Κεφάλαιο 4:
Μεταβλητές, Εκφράσεις, Εντολές

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας

Μεταβλητές (1)

- **Μεταβλητή**: Αφαιρετική αναπαράσταση διεύθυνσης μνήμης, ή συλλογής διευθύνσεων μνήμης του Η/Υ.
- **Αναγνωριστικό** ή **Όνομα** (**id**): Συνδυασμός από αλφαριθμητικούς χαρακτήρες. Είναι ένα μόνο από τα συστατικά μιας μεταβλητής. Είναι συστατικό και άλλων δομών: Υποπρογράμματα, παράμετροι, …
  - Σχεδιαστικά θέματα για τα ονόματα:
    - Μορφή. Διάκριση πεζών/κεφαλαίων γραμμάτων, άλλοι περιορισμοί.
    - Δεσμευμένες Λέξεις vs Λέξεις Κλειδιά.

Μεταβλητές (2)

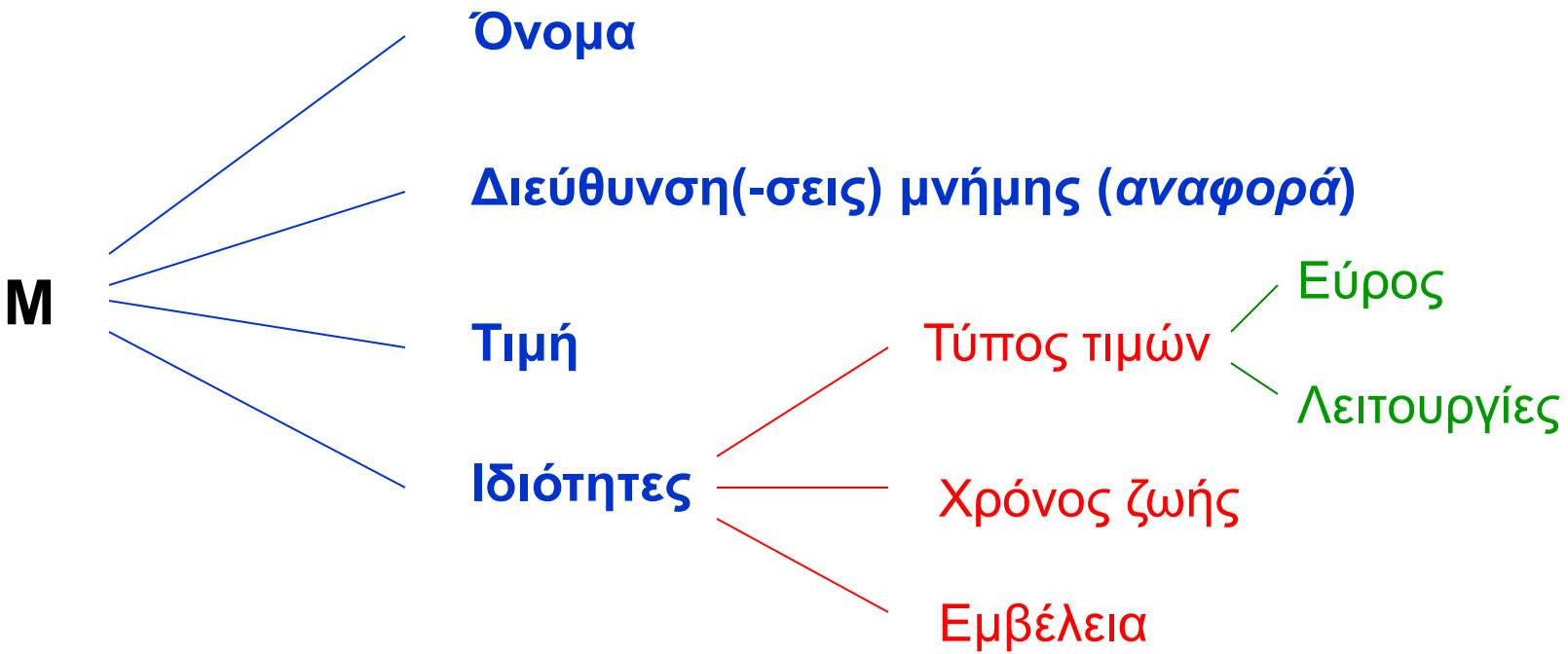
- X = 2.5;

  «*X είναι το όνομα (id) μιας θέσης (ή θέσεων) μνήμης που περιέχει(-ουν) πλέον τον αριθμό 2.5*»

  Και η πρόταση αυτή δεν είναι πλήρης…

- Μία μεταβλητή **M** αποτελείται από 4 στοιχεία:

Μεταβλητές (3)



Μεταβλητές (4)

- Η σχέση **ονόματος** – **διεύθυνσης** δεν είναι αμφιμονοσήμαντη.
- *Α. Ίδιο όνομα – Διαφορετικές διευθύνσεις*
  - Σε ένα πρόγραμμα, σε κάθε μία από δύο συναρτήσεις $\mathbf{f_1}$, $\mathbf{f_2}$, μπορεί να ορίζεται μία μεταβλητή (τοπική) με το όνομα **X**.
  - Όταν μία συνάρτηση **f** καλείται αναδρομικά, κάθε κλήση της **f** ορίζει διαφορετικές θέσεις μνήμης (και άρα μεταβλητές), για τα ονόματα που είναι τοπικές μεταβλητές της **f**.
  - Αν η συνάρτηση **f** με τοπική μεταβλητή την **Y**, κληθεί από τις συναρτήσεις $\mathbf{f_1}$, $\mathbf{f_2}$, η **Y** θα έχει συνδεθεί με διαφορετικές μνήμες στις $\mathbf{f_1}$, $\mathbf{f_2}$

Μεταβλητές (5)

