

# ΣΓΠ Σημειώσεις

by Μπαζ.

(Οι σελίδες με highlighted αριθμό μπορούν να σε βοηθήσουν με ασκήσεις, οι άλλες είναι θεωρία)

## Δομή Γλωσσών Προγραμματισμού

Η δομή των Γλωσσών Προγραμματισμού ορίζεται σε 3 επίπεδα:

- **Λέξεων**: πως οι χαρακτήρες σχηματίζουν *στοιχειώδεις λεξικές μονάδες* - *lexemes*
- **Σύνταξης**: πως συνδιάζονται σε πρόγραμμα τα διάφορα είδη φράσεων
- **Συμφραζόμενων**: πχ. ποιά ονόματα μεταβλητών είναι ενεργά σε ποιά σημεία, εγκυρότητα τύπων, κλπ... .

## Λεξική Δομή

Αφορά:

- Αναγνωρηστικά
- Λεξικές μονάδες

## Αλφάβητο Γλωσσών Προγραμματισμού

- Το σύνολο έγκυρων χαρακτήρων που μπορούν να χρησιμοποιηθούν σε μία γλωσσών προγραμματισμού
  - Για το μεταγλωττιστή κάθε χαρακτήρας αντιστοιχεί σε μία μοναδική τιμή (ASCII, UTF-16, κλπ..)
  - Κάποιες γλώσσες διακρίνουν πεζά-κεφαλαία

## Δεσμευμένες Λέξεις - Λέξεις Κλειδιά

- **Δεσμευμένες Λέξεις**: κανένα όνομα στο πρόγραμμα δεν μπορεί να είναι ίδιο με αυτές
- **Λέξεις Κλειδιά**: Λέξεις με ειδική σημασία στη γλώσσα (όχι απαραίτητα δεσμευμένες)
- Συχνά ταυτίζονται ή οι λέξεις κλειδιά είναι *υποσύνολο* των δεσμευμένων
- Μπορεί επίσης να υπάρχουν ονόματα που χρησιμοποιούνται από άλλα κομμάτια του προγράμματος (πχ. βιβλιοθήκες), αλλά η χρήση τους περιορίζεται από το μεταφραστή και όχι από την ίδια τη γλώσσα, οπότε δεν ταυτίζονται με τις παραπάνω κατηγορίες

## Ονόματα

- **Ονόματα**: χρησιμοποιούνται για την *ταυτοποίηση* μεταβλητών, τύπων, ετικετών, διαδικασιών, πακετών και άλλων οντοτήτων
  - Συνήθως επιβάλλονται περιορισμοί στη μορφή τους (πχ. να μην ξεκινούν από αριθμό)
- Είναι αναγνωρηστικά *μεταβλητού μήκους*
  - Ο διαχωρισμός τους ακολουθεί την **ταύτηση για τη λεξική μονάδα μεγαλύτερου δυνατού μήκους**
    - δλδ. ως όνομα αναγνωρίζεται πάντα η μεγαλύτερη δυνατή συμβολοσειρά, χωρίς κενά, με ένα μόνο αναγνωριστικό
    - πχ. η λέξη doif δεν ταυτίζεται με τις λέξεις κλειδιά do και if, αλλά ως ένα όνομα

## Τελεστές

- **Τελεστές:** λεξικές μονάδες που συμπεριφέρονται γενικά σαν συναρτήσεις, αλλά διαφέρουν συντακτικά και σημασιολογικά
  - αριθμητικοί
  - σύγκρισης
  - λογικοί
  - ανάθροσης
  - επιλογής πεδίου σε record (.)
  - ανάλυσης εμβέλειας (::)
- Η κάθε γλώσσα ορίζει σύνολο ενσωματωμένων τελεστών, και κάποιες φορές δίνει τη δυνατότητα στους προγραμματιστές
  - να τους υπερφορτώσουν
  - να ορίσουν καινούργιους

## Διαχωριστές και Κενά

- **Διαχωριστές ή κενά:** χαρακτήρες που όταν διαβάζονται, επιβάλλουν την απόσπαση αναγνωρηστικών από το υπόλοιπο κείμενο
- Στις περισσότερες γλώσσες, οι λεξικές μονάδες διαχωρίζονται με ένα πλό κενό
- Όλες οι γλώσσες διαθέτουν χαρακτήρες με σημασία
  - διαχωρισμού εντολών
  - τερματισμού εντολών
  - συνέχισης σε νέα γραμμή

## Κυριολεκτικές Σταθερές

- **Κυριολεκτικές σταθερές:** αναπαριστούν σταθερές τιμές, σε αντίθεση όμως με τις (ορισμένες) σταθερές, μπορούν να έχουν μόνο μία τιμή
  - Συνήθως χρησιμοποιούνται στην αρχικοποίηση μεταβλητών
  - πχ 14, 0.5, "hello", κλπ...

## Σχόλια

Διακρίνονται σε:

- *inline* και *block*
- Αυτά που αγνωρύνται τελείως και αυτά που έχουν κάποια σημασία
- Υποστηριζουν ένθεση ή όχι

# Συντακτικό Γλωσσών Προγραμματισμού

Τα βασικά συντακτικά μέρη των προγραμμάτων είναι:

- Εκφράσεις:** υπολογισμοί που καθορίζουν τις τιμές τους. Αν μία έκφραση αλλάζει τη συμπεριφορά του προγράμματος, λέμε ότι συνοδεύεται από παρενέργειες
- Εντολές:** οδηγίες που αλλάζουν ρητά τη συμπεριφορά του προγράμματος
  - πχ.  $x = x + 3$ , αλλάζει την τιμή της μεταβλητής  $x$
- Δηλώσεις:** εισάγουν νέα ονόματα και ορίζουν τις ιδιότητές τους

Για τον ορισμό των συντακτικών δομών των γλωσσών προγραμματισμού **χρησιμοποιούνται Γραμματικές Χωρίς Συμφραζόμενα** (δες παρακάτω)

## Σύνταξη Προστακτικών και Δηλωτικών Γλωσσών

- Στις **προστακτικές** γλώσσες προγραμματισμού τα προγράμματα αποτελούνται από ακολουθίες εντολών που:
  - εκτελούν υπολογισμούς
  - αναθέτουν τιμές σε μεταβλητές
  - δέχονται είσοδο ή παράγουν έξοδο
  - εκτρέπουν το έλεγχο σε κάποιο άλλο σημείο της ακολουθίας
- Τα βασικά στοιχεία στις προστακτικές γλώσσες είναι
  - εντολές ανάθεσης
  - επαναληπτικοί βρόγχοι
  - ακολουθίες (blocks) εντολών
  - διαδικασίες
  - υποθετικές εντολές
  - εντολές διαχείρησης εξαιρέσεων
- Στις **συναρτησιακές** γλώσσες προγραμματισμού, η εκτέλεση γίνεται συνήθως με υπολογισμό τιμής εκφράσεων που περιέχουν συναρτήσεις
- Υπάρχουν και γλώσσες με συνδυασμό προστακτικών και δηλωτικών δομών

	Προστακτικές γλώσσες	Δηλωτικές γλώσσες
Ανάθεση τιμών	Διαθέτουν εντολές ανάθεσης, σε ονόματα μεταβλητών που συνδέονται με τροποποιήσιμες τιμές.	Στις μεταβλητές ανατίθεται μόνο μία τιμή, που δεν αλλάζει κατά την εκτέλεση του προγράμματος. Για τις τιμές που προκύπτουν στους υπολογισμούς δημιουργούνται νέα ονόματα με ένθεση ή αναδρομή.
Δομές δεδομένων	Χρησιμοποιούνται δομές δεδομένων όπως πίνακες, records κ.α., που τα στοιχεία τους αλλάζουν τιμές με εντολές ανάθεσης.	Χρησιμοποιούνται ρητές αναπαραστάσεις δομών δεδομένων, όπως οι λίστες, που οι τιμές των στοιχείων δεν μπορούν να αλλάξουν.
Σειρά εκτέλεσης	Οι τιμές περνάνε μέσω κοινών μεταβλητών από μία έκφραση σε μία άλλη. Οι εκφράσεις μπορεί να έχουν παρενέργειες. Η σειρά εκτέλεσης είναι κρίσιμη.	Η σειρά με την οποία καλούνται οι ορισμοί ή οι εξισώσεις δεν επηρεάζει καθώς οι τιμές των μεταβλητών δεν αλλάζουν. Οι δηλωτικοί ορισμοί δεν έχουν παρενέργειες.
Εκφράσεις ως τιμές	Συνήθως δε γίνεται να περνάμε διαδικασίες ως παραμέτρους σε άλλες διαδικασίες ή να επιστρέφονται τα αποτελέσματα αυτών.	Επιτρέπεται ο χειρισμός των ορισμών και των εκφράσεων ως δεδομένα παραμέτρων.
Ροή ελέγχου	Η ροή ελέγχου στο πρόγραμμα είναι ευθύνη του προγραμματιστή.	Η ροή ελέγχου στο πρόγραμμα είναι ευθύνη της γλώσσας, και όχι ευθύνη του προγραμματιστή.

# Σημασία Γλωσσών Προγραμματισμού

- Αναφέρεται σε πληροφορίες σχετικές με τη συμπεριφορά και τα αποτελέσματα εκτέλεσης ενός προγράμματος
  - Σε αντίθεση με τη σύνταξη που αναφέρεται στην έγκυρη δομή προγραμμάτων
- Ένα πρόγραμμα μπορεί να είναι **συντακτικά ορθό** αλλά **σημασιολογικά λάθος** (εμφανίζει λάθη σε σχέση με τη σημασία της γλώσσας)
- Ένα πρόγραμμα μπορεί να είναι **σημασιολογικά ορθό**, αλλά **λογικά λάθος** (δεν ταιριάζει με τους στόχους του προγραμματιστή)
- Ο ορισμός της σημασίας είναι πιο δύσκολος από τον ορισμό της σύνταξης (μπορεί να περιέχει ασάφιες)
  - **Εγχειρίδιο Αναφοράς της Γλώσσας Προγραμματισμού**: Φυσική γλώσσα, ασαφές, ανακριβές, ελλειπές
  - **Προδιαγραφή και Ανάπτυξη Μεταφραστή**: Απαντά όλα τα ζητήματα σε σχέση με τη σημασία, αλλά επηρεάζεται από λάθη και εξαρτήσεις μηχανής εκτέλεσης
  - **Τυπικός Ορισμός**: μαθηματικές μέθοδοι

## Ιδιότητες Και Συνδέσεις

- **Ιδιότητα**: κανόνας σημασίας ονομάτων
  - μεταβλητών
  - σταθερών
  - διαδικασιών
- **πχ.**
  - **θέση**: περιοχή μνήμης (αφαιρετική θεώρηση διευθυνσιοδότησης, ξεκινάμε από το 0)
  - **τιμή**: η αποθηκευμένη ποσότητα
- **Σύνδεση (binding)**: Η συσχέτιση μίας ιδιότητας με ένα όνομα
- **Δηλώσεις**: συντακτικά στοιχεία που συνδέουν ονόματα σε ιδιότητες

### Παράδειγμα

#### Παράδειγμα 1 από τη γλώσσα C

Η δήλωση **const int n=7;**  
αποδίδει στο όνομα **n** σημασία ακέραιης σταθεράς με τιμή 7, δηλ.  
στο όνομα **n** συνδέονται

- η ιδιότητα τύπου δεδομένων "integer constant"
- η ιδιότητα τιμής 7

#### Παράδειγμα 2 από τη γλώσσα C

Η δήλωση **double f(int n){**  
                                  **...**  
                                  **}**  
συνδέει στο όνομα **f**

- την ιδιότητα "function"
- τις ιδιότητες αριθμός, ονόματα και τύποι παραμέτρων (μία παράμετρος με όνομα **n** και τύπο "integer")
- την ιδιότητα τύπου της επιστρεφόμενης τιμής ("double")
- την ιδιότητα που αναφέρεται στο σώμα του κώδικα που εκτελείται όταν καλείται η **f**

- Εντολές: μπορούν επίσης να συνδέουν νέες ιδιότητες ή να αλλάξουν υπάρχουσες

### Παράδειγμα

#### Παράδειγμα 3 από τη γλώσσα C

Η ανάθεση τιμής                             $x=2;$   
 συνδέει στο όνομα μεταβλητής  $x$   
 ➤ τη νέα ιδιότητα "τιμή 2"

#### Παράδειγμα 4 από τη γλώσσα C

Αν η  $y$  είναι μεταβλητή δείκτη που έχει δηλωθεί ως        **int\* y;**  
 τότε η εντολή                                 $y = \text{new int};$   
 συνδέει  
 ➤ μία ιδιότητα θέσης  $y$  (δεσμεύει μνήμη για μία ακέραιη μεταβλητή)  
 ➤ μία νέα ιδιότητα τιμής (εκχωρεί αυτή τη θέση στην  $*y$ )

