σύγκρισης

>	μεγαλύτερο
<	μικρότερο
>=	μεγαλύτερο ή ίσο
<=	μικρότερο ή ίσο
/=	διάφορο
==	ίσο (ΠΡΟΣΟΧΗ: ο τελεστής ισότητας συμβολίζεται με 2 =)

Λογικοί τελεστές

```
.AND.
.OR.
```

```
if ((συνθήκη1).AND.(συνθήκη2)) then
   .
   εντολές
   .
endif
```

```
if ((συνθήκη1).OR.(συνθήκη2)) then
   .
   εντολές
   .
endif
```

Έλεγχος της ροής του κώδικα (2)

Δομή 2

```
if (συνθήκη) then
   .
   εντολές
   .
else
   .
   εντολές
   .
endif
```

Παράδειγμα

```
program blockif
IMPLICIT NONE
real x

x = -4.8

if (x<0) then

  print *, 'Negative'

else

  print *, 'Positive'

endif

end
```

Έλεγχος της ροής του κώδικα (3)

Δομή 3

```
if (συνθήκη) εντολή
```

μόνο μια εντολή

Παράδειγμα

```
program single_line_if
IMPLICIT NONE
real x

x = -4.8

if (x<0) print *, 'Negative'

if (x>=0) print *, 'Positive'

end
```

Επανάληψη do (1)

```
do μετρητής = αρχική τιμή, τελική τιμή, βήμα
   .
   εντολές
   .
enddo
```

μετρητής : **είναι το όνομα μιας αριθμητικής μεταβλητής που έχουμε ορίσει**

αρχική τιμή, τελική τιμή, βήμα : **ορίζουν το σύνολο των τιμών που θα πάρει ο** *μετρητής*.

Παράδειγμα (1)

```
program do_loop
IMPLICIT NONE
integer i

do i = 4, 10, 2
  print *, i
enddo

end
```

Όταν τρέξει ο παραπάνω κώδικας τότε,
η εντολή **print *,** i εκτελείται 4 φορές

και στην οθόνη εμφανίζεται:

```
4
6
8
10
```

Παράδειγμα (2)

```
program do_loop
IMPLICIT NONE
integer i

do i = 4, 10
  print *, i
enddo

end
```

Αν το βήμα είναι 1 τότε μπορεί να παραληφθεί.

Όταν τρέξει ο παραπάνω κώδικας τότε,
η εντολή **print *,** i εκτελείται 7 φορές

Επανάληψη do (2)

Μπορούμε επίσης να χρησιμοποιήσουμε δυο επαναλήψεις do τη μία μέσα στην άλλη αρκεί να μην έχουν τους ίδιους μετρητές.

```
program do_loops
IMPLICIT NONE
integer i, j

do i = 1, 3
  do j = 1, 3
    print *, i, j
  enddo
enddo

end
```

Όταν τρέξει ο κώδικας αυτός τότε, η εντολή `print *, i, j` εκτελείται 9 φορές και στην οθόνη εμφανίζεται:

```
 1     1
 1     2
 1     3
 2     1
 2     2
 2     3
 3     1
 3     2
 3     3
```

Αέναη επανάληψη

```
do
   .
   εντολές
   .
   if (συνθήκη) exit
   .
   εντολές
   .
enddo
```

Παράδειγμα

```
program doloop
IMPLICIT NONE
integer i

i = 0
do
  i = i + 1
  print *, i
  if (i>5) exit
enddo

end
```

Όταν τρέξει ο κώδικας αυτός τότε στην οθόνη εμφανίζεται:

```
1
2
3
4
5
6
```

Διατεταγμένα σύνολα τιμών (arrays)

Οι arrays είναι αριθμητικές μεταβλητές που μπορούν να φιλοξενήσουν περισσότερες από μία τιμές.

Διανύσματα ή Μονοδιάστατες Arrays

Πίνακες ή Διδιάστατες Arrays

Διανύσματα ή Μονοδιάστατες Arrays

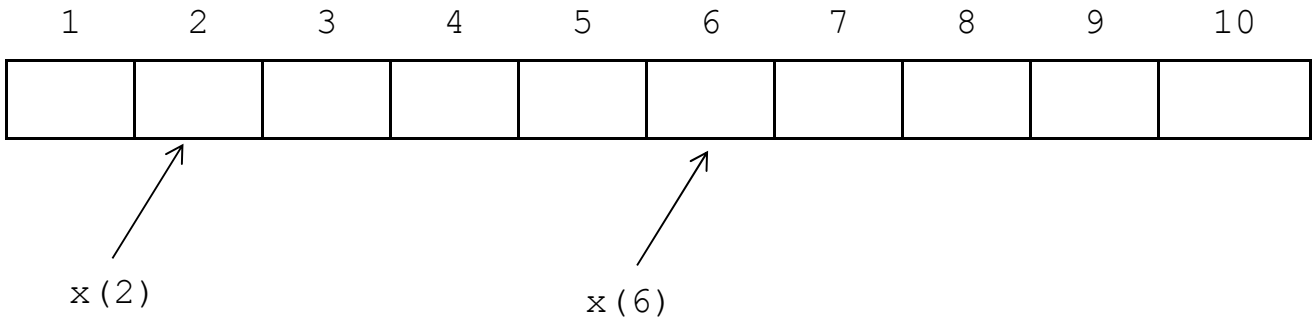
```
real, allocatable, dimension(:) :: variable_name

integer, allocatable, dimension(:) :: variable_name
```

```
allocate(variable_name(count))
```

```
deallocate(variable_name)
```

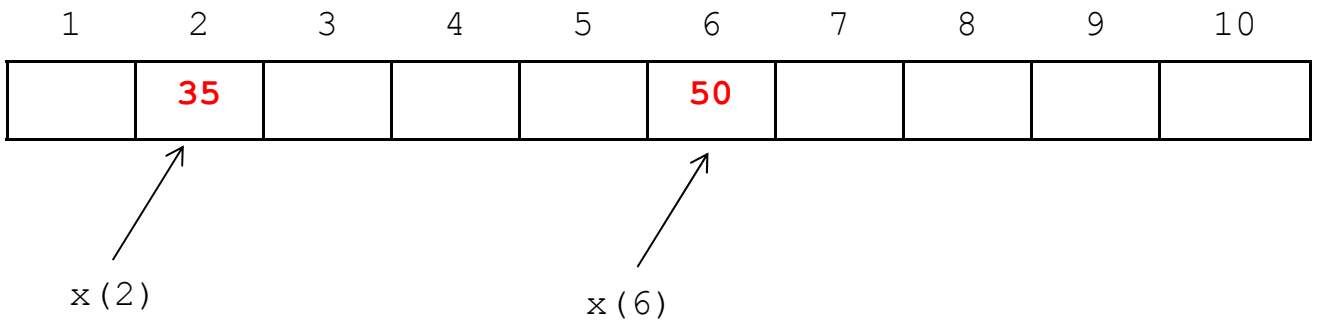
Διανύσματα ή Μονοδιάστατες Arrays

```
integer, allocatable, dimension(:) :: x

allocate(x(10))
```



Διανύσματα ή Μονοδιάστατες Arrays

```
integer, allocatable, dimension(:) :: x

allocate(x(10))
```




```
x(2)=35

x(6)=50
```

Παράδειγμα

```
program vector_array
IMPLICIT NONE
integer, allocatable, dimension(:) :: x

allocate(x(5))
x(1)= 3
x(2)= 6
x(3)= 4
x(4)= 1
x(5)= 7

print *, x(1)
print *, x(2)
print *, x(3)
print *, x(4)
print *, x(5)

deallocate(x)

end
```

ορίζουμε 5 ακέραιες μεταβλητές με ονόματα:
x(1) x(2) x(3) x(4) x(5)

Όταν τρέξει ο κώδικας αυτός τότε στην οθόνη εμφανίζεται:

```
3
6
4
1
7
```

Παράδειγμα

```
program vector_array
IMPLICIT NONE
integer i
integer, allocatable, dimension(:) :: x

allocate(x(5))
x(1)= 3
x(2)= 6
x(3)= 4
x(4)= 1
x(5)= 7

do i = 1, 5
  print *, x(i)
enddo

deallocate(x)

end
```