- Β. *Διαφορετικά ονόματα – Ίδια διεύθυνση*

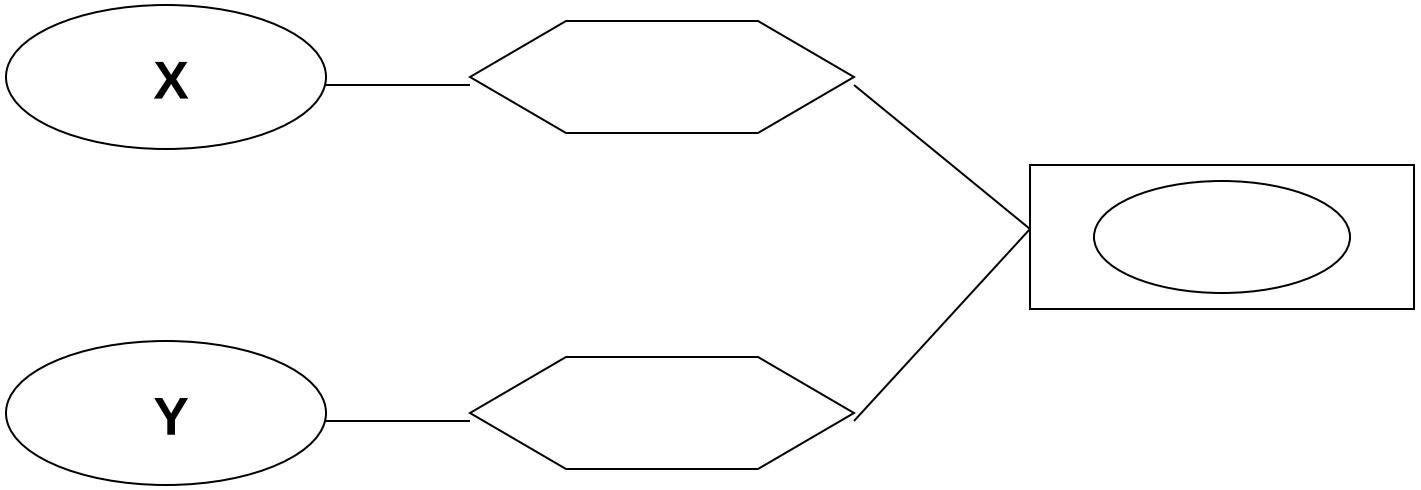
  Ψευδωνυμία (κακό για αναγνωσιμότητα…)

  Μπορεί να δημιουργηθεί με διάφορους τρόπους:

  - *Άμεσα*
    - Στη FORTRAN με την εντολή EQUIVALENCE
    - Στις C, C++ με union variables
  - *Έμμεσα*
    - Με την κλήση υποπρογραμμάτων (by reference)
    - Με τη χρήση pointers

Μεταβλητές (6)

Ψευδωνυμία

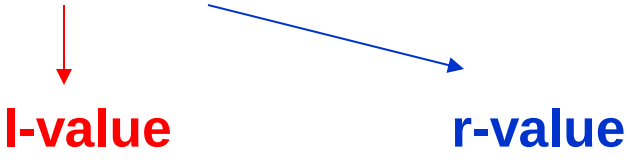


Μεταβλητές (7)

- Μία μεταβλητή **X** έχει 2 χρήσεις.

  Στην εντολή ανάθεσης:

  X = **X** + 1



- l-value (left value): Αναφορά στη θέση μνήμης
- r-value (right value): Αποαναφοροποίηση για να πάρουμε την τιμή

  Ουσιαστικά ο τελεστής **=** της εντολής ανάθεσης, είναι ένας τελεστής που δέχεται 2 ορίσματα: τιμή (r-value) και αναφορά (l-value)

Μεταβλητές (8)

- **Αποαναφοροποίηση** (dereferencing): Πρόσβαση στη θέση μνήμης, και λήψη της υπάρχουσας τιμής.
- Στις περισσότερες γλώσσες, ο r-value ρόλος μιας μεταβλητής, συνάγεται *εμμέσως* από τη θέση της μεταβλητής, δηλαδή όταν βρίσκεται δεξιά από τον τελεστή ανάθεσης **=**
- Υπάρχουν γλώσσες, στις οποίες υπάρχει *ρητή* δήλωση του r-value ρόλου.

  Π.χ. στην BLISS:

  X := .X + 1

Δέσμευση (1)

- **Δέσμευση** (Binding): Συσχέτιση δύο οντοτήτων. Π.χ. *ιδιότητας* με *μεταβλητή*, *operation* με *σύμβολο*.
- **Χρόνος Δέσμευσης** (Binding Time): Ο χρόνος που γίνεται η δέσμευση.
- Έχουμε τους εξής Χρόνους Δέσμευσης:
  1) Χρόνος Σχεδιασμού της Γλώσσας
  2) Χρόνος Υλοποίησης της Γλώσσας
  3) Χρόνος Γραφής του Προγράμματος
  4) Χρόνος Μετάφρασης του Προγράμματος
  5) Χρόνος Σύνδεσης του Προγράμματος
  6) Χρόνος Φόρτωσης του Προγράμματος
  7) Χρόνος Εκτέλεσης του Προγράμματος

Δέσμευση (2)

1. Χρόνος Σχεδιασμού της Γλώσσας
2. Χρόνος Υλοποίησης της Γλώσσας
3. Χρόνος Γραφής του Προγράμματος
4. Χρόνος Μετάφρασης του Προγράμματος
5. Χρόνος Σύνδεσης του Προγράμματος
6. Χρόνος Φόρτωσης του Προγράμματος
7. Χρόνος Εκτέλεσης του Προγράμματος

- Διάφορες οντότητες δεσμεύονται σε διάφορους χρόνους:
  - Ένα **σύμβολο** (π.χ. *) με μία **operation**, στο χρόνο (1)
  - Ένας **Τύπος Δεδομένων** (π.χ. int) με **εύρος τιμών**, στο (2)
  - Ένα **όνομα** (π.χ. A) με μία **έννοια**, στο (3)
  - Μία **μεταβλητή** με ένα **Τύπο Δεδομένων**, στο (4)
  - Μία **Συνάρτηση** βιβλιοθήκης με τον **κώδικα** χρήστη, στο (5)
  - Μία **μεταβλητή** με θέση(-εις) **μνήμης**, στο (6)
  - Μία **μεταβλητή** με **τιμή**, στο (7)

Δέσμευση (3)

1. Χρόνος Σχεδιασμού της Γλώσσας
2. Χρόνος Υλοποίησης της Γλώσσας
3. Χρόνος Γραφής του Προγράμματος
4. Χρόνος Μετάφρασης του Προγράμματος
5. Χρόνος Σύνδεσης του Προγράμματος
6. Χρόνος Φόρτωσης του Προγράμματος
7. Χρόνος Εκτέλεσης του Προγράμματος

- Παράδειγμα: Στον κώδικα

```
int count;
count = count + 5;
```

υπάρχουν (μεταξύ άλλων) οι δεσμεύσεις:

- Το σύνολο των πιθανών Τύπων Δεδομένων του count, **(1)**
- Ο Τύπος Δεδομένων του count στο πρόγραμμά μας, **(4)**
- Το σύνολο των πιθανών τιμών του count, **(2)**
- Η τιμή του count στο πρόγραμμά μας, **(7)**
- Το σύνολο των πιθανών εννοιών του +, **(1)**
- Η έννοια του + στο πρόγραμμά μας, **(4)**
- Η εσωτερική αναπαράσταση του 5, **(2)**