- **Συνδέσεις σε διαφορετικές γλώσσες**
  - Στις καθαρά συναρτησιακές η τιμή συνδέεται σε όνομα μόνο μία φορά και δεν χρειάζεται διαφορετική σύνδεση θέσης, αφού η τιμή δεν αλλάζει
  - Σε πιο δυναμικές γλώσσες, το όνομα μπορεί αν συνδέεται κατά την εκτέλεση με σειρά από διαφορετικές τιμές
- **Χρόνος Σύνδεσης (binding time)**: η χρονική στιγμή υπολογισμού και σύνδεσης ιδιότητας σε όνομα
  - Ανάλογα με το χρόνο σύνδεσης διακρίνουμε
    - **Στατικές Συνδέσεις**: πριν την εκτέλεση (*στατικές ιδιότητες*)
    - **Δυναμικές Συνδέσεις**: κατά την εκτέλεση (*δυναμικές ιδιότητες*)
  - Διαφορετικές γλώσσες είναι πιο κατάλληλες για την κάθε κατηγορία
    - Οι συναρτησιακές στηρίζουν περισσότερο τη δυναμική σύνδεση από τις προστακτικές
    - Οι μεταγλωττιστές στηρίζουν περισσότερο στατικές από τους διερμηνευτές

### Στατικές Συνδέσεις

- Συμβαίνουν κατά την:
  - **Συντακτική ή σημασιολογική ανάλυση** (Χρόνος Μετάφρασης)
  - **Διασύνδεση προγράμματος με βιβλιοθήκες** (Χρόνος Διασύνδεσης)
  - **Φόρτωση προγράμματος στη μνήμη για εκτέλεση** (Χρόνος Φόρτωσης)
  - Κάποιες φορές και πριν από τη μετάφραση, κατά τον ορισμό της γλώσσας
- Ο μεταφραστής δημιουργεί μία **δομή δεδομένων** στην οποία διατηρεί τις συνδέσεις, τον **πίνακα συμβόλων**
- Μπορεί να αναπαρασταθεί ως μία συνάρτηση που εκφράζει τη σύνδεση *ιδιοτήτων σε ονόματα*

*SymbolTable : Names → Attributes*

- Ο ορισμός της συνάρτησης αλλάζει καθώς εξελίσσεται η μετάφραση, ανάλογα με τη δημιουργία και τη διαγραφή συνδέσεων
- Στη στατική σημασιολογική ανάλυση καθορίζονται οι ιδιότητες ονομάτων που εμφανίζονται σε δηλώσεις και επιβεβαιώνεται πως η **χρήση τους συμμορφώνεται** με τις δηλωμένες ιδιότητες

## Δυναμικές Συνδέσεις

- Συμβαίνουν κατά την:
  - *Είσοδο/Έξοδο* από συναρτήσεις
  - *Είσοδο/Έξοδο* από το πρόγραμμα
- Κατά την εκτέλεση παράγεται κώδικας (από το μεταγλωττιστή) που διατηρεί πληροφορίες για τις ιδιότητες ονομάτων, όπως θέσεις και τιμές, κατά την εκτέλεση
- Το μέρος που συνδέει τις ιδιότητες θέσης με ονόματα, ονομάζεται **περιβάλλον**

*Environment : Names → Locations*

- Το μέρος που συνδέει τις ιδιότητες τιμής με θέσεις, ονομάζεται **μνήμη**

*Memory : Locations → Values*

## Ρητές και Έμμεσες Συνδέσεις

- Σε μία δήλωση οι συνδέσεις μπορεί να καθορίζονται
  - *ρητά*: από τον προγραμματιστή
  - *έμμεσα*: από τους κανόνες της γλώσσας
- Η ίδια σύνδεση μπορεί να γίνει και ρητά και έμμεσα ανάλογα με το πηγαίο πρόγραμμα
- Σε κάποιες γλώσσες μία απλή αναφορά σε όνομα, προκαλεί τη δήλωσή της (συνήθως σε μία εντολή ανάθεσης)
- Στις C/C++ έχουμε *ορισμούς*, δηλώσεις που συνδέουν όλες τις σχετικές ιδιότητες ενός ονόματος
- Σε γλώσσες με *δυναμικούς τύπους*, τα ονόματα έχουν μόνο *σύνδεση τιμής*, ενώ οι ιδιότητες τύπων συνδέονται με τις ίδιες τις τιμές

## Blocks

- *Συντακτικές δομές* που αποτελούνται από μία αλληλουχία δηλώσεων, ακολουθούμενη από μία αλληλουχία εντολών
  - Η αλληλουχία εντολών περικλείεται συνήθως από *στοιχεία σύνταξης* όπως '{' και '}' ή 'begin' και 'end'
- Στη C τα *blocks* ονομάζονται και *σύνθετες εντολές*, και εμφανίζονται στο σ' σμα συναρτήσεων, σε ορισμούς συναρτήσεων και σε θέση οποιασδήποτε εντολής
- Στη C η θέση μίας δήλωσης σε σχέση με ένα block, την κατηγοριοποιεί ως
  - *Τοπική* όταν σχετίζεται με ένα συγκεκριμένο block
  - *Μη Τοπική* (εξωτερική ή καθολική) όταν συμβαίνει εκτός του block

### Παράδειγμα

Παράδειγμα 6 από τη γλώσσα C

```
void p() {
    double r, z;
    ...
    {int x, y;
     x=2;
     y=0;
     x+=1;
    }
    ...
}
```

δηλώσεις τοπικές  
στην p και μη  
τοπικές στο  
εσωτερικό block,  
που είναι ένθετο  
στην p

- Μπορούμε επίσης να έχουμε ένθεση ενός block σε ένα άλλο, σε οποιοδήποτε βάθος
  - Τα ίδια ονόματα μπορούν αν δηλώνονται ξανά σε ένθετα blocks
  - Κάθε νέο όνομα που δηλώνεται το συσχετίζουμε με μία λεξική διεύθυνση, δλδ ένα ζεύγος αριθμών που αποτελείται από
    - τον **αριθμό επιπέδου**: ξεκινάει από το 0 στο εξωτερικό επίπεδο και αυξάνεται κατεβαίνοντας επίπεδα
    - τον **αριθμό μετατόπισης**: ξεκινάει από το 0 για κάθε block και αυξάνεται για κάθε όνομα

#### Παράδειγμα

**Παράδειγμα 7 από τη γλώσσα Ada**

```

declare    (x: integer;
           y: boolean;
begin      x:=2;           λεξ. δ/νση (0,0)
           y:=true;        λεξ. δ/νση (0,1)
           x:=x+1;
declare    (x: integer;
           y: boolean;
begin      x:=3;           λεξ. δ/νση (1,0)
           y:=false;       λεξ. δ/νση (1,1)
           x:=x*6;
end;
end;

```

18

## Structs και Κλάσεις

- Στη C ένας **ορισμός struct** αποτελείται από δηλώσεις τοπικών μεταβλητών
  - Η σύνταξη μοιάζει με block (άγκιστρα), αλλά στο τέλος έχει ερωτηματικό (σύνθετη δήλωση ?)

#### Παράδειγμα

**Παράδειγμα 8 από τη γλώσσα C**

```

struct A{
        int x;
        double y;
        struct{
                int* x;
                char y;
        } z;
}

```

- Στις αντικειμενοστρεφείς γλώσσες μπορούμε και να δηλώσουμε **κλάσεις**
  - Είναι η μόνη δήλωση που δεν χρειάζεται να γίνει μέσα σε άλλη κλάση

#### Παράδειγμα

**Παράδειγμα 9 από τη γλώσσα Java**

```

Public class IntWithGcd{
    ...
    public int intValue() {           }
                                return value;
}
private int value;

```

## Εμβέλεια Συνδέσεων

- **Εμβέλεια Σύνδεσης (scope of binding)**: το μέρος του προγράμματος μέσα στο οποίο διατηρείται μία σύνδεση
  - **Εμβέλεια δήλωσης**: όρος που χρησιμοποιείται αν όλες οι συνδέσεις μίας δήλωσης έχουν την ίδια εμβέλεια
  - **Εμβέλεια ονόματος**: όρος που χρησιμοποιείται καταχρηστικά, καθώς το ίδιο όνομα μπορεί να εμφανίζεται σε διαφορετικές δηλώσεις με διαφορετική εμβέλεια για την κάθε μία
- **Κανόνας λεξικής εμβέλειας**:
  - Σε μία γλώσσα που υποστηρίζει ένθετα blocks, η εμβέλεια μίας σύνδεσης περιορίζεται
    - στο block της σχετικής δήλωσης
    - σε κάθε block που περιέχεται σε αυτό
- **Κανόνας δήλωσης πριν τη χρήση**:
  - Η εμβέλεια μίας δήλωσης ξεκινά ακριβώς μετά το σημείο της δήλωσης, και τελειώνει στο τέλος του block στο οποίο βρίσκεται

### Παράδειγμα

#### Παράδειγμα 11 από τη γλώσσα C

```

1 int x;
2 void p() {
3     char y;
4     ...
5 } /* p */
6 void q() {
7     double z;
8     ...
9 } /* q */
10 main() {
11     int w[10];
12     ...
13 }
```

- Οι δηλώσεις σε ένθετα block έχουν προτεραιότητα έναντι των προηγούμενων δηλώσεων

### Παράδειγμα

#### Παράδειγμα 12 από τη γλώσσα C

```

int x;

void p() {
    char x;
    x='a'; → ανάθεση τιμής
    ...           στην τοπική x
}

main() {
    x=2; → ανάθεση τιμής
    ...           στην καθολική x
}
```

- Στο πράδειγμα λέμε ότι η καθολική δήλωση της x έχει μία τρύπα στην εμβέλεια μέσα στην p, και ότι η τοπική δήλωση της x βρίσκεται στη σκιά της καθολικής δήλωσης
  - Η εμβέλεια μπορεί να περιλαμβάνει και τρύπες εμβέλειας: εκεί που οι συνδέσεις εξακολουθούν να υπάρχουν αλλά δεν είναι ορατές
  - η ορατότητα μίας δήλωσης περιλαμβάνει μόνο τα μέρη του προγράμματος όπου η συνδέσης της δήλωσης εφαρμόζονται
- στη C++, για να προσπελαύνονται κρυμμένες δηλώσεις, διαθέτιε τον τελεστή ανάλυσης εμβέλειας ":"

## Εμβέλεια και C

### Ιδιότητα Διασύνδεσης και Κλάσεις Αποθήκευσης

- Ένα πρόγραμμα C μπορεί να αποτελείται από πολλά πηγαία αρχεία
  - Ο προεπεξεργαστής ενσωματώνει τα αρχεία που αναφέρονται στα #include, σχηματίζοντας διαφορετικές μονάδες μεταγλώπισης
- Η **ιδιότητα διασύνδεσης** εξαναγκάζει ένα όνομα, που δηλώνεται σε περισσότερες από μία εμβέλειες, ή περισσότερες από μία φορά στην ίδια εμβέλεια, να αναφέρεται στην ίδια οντότητα (μεταβλητή, συνάρτηση, struct)

Αναφέρεται από ονόματα που δηλώνονται . . . .	στην ίδια εμβέλεια	σε άλλες εμβέλειες στην ίδια μον. μεταγλώττισης	σε άλλες μονάδες μεταγλώττισης
χωρίς διασύνδεση (no linkage)	✓	✗	✗
εσωτερική διασύνδεση (internal linkage)	✓	✓	✗
εξωτερική διασύνδεση (external linkage)	✓	✓	✓

- Η εμβέλεια δήλωσης ενός ονόματος επηρεάζεται και από την **κλάση αποθήκευσης** με την οποία δηλώνεται:
  - extern* - προκαθορισμένη κλάση για ονόματα που δε βρίσκονται μέσα σε συνάρτηση και για δηλώσεις συναρτήσεων
  - auto* - προκαθορισμένη κλάση για δηλώσεις εντός συνάρτησης
  - static* - δηλώνει ότι η μνήμη εκχωρείται άπαξ (μόνο μία φορά)
  - register* - δηλώνει ότι μπορεί να αποθηκευτεί σε καταχωριτή αντί για μνήμη
- Αν μία δήλωση είναι μέσα σε συνάρτηση και δεν περιέχει τις λέξεις κλειδιά static ή extern, τότε η μνήμη εκχωρείται και απελευθερώνεται με προσέγγιση λειτουργίας στοιβας