Όταν τρέξει ο κώδικας αυτός τότε στην οθόνη εμφανίζεται:

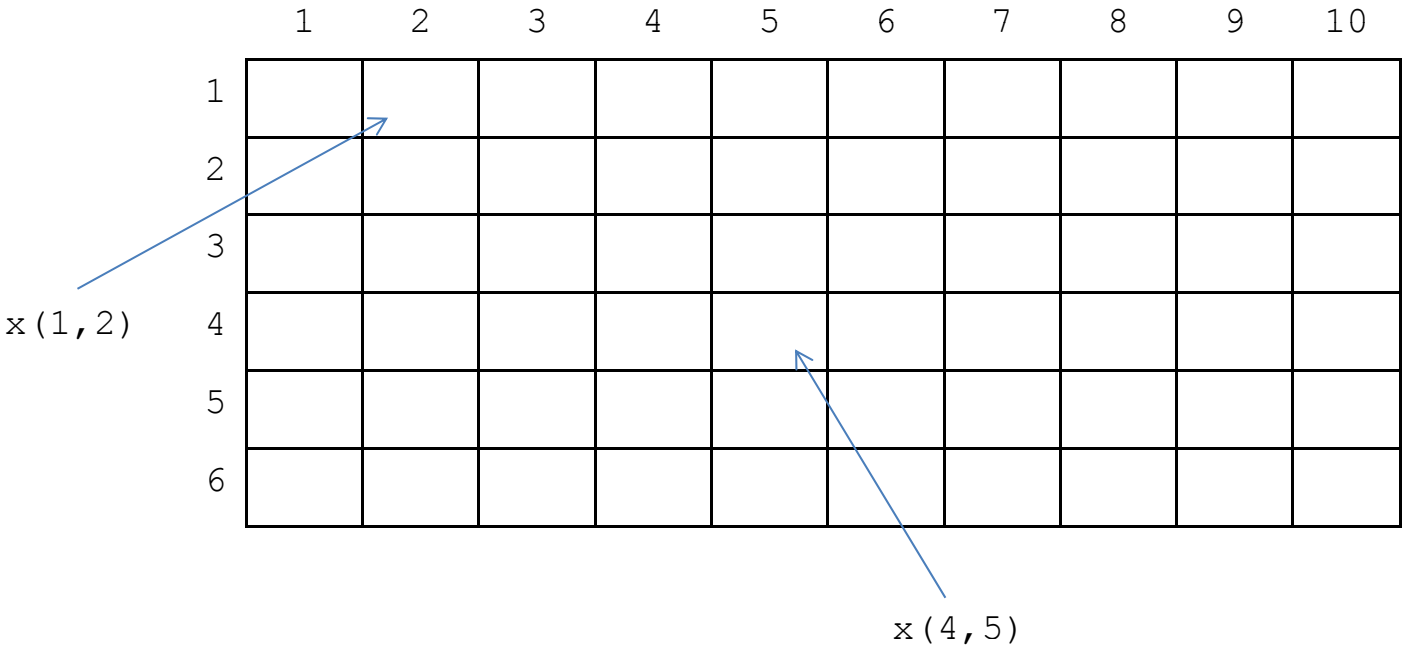
```
3
6
4
1
7
```

Πίνακες ή Διδιάστατες Arrays

```
real, allocatable, dimension(:,:) :: variable_name

integer, allocatable, dimension(:,:) :: variable_name


allocate(variable_name(count1,count2))


deallocate(variable_name)
```

Πίνακες ή Διδιάστατες Arrays

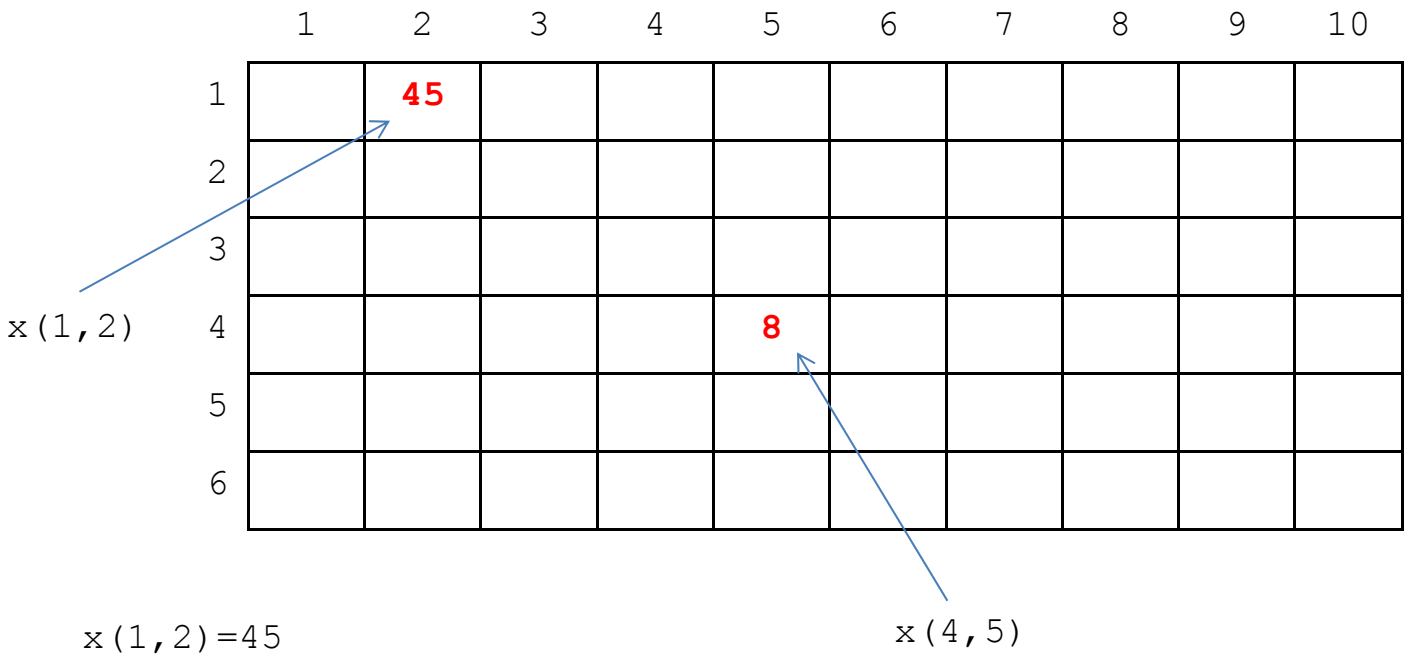
```
integer, allocatable, dimension(:,:) :: x

allocate(x(6,10))
```



Πίνακες ή Διδιάστατες Arrays

```
integer, allocatable, dimension(:,:) :: x
```

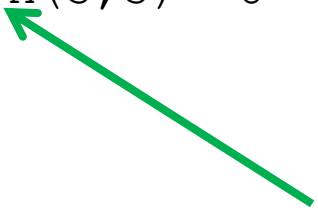
```
allocate(x(6,10))
```




```
x(1,2)=45

x(4,5)=8
```

Παράδειγμα

```
program matrix_array
IMPLICIT NONE
integer i,j
integer, allocatable, dimension(:,:) :: x

allocate(x(3,3))
x(1,1)= 3 ; x(1,2)= 6 ; x(1,3)= 4
x(2,1)= 1 ; x(2,2)= 9 ; x(2,3)= 7
x(3,1)= 4 ; x(3,2)= 3 ; x(3,3)= 8

do i = 1, 3
  do j = 1, 3
    print *, x(i,j)
  enddo
enddo

deallocate(x)

end
```



Με το σύμβολο ; μπορούμε να γράψουμε διαφορετικές εντολές στην ίδια γραμμή

Συνεπαγόμενο do

```
program matrix_array
IMPLICIT NONE
integer i,j
integer, allocatable, dimension(:,:) :: x

allocate(x(3,3))
x(1,1)= 3 ; x(1,2)= 6 ; x(1,3)= 4
x(2,1)= 1 ; x(2,2)= 9 ; x(2,3)= 7
x(3,1)= 4 ; x(3,2)= 3 ; x(3,3)= 8

do i = 1, 3
    print *, (x(i,j),j=1,3)
enddo

deallocate(x)

end
```