Δέσμευση (4)

- **Διάρκεια Ζωής της δέσμευσης**: Η χρονική περίοδος μεταξύ της δημιουργίας και της καταστροφής μιας δέσμευσης (π.χ. ενός ονόματος και μιας μεταβλητής)
- **Διάρκεια Ζωής αντικειμένου**: Η χρονική περίοδος μεταξύ της δημιουργίας και της καταστροφής του αντικειμένου (π.χ. μεταβλητής)
- Δεν συμπίπτουν απαραίτητα:
  - *Ένα αντικείμενο μπορεί να έχει μεγαλύτερη διάρκεια από αυτήν της δέσμευσης*. Π.χ. όταν μια μεταβλητή μεταβιβάζεται με αναφορά σε μια υπορουτίνα, η δέσμευση του ονόματος της παραμέτρου και της μεταβλητής έχει μικρότερη διάρκεια από αυτήν της μεταβλητής.
  - *Μία δέσμευση μπορεί να έχει μεγαλύτερη διάρκεια από αυτήν του αντικειμένου*. Π.χ. αν ένα αντικείμενο που έχει δημιουργηθεί με new στην C++, μεταβιβαστεί ως παράμετρος με & και μετά καταστραφεί με delete πριν επιστρέψει η υπορουτίνα, έχουμε **αιωρούμενη αναφορά** (dangling reference).

Δέσμευση (5)

- **Στατική** Δέσμευση: Όταν γίνεται πριν το χρόνο εκτέλεσης του προγράμματος (6) και δεν αλλάζει κατά τη διάρκεια της εκτέλεσης.
- **Δυναμική** Δέσμευση: Αλλιώς.

Α. Δέσμευση Μεταβλητής με Τύπο Δεδομένων (ΤΔ)

- Στατική Δέσμευση
  - Με **Ρητή** (explicit) δήλωση
    - Π.χ. στη C: int A;
  - Με **Έμμεση** (implicit) δήλωση
    - Π.χ. στη συναρτησιακή γλώσσα ML (Meta Language):

Δέσμευση (6)

Οι ΤΔ των εκφράσεων καθορίζονται από τις (τυχούσες) σταθερές:
Στην εντολή fun circ(r) = 3.14 * r * r, το circ θεωρείται real
Στην εντολή fun times(x) = 10 * x, το times θεωρείται integer
Αν δεν συνάγεται ο ΤΔ, συντακτικό λάθος. Π.χ στο fun sq(x) = x * x
χρειάζεται να γίνει ρητή δήλωση:
fun sq(x): int = x * x **ή** fun sq(x: int) = x * x **ή** fun sq(x) = (x: int) * x

Τώρα, όλες οι γλώσσες (εκτός ML, Perl) έχουν ρητή δήλωση.

Perl: $a (βαθμωτός ΤΔ), @a (array ΤΔ), διαφορετικές μετ/τές.

Ορισμένες αρχικές γλώσσες είχαν ένα είδος έμμεσης δήλωσης.

Π.χ. στην FORTRAN αν υπήρχε id που δεν είχε δηλωθεί ρητά, συναγόταν έμμεσα ο ΤΔ ως εξής:

- Αν το όνομα αρχίζει από I, J, K, L, M, N είναι INTEGER
- Αλλιώς είναι REAL

Δέσμευση (7)

- Δυναμική Δέσμευση
  - Η μεταβλητή δεσμεύεται με ΤΔ κάθε φορά που παίρνει τιμή, δηλαδή όταν είναι l-value.
  - Δεσμεύεται με τον ΤΔ που έχει η τιμή της μεταβλητής r-value, ή της έκφρασης που είναι δεξιά του =
  - Γλώσσες: APL, SNOBOL, PHP, JavaScript
  - ΘΕΤΙΚΑ:
    - Ευελιξία
    - Δυνατότητα για generic συναρτήσεις
  - ΑΡΝΗΤΙΚΑ:
    - Δεν υπάρχει δυνατότητα εντοπισμού λαθών στη μετάφραση
    - Μεγάλο κόστος για έλεγχο των ΤΔ κατά την εκτέλεση

Δέσμευση (8)

Β. Δέσμευση Μεταβλητής με Διεύθυνση(-εις) Μνήμης

- **Εκχώρηση** (allocation): Η διαδικασία δέσμευσης της μεταβλητής με Δ/νση Μνήμης από τις διαθέσιμες.
- **Αποδέσμευση** (de-allocation): Η διαδικασία αποδέσμευσης και επιστροφής της μνήμης στις διαθέσιμες.
- **Διάρκεια Ζωής**: Ο χρόνος κατά τον οποίο η μεταβλητή είναι δεσμευμένη με διεύθυνση(-εις) μνήμης. Δηλαδή:

Διάρκεια Ζωής = (Χρόνος Αποδέσμευσης) – (Χρόνος Εκχώρησης)

Δέσμευση (9)