### Κανόνες Εμβέλειας

- Ένα όνομα έχει **τοπική εμβέλεια**, αν αυτό δηλώνεται σε συνάρτηση ή block, άρα μπορεί να χρησιμοποιηθεί μόνο στο block/συνάρτηση που το περικλείει
- Τα ονόματα **παραμέτρων συνάρτησης** έχουν εμβέλεια σε όλο το block με τον κώδικα της συνάρτησης
- Αν ένα όνομα σε **ένθετο block** είναι ίδιο με όνομα σε block που το περικλείει, τότε η ένθετη δήλωση, **αποκρύπτει** την εμβέλεια της δήλωσης του περικλείοντος block
- Τα ονόματα **ετικετών (labels)** είναι το μόνα που έχουν εμβέλεια συνάρτησης, δλδ η ετικέτα μπορεί να αναφερθεί σε εντολή goto που μπορεί να βρίσκεται ακόμα και πριν από τη δήλωσή της
- Ένα όνομα έχει **εμβέλεια αρχείου (καθολική)** αν δε δηλώνεται μέσα σε block
- Για δηλώσεις ονομάτων με **εμβέλεια αρχείου**, η μνήμη εκχωρείται **άπαξ** και κλάση αποθήκευσης
  - την **προκαθορισμένη** ή *auto*,
    - για ονόματα συνάρτησης (δεν μπορούν να είναι auto ή register), η δήλωση έχει το ίδιο επίπεδο διασύνδεσης με οποιαδήποτε άλλη ορατή δήλωση, με εμβέλεια αρχείου για το ίδιο όνομα, αν υπάρχει, αλλιώς **εξωτερική διασύνδεση**
    - για ονόματα **μεταβλητών**, *struct*, *κα.* έχουμε **εξωτερική διασύνδεση**
  - extern*, η δήλωση έχει το ίδιο επίπεδο διασύνδεσης με οποιαδήποτε άλλη ορατή δήλωση, με εμβέλεια αρχείου αν υπάρχει, αλλιώς **εξωτερική διασύνδεση**
  - static*, η δήλωση έχει **εξωτερική διασύνδεση**
- Για δηλώσεις ονομάτων με **τοπική εμβέλεια** και κλάση αποθήκευσης
  - την προκαθορισμένη *auto* ή *register* ή *static*, τα ονόματα είναι **χωρίς διασύνδεση**
  - extern* ισχύουν οι ίδιοι κανόνες για δήλωση με εμβέλεια αρχείου, και κλάση αποθήκευσης *extern*

Οι κανόνες 5. και 6. περιληπτικά:

- Η **μνήμη εκχωρείται άπαξ**, για όλες τις δηλώσεις με εμβέλεια αρχείου και τις δηλώσεις *extern* με τοπική εμβέλεια
- **Χωρίς διασύνδεση**, οι δηλώσεις τοπικής εμβέλειας, *auto*, *register*, *static*
- Με **εξωτερική διασύνδεση** όλες οι άλλες, **εκτός από**
  - Δηλώσεις *συναρτήσεων*, *auto*
  - Δηλώσεις *extern*
  - για τις οποίες υπάρχει ορατή δήλωση με εμβέλεια αρχείου για το ίδιο όνομα -> έχουν **ίδια διασύνδεση** με τη άλλη δήλωση

## Πίνακας Συμβόλων

- Υποστηρίζει την **εισαγωγή**, **αναζήτηση** και **διαγραφή** ονομάτων, μαζί με τις **ιδιότητές** τους και τις συνδέσεις που υλοποιούν οι δηλώσεις τους
- **Επεξεργασία στοίβας** των πληροφοριών εμβέλειας, σε γλώσσα με δομή *block*
  - κατά την είσοδο σε *block*, όλες οι δηλώσεις του *block* υποβάλλονται σε επεξεργασία και οι αντίστοιχες συνδέσεις εισάγονται στον πίνακα συμβόλων
  - κατά την έξοδο από *block*, οι συνδέσεις που δημιουργήθηκαν από τις δηλώσεις αφαιρούνται, αποκαθιστώντας τυχόν προυπάρχουσες συνδέσεις
- Αν ο πίνακας συμβόλων,
  - δημιουργείται κατά τη μεταγλώττιση, παρέχει πληροφορίες για τη στατική εμβέλεια των ονομάτων
  - διαχειρίζεται πληροφορίες κατά την εκτέλεση του προγράμματος, εκφράζει τη δυναμική εμβέλεια των ονομάτων

## Παραδείγματα Πινάκων Συμβόλων και Εμβέλειας Ονομάτων

### Στατική Εμβέλεια

#### Παράδειγμα

##### Παράδειγμα 13 από τη γλώσσα C

```

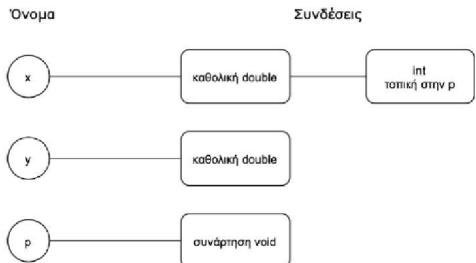
1 double x;
2 double y;

4 void p(){
5     int x;
6     ...
7     { int y[100];
8     ...
9     }
10    ...
11 }

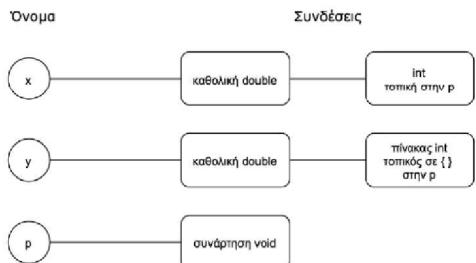
13 void q(){
14     char y;
15     ...
16 }

18 main(){
19     int x;
20     ...
21 }
```

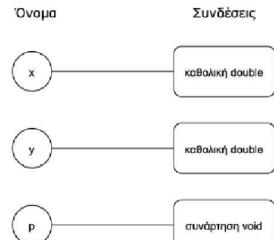
- Πίνακας συμβόλων στη γραμμή 6



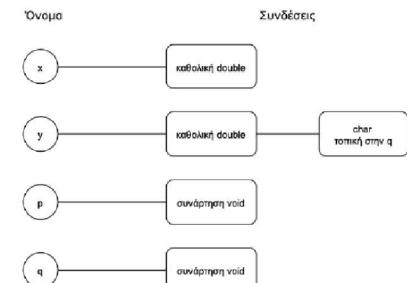
- Πίνακας συμβόλων στη γραμμή 8



- Πίνακας συμβόλων στη γραμμή 11

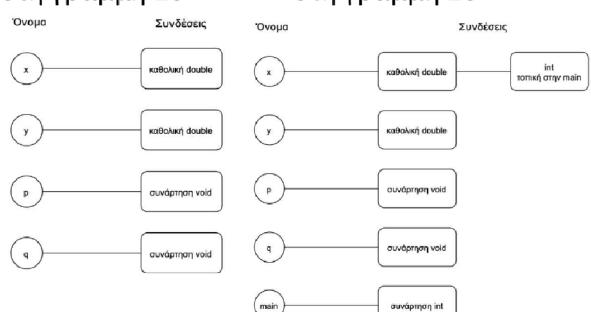


- Πίνακας συμβόλων στη γραμμή 15



- Πίνακας συμβόλων στη γραμμή 16

- Πίνακας συμβόλων στη γραμμή 20



## Δυναμική Εμβέλεια

### Παράδειγμα

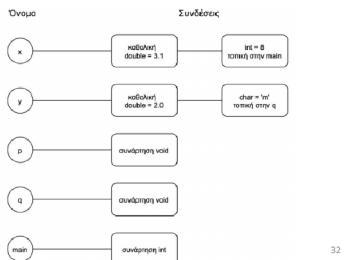
#### Παράδειγμα 14 από τη γλώσσα C

```

1 #include <stdio.h>
2
3 double x = 3.1;
4 double y = 2.0;
5
6 void p(){
7     int x = 1;
8     printf("%lf\n",y);
9     { int y[100];
10 }
11 }
12
13 void q(){
14     char y = 'm';
15     printf("%lf\n",x);
16     p();
17 }
18
19 int main(){
20     int x = 8;
21     q();
22     return 0;
23 }
```

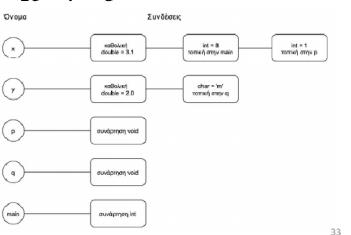
- Πίνακας συμβόλων στη γραμμή 15

διαδρομή εντολών μέχρι τη γραμμή 15:  
 $3 \rightarrow 4 \rightarrow 6 \rightarrow 13 \rightarrow 19 \rightarrow 20 \rightarrow 21 \rightarrow 14 \rightarrow 15$



- Πίνακας συμβόλων στη γραμμή 8

διαδρομή εντολών μέχρι τη γραμμή 8:  
 $3 \rightarrow 4 \rightarrow 6 \rightarrow 13 \rightarrow 19 \rightarrow 20 \rightarrow 21 \rightarrow 14 \rightarrow 15 \rightarrow 16 \rightarrow 7 \rightarrow 8$



## Σύνθετες Δομές Δεδομένων

### Παράδειγμα

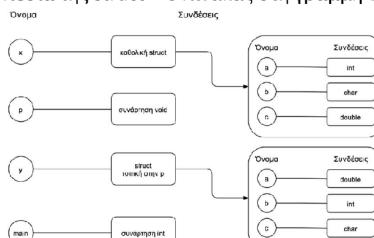
- Όταν υποστηρίζεται ο ορισμός νέων τύπων ή τύπου struct, ο πίνακας συμβόλων πρέπει να διατηρεί πληροφορίες για κάθε μεταβλητή, που εμπεριέχεται στη δομή struct.

#### Παράδειγμα 15 από τη γλώσσα C

```

1 struct{
2     int a;
3     char b;
4     double c;
5 } x = {1,'a',2.5};
6
7 void p(){
8     struct{
9         double a;
10        int b;
11        char c;
12    } y = {1,2,'b'};
13    printf("%d,%c,%g\n",x.a,x.b,x.c);
14    printf("%d,%c,%g\n",y.a,y.b,y.c);
15 }
16
17 main(){
18     p();
19     return 0;
20 }
```

- Δημιουργία υποπίνακα συμβόλων για τα πεδία της struct – ο πίνακας στη γραμμή 16



## Ανάλυση Ονομάτων

- **Ανάλυση ονομάτων:** η διαδικασία που συσχετίζει κάθε αναφορά ονόματος με τη σωστή δήλωση, σύμφωνα με τη σημασία της γλώσσας
- Χρησιμοποιείται σε πολλές φάσεις της μεταγλώττισης ένας μηχανισμός επεξεργασίας δηλώσεων μέσω του πίνακα συμβόλων
  - Μπορεί ένα όνομα να αναφέρεται σε διάφορες οντότητες του προγράμματος, ανάλογα με το πως χρησιμοποιείται

## Υπερφόρτωση

- Σε πολλές γλώσσες προγραμματισμού επιτρέπεται η **υπερφόρτωση** τελεστών και ονομάτων συναρτήσεων
- Η επαναχρησιμοποίηση του ίδιου συμβόλου δεν προκαλέι σύγχυση, επειδή οι διαφορετικές πράξεις σχετίζονται από μαθηματική άποψη μεταξύ τους
  - Η αποσαφήνιση από τον μεταφραστή της χρήσης ενός συμβόλου, γίνεται εξετάζοντας τους τύπους δεδομένων των τελεστών
- Οι C++, Java, Ada επιτρέπουν εκτεταμένη υπερφόρτωση ονομάτων και τελεστών
- Η Haskell επιτρέπει την υπερφόρτωση των συναρτήσεων, των τελεστών, καθώς και τον ορισμό νέων τελεστών

## Υπερφόρτωση Συναρτήσεων

- Υποστηρίζεται από μία λειτουργία **αναζήτησης** στον πίνακα συμβόλων και όχι μόνο με βάση το όνομα μίας συνάρτησης, αλλά λαμβάνοντας υπόψη και τον *αριθμό* και *τύπο* των παραμέτρων
- Αυτή η διαδικασία λέγεται **ανάλυση υπερφόρτωσης**
- Στην Ada στηρίζεται επίσης και ανάλυση με βάση
  - τον τύπο της επιστρεφόμενης τιμής
  - **ονόματα παραμέτρων στον ορισμό** της συνάρτησης
- Σε καποιες περιπτώσεις η υπερφόρτωση συνδιάζεται με τους κανόνες **αυτόματης μετατρωπής** τύπων