Όταν τρέξει ο κώδικας αυτός τότε στην οθόνη εμφανίζεται:

```
3       6       4
1       9       7
4       3       8
```

```
real, parameter :: variable_name = value
integer, parameter :: variable_name = value
```

```
program matrix_array
IMPLICIT NONE
integer i,j
integer, parameter :: N = 3
integer, allocatable, dimension(:,:) :: x

allocate(x(N, N))
x(1,1)= 3 ; x(1,2)= 6 ; x(1,3)= 4
x(2,1)= 1 ; x(2,2)= 9 ; x(2,3)= 7
x(3,1)= 4 ; x(3,2)= 3 ; x(3,3)= 8

do i = 1, N
   print *, (x(i,j),j=1,N)
enddo

deallocate(x)

end
```

Επανάληψη do

Ο μετρητής μιας επανάληψης do δεν μπορεί να μεταβάλλεται μέσα στο σώμα της επανάληψης. Για παράδειγμα οι παρακάτω επαναλήψεις do **είναι λάθος**

```
do i=1,5          do i=1,5
  .                 .
  .                 .
  i=32              i=i+1
  .                 .
enddo             enddo
```

Κάθε do πρέπει να συνδυάζεται με enddo . Για παράδειγμα:

Λάθος

```
do i=1,5
  .
  do j=4,6
  .
  .
enddo
```

Σωστό

```
do i=1,5
  .
  do j=4,6
  .
  enddo
  .
enddo
```

Επανάληψη do

Κάθε δομή if πρέπει να αρχίζει και να τελειώνει μέσα στο σώμα μιας επανάληψης do. Για παράδειγμα:

Λάθος

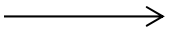
```
do i=1,5
   .
   if(x>35) then
   .
   .
enddo
   .
   endif
```

Σωστό

```
do i=1,5
   .
   if(x>35) then
   .
   endif
   .
enddo
```

Παράδειγμα - Διδιάστατες Arrays

```
N = 3
do i=1,N
  do j=1,N
    if(i==j) then
      A(i,j)=1
    else
      A(i,j)=0
    endif
  enddo
enddo
```



i=1

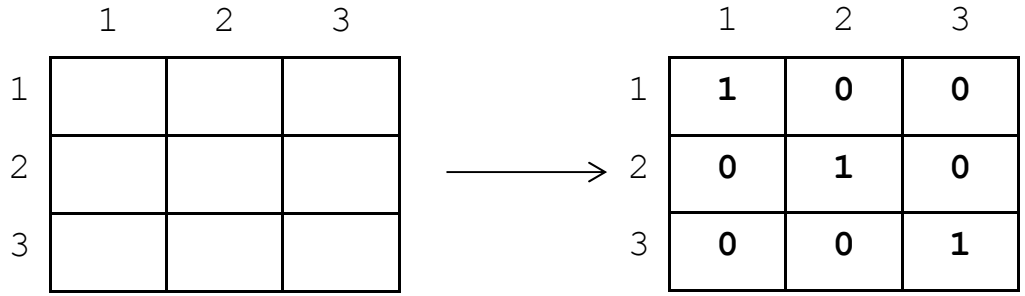
j=1 → A(**1**,**1**)=1
j=2 → A(**1**,**2**)=0
j=3 → A(**1**,**3**)=0

i=2

j=1 → A(**2**,**1**)=0
j=2 → A(**2**,**2**)=1
j=3 → A(**2**,**3**)=0

i=3

j=1 → A(**3**,**1**)=0
j=2 → A(**3**,**2**)=0
j=3 → A(**3**,**3**)=1



	1	2	3
1			
2			
3			

	1	2	3
1	**1**	**0**	**0**
2	**0**	**1**	**0**
3	**0**	**0**	**1**

Πρότυπο IEEE 754 κινητής υποδιαστολής

IEEE Standard 754

Single precision

$\text{realmin} = 2^{e_{\min}} = 2^{-126} = 1.1755\text{e-}38$

$\text{realmax} = (2 - 2^{-t}) \cdot 2^{e_{\max}} = (2 - 2^{-23}) \cdot 2^{127} = 3.4028\text{e+}38$

$eps = 2^{-23} = 1.1921\text{e-}07$

Double precision

$\text{realmin} = 2^{e_{\min}} = 2^{-1022} = 2.2251\text{e} - 308$

$\text{realmax} = (2 - 2^{-t}) \cdot 2^{e_{\max}} = (2 - 2^{-52}) \cdot 2^{1023} = 1.7977\text{e} + 308$

$eps = 2^{-52} = 2.2204\text{e} - 16$

Οι αριθμοί που εμφανίζονται στους υπολογισμούς και κατ΄ απόλυτη τιμή είναι μεγαλύτεροι από την τιμή **realmax** δημιουργούν ***overflow***. Οι αριθμοί αυτοί τίθενται ίσοι με **Infinity** ή **-Infinity** ανάλογα με το αν είναι θετικοί οι αρνητικοί αντίστοιχα.

Οι αριθμοί που εμφανίζονται στους υπολογισμούς και κατ΄ απόλυτη τιμή είναι μικρότεροι από την τιμή **realmin** δημιουργούν **underflow**. Οι αριθμοί αυτοί τίθενται ίσοι με το **μηδέν**.

Ποσότητες που δεν μπορούν να αναπαρασταθούν (π.χ. σε ορισμένα αποτελέσματα μαθηματικών συναρτήσεων) τίθενται ίσοι με **NaN** (Not-a-Number).

Πεδίο τιμών των μεταβλητών

	Σημαντικά ψηφία *my_variable*	Πεδίο τιμών	Απαίτηση σε μνήμη (1 byte = 8 bits)
real *my_variable* (single precision)	6-7	$10^{-38} \ldots 10^{+38}$ (απόλυτες τιμές)	4 bytes = 32 bits
real(8) *my_variable* (double precision)	15-16	$10^{-308} \ldots 10^{+308}$ (απόλυτες τιμές)	8 bytes = 64 bits
integer *my_variable* (single precision)	–	-2147483648 ... 2147483647	4 bytes = 32 bits
integer(8) *my_variable* (double precision)	–	**-9223372036854775808 ... 9223372036854775807**	8 bytes = 64 bits

Δηλώσεις μεταβλητών

single precision
real
real*4
real(4)
real(KIND=4)

double precision
real*8
real(8)
real(KIND=8)

Αποθήκευση των arrays

	Απαίτηση σε μνήμη
`real, allocatable, dimension (:) :: my_variable` `allocate(my_variable(N))`	N·4 bytes
`real(8), allocatable, dimension (:) :: my_variable` `allocate(my_variable(N))`	N·8 bytes
`real, allocatable, dimension (:,:) :: my_variable` `allocate(my_variable(N,N))`	N·N·4 bytes
`real(8), allocatable, dimension (:,:) :: my_variable` `allocate(my_variable(N,N))`	N·N·8 bytes

Παράδειγμα

Αν ορίσουμε τη μεταβλητή `my_variable` ως

```
real(8), allocatable, dimension (:,:) :: my_variable

allocate(my_variable(10000,10000))
```

τότε η `my_variable` έχει απαίτηση σε μνήμη 800 MB

Εκθετική μορφή

$a \cdot 10^b = aEb$

Αριθμητικές Εκφράσεις	Εκθετική γραφή
$15.23 \cdot 10^2 = 152.3 \cdot 10 = 1523$	15.23E+2 = 152.3E+1 = 1523E+0
$152300 \cdot 10^{-2} = 15230 \cdot 10^{-1} = 1523$	152300E-2 = 15230E-1 = 1523E+0

Παράδειγμα

Single precision: `15.23e2`

Double precision: `15.23d2`

Single (32 bits) vs Double precision (64 bits)

"Τρέξτε" τους παρακάτω κώδικες. Τι αποτέλεσμα περιμένετε για την τιμή της μεταβλητής **s** και τι τυπώνεται;

```
program single_precision_s
implicit none
integer i
real s

s=0.

do i=1,1000
  s = s + 0.1
enddo

print *, s
print *, abs(s-100)

end
```

32 bits

Μεταβλητή single precision

single precision

```
program double_precision_s
implicit none
integer i
real(8) s

s=0.

do i=1,1000
  s = s + 0.1_8
enddo

print *, s
print *, abs(s-100)

end
```