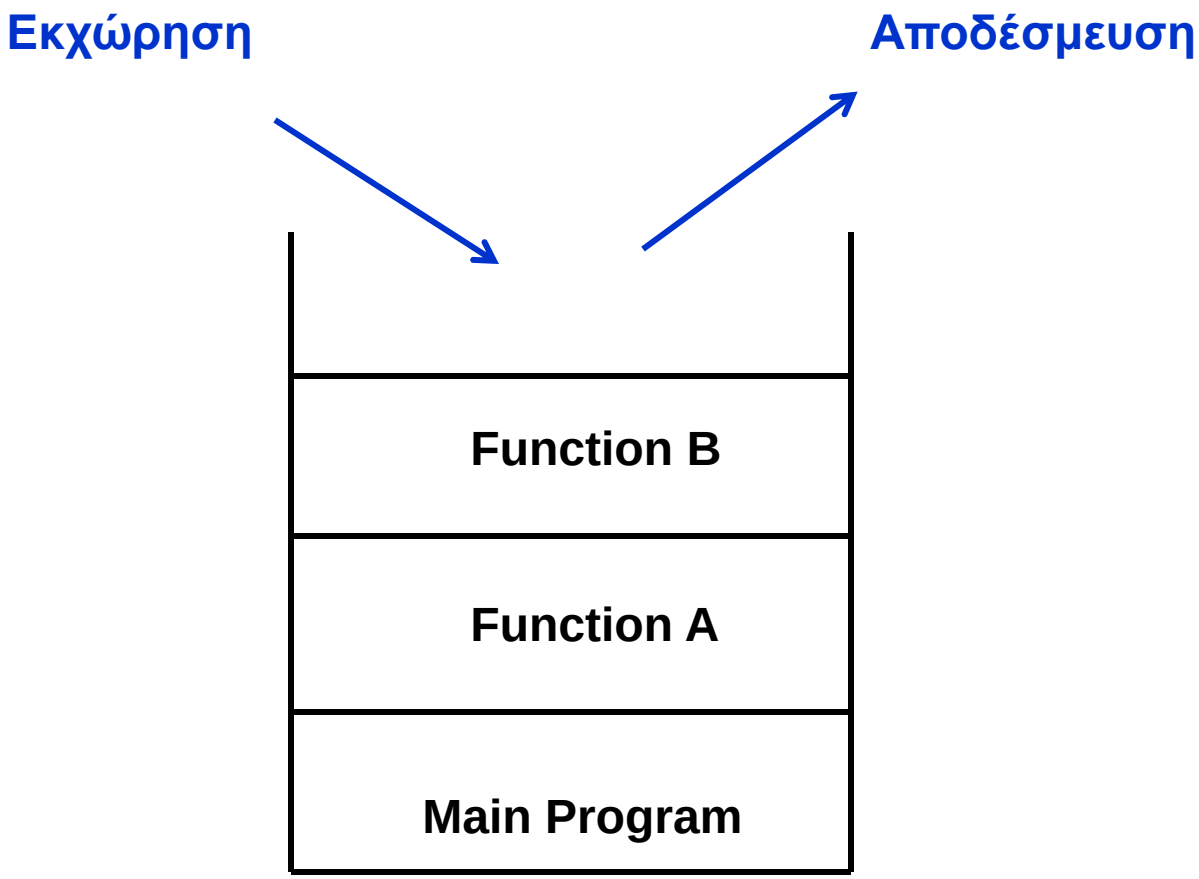
Με βάση τον τρόπο δέσμευσης και τους χρόνους εκχώρησης και αποδέσμευσης, έχουμε διαφορετικού τύπου μεταβλητές:

- **Static Variables**
  - Η εκχώρηση γίνεται πριν την εκτέλεση του προγράμματος και δεν αλλάζει στη διάρκειά της.
  - Global μεταβλητές.
  - Στις C, C++, Java με static. Η Pascal όχι. Στις αρχικές FORTRAN (από I ως και IV), όλες οι μεταβλητές static.
  - Πολλές φορές είναι history sensitive.
  - ΠΛΕΟΝΕΚΤΗΜΑ: Αποδοτικές (μικρό overhead)
  - ΜΕΙΟΝΕΚΤΗΜΑ: Μικρή ευελιξία (όχι αναδρομικότητα)

Δέσμευση (10)

- **Stack – Dynamic Variables**
  - Η εκχώρηση γίνεται όταν «εκτελείται» η εντολή δήλωσης της μεταβλητής (ο ΤΔ είναι στατικά συνδεδεμένος)
  - Στη C και στη C++ είναι τέτοιες οι μεταβλητές
  - Στην αρχή εκτέλεσης του υποπρογράμματος γίνεται εκχώρηση της τοπικής μεταβλητής, αποδέσμευση μόλις λήξει η εκτέλεση.
  - Εκχώρηση από την run-time stack μνήμη (οργανωμένη)
  - Εκχώρηση και Αποδέσμευση με τρόπο *Last-In First-Out* (LIFO), συνήθως σε συνδυασμό με κλήση υπ/τος.
  - ΜΕΙΟΝΕΚΤΗΜΑ: Δεν μπορούν να είναι history sensitive.
  - ΠΛΕΟΝΕΚΤΗΜΑ: Αναδρομή

Δέσμευση (11)



ΠΑΡΑΔΕΙΓΜΑ:
Η run-time stack, αφού το **Main** έχει καλέσει τη συνάρτηση **A**, η οποία με τη σειρά της έχει καλέσει τη συνάρτηση **B**

Δέσμευση (12)

- **Explicit Heap – Dynamic Variables**
  - Είναι ανώνυμα αντικείμενα των οποίων οι διευθύνσεις εκχωρούνται και αποδεσμεύονται με εντολές που εκτελούνται κατά την εκτέλεση του προγράμματος.
  - Χρησιμοποιούν τη heap storage: Ανοργάνωτη συλλογή θέσεων μνήμης με «απρόβλεπτη» χρήση.
  - Οι μεταβλητές αυτές μπορούν να προσπελασθούν μόνο από pointers ή objects.
  - Εκχώρηση και Αποδέσμευση σε οποιαδήποτε χρονική στιγμή.
  - Στην Java όλα τα objects είναι explicit heap – dynamic variables

Δέσμευση (13)

- **Implicit Heap – Dynamic Variables**
  - Γίνεται εκχώρηση από την heap storage μόνο όταν η μεταβλητή παίρνει τιμή με εντολή ανάθεσης (l-value).
  - Μαζί δεσμεύονται με τη μεταβλητή όλες οι ιδιότητες του ΤΔ της τιμής (δυναμική δέσμευση της μεταβλητής με ΤΔ)
  - Ουσιαστικά είναι μόνο ονόματα που δεσμεύονται με ότι θέλουμε.
  - APL, ALGOL-68
  - ΘΕΤΙΚΑ:
    - Ευελιξία
    - Δυνατότητα για generic συναρτήσεις
  - ΑΡΝΗΤΙΚΑ:
    - Μεγάλο κόστος για συντήρηση των δυναμικών ιδιοτήτων κατά την εκτέλεση

Σταθερές (1)

- Είναι «μεταβλητές» που δεσμεύονται με τιμή, **μόνο** όταν δεσμεύονται με μνήμη.
- Η τιμή δεν μπορεί να αλλάξει με εντολή ανάθεσης.
- C, C++ : const int s = 100; Pascal : const s = 100;
- Η αρχική C δεν είχε σταθερές. Όμως ο C preprocessor περιλαμβάνει macro:

  #define LEN 100 : αντικαθίστανται όλες οι εμφανίσεις του LEN με το 100 (named literal)

Σταθερές (2)

- Οι σταθερές βοηθούν την αναγνωσιμότητα και την αξιοπιστία του προγράμματος.
- Εύκολη αλλαγή μεγέθους arrays και άλλων δομών.
- Στην Ada υπάρχει μεγάλη ευελιξία:

  MAX: constant integer := 2 * WIDTH + 1; (dynamic)