### Παράδειγμα

#### Παράδειγμα 1 από τη γλώσσα C++

```
int max(int x, int y){           // max #1
    return x>y?x:y;
}

double max(double x, double y){ // max #2
    return x>y?x:y;
}

int max(int x, int y, int z){ // max #3
    return x>y?(x>z?x:z):(y>z?y:z);
}
```

Στις παρακάτω κλήσεις της max, ο πίνακας συμβόλων μπορεί να αποφασίσει ποια είναι η κατάλληλη συνάρτηση βάσει των παραμέτρων:

```
max(2,3);      // max #1
max(2.1,3.2); // max #2
max(1,3,2);   // max #3
```

## Ανάλυση υπερφόρτωσης

Τι θα γίνει όμως στην παρακάτω κλήση; `max(2.1, 3);`

Εξαρτάται από τους κανόνες της γλώσσας που καθορίζουν τις έγκυρες «αυτόματες» μετατροπές τύπων. Π.χ. στην C++ η κλήση αυτή είναι διφορούμενη καθώς η γλώσσα επιτρέπει μετατροπές από ακέραιο σε `double` και το ανάποδο (αυτόματη περικοπή), αλλά δεν αναφέρεται ποια μετατροπή έχει προτεραιότητα. Μία λύση είναι η περαιτέρω υπερφόρτωση των συναρτήσεων `max`. Κάποιες γλώσσες (π.χ. Ada) δεν υποστηρίζουν αυτόματες μετατροπές τύπων.

```
double max(int x, double y){ // max #4
    return x > y ? (double) x : y;
}

double max(double x, int y){ // max #5
    return x > y ? x : (double) y;
}
```

6

## Υπερφόρτωση Τελεστών

- Κατά την υπερφόρτωση τελεστών, δεν επηρεάζεται ότι σχετίζεται με την σύνταξή τους (προτεραιότητα και προσεταιριστικότητα)
- Οι τελεστές δε διαφέρουν σημασιολογικά από τις συναρτήσεις, αλλά συντακτικά
  - Οι τελεστές συνήθως εκφράζονται με ένθετη μορφή  
πχ. `2+3`
  - Οι συναρτήσεις έχουν προθεματική σύνταξη  
πχ. `add(2,3)`

### Παράδειγμα

#### Παράδειγμα 2 από τη γλώσσα C++

```
1 #include <iostream>
2 using namespace std;
3 typedef struct{
4     int i;
5     double d;} typos;
6
7 bool operator < (typos x, typos y){
8     return x.i < y.i && x.d < y.d;
9 }
10
11 typos operator + (typos x, typos y){
12     typos z;
13     z.i = x.i + y.i;
14     z.d = x.d + y.d;
15     return z;
16 }
```

```
18 int main(){
19     typos x = {2,3.7}, y = {10,5.1};
20     if (x < y) x = x + y;
21     else y = x + y;
22     cout << x.i << " " << x.d << endl;
23     return 0;
24 }
```

Παράδειγμα υπερφόρτωσης των τελεστών '`+`' & '`<`' για την πρόσθεση δύο `struct` τύπου `typos`

## Υπερφόρτωση Ονόματος για Διαφορετικού Τύπου Οντότητες

- Το ίδιο όνομα μπορεί να χρησιμοποιείται για διαφορετικού είδους οντότητες μέσα στο ίδιο πρόγραμμα
  - πχ. μία μεταβλητή, έναν τύπο και μία συνάρτηση
    - Δεν επιτρέπεται στις περισσότερες γλώσσες προγραμματισμού
    - Χρήσιμο αν για κάποιο λόγο πρέπει να περιοριστούν τα ονόματα που επιτρέπεται να χρησιμοποιήσει ο προγραμματιστής

### Παράδειγμα

#### Παράδειγμα 3 από τη γλώσσα Java

Στο διπλανό παράδειγμα βλέπουμε ότι ένας ορισμός κλάσης (class A), μίας μεθόδου (A A (A A)) και μίας ετικέτας (A:) μοιράζονται το όνομα A

```

1 class A{
2     A A(A A){
3         A:
4             for (;;) {
5                 if (A.A(A) == A) break A;
6             }
7         return A;
8     }
9 }
```

## Περιβάλλον

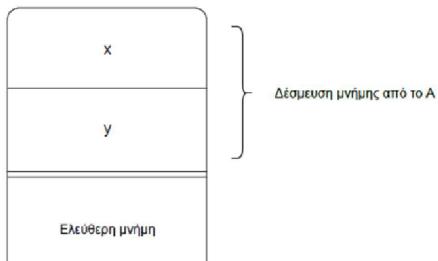
- Το **Περιβάλλον** διατηρεί τις συνδέσεις ονομάτων με τις ιδιότητες θέσης τους

*Environment : Names → Locations*

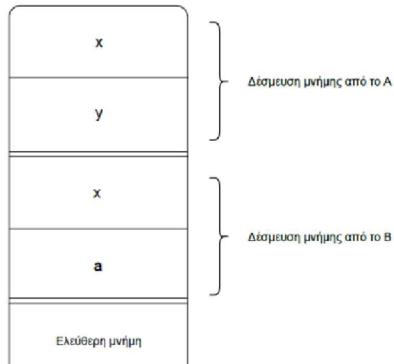
- Ανάλογα με τη γλώσσα το περιβάλλον εκτέλεσης μπορεί να κατασκευάζεται
  - Στατικά:** σε χρόνο φόρτωσης (FORTRAN)
  - Δυναμικά:** σε χρόνο εκτέλεσης (LISP)
  - Σε συνδιασμό των δύο (C, C++, Ada, Java)**
- Δε συνδέονται θέσεις μνήμης σε όλα τα ονόματα
  - πχ οι σταθερές και οι τύποι που έχουν ονόματα, είναι ονόματα που υπάρχουν μόνο κατά τη μεταγλώτιση
- Η κατασκευή του περιβάλλοντος εκτέλεσης εξαρτάται από τις δηλώσεις
  - μεταγλωτιστής** -> αναλύονται για να δημιουργηθεί ο κατάλληλος κώδικας εκχώρησης μνήμης για κάθε περίπτωση
  - διερμηνευτής** -> οι συνδέσεις ιδιοτήτων μίας δήλωσης, συμπεριλαμβάνουν και τη σύνδεση ιδιότητας θέσης
- Σε γλώσσες με δομή **block**
  - Στις καθολικές μεταβλητές μπορεί να εκχωρούνται και θέσεις **στατικά** (οι ιδιότητές τους δεν μεταβάλλονται κατά την εκτέλεση)
  - Στις τοπικές μεταβλητές εκχωρούνται θέσεις **δυναμικά**, όταν κατά την εκτέλεση προσεγγίζεται το σχετικό **block**
  - Η σύνδεση θέσεων σε ονόματα γίνεται με **προσέγγιση στοίβας** (όπως στον πίνακα συμβόλων)
- Για ένα όνομα σε κάποια στιγμή μπορεί να υπάρχουν πολλες διαφορετικές συνδέσεις θέσης, αλλά μόνο μία είναι προσπελάσιμη
  - αυτό ισχύει για όλες τις γλώσσες με δομή **block** και λεξική εμβέλεια
- Πρέπει για ένα όνομα να μπορούμε να διακρίνουμε τη δήλωση που έχει προκαλέσει τη σύνδεση συγκεκριμένης θέσης στο όνομα
  - όταν μία θέση έχει εκχωρηθεί ως συνέπεια μίας δήλωσης, λέγεται **αντικείμενο**
  - στη C οι μεταβλητές σε συναρτήσεις συσχετίζονται με αντικέιμενα, σε αντίθεση με τις καθολικές (δεν συνδέονται με θέση μνήμης)

## Παράδειγμα

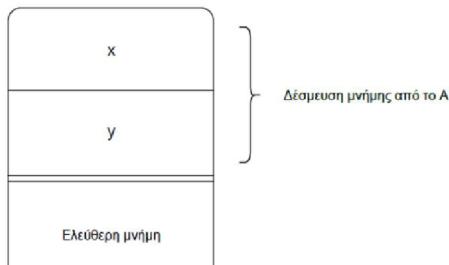
```
1 A: {      int x;  
2           char y;  
3           ...  
4 B: {      double x;  
5           int a;  
6           ...  
7           } /* end B */  
8 C: {      char y;  
9           int b;  
10          ...  
11 D: {      int x;  
12           double y;  
13           ...  
14           } /* end D */  
15           ...  
16           } /* end C */  
17           ...  
18       } /* end A */
```



Κατάσταση μνήμης στην γραμμή 3 του προγράμματος



Κατάσταση μνήμης στην γραμμή 6 του προγράμματος



Κατάσταση μνήμης στην γραμμή 7 του προγράμματος



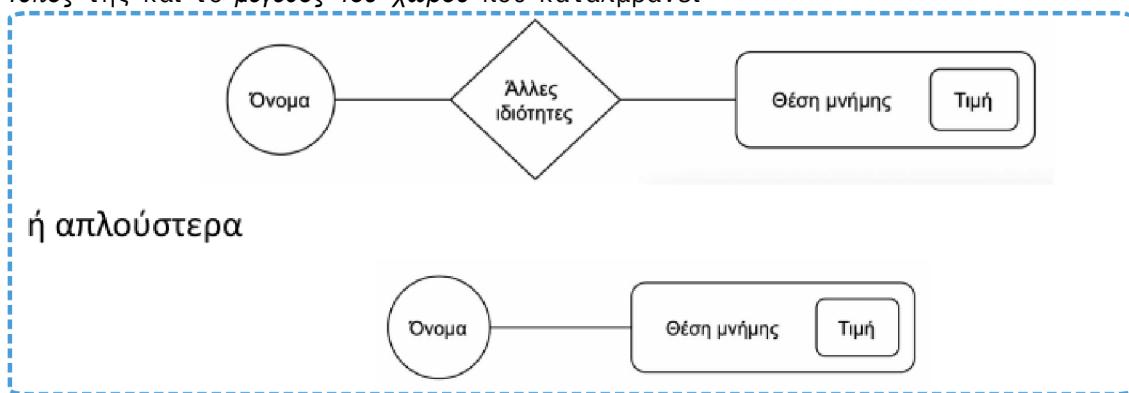
Κατάσταση μνήμης στην γραμμή 13 του προγράμματος

## Χρόνος Ζωής

- Ο **Χρόνος Ζωής** ενός αντικειμένου αντιπροσωπεύει τη διάρκεια της εκχώρησης από το περιβάλλον
- Οι χρόνοι ζωής των αντικειμένων μπορεί να εκτείνονται έξω από την περιοχή που έιναι δυνατό να προσπελαστούν
- Αν η γλώσσα προγραμματισμού υποστηρίζει τη χρήση δεικτών, τότε το περιβάλλον εκτέλεσης διαθέτει τη δομή για να παρέχει τις απαραίτητες λειτουργίες
  - Ένας **δεικτης** είναι αντικείμενο, που η αποθήκευμένη τιμή του έιναι αναφορά, που παραπέμπει σε άλλο αντικείμενο
- Σε γλώσσες με δομή block και δυνατότητα εκχώρησης σε σωρό (heap), έχουμε τρία είδη εκχώρησης θέσεων στο περιβάλλον εκτέλεσης
  - **Στατική** (καθολικές μεταβλητές)
  - **Αυτόματη** (τοπικές μεταβλητές)
  - **Δυναμική** (εκχώρηση σε σωρό)
- Αυτές οι κατηγορίες λέγονται και κλάσεις αποθήκευσης
- Σε γλώσσες όπως η C επιτρέπεται και η δήλωση κλάσης αποθήκευσης, μαζί με τον τύπο δεδομένων

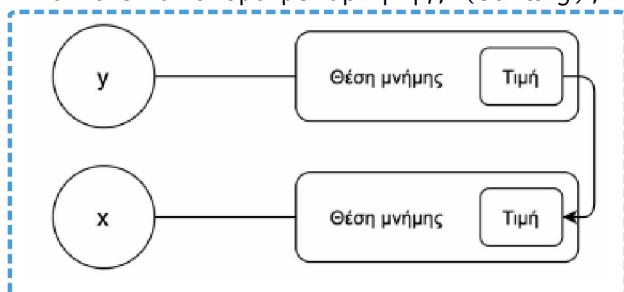
## Μεταβλητές

- **Μεταβλητή**: είναι ένα αντικείμενο, που η αποθήκευμενη τιμή του αλλάζει κατά την εκτέλεση του προγράμματος
- Κάθε μεταβλητή ορίζεται πλήρως από τις ιδιότητές της, που έιναι το όνομα, η θέση της και ο τύπος της και το μέγεθος του χώρου που καταλμβάνει