64 bits

Μεταβλητή double precision

double precision

Machine epsilon (1)

```
program find_machine_precision
IMPLICIT NONE
real eps
real a

eps=1.

do
  a=1.+eps
  if(a==1.) exit
  eps=eps/2.
enddo

print *, 2*eps

end
```

Machine epsilon (2)

```
program find_machine_precision
IMPLICIT NONE
real(8) eps
real(8) a

eps=1.

do
  a=1.+eps
  if(a==1.) exit
  eps=eps/2.
enddo

print *, 2*eps

end
```

Double precision

Ισότητα αριθμών κινητής υποδιαστολής ή "how close is close"

a == b → **match bit to bit**

Η ερώτηση της ισότητας "γεννάει" μια άλλη ερώτηση : "Τι εννοούμε με τον όρο ισότητα";

Στον προγραμματισμό η ισότητα σημαίνει "αρκετά κοντά".
Ο έλεγχος της "ισότητας" γίνεται ορίζοντας κάποια μικρή απόσταση **e** ως "αρκετά κοντά"

Όμως:
e = 0.00001
είναι κατάλληλο για αριθμούς γύρω από το 1
πολύ μεγάλο για αριθμούς γύρω από το 0.00001
και πολύ μικρό για αριθμούς γύρω από το 10000

Equality test
Όχι: if (a==b)
Ναι: if (abs(a-b)< e)

ΠΡΟΣΟΧΗ

Αφαιρέσεις "κοντινών" αριθμών

Προσθέσεις/Αφαιρέσεις αριθμών με μεγάλη διαφορά

Fortran single precision

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{1000000} = 14.3573580$$

$$\frac{1}{1000000}+\cdots+\frac{1}{3}+\frac{1}{2}+1 = 14.3926516$$

Λογικές

Δήλωση: `logical :: variable_name`
Ανάθεση:
```
variable_name = .true.
variable_name = .false.
```

Αλφαριθμητικές

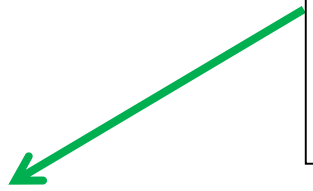
Δήλωση: `character(len=k) :: variable_name`
Ανάθεση:
```
variable_name = 'Enter text here'
```

όπου k το μέγιστο πλήθος των χαρακτήρων που θα αποθηκευτούν στη μεταβλητή

Μορφοποίηση (1)

```
program format
IMPLICIT NONE
integer j

j = 32

print *, j

end
```

```
print '(I2)', j

print '(I4)', j

print '(I1)', j
```

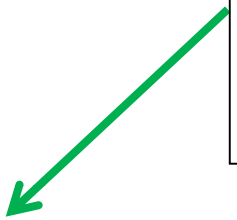


Iw : τύπωσε έναν ακέραιο στις επόμενες w θέσεις

Μορφοποίηση (2)

```
program format
IMPLICIT NONE
integer i,j

i = 3
j = 5

print *, i, j

end
```

```
print '(I1,I1)', i,j

print '(I1,I2)', i,j

print '(I1,2X,I1)', i,j
```



*n*X: κατά την εκτύπωση αγνόησε *n* θέσεις (δημιουργεί *n* κενά)

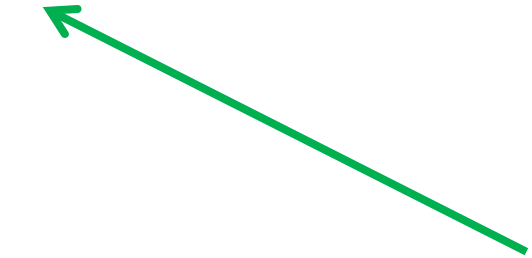
Μορφοποίηση (3) **Iw και Iw.m**

```
integer :: a=123, b=-123, c=123456
```

`print '(I5)',  a`			**1**	**2**	**3**
`print '(I5.2)', a`			**1**	**2**	**3**
`print '(I5.4)', a`		**0**	**1**	**2**	**3**
`print '(I5.5)', a`	**0**	**0**	**1**	**2**	**3**
`print '(I5)',  b`		**-**	**1**	**2**	**3**
`print '(I5.2)', b`		**-**	**1**	**2**	**3**
`print '(I5.4)', b`	**-**	**0**	**1**	**2**	**3**
`print '(I5.5)', b`	*****	*****	*****	*****	*****
`print '(I5)',  c`	*****	*****	*****	*****	*****

Μορφοποίηση (4)

```
program format
IMPLICIT NONE
integer j

j = 3

print *, 'The value of j is',j

end
```



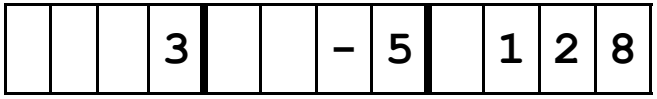
```
print '(A,I2)', 'The value of j is',j
```

A : τύπωσε μια σειρά από χαρακτήρες (character string)

Μορφοποίηση (5)

rIw

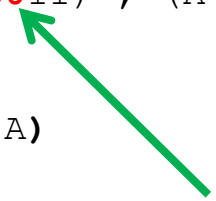
```
integer :: a=3, b=-5, c=128
print '(3I4)', a, b, c
```



Μορφοποίηση (6)

rIw

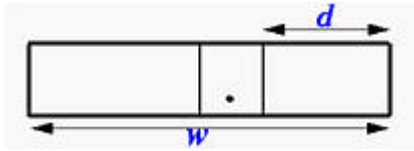
```
program xiasti
IMPLICIT NONE
integer, parameter :: M = 20
integer, allocatable, dimension(:,:) :: A
integer i, j

allocate(A(M,M))
A = 0

do i = 1, M
  do j = 1, M
    if(i==j) A(i,j) = 1
    if((i+j)==(M+1)) A(i,j) = 1
  enddo
enddo

do i = 1, M
  print '(50I1)', (A(i,j),j=1,M)
enddo

deallocate(A)
end
```



Μορφοποίηση (7) **Fw.d**

```
program format
IMPLICIT NONE
real j

j = 32.5678

print *, j

end
```

```
print '(F7.4)', j

print '(F7.2)', j

print '(F7.6)', j
```



`Fw.d`: τύπωσε έναν πραγματικό στις επόμενες `w` θέσεις με `d` δεκαδικά ψηφία

Μορφοποίηση (8) **Fw.d**

```
real :: a=123.346, b=-123.346
```

print '(F10.0)', a							1	2	3	.
print '(F10.2)', a					1	2	3	.	3	5
print '(F10.4)', a			1	2	3	.	3	4	6	0
print '(F10.6)', a	1	2	3	.	3	4	6	0	0	0
print '(F10.7)', a	*	*	*	*	*	*	*	*	*	*
print '(F10.4)', b		-	1	2	3	.	3	4	6	0
print '(F10.5)', b	-	1	2	3	.	3	4	6	0	0
print '(F10.6)', b	*	*	*	*	*	*	*	*	*	*

Μορφοποίηση (9)

rFw.d

```
real :: a=12.34, b=-0.946, c=100.
print '(3F6.2)', a, b, c
```

	1	2	.	3	4		-	0	.	9	5	1	0	0	.	0	0

Εκθετική μορφή

Για να τυπωθεί ένας αριθμός στην εκθετική του μορφή μετατρέπεται πρώτα στην κανονικοποιημένη του μορφή

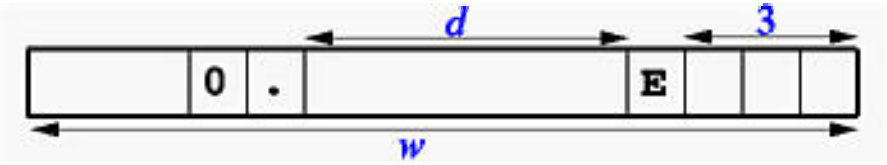
$$s0.xxxxxx \times 10^{sxx}$$

όπου *s* το πρόσημο

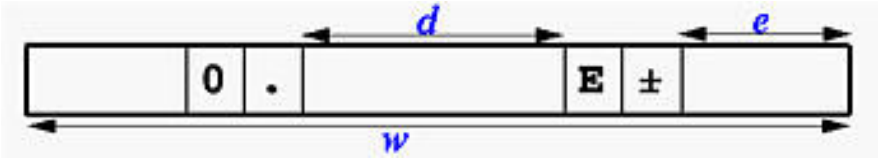
Για παράδειγμα οι αριθμοί 12.345, -12.345, 0.00123 μετατρέπονται σε