Αρχικές Τιμές Μεταβλητών

- **Αρχικοποίηση**: Δέσμευση μεταβλητής με τιμή, τη στιγμή που δεσμεύεται με μνήμη (όπως η σταθερά).
- Διαφορά από σταθερά: Η σταθερά δεν αλλάζει τιμή.
- FORTRAN: REAL PI, DATA PI /3.14159/
- C: int i = 0; float e = 1.0e-5;
- Pascal: Όχι…
- Ada: SUM: integer := 0;
  (Σταθερά: SUM: constant integer := 0;)
- ALGOL 68: int first := 10;
  (Σταθερά: int first = 10;)
  Κακή αναγνωσιμότητα…

Εκφράσεις (1)

- Βασικός τρόπος προσδιορισμού υπολογισμών σε μια ΓΠ
- Τρεις κατηγορίες Εκφράσεων:
  1. Αριθμητικές
  2. Σχεσιακές
  3. Λογικές

Εκφράσεις (2)

1. Αριθμητικές Εκφράσεις

   Κατασκευές από:

   - Τελεστές (operators)
   - Τελεστέους (operands)
   - Παρενθέσεις
   - Κλήσεις Συναρτήσεων

   Π.χ. A + (B * C – D) – SYN(X)

   Τελεστές:

   - Μοναδιαίοι (unary) π.χ. **–** X
   - Δυαδικοί (binary) π.χ. X **+** Y
   - Τριαδικοί (ternary) π.χ. A **?** B **:** C

Εκφράσεις (3)

Η σειρά υπολογισμών σε μια αριθμητική έκφραση καθορίζεται από τα εξής:

Α. Ιεραρχία Τελεστών

FORTRAN	Pascal	C	Ada
**	* / div mod	Postfix ++ --	** abs
* /	+ -	Prefix ++ --	* / mod
+ -		Unary -	Unary + -
		* / %	Binary + -
		Binary + -	

Εκφράσεις (4)

Β. Associativity (Σειρά υπολογισμών στο ίδιο επίπεδο ιεραρχίας)

ΓΕΝΙΚΑ: Από αριστερά προς τα δεξιά

Εκτός από την ύψωση σε δύναμη:

A**B**C

- FORTRAN: από δεξιά προς αριστερά
- Ada: δεν υπάρχει κανόνας, χρειάζονται παρενθέσεις

Εκφράσεις (5)

FORTRAN	*Αριστερά*: * / + - *Δεξιά*: **
Pascal	*Αριστερά*: Όλα
C	*Αριστερά*: postfix (++ --) * / % binary (+ -) *Δεξιά*: prefix (++ --) unary -
C++	*Αριστερά*: * / % binary (+ -) *Δεξιά*: ++ -- unary -
Ada	*Αριστερά*: Όλα εκτός ** *Χωρίς κανόνα*: **

Γ. Παρενθέσεις

Αλλάζουν τη σειρά υπολογισμού των A και B

Εκφράσεις (6)

Εκφράσεις υπό συνθήκη

Μπορούν να χρησιμοποιηθούν όπως κάθε έκφραση

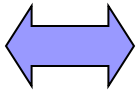
Π.χ. ο τριαδικός τελεστής των C, C++, Java:

<έκφραση_1> **?** <έκφραση_2> **:** <έκφραση_3>

(αν <έκφραση_1> = TRUE , τότε <έκφραση_2> , αλλιώς <έκφραση_3>)

μπορεί να χρησιμοποιηθεί και σε εντολή ανάθεσης:

```
if (c == 0)
  then av = 0
  else av = sum/count
```



```
av = (c==0) ? 0 : sum/count
```

Εκφράσεις (7)

- Παράδειγμα:

  M = (Vg < Va) **?** Vg **:** Va

  B = (M < 5) **?** M **:** 0.7 * Vg + 0.3 * Va

  Ο βαθμός σας στο μάθημα…

  Vg = Βαθμός γραπτού

  Va = Βαθμός άσκησης

B = (((Vg < Va) ? Vg : Va) < 5) **?** ((Vg < Va) ? Vg : Va) **:** 0.7 * Vg + 0.3 * Va

Εκφράσεις (8)

2. Σχεσιακές Εκφράσεις
   (Relational Expressions)

   Αποτελούνται από:

- *Σχεσιακούς Τελεστέους* (Αριθμητικές ή Λογικές Εκφράσεις)
- *Σχεσιακούς Τελεστές*

- Η τιμή τους είναι *Boolean* (εκτός από τη C)
- Σχεσιακός Τελεστής: Συγκρίνει τις τιμές δύο τελεστέων.

Εκφράσεις (9)

Σχεσιακοί Τελεστές

Λειτουργία	Pascal	Ada	C	FORTRAN 77
Ίσο	=	=	==	.EQ.
Άνισο	<>	/=	!=	.NE.
Μεγαλύτερο από	>	>	>	.GT.
Μικρότερο από	<	<	<	.LT.
Μεγαλύτερο ή ίσο	>=	>=	>=	.GE.
Μικρότερο ή ίσο	<=	<=	<=	.LE.

Εκφράσεις (10)

- Η FORTRAN αρχικά χρησιμοποιούσε τα παραπάνω σύμβολα, διότι οι διατρητικές μηχανές δεν είχαν τα σύμβολα < >
- Από τη FORTRAN 90 και μετά, χρησιμοποιεί τα σύμβολα της Pascal, με == για το «ίσο»
- Με ποια σειρά θα εκτελεστεί το παρακάτω;

A + 1 > B * 2

(A + 1) > (B * 2)

Σειρά: *2* *3* *1*

Οι σχεσιακοί τελεστές έχουν χαμηλότερη προτεραιότητα από τους αριθμητικούς.

Εκφράσεις (11)

- Οι JavaScript και PHP έχουν δύο ακόμα σχεσιακούς τελεστές: **===** και **!==**
- Διαφέρουν από τα == και != ως προς το ότι δεν κάνουν μετατροπή τύπου κατά τη σύγκριση. Π.χ.
  - “7” == 7 και “” == 0 είναι **true**

    δηλαδή, μετατροπή του “7” σε 7 και του “” σε 0 (!)
  - “7” === 7 και “” === 0 είναι **false**
- Αντίστοιχα η Ruby έχει το **eql?** για έλεγχο ισότητας χωρίς μετατροπή τύπου. Χρησιμοποιεί και το === αλλά μόνο στο when της εντολής case (…)

Εκφράσεις (12)

3. Λογικές Εκφράσεις (Boolean Expressions)

Αποτελούνται από:

- *Boolean Τελεστέους*, δηλαδή:
  - *Boolean Μεταβλητές*
  - *Boolean Σταθερές* (π.χ. TRUE, FALSE)
  - *Σχεσιακές Εκφράσεις* (έχουν Boolean τιμή)
- *Boolean Τελεστές*

- Η τιμή τους είναι *Boolean* (εκτός από τη C)
- Boolean Τελεστής: Εκτελεί Boolean πράξη (AND, OR, NOT) στις τιμές δύο Boolean τελεστέων.