## Εντολές Ανάθεσης με Σημασία Αντιγραφής

- Η τιμή μίας μεταβλητής αλλάζει, με εντολή ανάθεσης  $x = e$ , όπου  $x$  είναι το όνομα μεταβλητής και  $e$  είναι μία έκφραση
  - Η σημασία αυτής της εντολής είναι ότι η  $e$  αποτιμάται σε μία τιμή και στη συνέχεια αντιγράφεται στη θέση της  $x$
  - Εάν  $e$  είναι όνομα μεταβλητής, (έστω  $y$ ), τότε η ανάθεση  $x = y$  απεικονίζεται ως

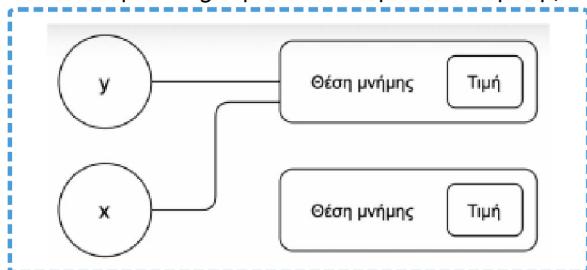


## I-τιμές και r-τιμές

- Καθώς μία μεταβλητή βρίσκεται σε μία θέση και έχει μία αποθηκευμένη τιμή, πρέπει να γίνεται διάκριση μεταξύ αυτών
  - η **τιμή** που βρίσκεται στη θέση μίας μεταβλητής, αποκαλείται συχνά **r-τιμή** (εμφανίζεται δεξιά στην εντολή ανάθεσης)
  - η **θέση** μίας μεταβλητής λέγεται και **I-τιμή** (εμφανίζεται δεξιά στην εντολή ανάθεσης)
- Η C επιτρέπει την ανάμειξη εκφράσεων με αναθέσεις, όπου μπορεί να έχουμε την r-τιμή και την I-τιμή μίας μεταβλητής
  - πχ  $x = x + 1;$
- Μπορούμε επίσης να ανακτήσουμε τη διεύθυνση μίας μεταβλητής με τη μορφή **δείκτη**
  - $\&x$  είναι ένας δείκτης στη διεύθυνση της  $x$
  - με  $*\&x$  παίρνουμε πάλι την τιμή της  $x$  ως I-τιμή αλλά και r-τιμή
- Αυτά τα χαρακτηριστικά καθιστούν τον προγραμματισμό σε C επιρρεπή σε μη προβλεπόμενες χρήσεις μεταβλητών, που δεν εντοπίζονται εύκολα

## Εντολές Ανάθεσης με Σημασία Κοινής Χρήσης

- Σε κάποιες γλώσσες, η εντολή ανάθεσης έχει διαφορετική σημασία: αντί για τις τιμές, αντιγράφονται οι θέσεις
- Αυτή η περίπτωση λέγεται **ανάθεση με κοινή χρήση**
- Η εντολή  $x = y$  προκαλεί τη σύνδεση της θέσης της  $y$  στην  $x$



- Η Java υποστηρίζει την ανάθεση με κοινή χρήση για όλες τις μεταβλητές **αντικειμένων**, αλλά όχι για απλά δεδομένα

## Σταθερές

- Μία **σταθερά** είναι μία οντότητα που έχει **σταθερή τιμή** σε όλη τη διάρκεια ύπαρξής της σε ένα πρόγραμμα
- Διαφέρει από της μεταβλητές, στο ότι η σταθερά δεν έχει συνδεδεμένη ιδιότητα θέσης, αλλά μόνο μία τιμή. Λέμε ότι μία σταθερά έχει **σημασία τιμής αντί για σημασία αποθήκευσης μίας μεταβλητής**



- Η σταθερά μπορεί να παίρνει τιμή, που γίνεται γνωστή μόνο κατά την εκτέλεση του προγράμματος. Τότε θα πρέπει να αποθηκευτεί η τιμή της στην μνήμη
  - Αντίθετα από μία μεταβλητή, όταν η τιμή μίας σταθεράς υπολογίζεται, δεν μπορεί να μεταβληθεί
- Σε πολλές γλώσσες, και ιδιέταιρα στις **συναρτησιακές**, δίνεται έμφαση στη χρήση σταθερών αντί για μεταβλητές, λόγω της πολύ **απλούστερης σημασίας** τους
  - Κάθε όνομα σταθεράς εκφράζει μόνο μία τιμή, που μετά τη δημιουργία της παραμένει αμετάβλητη, ανεξάρτητα από τη θέση ή τη χρήση της στο πρόγραμμα
- Μερικές γλώσσες όπως η Haskell, απαγορεύουν εντελώς τη χρήση μεταβλητών και στηρίζονται αποκλειστικά στη χρήση σταθερών για υπολογισμούς

## Στατικές και Δυναμικές Σταθερές

- Οι σταθερές μπορεί να είναι
  - **Στατικές:** η τιμή τους μπορεί να υπολογιστεί πριν από την εκτέλεση
    - **Χρόνου Μεταγλώττισης**
    - **Χρόνου Φόρτωσης:** υπολογίζονται σε χρόνο φόρτωσης ή στην αρχή εκτέλεσης του προγράμματος
  - **Δυναμικές:** αυτές που μπορούν να υπολογιστούν μόνο στο **χρόνο εκτέλεσης**

## Ψευδωνυμία

- Όταν ένα αντικείμενο συνδέεται ταυτόχρονα με δύο διαφορετικά ονόματα, λέμε ότι έχουμε ένα **ψευδώνυμο**
- Η ψευδωνυμία μπορεί να συμβέι με διαάφορους τρόπους, όπως κατά την **κλήση διαδικασιών**, χρήση μεταβλητών **δείκτη**, ή όταν γίνεται **ανάθεση με κοινή χρήση**
- Τα ψευδώνυμα μπορεί να έχουν πιθανώς **επιβλαβείς παρενέργειες**
  - ως παρενέργεια εντολής ορίζουμε κάποια μεταβολή στην τιμή μίας μεταβλητής, που διατηρείται μετά την εκτέλεσή της
  - δεν είναι όλες οι παρενέργειες επιβλαβείς: η ανάθεση τιμής σε μία μεταβλητή αλλάζει την τιμή της, όπως ρητά επιδιώκεται από τη σημασία της εντολής
  - όμως παρενέργειες, όπως αλλαγές σε μεταβλητές, που δεν αναφέρονται ρητά τα ονόματά τους στην εντολή, είναι πιθανώς επιβλαβείς

Παραδείγματα

### Παράδειγμα 4 από τη γλώσσα C

```

1 int *x, *y;
2 x = (int *) malloc(sizeof(int));
3 *x = 3;
4 y = x; /* *x and *y now aliases */
5 *y = 5;
6 printf("%d\n", *x);

```

Μετά την ανάθεση του pointer x στον pointer y, οι δύο pointers δείχνουν στην ίδια μεταβλητή και τελικά στην γραμμή 6 εκτυπώνεται η τιμή 5

### Παράδειγμα 5 από τη γλώσσα Java

```

1 class AliasTest{
2     public static void main(String[] args){
3         int[] x = {10,20,30};
4         int[] y = x;
5         x[0] = 5;
6         System.out.println(y[0]);
7     }
8 }

```

Στην γραμμή 4 ορίζεται ο y και αρχικοποιείται με x. Στην γραμμή 5 το x[0] αλλάζει τιμή και στην γραμμή 6 εκτυπώνεται το y[0] με την τιμή 5, εξαιτίας της προηγούμενης ανάθεσης με κοινή χρήση

## Αιωρούμενες Αναφορές

- **Αιωρούμενη Αναφορά** είναι μία θέση μνήμης, που έχει ανακληθεί η εκχώρησή της από το περιβάλλον, αλλά εξακολουθεί να είναι προσπελάσιμη μέσα από το πρόγραμμα (δηλαδή όταν ένα αντικείμενο είναι προσπελάσιμο πέρα από το χρόνο ζωής του στο περιβάλλον)
- Το πρόβλημα παρουσιάζεται με τη χρήση μεταβλητών δείκτη
- Στη C μπορεί επίσης να έχουμε αιωρούμενες αναφορές λόγω αυτόματης ανάκλησης της εκχώρησης τοπικών μεταβλητών κατά την έξοδο από το block στο οποίο δηλώνονται. Αυτό συμβαίνει λόγω του τελεστή διεύθυνσης "&"

### Παραδείγματα

#### Παράδειγμα 6 από τη γλώσσα C

Αιωρούμενη αναφορά στην γλώσσα C

```

1 int *x, *y;
2 ...
3 x=(int *) malloc(sizeof(int));
4 ...
5 *x=5;
6 ...
7 y=x;           /* *y and *x are now aliases */
8 free(x);      /* *y now is dangling reference */
9 ...
10 printf("%d\n",*y); /* illegal reference */

```

Απλό παράδειγμα αιωρούμενης αναφοράς μέσω δείκτη (γραμμή 10) που δείχνει σε αντικείμενο που έχει ήδη ανακληθεί από την free()

#### Παράδειγμα 7 από τη γλώσσα C

Αιωρούμενη μεταφορά μετά από έξοδο από block

```

1 { int * x;
2   {
3     int y;
4     y = 2;
5     x = &y;
6   }
7 } /* *x is now a dangling reference */

```

#### Παράδειγμα 8 από τη γλώσσα C++

- Έστω κλάση String ορισμένη όπως φαίνεται στην εικόνα
- Παρατηρήστε ότι member variable m\_Buffer είναι δείκτης σε διεύθυνση μνήμης στο heap.
- Τι γίνεται όταν τρέξουμε τον παρακάτω κώδικα;

```

int main()
{
    String string = "Cherno";
    String second = string;

    std::cout << string << std::endl;
    std::cout << second << std::endl;
}
std::cin.get();

```

```

class String
{
private:
    char* m_Buffer;
    unsigned int m_Size;
public:
    String(const char* string)
    {
        m_Size = strlen(string);
        m_Buffer = new char[m_Size + 1];
        memcpy(m_Buffer, string, m_Size);
        m_Buffer[m_Size] = 0;
    }

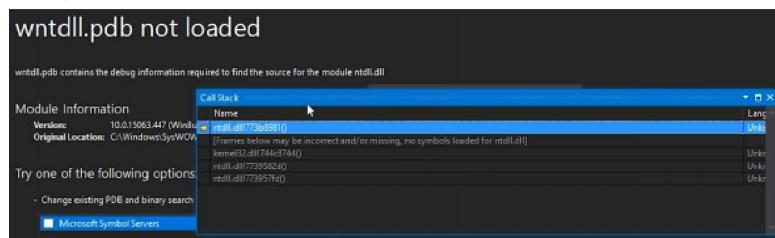
    ~String()
    {
        delete[] m_Buffer;
    }
};

```

- Αρχικά εκτυπώνονται τα δύο string όπως ήταν αναμενόμενο:



- Στην έξοδο του προγράμματος όμως βλέπουμε το παρακάτω error:



- Που είναι το πρόβλημα;
- Στην έξοδο του προγράμματος καθώς καλούνται οι destructors των αντικειμένων String, ελευθερώνεται η μνήμη που δείχνει ο pointer m\_Buffer.
- Όταν δημιουργήθηκε το αντικείμενο second, αντιγράφηκε η τιμή του pointer m\_Buffer του πρώτου αντικειμένου string, και άρα τα 2 αντικείμενα έδειχναν στην ίδια μνήμη.
- Όταν διαγράφηκε το αντικείμενο string απελευθερώθηκε η μνήμη του. Επομένως όταν το second πάει προς καταστροφή, προσπαθεί να απελευθερώσει μνήμη που έχει ήδη απελευθερωθεί.