$0.12345 \times 10^{2},\ -0.12345 \times 10^{2},\ 0.123 \times 10^{-2}$

Ew.d



Ew.dEe



Μορφοποίηση (11)

```
real :: pi=3.1415926
```

`print '(E12.5)',  pi`		**0**	**.**	**3**	**1**	**4**	**1**	**6**	**E**	**+**	**0**	**1**
`print '(E12.3E3)', pi`			**0**	**.**	**3**	**1**	**4**	**E**	**+**	**0**	**0**	**1**
`print '(E12.7E1)', pi`	**0**	**.**	**3**	**1**	**4**	**1**	**5**	**9**	**3**	**E**	**+**	**1**

Εκτύπωση σε αρχείο

Χρησιμοποιείται όπως και το * της `print *,` δηλαδή για μορφοποίηση των αποτελεσμάτων

```
write(*,*)
```

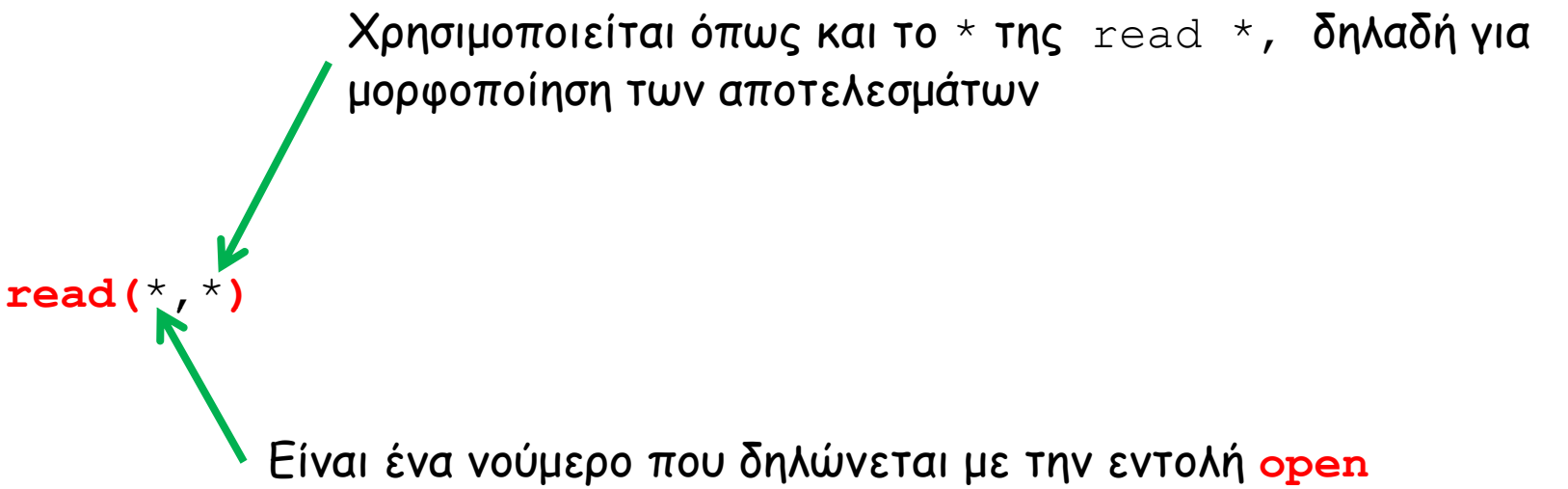
Είναι ένα νούμερο που δηλώνεται με την εντολή **open**

```
open(UNIT=1,FILE='myfile')
```

Παράδειγμα

```
program write
IMPLICIT NONE
integer i

open(UNIT=1,FILE='myfile')

do i=1,10
  write(1,*) i
enddo

end
```

Διάβασμα από αρχείο




```
open(UNIT=1,FILE='myfile')
```

Παράδειγμα

```
program write
IMPLICIT NONE
integer i,j

open(UNIT=1,FILE='myfile')

do i=1,10
  read(1,*) j
  print *, j
enddo

end
```

Υποπρογράμματα

Ανεξάρτητα προγράμματα

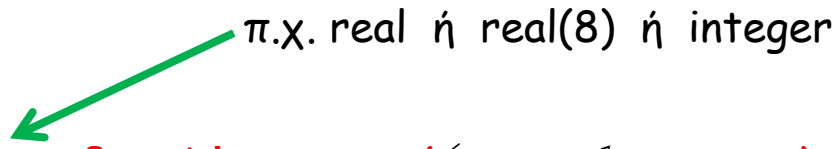
Επικοινωνούν με το κυρίως πρόγραμμα μέσω των ορισμάτων τους

Κατηγορίες
- Εσωτερικά (internal)
- Εξωτερικά (external)
- Εγγενή (intrinsic)

Μπορεί να είναι είτε υπορουτίνες (subroutines) είτε συναρτήσεις (functions)

Υποπρογράμματα

```
subroutine name(όρισμα1, … , …)
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

end subroutine
```




```
τύπος function name(όρισμα1, … , …)
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

end function
```

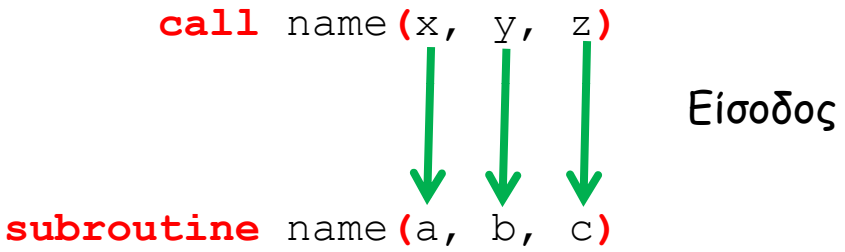
Κλήση μιας υπορουτίνας (subroutine)

```
program main
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

call name(όρισμα1, … , …)

Εκτελέσιμες εντολές

end


subroutine name(όρισμα1, … , …)
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

end subroutine
```

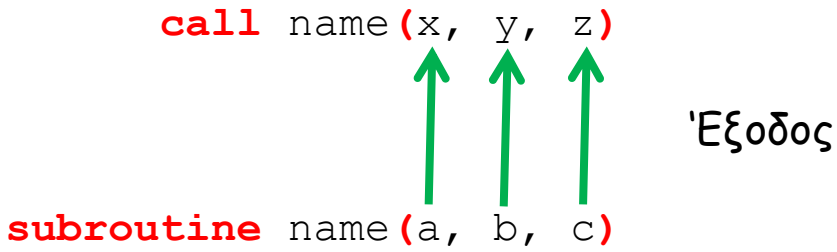
Επικοινωνία μέσω ορισμάτων (1) - subroutines

Το **a** πρέπει να έχει τον ίδιο τύπο (π.χ. real ή real(8) ή integer) με το **x**
Το **b** πρέπει να έχει τον ίδιο τύπο με το **y**
Το **c** πρέπει να έχει τον ίδιο τύπο με το **z**

Κατά την είσοδο (κλήση) στην υπορουτίνα:
το **a** θα πάρει την τιμή του **x**
το **b** θα πάρει την τιμή του **y**
το **c** θα πάρει την τιμή του **z**



Κατά την έξοδο από την υπορουτίνα:
το **x** θα πάρει την τιμή του **a**
το **y** θα πάρει την τιμή του **b**
το **z** θα πάρει την τιμή του **c**



Επικοινωνία μέσω ορισμάτων (2) - subroutines

```
subroutine name(a, b, c)
```

Αποδεκτές κλήσεις

```
call name(x, y, z)
call name(a, b, c)
call name(a, y, z)
```