Εκφράσεις (13)

Ιεραρχία Boolean Τελεστών

Pascal	Ada	C	FORTRAN 77
NOT	NOT	!	.NOT.
AND	AND, OR	&&	.AND.
OR		||	.OR.

- Οι αριθμητικές εκφράσεις είναι τελεστέοι σε σχεσιακές εκφράσεις, οι σχεσιακές εκφράσεις σε λογικές, και οι λογικές σε σχεσιακές.
- Συνεπώς χρειαζόμαστε ιεράρχηση όλων των τελεστών και των τριών τύπων εκφράσεων.

Εκφράσεις (14)

Ιεραρχία όλων των Τελεστών

FORTRAN 77	C	Pascal
**	!	NOT
* /	Postfix ++, --	* / div mod AND
+ -	Prefix ++, --	+ - OR
.EQ. .NE. .GT. .LT. .LE. .GE.	Unary -	= <> < <= > >=
.NOT.	* / %	
.AND.	Binary + -	
.OR.	< <= > >=	
	== !=	
	&&	
	||	

Εκφράσεις (15)

- Παράδειγμα στη **FORTRAN 77**:

  A+B.GT.2*C.AND.K.NE.0

  Σειρά: 2 3 1 5 4

  [(A+B).GT.(2*C)].AND.(K.NE.0)

- Στην **Pascal**, οι Boolean τελεστές έχουν υψηλότερη προτεραιότητα από τους σχεσιακούς. Έτσι, η έκφραση

  A > 5 OR A < 0 είναι απαράδεκτη.

  Πρέπει να γραφεί: (A > 5) OR (A < 0)

Εκφράσεις (16)

- Boolean τιμές
  - Στις περισσότερες γλώσσες: TRUE, FALSE
  - **Java**: Οι *boolean* τιμές είναι 1 bit (αρκεί). Δεν συνδέεται με int Τύπο Δεδομένων
  - **C++** : Έχει Τύπο Δεδομένων *bool*. Συνδέεται με int.
  - **C** : Δεν έχει Boolean Τύπο Δεδομένων. Χρησιμοποιούνται αριθμητικές τιμές:

    0 → FALSE

    άλλο → TRUE (παράγει το 1 για TRUE)

Η έκφραση A > B > 4 είναι νόμιμη στις C, C++

1 2

Εκφράσεις (17)

- **Υπολογισμός Περιορισμένης Έκτασης**

  (Short – Circuit Evaluation)

  Υπολογισμός μιας έκφρασης, χωρίς να χρειάζεται να υπολογιστούν όλοι οι τελεστέοι.

Α. Σε αριθμητικές εκφράσεις:

(4 * A) * (B / 3 -1) Τι επίπτωση υπάρχει εδώ;

Αν A = 0, δεν χρειάζεται να υπολογιστεί το (B / 3 -1).

- Δύσκολο να εντοπιστεί
- Χρησιμοποιείται σπάνια

Εκφράσεις (18)

Β. Σε λογικές εκφράσεις:

(A >= 0) AND (B < 10)

Αν A < 0, τότε (A >= 0) = FALSE και δεν χρειάζεται να υπολογιστεί το (B < 10)

- Πιθανές Παράπλευρες Συνέπειες (Side Effects):

  (A > B) && (B++/2)

  Το B θα αλλάζει τιμή (B++), μόνο όταν A > B

- C, C++, Java χρησιμοποιούν υπολογισμό περιορισμένης έκτασης

Εντολές Ανάθεσης (1)

Μηχανισμοί με τους οποίους ο χρήστης αλλάζει δυναμικά τη σύνδεση μεταβλητών με τιμές.

1. Απλή Ανάθεση

Γενική σύνταξη:

<μεταβλητή-στόχος> <τελεστής ανάθεσης> <έκφραση>

- Οι FORTRAN, BASIC, PL/1, C, C++ χρησιμοποιούν ως τελεστή ανάθεσης το =
- Μπορεί να δημιουργηθεί σύγχυση αν χρησιμοποιείται και ως σχεσιακός τελεστής (PL/1, BASIC). Π.χ. στην PL/1:

  A = B = C : Βάζει ως τιμή του A, την Boolean τιμή της σχεσιακής έκφρασης B = C

Εντολές Ανάθεσης (2)

- Η ALGOL και στη συνέχεια οι Pascal, Ada χρησιμοποίησαν το :=
- Οι FORTRAN, Pascal, Ada χρησιμοποιούν την εντολή ανάθεσης, μόνο ως κανονική εντολή ανάθεσης.
- Στις C, C++, Java, χρησιμοποιείται και ως *δυαδικός τελεστής*: Μπορεί να χρησιμοποιηθεί μέσα σε εκφράσεις.

2. Πολλαπλές μεταβλητές – στόχος

PL/1: SUM, TOTAL = 0

C, C++ : SUM = COUNT = 0

(πρώτα COUNT = 0, μετά SUM = COUNT)

Εντολές Ανάθεσης (3)

3. Μεταβλητές – στόχος υπό συνθήκη

C, C++ : FLAG **?** COUNT1 **:** COUNT2 = 0

(if FLAG then COUNT1 = 0 else COUNT2 = 0)

4. Περιληπτικές εντολές ανάθεσης

Πρώτη η ALGOL 68, ακολούθησε η C:

A+=B ↔ A = A + B

Εντολές Ανάθεσης (4)

5. Μοναδιαίος τελεστής ανάθεσης

- SUM = ++COUNT ↔ COUNT = COUNT + 1
  SUM = COUNT
- SUM = COUNT++ ↔ SUM = COUNT
  COUNT = COUNT + 1
- COUNT++ ↔ COUNT = COUNT + 1

Όταν εφαρμόζονται δύο μοναδιαίοι τελεστές ανάθεσης στον ίδιο τελεστέο, η σειρά είναι από δεξιά προς αριστερά:

-- COUNT++ ↔ -- (COUNT++)

Εντολές Ανάθεσης (5)

6. Εντολή ανάθεσης ως έκφραση (C, C++, Java)

Η εντολή ανάθεσης δημιουργεί ένα αποτέλεσμα (η τιμή που παίρνει η μεταβλητή – στόχος). Συνεπώς, μπορεί να χρησιμοποιηθεί ως έκφραση, ή ως τελεστέος σε μια άλλη έκφραση.

```
while (CH = getchar() != EOF) {…}
```

Σύγκριση νέου χαρακτήρα με EOF, και το αποτέλεσμα (0 ή 1) δίνεται ως τιμή στο CH. Ο τελεστής ανάθεσης είναι χαμηλότερης προτεραιότητας από σχεσιακούς τελεστές.