# Γραμματικές Χωρίς Συμφραζόμενα

## Γραμματική Χωρίς Συμφραζόμενα (Γ.Χ.Σ)

$$G = (V, \Sigma, R, S)$$

όπου :

- $V$  το αλφάβητο, ένα πεπερασμένο σύνολο συμβόλων
- $\Sigma \subseteq V$  τα τερματικά σύμβολα, ενώ  $V - \Sigma$  τα μη-τερματικά σύμβολα
- $R$  ένα σύνολο κανόνων παραγωγής της μορφής
  - $A \rightarrow u$ , με
    - $A \in V - \Sigma$
    - $u \in V^*$  μία συμβολοσειρά από σύμβολα
- $S \in V - \Sigma$  ένα μη τερματικό σύμβολο που λέγεται αρχή της  $G$

## Παραγωγή

- Αν η συμβολοσειρά  $w \in \Sigma^*$  παράγεται από την αρχή  $S$  μέσω κανόνων της  $G$ , γράφουμε  $S \Rightarrow^* w$  και  $w \in L(G)$ -η γλώσσα της  $G$
- Σ' αυτή την περίπτωση λέμε ότι η  $w$  συμμορφώνεται στη σύνταξη των προτάσεων της γλώσσας της  $G$

### Παράδειγμα

$assign-stmt \rightarrow var := expr$	(1)
$expr \rightarrow term$	(2)
$expr + term$	(3)
$expr - term$	(4)
$term \rightarrow prim-expr$	(5)
$term \times prim-expr$	(6)
$term / prim-expr$	(7)
$prim-expr \rightarrow var$	(8)
<b>NUM</b>	(9)
$( expr )$	(10)
$var \rightarrow ID$	(11)
$ID [sbscr-lst]$	(12)
$sbscr-lst \rightarrow expr$	(13)
$sbscr-lst, expr$	(14)

- Αν η συμβολοσειρά  $w \in \Sigma^*$  παράγεται από την αρχή  $S$  μέσω κανόνων της  $G$ , τότε γράφουμε  $S \Rightarrow^* w$  και  $w \in L(G)$  - η γλώσσα της  $G$ .
- Σ' αυτή την περίπτωση λέμε ότι η  $w$  συμμορφώνεται στη σύνταξη των προτάσεων της γλώσσας της  $G$ .
- Παραγωγή του προγράμματος:  $a [ 4 ] := 3 \times 5$ 

$$\begin{aligned} assign-stmt &\stackrel{(1)}{\Rightarrow} var := expr \stackrel{(12)}{\Rightarrow} ID[sbscr-lst] := expr \stackrel{(13)}{\Rightarrow} ID[expr] := expr \\ &\stackrel{(2)}{\Rightarrow} ID[term] := expr \stackrel{(5)}{\Rightarrow} ID[prim-expr] := expr \stackrel{(9)}{\Rightarrow} ID[NUM] := expr \\ &\stackrel{(2)}{\Rightarrow} ID[NUM] := term \stackrel{(6)}{\Rightarrow} ID[NUM] := term \times prim-expr \\ &\stackrel{(5)}{\Rightarrow} ID[NUM] := prim-expr \times prim-expr \\ &\stackrel{(9)}{\Rightarrow} ID[NUM] := NUM \times prim-expr \stackrel{(9)}{\Rightarrow} ID[NUM] := NUM \times NUM \end{aligned}$$

- **Αριστερή Παραγωγή:** όταν σε κάθε βήμα εφαρμόζουμε κανόνα στο πρώτο μη-τερματικό από τα αριστερά
- **Δεξιά Παραγωγή:** όταν σε κάθε βήμα εφαρμόζουμε κανόνα στο πρώτο μη-τερματικό από τα δεξιά

# Παράγωγα ή Συντακτικά Δέντρα

- Η σύνταξη ενός προγράμματος αποτυπώνεται στο **παράγωγο/συντακτικό δέντρο**

## Παράγωγο/Συντακτικό Δέντρο

Έστω  $\Gamma\Delta G = (V, \Sigma, R, S)$

- Για κάθε σύμβολο  $a \in \Sigma$ , το γράφημα με ένα μόνο κόμβο  $a$  είναι πράγωγο δέντρο, με ρίζα και μοναδικό φύλλο τον  $a$

$a$

- Αν  $A \rightarrow \epsilon$  είναι κανόνας του  $R$  με  $\epsilon$  την κενή συμβολοσειρά, τότε το παρακάτω είναι παράγωγο δέντρο, με ρίζα τον  $A$  και μοναδικό φύλλο το  $\epsilon$

$A$

$\epsilon$

- Αν τα πρακάτω είναι παράγωγα δέντρα με ρίζες τα  $A_1, A_2, \dots, A_n$  και παραγόμενες συμβολοσειρές

ΤΙΣ  $y_1, y_2, \dots, y_n$

$A_1$

$y_1$

$A_2$

$y_2$

$\dots$

$A_n$

$y_n$

και αν  $A \rightarrow A_1A_2\dots A_n$  είναι κανόνας του  $R$ , τότε το παρακάτω είναι παράγωγο δέντρο με ρίζα  $A$  και παραγόμενη συμβολοσειρά την  $y_1 \cdot y_2 \cdot \dots \cdot y_n$

$A$

$A_1$

$y_1$

$A_2$

$y_2$

$\dots$

$A_n$

$y_n$

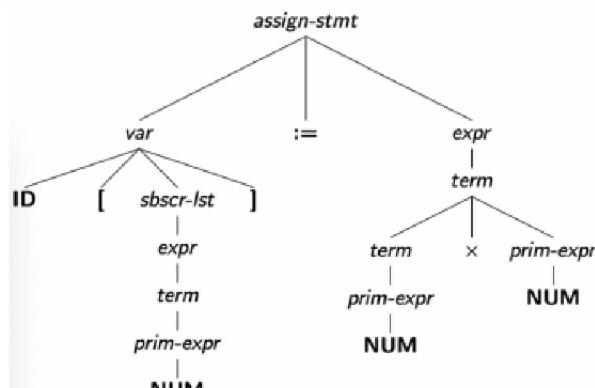
## Παράδειγμα

### Παράγωγα ή Συντακτικά Δέντρα

$assign-stmt \rightarrow var := expr$	(1)
$expr \rightarrow term$	(2)
$expr + term$	(3)
$expr - term$	(4)
$term \rightarrow prim-expr$	(5)
$term \times prim-expr$	(6)
$term / prim-expr$	(7)
$prim-expr \rightarrow var$	(8)
$NUM$	(9)
$( expr )$	(10)
$var \rightarrow ID$	(11)
$ID [ sbscr-lst ]$	(12)
$sbscr-lst \rightarrow expr$	(13)
$sbscr-lst, expr$	(14)

- Παραγωγή του προγράμματος:  $a [ 4 ] := 3 \times 5$

$$\begin{aligned}
 assign-stmt &\xrightarrow{(1)} var := expr \xrightarrow{(12)} ID[sbscr-lst] := expr \xrightarrow{(13)} ID[expr] := expr \\
 &\xrightarrow{(2)} ID[term] := expr \xrightarrow{(5)} ID[prim-expr] := expr \xrightarrow{(9)} ID[NUM] := expr \\
 &\xrightarrow{(2)} ID[NUM] := term \xrightarrow{(6)} ID[NUM] := term \times prim-expr \\
 &\xrightarrow{(5)} ID[NUM] := prim-expr \times prim-expr \\
 &\xrightarrow{(9)} ID[NUM] := NUM \times prim-expr \xrightarrow{(9)} ID[NUM] := NUM \times NUM
 \end{aligned}$$

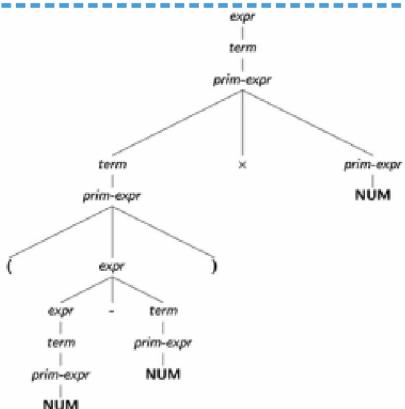


22

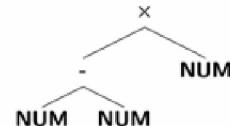
## Αιφαρετικά Συντακτικά Δέντρα

- Τα παράγωγα δέντρα διατηρούν όλη την πληροφορία σχετικά με την παραγωγή μίας συμβολοσειράς από μία γραμματική
- Μέρος αυτής της πληροφορίας δεν επηρεάζει τη σημασία της γλώσσας προγραμματισμού και μπορεί να παραληφθεί για την απλούστευση του δέντρου
- Το **Αιφαρετικό συντακτικό δέντρο** δεν περιλαμβάνει όλες τις πληροφορίες

Παράδειγμα



π.χ. μπορούμε να παραλείψουμε τις παρενθέσεις στις εκφράσεις και οι πράξεις να ομαδοποιούνται απευθείας στη δομή του δέντρου



23

## Διφορούμενες ή Ασαφείς Γραμματικές

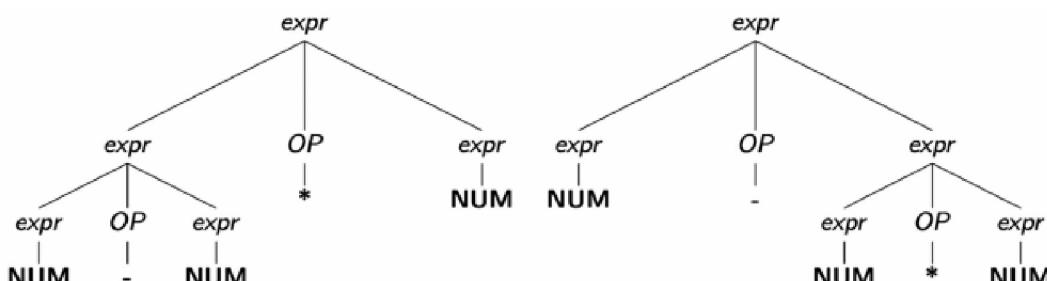
- Μία ΓΧΣ λέγεται **διφορούμενη ή ασαφής** όταν υπάρχει πρόταση της γλώσσας με τουλάχιστον δύο παράγωγα δέντρα (δύο διαφορετικές αριστερές ή δεξιές παραγωγές)
- Η ΓΧΣ που περιγράφει τη σύνταξη μίας γλώσσας δεν πρέπει να είναι διφορούμενη, ή αν είναι, το πρόβλημα πρέπει να αντιμετωπίζεται κατά τη μεταγλώττιση

## Ασάφεια Προτεραιότητας Τελεστών

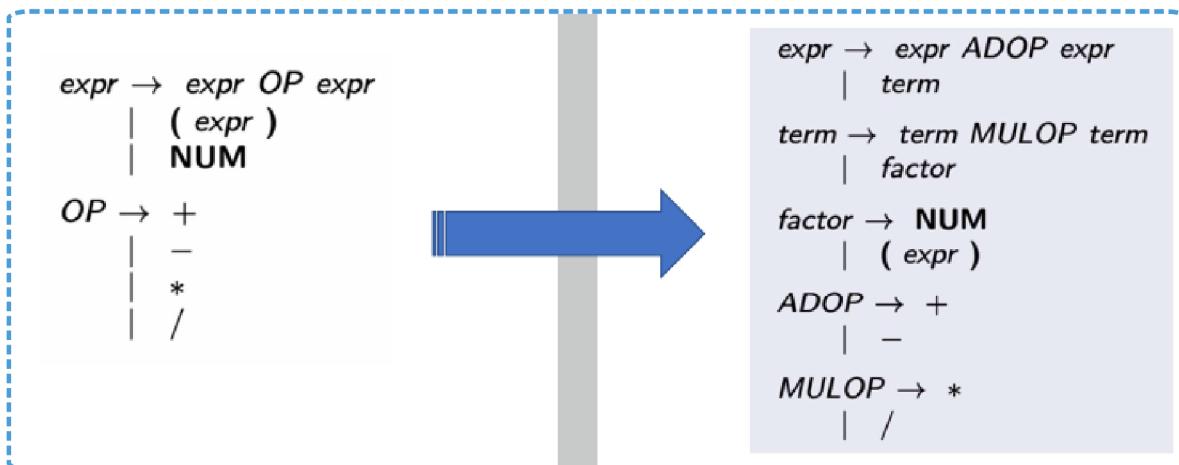
Παράδειγμα

$expr \rightarrow expr \text{ } OP \text{ } expr$	(1)
$( \text{ } expr \text{ } )$	(2)
$\text{NUM}$	(3)
$OP \rightarrow +$	(4)
$-$	(5)
$*$	(6)
$/$	(7)

Δέντρα της ΓΧΣ για τις δύο παραγωγές της  $27-5*8$  (η σωστή σημασία/αποτέλεσμα δίνεται στο δεξί δέντρο):



- Για να διορθώσουμε το πρόβλημα ασάφειας προτεραιότητας τελεστών:
  - Ωμαδοποιούμε τους τελεστές με την ίδια προτεραιότητα σε κανόνα με κοινό μη-τερματικό σύμβολο



## Ασάφεια Πορσεταιριστικότητας Τελεστών

### Παράδειγμα

- ΓΧΣ με πρόβλημα στον ορισμό της προσεταιριστικότητας των τελεστών

Δέντρα για δύο αριστερές παραγωγές της πρότασης 27-5-8 (η σωστή σημασία δίνεται από το δεξί δέντρο)