Λάθος κλήση

```
call name(x, y)
```

Υπολογισμός του N!

```
program factorial_of_N
IMPLICIT NONE
integer N,i
integer(8) factorial

print *, 'This program computes the factorial of &
&a positive integer number N'
print *, 'Give the N'
read *, N

factorial=1

do i = 1, N
  factorial = factorial*i
enddo

print *, 'The factorial of N is'
print *, factorial

end
```

Υπολογισμός του N! με χρήση subroutine

```
program factorial_of_N
IMPLICIT NONE
integer N
integer(8) factorial

print *, 'This program computes the factorial of &
&a positive integer number N'
print *, 'Give the N'
read *, N

call calculate_factorial(N,factorial)

print *, 'The factorial of N is'
print *, factorial

end

subroutine calculate_factorial(a,b)
IMPLICIT NONE
integer a,i
integer(8) b

b=1

do i = 1, a
  b = b*i
enddo

end subroutine
```

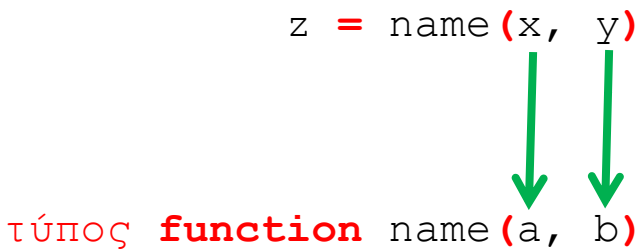
Κλήση μιας συνάρτησης (function)

```
program main
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

μεταβλητή = name(όρισμα1, … , …)

Εκτελέσιμες εντολές

end


τύπος function name(όρισμα1, … , …)
IMPLICIT NONE
Ορισμοί μεταβλητών

Εκτελέσιμες εντολές

end function
```

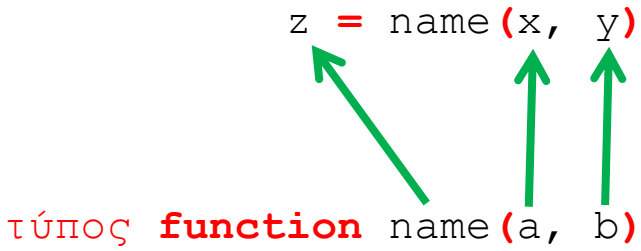
Επικοινωνία μέσω ορισμάτων (1) - functions

Το **a** πρέπει να έχει τον ίδιο τύπο με το **x**
Το **b** πρέπει να έχει τον ίδιο τύπο με το **y**
Το **name** πρέπει να έχει τον ίδιο τύπο με το **z**

Κατά την είσοδο (κλήση) στη συνάρτηση:
το **a** θα πάρει την τιμή του **x**
το **b** θα πάρει την τιμή του **y**



Κατά την έξοδο από τη συνάρτηση:
το **x** θα πάρει την τιμή του **a**
το **y** θα πάρει την τιμή του **b**
το **z** θα πάρει την τιμή του **name**



Υπολογισμός του N! με χρήση function

```
program factorial_of_N
IMPLICIT NONE
integer N
integer(8) factorial
integer(8) calculate_factorial

print *, 'This program computes the factorial of &
&a positive integer number N'
print *, 'Give the N'
read *, N

factorial=calculate_factorial(N)

print *, 'The factorial of N is'
print *, factorial

end

integer(8) function calculate_factorial(a)
IMPLICIT NONE
integer a,i

calculate_factorial=1

do i = 1, a
  calculate_factorial = calculate_factorial*i
enddo

end function
```

Σκοπός (INTENT) των ορισμάτων (1)

```
program main
IMPLICIT NONE
integer a,b,c

a = 5
b = 6
call add(a,b,c)

print *, 'The sum of',a,b
print *, 'is',c

end

subroutine add(x,y,z)
IMPLICIT NONE
integer x,y,z

z = x + y

end subroutine
```

Σκοπός (INTENT) των ορισμάτων (2)

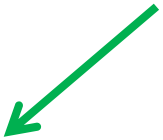
```
program main
IMPLICIT NONE
integer a,b,c

a = 5
b = 6
call add(a,b,c)

print *, 'The sum of',a,b
print *, 'is',c

end

subroutine add(x,y,z)
IMPLICIT NONE
integer x,y,z

y = x + z

end subroutine
```

Σκοπός (INTENT) των ορισμάτων (3)

```
program main
IMPLICIT NONE
integer a,b,c

a = 5
b = 6
call add(a,b,c)

print *, 'The sum of',a,b
print *, 'is',c

end

subroutine add(x,y,z)
IMPLICIT NONE
integer x,y,z

z = x + y

x = 100

end subroutine
```

Σκοπός (INTENT) των ορισμάτων (3)

```
program main
IMPLICIT NONE
integer a,b,c

a = 5
b = 6
call add(a,b,c)

print *, 'The sum of',a,b
print *, 'is',c

end

subroutine add(x,y,z)
IMPLICIT NONE
integer, intent(IN)  :: x,y
integer, intent(OUT) :: z

z = x + y

x = 100

end subroutine
```



Τοπικές μεταβλητές

```
program main
IMPLICIT NONE
integer a,b,c

a = 5
b = 6
call add(a,b,c)

print *, 'The sum of',a,b
print *, 'is',c

end

subroutine add(x,y,z)
IMPLICIT NONE
integer, intent(IN)  :: x,y
integer, intent(OUT) :: z
integer a

z = x + y

a = 100

end subroutine
```

τοπική μεταβλητή

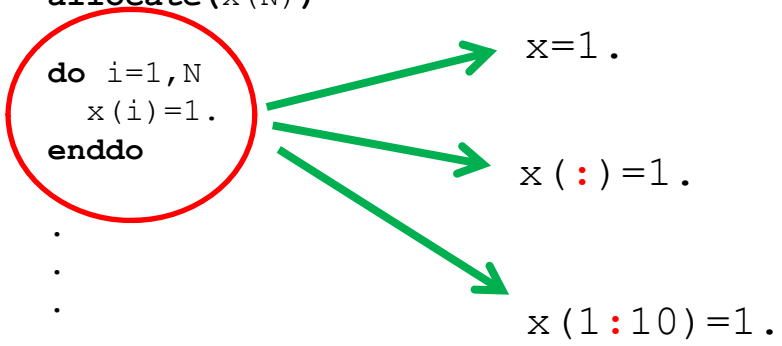
Arrays ως ορίσματα

```
program main
IMPLICIT NONE
integer, allocatable, dimension(:,:) :: A
integer N,i,j

N=5

allocate(A(N,N))

call identity(N,A)

do i = 1, N
  print *, (A(i,j),j=1,N)
enddo

deallocate(A)

end
```

```
subroutine identity(N,A)
IMPLICIT NONE
integer, intent(IN) :: N
integer, intent(OUT), dimension(N,N) :: A
integer i,j

A=0

do i = 1, N
  do j = 1, N
    if(i==j) A(i,j)= 1
  enddo
enddo

end subroutine
```

Απλοποίηση των πράξεων με arrays(1)

```
program simple1
IMPLICIT NONE
real, allocatable, dimension(:) :: x
integer N, i

N=10

allocate(x(N))

do i=1,N
  x(i)=1.
enddo

.
.
.
```



x(*a*:*b*)=*value*

όπου a, b integers στο διάστημα [1 N] με a<b

Απλοποίηση των πράξεων με arrays(2)

x(*a*:*b*:*s*)=*value*

a: αρχική τιμή
b: τελική τιμή
s: βήμα

```
program simple2
IMPLICIT NONE
real, allocatable, dimension(:) :: x
integer N, i

N=10

allocate(x(N))

do i=2,6,2
  x(i)=1.
enddo

.
.
.
```

x(2:6:2)=1.