Αλλιώς: `while ((CH = getchar()) != EOF) {…}`

Τώρα, πρώτα θα πάρει τιμή το CH, μετά θα συγκριθεί η τιμή αυτή με το EOF.

Εντολές Ανάθεσης (6)

- Μειονέκτημα χρήσης εντολής ανάθεσης ως δυαδικός τελεστής: Ένα είδος side effect.
- Αποτέλεσμα, δύσκολες στην ανάγνωση εκφράσεις. Π.χ.

```
a = b + (c = d / b) -1
```

Σειρά εκτέλεσης:

```
c = d / b
temp = b + c
a = temp - 1
```

Εντολές Ανάθεσης (7)

Στη C είναι αποδεκτές οι δύο παρακάτω εντολές; Τι αποτέλεσμα θα έχουν;

1. if (X == Y) then A = 1 else A = 0;
2. if (X = Y) then A = 1 else A = 0;

Αποτέλεσμα:

1. Θα ελεγχθεί η ισότητα των X, Y. Αν X = Y τότε A=1.

2. Θα ελεγχθεί η τιμή που θα πάρει το X. Αν δεν είναι 0, τότε A=1. Αν είναι 0, τότε A=0.

Η Java επιτρέπει μόνο Boolean εκφράσεις στην εντολή if, οπότε δεν επιτρέπεται η 2.

Δομές και Εντολές Ελέγχου Ροής Προγράμματος (1)

Ο έλεγχος ροής προγράμματος μπορεί να γίνει σε **τρία διαφορετικά επίπεδα**:

Α. Μέσα σε μια **έκφραση**, με βάση την ιεραρχία των τελεστών και τους κανόνες προτεραιότητας.

Β. Μεταξύ **εντολών**.

Γ. Μεταξύ των **τμημάτων** του προγράμματος

Το Α (το έχουμε εξετάσει) είναι το «χαμηλότερο» επίπεδο. Τώρα θα δούμε το Β.

Δομές και Εντολές Ελέγχου Ροής (2)

Σε ένα πρόγραμμα, εκτός από:

- Υπολογισμό εκφράσεων
- Ανάθεση τιμών

χρειάζονται:

1. Τρόποι επιλογής μεταξύ **εναλλακτικών** επιλογών
2. Τρόποι **επαναληπτικής** εκτέλεσης ομάδων εντολών

Για τα 1, 2 χρειαζόμαστε Εντολές Ελέγχου

ΔΟΜΗ ΕΛΕΓΧΟΥ:

Εντολή Ελέγχου **+** Ομάδα εντολών που ελέγχει

Δομές και Εντολές Ελέγχου Ροής (3)

Έχει αποδειχθεί θεωρητικά, ότι μια Γλώσσα Προγραμματισμού χρειάζεται **μόνο**:

- Είτε εντολή GOTO με επιλογή
- Είτε εντολή επιλογής 1 από 2, και λογικά ελεγχόμενη εντολή επανάληψης

A. ΣΥΝΘΕΤΕΣ ΕΝΤΟΛΕΣ

- ALGOL, Pascal: **begin** <εντολή_1>
  ….. <εντολή_κ> **end**
- C, C++, Java: **{** <εντολή_1>
  ….. <εντολή_κ> **}**

Δομές και Εντολές Ελέγχου Ροής (4)

Β. ΕΝΤΟΛΕΣ ΕΠΙΛΟΓΗΣ

Ι. Δύο Επιλογών

Σχεδιαστικά Θέματα:

- Μορφή και τύπος της έκφρασης που ελέγχει την επιλογή.
- Επιλογή απλής εντολής, ακολουθίας εντολών, ή σύνθετης εντολής;
- Τρόπος υλοποίησης φωλιασμένων (nested) επιλογών.

Αρχικά στη FORTRAN: IF **<Boolean expression> <εντολή>**

Για πολλαπλές εντολές:

```
IF  (FLAG .NE. 1)  GO TO 20
I = 1
J = 2
20  CONTINUE
```

(Αρνητική λογική, δύσκολη ανάγνωση)

Δομές και Εντολές Ελέγχου Ροής (5)

Στη συνέχεια, η ALGOL:

if <Boolean expression> then <εντολή_1>

else <εντολή_2>

- Οι <εντολή_1> και <εντολή_2> μπορούν να είναι και Σύνθετες Εντολές
- Όλες οι γλώσσες στη συνέχεια, ακολουθούν την ίδια λογική

Δομές και Εντολές Ελέγχου Ροής (6)

II. Πολλαπλών Επιλογών

- Μπορεί να κατασκευαστεί από 2 επιλογών, αλλά
- Βοηθάει αναγνωσιμότητα και ευκολία γραφής

Αρχικά FORTRAN: IF **<αριθμ_έκφραση>** L1, L2, L3

αν <0 =0 >0

Η ALGOL-W εισήγαγε την εντολή case

C, C++, Java:

```
switch (<έκφραση_int>) {
    case <const_expres_int_1> : <εντολή_1> ;
    …
    case <const_expres_int_n> : <εντολή_n> ;
    break;
    [default : <εντολή_n+1>]
}
```

Δομές και Εντολές Ελέγχου Ροής (7)

C:

```
switch (index)  {
    case 1:
    case 3: d+=1;
            s+=index;
            break;
    case 2 :
    case 4: e+=1;
            v+=index;
            break;
    default: printf(“Error”);
 }
```

Pascal:

```
case index of
    1, 3: begin
            d:=d+1;
            s:=s+index
          end;
    2, 4: begin
            e:=e+1;
            v:=v+index
          end
    else  writeln(‘Error’)
end
```

Δομές και Εντολές Ελέγχου Ροής (8)

C. ΕΝΤΟΛΕΣ ΕΠΑΝΑΛΗΨΗΣ

- Βασική λειτουργία των γλωσσών προγ/σμού.
- Κάνουν μια εντολή (ή ομάδα εντολών) να εκτελεστεί 0, 1 ή περισσότερες φορές.
- Οι Συναρτησιακές γλώσσες χρησιμοποιούν την αναδρομή για επανάληψη.
- Οι βασικές κατηγορίες εντολών επανάληψης, καθορίστηκαν από τις απαντήσεις των σχεδιαστών σε δύο βασικά ερωτήματα:

Δομές και Εντολές Ελέγχου Ροής (9)

- Πως ελέγχεται η επανάληψη;
  - Λογικά ελεγχόμενη (logically controlled)
  - Με μέτρηση (counter controlled)
- Που θα εμφανίζεται ο μηχανισμός ελέγχου στο βρόχο;
  - Στην αρχή (pre-test)
  - Στο τέλος (post-test)

Δομές και Εντολές Ελέγχου Ροής (10)

I. Επανάληψη με Μέτρηση

Μεταβλητή Βρόχου (ΜΒ) : Συντηρεί την τιμή μέτρησης.