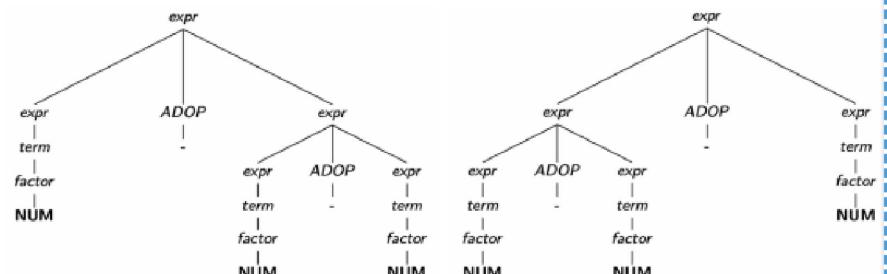
$expr \rightarrow expr \text{ } ADOP \text{ } expr$   
   |  
   term

$term \rightarrow term \text{ } MULOP \text{ } term$   
   |  
   factor

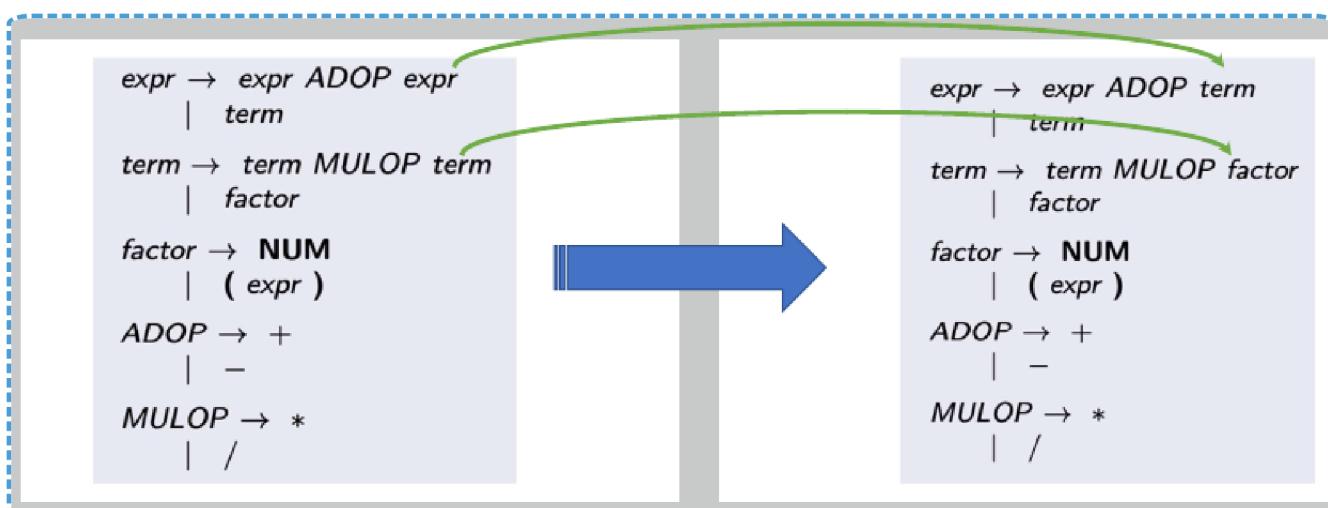
$factor \rightarrow \text{NUM}$   
   |  
   (  $expr$  )

$ADOP \rightarrow +$   
   |  
   -

$MULOP \rightarrow *$   
   |  
   /



- Για να διορθώσουμε το πρόβλημα ασάφειας προσεταιριστικότητας τελεστών:
  - Ένας κανόνας ΓΧΣ παράγει δέντρο με αριστερή προσεταιριστικότητα αν αυτός είναι αποκλειστικά αριστερά αναδρομικός



## Ασάφεια Μετέωρου Else

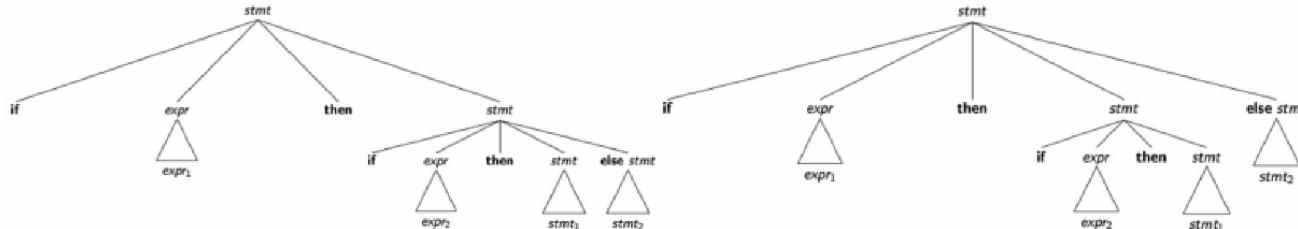
### Παράδειγμα

- Διφορούμενη ΓΧΣ με το πρόβλημα του μετέωρου else

$$\begin{aligned}stmt \rightarrow & \text{ if } expr \text{ then } stmt \\& | \quad \text{if } expr \text{ then } stmt \text{ else } stmt\end{aligned}$$

Δύο δέντρα με διαφορετική σημασία για το πρόγραμμα

$$\text{if } expr_1 \text{ then if } expr_2 \text{ then } stmt_1 \text{ else } stmt_2$$



Διόρθωση ΓΧΣ που έχει το πρόβλημα ασάφειας μετέωρου else:

- Ισχύει – σε όλες τις γλ. προγραμματισμού – η **αρχή του πλησιέστερου if**  
Αυτό σημαίνει ότι όταν δε γίνεται ένθεση εντολών if με χρήση διαχωριστών της γλώσσας, τότε κάθε else θα πρέπει να αναφέρεται στο πλησιέστερο προαναφερόμενο if.

$$\begin{aligned}stmt \rightarrow & \text{ if } expr \text{ then } stmt \\& | \quad \text{if } expr \text{ then } stmt \text{ else } stmt\end{aligned}$$



$$\begin{aligned}stmt \rightarrow & \text{ bal\_stmt} \\& | \quad \text{unbal\_stmt} \\bal\_stmt \rightarrow & \text{ if } expr \text{ then } bal\_stmt \text{ else } bal\_stmt \\unbal\_stmt \rightarrow & \text{ if } expr \text{ then } stmt \\& | \quad \text{if } expr \text{ then } bal\_stmt \text{ else } unbal\_stmt\end{aligned}$$

## Συμβολισμός BNF/EBNF

- BNF (Backus Normal Form)**
  - Τα μη-τερματικά σύμβολα περικλείονται από < και >
  - Για κανόνες παραγωγής χρησιμοποιείται το → ή κάποιες φορές το ::=
  - Οι κανόνες παραγωγής για το ίδιο μη-τερματικό αναφέρονται σε έναν ενιαίο κανόνα
- EBNF (Extended BNF)**

### Συμβολισμός EBNF του N. Wirth

Μη τερματικά σύμβολα	Κανονικοί χαρακτήρες, που δεν περικλείονται σε < και >.
Τερματικά σύμβολα	Περιλαμβάνονται σε " και ", όπως π.χ. το "+".
	Συμβολισμός εναλλακτικών περιπτώσεων.
( και )	Ομαδοποίηση συμβόλων και αλλαγή της προτεραιότητας εφαρμογής των άλλων μετασυμβόλων.
[ και ]	Προαιρετική εμφάνιση ενός συμβόλου ή μιας ομάδας συμβόλων.
{ και }	Προαιρετική επανάληψη ενός συμβόλου ή μιας ομάδας συμβόλων.
=	Χρησιμοποιείται στους ορισμούς των κανόνων παραγωγής.
.	Σημειώνει το τέλος ενός κανόνα παραγωγής.
(* και *)	Χρησιμοποιούνται για τη διατύπωση σχολίων.

## Παράδειγμα

```

 $expr \rightarrow expr \text{ ADOP } expr$ 
| term
 $term \rightarrow term \text{ MULOP } term$ 
| factor
 $factor \rightarrow \text{NUM}$ 
| ( expr )
 $\text{ADOP} \rightarrow +$ 
| -
 $\text{MULOP} \rightarrow *$ 
| /

```



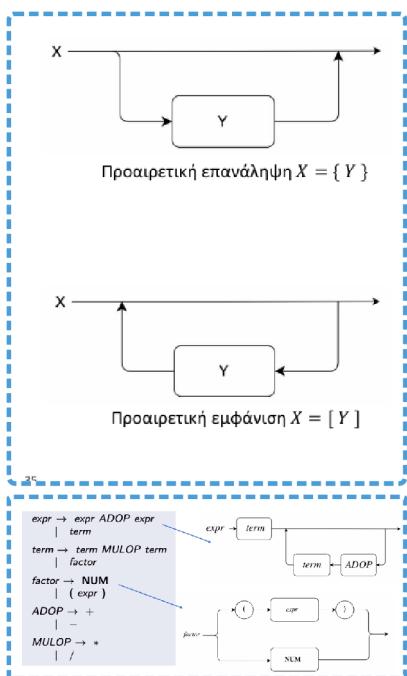
$expr$	= $term \{ \text{ADOP } term \}$
$term$	= $factor \{ \text{MULOP } factor \}$
$factor$	= “(” $expr$ “)”   “NUM”
$\text{ADOP}$	= “+”   “-”
$\text{MULOP}$	= “*”   “/”

24

## Συντακτικά Διαγράμματα

- **Συντακτικό Διάγραμμα:** μία γραφική παράσταση κανόνων EBNF
  - **τετράγωνα:** μη-τερματικά
  - **κύκλοι:** τερματικά
  - **τόξα:** δομές ακολουθίας ή επιλογής
  - **αριστερό μέρος:** μη-τερματικό του οποίου ο κανόνας αναπαριστάται

## Παραδείγματα



# Κανονικές Εκφράσεις

## Κανονικές Εκφράσεις (Ορισμός)

- Δοθέντος πεπερασμένου αλφαβήτου  $\Sigma$ , **Κανονικές Εκφράσεις** είναι οι σταθερές
  - $\emptyset$ : το **κενό σύνολο**
  - $\varepsilon$ : το σύνολο με την **κενή συμβολοσειρά**
  - $a$ : το σύνολο με τη συμβολοσειρά  $a$ , όπου  $a \in \Sigma$
- Δοθέντων των *Κανονικών Εκφράσεων*  $R$  και  $S$ , ορίζουμε επίσης τις ακόλουθες πράξεις ως K.E.
  - $R \cdot S = \{ab | a \in R, b \in S\}$ : η **Παράθεση** της  $R$  με την  $S$
  - $R|S = \{a | a \in R \cup S\}$ : η **Διάζευξη** της  $R$  με την  $S$
  - $R^*$ : η **Kleene Star** της  $R$ , το ελάχιστο υπερσύνολο της  $R$  που περιλαμβάνει την  $\varepsilon$  και είναι **κλειστό προς την παράθεση**
- Η **Kleene Star**  $R^*$  εκφράζει το σύνολο όλων των συμβολοσειρών που σχηματίζονται με παράθεση πεπερασμένου αριθμού (ή μηδέν) συμβολοσειρών της  $R$ 
  - πχ  

$$\{\text{"ab"} \mid \text{"c"}\}^* = \{\varepsilon, \text{"ab"}, \text{"c"}, \text{"abab"}, \text{"abc"}, \text{"cab"}, \text{"cc"}, \text{"ababab"}, \dots\}$$
- Προτεραιότητα Τελεστών:** υποθέτουμε πως η προτεραιότητα πράξεων πάει ως εξής:
  1. Kleene Star
  2. Παράθεση
  3. Διάζευξη

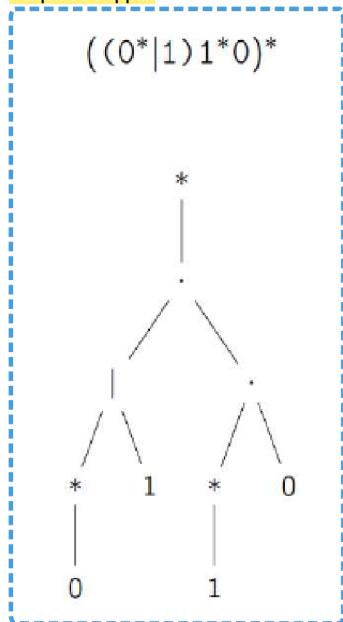
## K.E. -> Ελάχιστο ΝΠΑ

### K.E. -> Συντακτικό Δέντρο

- Απλό

Παράδειγμα

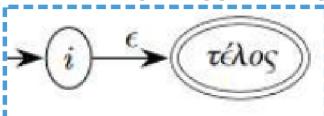
$$((0^*|1)1^*0)^*$$



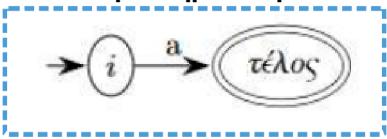
### Συντακτικό Δέντρο -> ΜΝΠΑ

- Ανάπτυξη Thompson

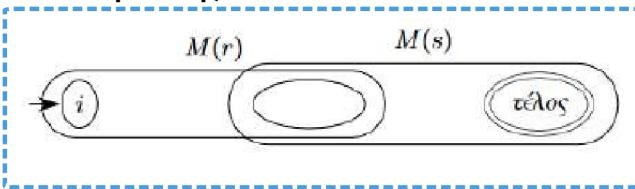
- ΜΝΠΑ Κενής Συμβολοσειράς  $\varepsilon$