Απλοποίηση των πράξεων με arrays(3)

```
program simple3
IMPLICIT NONE
real, allocatable, dimension(:) :: x, y, z
integer N, i

N=10

allocate(x(N), y(N), z(N))
.
.
.
do i=1,N
  z(i) = x(i) + y(i)
enddo
.
.
.
```

```
z = x + y
```

```
z(:) = x(:) + y(:)
```

```
z(1:size(x)) = x(1:size(x)) + y(1:size(x))
```

Η εσωτερική συνάρτηση της Fortran **size(*array*)**
επιστρέφει το μέγεθος της *array*

Απλοποίηση των πράξεων με arrays(4)

Στη Fortran οι πράξεις `+ - * / **` μεταξύ των arrays γίνονται στοιχείο προς στοιχείο

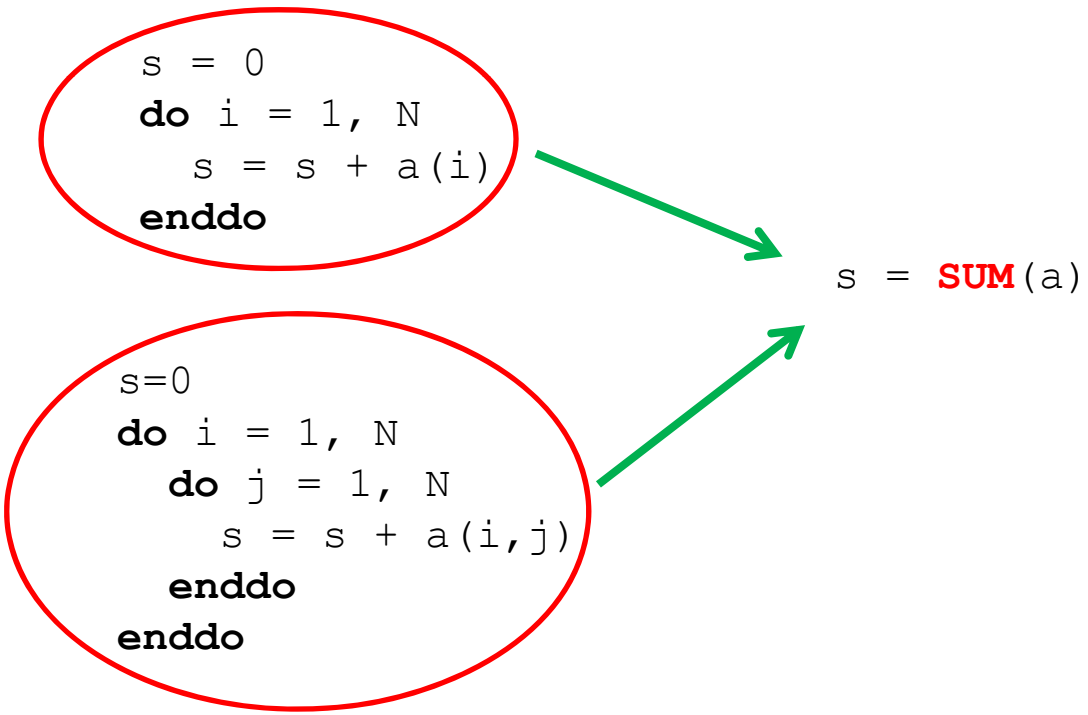
$$\overbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}}^{x} \odot \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}}^{y} = \overbrace{\begin{bmatrix} x_1 \odot y_1 \\ x_2 \odot y_2 \\ x_3 \odot y_3 \\ x_4 \odot y_4 \\ x_5 \odot y_5 \end{bmatrix}}^{z}$$

$\odot$: + − * / **

Απλοποίηση των πράξεων με arrays(5)

Χρήση εσωτερικών συναρτήσεων (intrinsic functions)

Υπολογισμός του αθροίσματος των στοιχείων μιας array



Απλοποίηση των πράξεων με arrays(6)

Χρήση εσωτερικών συναρτήσεων (intrinsic functions)

Παράδειγμα

Υπολογισμός του αθροίσματος της 2ης στήλης των στοιχείων μιας διδιάστατης array

```
s=0
do i = 1, N
  s = s + a(i,2)
enddo
```

```
s = SUM( a(:,2) )
```

Απλοποίηση των πράξεων με arrays(7)

Χρήση εσωτερικών συναρτήσεων (intrinsic functions)

SIZE(array)	Το πλήθος των στοιχείων
MAXVAL(array)	Η τιμή του μεγαλύτερου στοιχείου
MINVAL(array)	Η τιμή του μικρότερου στοιχείου
SUM(array)	Το άθροισμα των στοιχείων
PRODUCT(array)	Το γινόμενο των στοιχείων
MAXLOC(array)	Η θέση του μεγαλύτερου στοιχείου
MINLOC(array)	Η θέση του μικρότερου στοιχείου
DOT_PRODUCT(vector_a, vector_b)	Το γινόμενο δυο διανυσμάτων
MATMUL(matrix_A, matrix_B)	Το γινόμενο δυο πινάκων
MATMUL(matrix_A, vector_a)	Το γινόμενο πίνακα με διάνυσμα
TRANSPOSE(matrix)	Ο ανάστροφος ενός πίνακα

vector_a, vector_b: **μονοδιάστατες (rank-one) arrays**
matrix_A, matrix_B, matrix: **διδιάστατες arrays**

Πολλαπλασιασμός πίνακα με πίνακα

Έστω *A*, *B*, *C* πίνακες NxN και *C*=*A*·*B*

N=3

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj}$$

```
do i=1,N
  do j=1,N
    s=0.
    do k=1,N
      s=s+a(i,k)*b(k,j)
    enddo
    c(i,j)=s
  enddo
enddo
```

```
c = MATMUL(a,b)
```

Array constructor (1)

```
(/value1,value2, . . ./)
```

```
program smoothing
IMPLICIT NONE
real, allocatable, dimension(:) :: F, Fsmooth
integer N,i

N=10

allocate(F(N), Fsmooth(N))

F(1)=0.  ;F(2)=100.;F(3)=250.;F(4)=290.; F(5)=460.
F(6)=500.;F(7)=605.;F(8)=760.;F(9)=830.;F(10)=910.
```

```
F=(/0.,100.,250.,290.,460.,500.,605.,760.,830.,910./)
```

Array constructor (2)

(/ (*variable, implied_do_control*) /)

```
program array_constructor
IMPLICIT NONE
real, allocatable, dimension(:) :: F
integer N,i

N=10

allocate(F(N))

F(1)=10.
do i=2,9
  F(i)=0.
enddo
F(N)=100.
```

F=(/ 10.,(0.,i=2,9),100. /)