Περιλαμβάνει επίσης:

- Αρχική Τιμή
- Τελική Τιμή
- Διαφορά Διαδοχικών Τιμών

} *Παράμετροι βρόχου* (ΠΒ)

Σχεδιαστικά Θέματα:

- Τύπος και εύρος τιμών της ΜΒ.
- Η τιμή της ΜΒ στο τέλος της επανάληψης.
- Μπορεί η ΜΒ και οι ΠΒ να αλλάξουν μέσα στο βρόχο, και πως επηρεάζεται ο έλεγχος;
- Οι ΠΒ υπολογίζονται 1 φορά, ή σε κάθε επανάληψη;

Δομές και Εντολές Ελέγχου Ροής (11)

- C: Γενική μορφή:

  for (<έκφραση_1> ; <έκφραση_2> ; <έκφραση_3>)

  <Σώμα_Βρόχου>

<έκφραση_1> : Ορίζεται η ΜΒ και η Αρχική τιμή της. Υπολογίζεται 1 φορά στην αρχή.

<έκφραση_2> : Έλεγχος βρόχου. Υπολογίζεται πριν από κάθε εκτέλεση του βρόχου. Σχεσιακή έκφραση (αν = 0, τότε τέλος).

<έκφραση_3> : Διαφορά διαδοχικών τιμών. Εκτελείται μετά από κάθε εκτέλεση. Αυξάνει τη Μεταβλητή Βρόχου.

Π.χ. for (index=0; index<=10; index++)

sum = sum + list[index];

Δομές και Εντολές Ελέγχου Ροής (12)

- Οι <έκφραση_2> και <έκφραση_3> μπορούν να είναι εντολές, ή πολλαπλές εντολές (χωρίζονται με **,**).
- Όλες οι <εκφράσεις> είναι προαιρετικές. Η default τιμή της <έκφραση_2> είναι TRUE (δηλαδή, 1). Των άλλων, τίποτα.
- Οι ΜΒ, ΠΒ μπορούν να αλλάξουν μέσα στο σώμα του βρόχου.
- Μπορεί να γίνει «είσοδος» στο σώμα του βρόχου.
- Ουσιαστικά είναι λογικά ελεγχόμενη επανάληψη.

C++ : Η <έκφραση_2> μπορεί να είναι και Boolean. Η <έκφραση_1> μπορεί να περιλαμβάνει ορισμούς μεταβλητών [π.χ. for (int count = 0 ; …)], με εμβέλεια το *σώμα του βρόχου* (σε παλιότερες υλοποιήσεις της C++, η εμβέλεια ήταν ως το τέλος της *περιβάλλουσας συνάρτησης*).

Java : Η <έκφραση_2> είναι μόνο Boolean. Η <έκφραση_1> μπορεί να περιλαμβάνει ορισμούς μεταβλητών, με εμβέλεια το *σώμα του βρόχου*.

Δομές και Εντολές Ελέγχου Ροής (13)

II. Λογικά Ελεγχόμενη Επανάληψη

- Είναι πιο γενικές εντολές επανάληψης από τις εντολές επανάληψης με μέτρηση.
- C, C++, Java: Έχουν και *pre-test* και *post-test* λογικά ελεγχόμενες επαναλήψεις:

Pre-test: while (i >= 0) { …
… }

Post-test: do { …
… } while (value > 0)

Διαφορά: Η post-test θα εκτελεστεί τουλάχιστον 1 φορά.

Java: Η έκφραση ελέγχου πρέπει να είναι Boolean.

Δεν έχει goto, οπότε δεν γίνεται «είσοδος» στο σώμα του βρόχου.

Δομές και Εντολές Ελέγχου Ροής (14)

III. Μηχανισμοί Ελέγχου του Χρήστη

Ορισμένες γλώσσες, όπως η Ada, έχουν βρόχους χωρίς έλεγχο της επανάληψης με μέτρηση ή λογικό:

```
loop
…
if sum >= 100 then exit;          (ή  exit when sum >= 100)
…
end loop
```

Υπό-Συνθήκη (conditional), ή Χωρίς Συνθήκη (unconditional) exit:

exit [<label>] [when <συνθήκη>]

Δομές και Εντολές Ελέγχου Ροής (15)

C, C++, Java: Unconditional unlabeled break

C, C++, Java: Παράλειψη εντολών που ακολουθούν: continue

Java: Unconditional labeled break και continue

Παράδειγμα:

```
while (sum < 1000) {
    getnext(value);
    if (value < 0)  continue;
                    (αν value < 0, στην αρχή του βρόχου)
                     break;
                    (αν value ≥ 0, έξοδος από το βρόχο)
    sum += value;
    (δεν εκτελείται ποτέ…)
}
```

Δομές και Εντολές Ελέγχου Ροής (16)

D. ΔΙΑΚΛΑΔΩΣΗ ΧΩΡΙΣ ΣΥΝΘΗΚΗ

(unconditional branching)

- Εντολή goto : Μεγάλη συζήτηση για τη χρησιμότητά της.
- Η Java δεν έχει…

EXAMPLES

- **The syntax of C in Backus-Naur Form**
- <translation-unit> ::= {<external-declaration>}*
- <external-declaration> ::= <function-definition> | <declaration>
- <function-definition> ::= {<declaration-specifier>}* <declarator> {<declaration>}* <compound-statement>
- <declaration-specifier> ::= <storage-class-specifier> | <type-specifier> | <type-qualifier>
- <storage-class-specifier> ::= auto | register | static | extern | typedef
- <type-specifier> ::= void | char | short | int | long | float | double | signed | unsigned | <struct-or-union-specifier> | <enum-specifier> | <typedef-name>
- <struct-or-union-specifier> ::= <struct-or-union> <identifier> { {<struct-declaration>}+ } | <struct-or-union> { {<struct-declaration>}+ } | <struct-or-union> <identifier>
- <struct-or-union> ::= struct | union
- <struct-declaration> ::= {<specifier-qualifier>}* <struct-declarator-list>
- <specifier-qualifier> ::= <type-specifier> | <type-qualifier>
- <struct-declarator-list> ::= <struct-declarator> | <struct-declarator-list> , <struct-declarator>
- …….e.t.c..