Writing real programs

real programs tend to be large

```
! Global variables
INTEGER, PARAMETER :: real8=8
REAL(KIND=real8), PARAMETER :: zero=0.0_real8
REAL(KIND=real8), PARAMETER :: one=1.0_real8
REAL(KIND=real8), PARAMETER :: eps=epsilon(1.0_real8)
! Global definitions for GMRES
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: hess ! Upper Hessenberg matrix
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: vm   ! Krylov subspace basis
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:),   SAVE :: rs   ! Right hand side of Hess system
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:),   SAVE :: c,s  ! Rotation coefficients
INTEGER, SAVE :: outer_iter                                 ! counter of GMRES outer itarations
INTEGER, SAVE :: inner_iter                                 ! counter of GMRES inner iterations
INTEGER, SAVE :: gmres_status                               ! check point for the reverse commun
REAL(KIND=real8), SAVE :: beta,conv                         ! conv=tol*norm2(rhs)
                                                            ! tol: input user tolerance
                                                            ! beta: norm2(rhs-Ax)
! Global definitions for DEFLATION
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: hess_original ! original hess for deflation
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: au_defl       ! stores AU (see Erhel 1996)
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: u_defl        ! stores U (see Erhel 1996)
REAL(KIND=real8), ALLOCATABLE, DIMENSION(:,:), SAVE :: t_defl        ! stores T (see Erhel 1996)
LOGICAL,          ALLOCATABLE, DIMENSION(:),   SAVE :: inner_contris ! if allocated by the user then use the
                                                                     ! user contributions(array inner_cotris)
                                                                     ! for the computation of the local inner products
LOGICAL, SAVE :: precond      ! if true use the preconditioner
LOGICAL, SAVE :: update       ! if true update the preconditioner
LOGICAL, SAVE :: adapt_update ! if true use adaptive approach of u_defl after dim_defl>maxdefl
LOGICAL, SAVE :: use_precond  ! is false if maxdefl=0 else is always true
LOGICAL, SAVE :: harmonic     ! if true then (Morgan 2000) else (Erhel 1996) for the
                              ! computation of the smallest eigenvalue of Hessenberg
LOGICAL, SAVE :: rcond_check  ! if true compute the condition number of Hessenberg
LOGICAL, SAVE :: pick_second  ! if true pick 2 eigenvalues in each outer gmres iteration
LOGICAL, SAVE :: mem_check    ! if true calculate the maximum required memory
LOGICAL, SAVE :: print_conv   ! if true print the convergence history
LOGICAL, SAVE :: print_eigs   ! if true print the approximated eigenvalues
LOGICAL, SAVE :: print_cond   ! if true print the approximated condition number
LOGICAL, SAVE :: keep_eigenpairs ! if true dim_defl,u_defl,eigen_defl are available to the calling program
INTEGER, SAVE :: conv_unit    ! the UNIT number for printing the convergence histoy
INTEGER, SAVE :: eigs_unit    ! the UNIT number for printing the eigenvalue history
INTEGER, SAVE :: cond_unit    ! the UNIT number for printing the condition number histoy
INTEGER, SAVE :: dim_defl     ! the current dimension of the deflation subspace
INTEGER, SAVE :: increase     ! the number of the new eigenvectors which added to u_defl
INTEGER, SAVE :: increaseT    ! increaseT copy of increase
TYPE compl
   REAL(KIND=real8) :: r,i
   INTEGER :: ind
END TYPE compl
REAL(KIND=real8), SAVE :: largest
LOGICAL, SAVE :: was,updateT
LOGICAL, SAVE :: reortho
TYPE(compl), ALLOCATABLE, DIMENSION(:), SAVE :: eigen_defl ! This is the array of eigenvalues returned to user
                                                           ! when keepeigenpairs=.TRUE.
INTEGER, SAVE :: MyID,Nprocs  ! The MPI process ID and number of total processes
REAL(KIND=real8), SAVE :: stime_inner,etime_inner,total_inner,stime_com,total_inner_com
INTEGER, SAVE :: OS_use       ! Orthogonalization Scheme for GS  (OS_use=1)
                              !                               MGS (OS_use=2) (Default)
                              !                               HO  (OS_use=3) (Not yet implemeted)
CONTAINS

SUBROUTINE pgmres(MyID_user,Nprocs_user,n,maxm,maxiter,maxdefl,tol,rhs,x,vec_out,vec_in,icode,    &
                  secondeig,harmoniceig,adapt,condition,reorthog,memcheck,                        &
                  printconv,convunit,printeigs,eigsunit,printcond,condunit,                       &
                  keepeigenpairs,OS)
IMPLICIT NONE
INCLUDE 'mpif.h'
! This subroutine solves the sparse linear system A x = rhs of order n
! Dummy Arguments
INTEGER, INTENT(IN) :: MyID_user  ! The MPI process ID
```

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (1)

Οι συστάσεις ισχύουν για κάθε γλώσσα προγραμματισμού που μπορείτε να χρησιμοποιήσετε.

1. Ψυχολογία πίσω από τον προγραμματισμό Η/Υ

• Μικρά λάθη (π.χ. λείπει ένα κόμμα, μεταβλητές με λάθος δήλωση) αποτρέπουν τη γρήγορη ανάπτυξη του κώδικα. → Εξάσκηση, Υπομονή

• Είναι γεγονός ότι ο προγραμματισμός Η/Υ είναι δεύτερη φύση για ορισμένα άτομα. Αν ανήκετε σε αυτή την ομάδα, συγχαρητήρια. Αν όχι, μην ανησυχείτε γι' αυτό. Απλά μη συγκρίνετε τον εαυτό σας με τους άλλους ανθρώπους. → Εξάσκηση

• Computer Programming Anxiety. Αυτό το άγχος είναι φυσιολογικό.

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (2)

2. Η σημασία του σχεδιασμού

Είναι εξαιρετικά σημαντικό να κατανοήσουμε το πρόβλημα που πρέπει να επιλυθεί.

Πρέπει να σκεφτούμε και να σχεδιάσουμε τον κώδικα.

Στο τέλος ξεκινάμε τη συγγραφή του κώδικα.

Συχνά, κάποιος μπορεί να περάσει ένα σημαντικό χρονικό διάστημα στη συγγραφή κώδικα που δε λύνει το πρόβλημα που του έχει ανατεθεί.

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (3)

3. Σταδιακή ανάπτυξη κώδικα (incremental code development)

Αυτός είναι ο πιο σημαντικός κανόνας του προγραμματισμού ηλεκτρονικών υπολογιστών!

Το σημαντικότερο λάθος που μπορείτε να κάνετε είναι να προσπαθήσετε να γράψετε τον κώδικα με τη μία.

Σταδιακή ανάπτυξη κώδικα σημαίνει ότι κάθε φορά:
προσθέτουμε ένα μικρό κομμάτι κώδικα,
ελέγχουμε ότι το πρόγραμμα παράγει τα αναμενόμενα αποτελέσματα,
και συνεχίζουμε την ίδια διαδικασία με νέο κομμάτι κώδικα.

Για παράδειγμα, εάν το πρόγραμμά σας λειτουργεί, και προσθέσετε δύο γραμμές κώδικα και το πρόγραμμα σταματήσει να λειτουργεί, μπορείτε να στοιχηματίσετε ότι τα υπεύθυνα μέρη είναι οι δύο γραμμές που μόλις προσθέσατε.

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (3)

4. Μην κάνετε υποθέσεις

Η είσοδος των δεδομένων είναι σωστή!
Αυτό συνήθως οδηγεί σε ένα πρόγραμμα που χρησιμοποιεί σκουπίδια για να υπολογίσει σκουπίδια ή διαφορετικά GIGO (Garbage In, Garbage Out).
Σιγουρευτείτε ότι τα δεδομένα έχουν διαβάσει σωστά.

Όλες οι βιβλιοθήκες λειτουργούν σωστά!
Εάν έχετε αμφιβολίες σχετικά με το τι κάνει ένα υποπρόγραμμα ή αν ξέρετε να το χρησιμοποιείτε σωστά, σταματήσετε(!) τη συγγραφή του κώδικα. Κοιτάξτε τις οδηγίες χρήσης, γράψετε ένα μικρό πρόγραμμα που ελέγχει τι κάνει το υποπρόγραμμα.

4. Δοκιμή, δοκιμή, δοκιμή . . .

Η δοκιμή είναι μια διαδικασία που μπορεί να είναι κουραστική, αλλά είναι εξαιρετικά σημαντική. Θα πρέπει να ελέγχετε τον κώδικα σας όσο το δυνατόν συχνότερα. Δεν πρέπει να περιμένετε μέχρι το τέλος για να δοκιμάσετε τον κώδικά σας. Σε βάθος χρόνου θα σας εξοικονομήσει χρόνο, διότι:

- Θα διαπιστώσετε προβλήματα νωρίς. Αυτό θα απλοποιήσει τη διαδικασία εντοπισμού σφαλμάτων,
- Θα έχετε εμπιστοσύνη στον κώδικά σας,
- Θα διαπιστώσετε προβλήματα σχεδιασμού νωρίς.

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (5)

5. Δοκιμή με ένα απλό σύνολο δεδομένων

6. Αναπτύξτε ένα οργανωμένο σύνολο δοκιμών έτσι ώστε να καλύπτουν όσο το δυνατόν περισσότερες περιπτώσεις

7. Σωστή διαχείριση χρόνου

8. Ακολουθήστε συμβάσεις στην ονομασία μεταβλητών και υποπρογραμμάτων
calulate_factorial vs sfafrewfevfww

9. Ακολουθήστε το μοντέλο του προληπτικού προγραμματισμού

10. Γράψτε τον κώδικα σαν να πρόκειται να τον ξεχάσετε αύριο!

11. Μην διστάζετε να ανακαλύπτετε λάθη (bugs) στον κώδικά σας

12. Κατανοήστε τι έχετε κάνει και γιατί λειτουργεί ο κώδικας

13. Κρατήστε αντίγραφα παλιότερων εκδόσεων του κώδικά σας

Συστάσεις για συγγραφή προγραμμάτων σε Η/Υ (6)

Be patient!

Ο προγραμματισμός απαιτεί χρόνο, προσπάθεια και εξάσκηση, με την πάροδο του χρόνου θα είστε σε θέση να αναπτύξετε κώδικα γρήγορα και αποδοτικά.