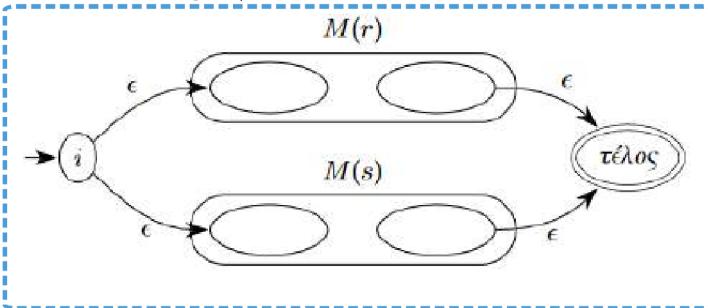
- ΜΝΠΑ Χαρακτήρα - Πρότυπο  $a$



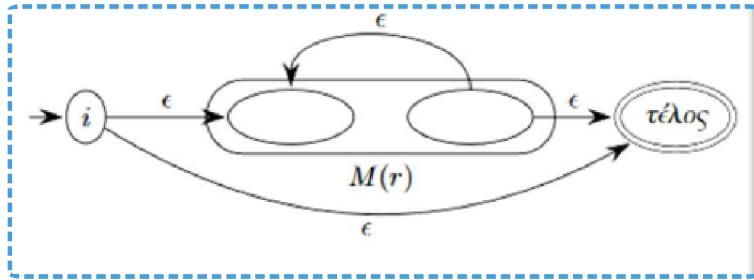
- ΜΝΠΑ Παράθεσης  $r \cdot s$



- ΜΝΠΑ Διάζευξης  $r|s$

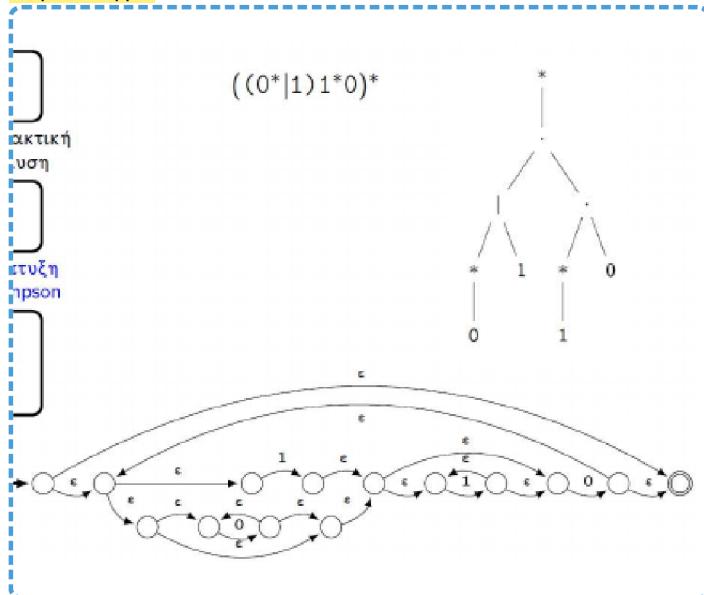


- ΜΝΠΑ Kleene Star  $r^*$



- Συνδιάζοντας τα παραπάνω ΜΝΠΑ χτίζουμε το τελικό ΜΝΠΑ από το φύλλα πρός τη ρίζα του Συντακτικού Δέντρου

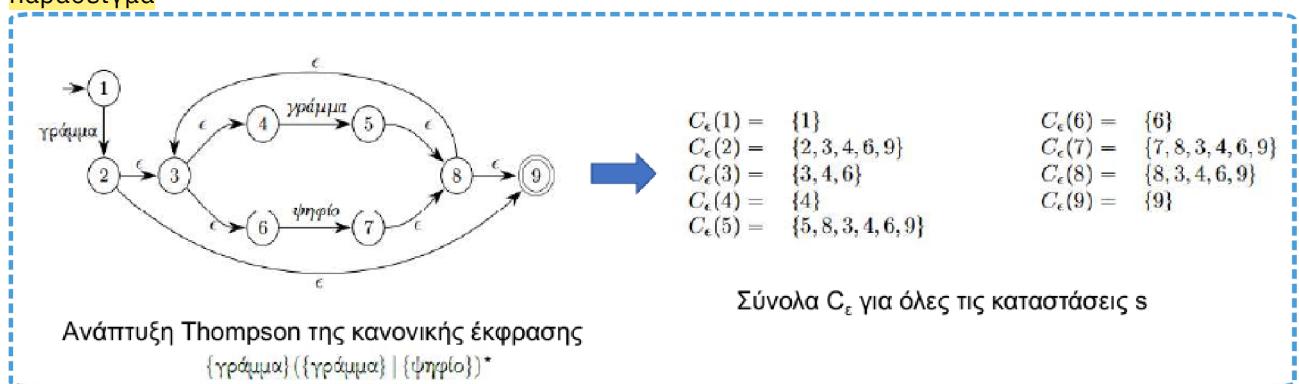
**Παράδειγμα**



## ΜΝΠΑ -> ΝΠΑ

- Βρίσκουμε την  **$\epsilon$ -Κλειστότητα** κάθε κατάστασης του ΜΝΠΑ
  - **$\epsilon$ -Κλειστότητα** της  $s$  :  $C_\epsilon(s)$  το σύνολο όλων των καταστάσεων που μπορούν να προσεγγιστούν αποκλειστικά με  $\epsilon$ -μεταβάσεις από την  $s$

**Παράδειγμα**



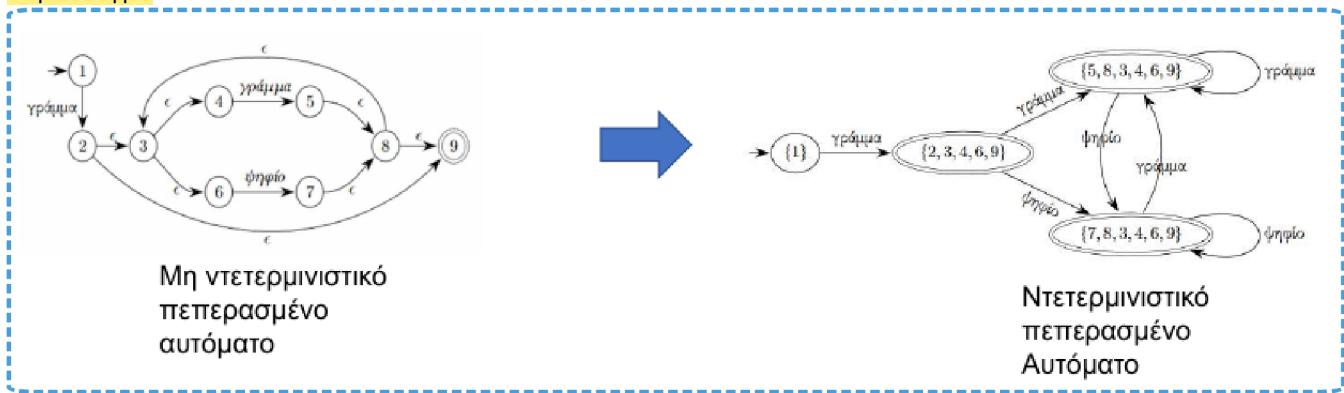
- Το νέο σύνολο καταστάσεων είναι η  **$\epsilon$ -κλειστότητα** κάθε κατάστασης του αρχικού Αυτόματου, δηλαδή σύνολο από καταστάσεις
- Η νέα **Συνάρτηση Μετάβασης** ορίζεται ως

$$\Delta(R, s) = \cup_{r \in R} C_\epsilon(\delta(r, s))$$

- Για κάθε σύνολο από καταστάσεις  $R$ , η μετάβαση του μέσω κάποιου συμβόλου  $s$ , οδηγεί στην ένωση, από τις  **$\epsilon$ -κλειστότητες**, των καταστάσεων στις οποίες οδηγούμασταν, από τις καταστάσεις του  $R$
- Η **Αρχή** του ΝΠΑ είναι η  **$\epsilon$ -κλειστότητα** της αρχής του ΜΝΠΑ

- Οι τελικές καταστάσεις του ΝΠΑ είναι όλες οι καταστάσεις που περιέχουν τελικές καταστάσεις του ΜΝΠΑ

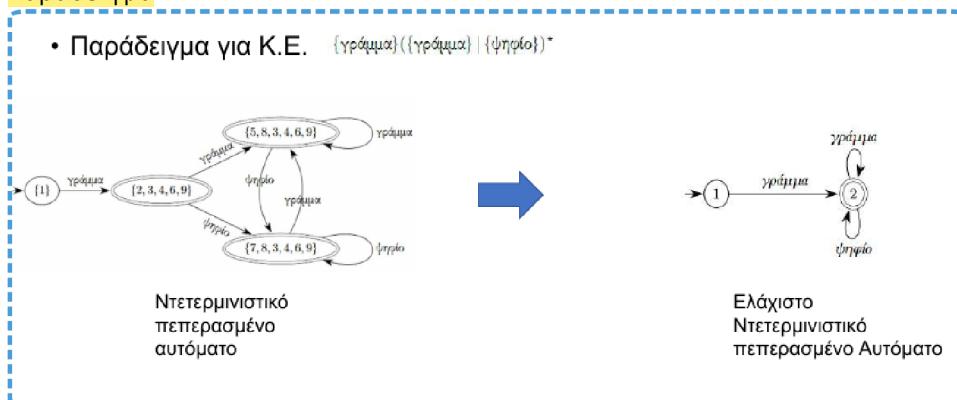
### Παράδειγμα



### ΝΠΑ -> Ελάχιστο ΝΠΑ

- Χωρίζουμε τις καταστάσεις σε δύο κλασεις  $A, B$ , όπου
  - $A$  οι μη-τελικές καταστάσεις του ΝΠΑ
  - $B$  οι τελικές καταστάσεις του ΝΠΑ
- Αναδρομικά ελέγχουμε αν σε κάθε κλάση, όλες οι μεταβάσεις, οδηγούν στην ίδια κλαση
  - δλδ αν  $\forall a \in A, \Delta(a, s) \in A \wedge \Delta(a, s) \in B, \forall s \in \Sigma$  η  $A$  είναι εντάξει
- Αν ένα υποσύνο μίας κλάσεις έχει διαφορετικά αποτελέσματα, δημιουργούμε νέα κλαση και επαναλαμβάνουμε τη διαδικασία, μέχρι κάθε κλαση να χαρακτηρίζεται από όμοιες μεταβάσεις σε άλλες κλάσεις
- Το **Ελαχιστοποιημένο ΝΠΑ** έχει μία κατάσταση για κάθε κλαση του ΝΠΑ, και η μετάβαση από κάθε κατάσταση είναι προφανής

### Παράδειγμα



### Γεννήτρια Λεξικού Αναλυτή

- Η **Γεννήτρια Λεξικού Αναλυτή** είναι ένα πρόγραμμα που δέχεται ως είσοδο ένα σύνολο από ορισμούς αναγνωριστικών σε μορφή *Κανονικών Εκφράσεων* και παράγει ως έξοδο έναν λεξικό αναλυτή
- Ένα πρόγραμμα λεξικού αναλυτή πρέπει να μπορεί να
  - Διακρίνει διάφορους τύπους αναγνωρηστικών (πχ αριθμούς, μεταβλητές, λέξεις κλειδιά, κλπ...)
  - Αποκόπτει τη σευμβολοσειρά εισόδου σε πολλά μέρη, που το καθένα ανήκει σε μία από τις γλώσσες των Κανονικών Εκφράσεων
  - Αποφασίζει με ποιό τρόπο θα προχωρήσει αν υπάρχουν πολλοί διαφορετικοί τρόποι αποκοπής της εισόδου σε επιμέρους συμβολοσειρές

# Τρόπος Λειτουργείας

- **Απλή Προσέγγιση**

- Σύνθεση ενός ΝΠΑ για κάθε ορισμό αναγνωρηστικού και προσωμοίωσή, με τη σειρά, ένα προς ένα
- Αρκετά αργή λύση. Προτιμάται η σύνθεση ενός μόνο ΝΠΑ από το σύνολο ορισμών, που προσομοιώνει ταυτόχρονα όλους τους ορισμούς αναγνωρηστικών

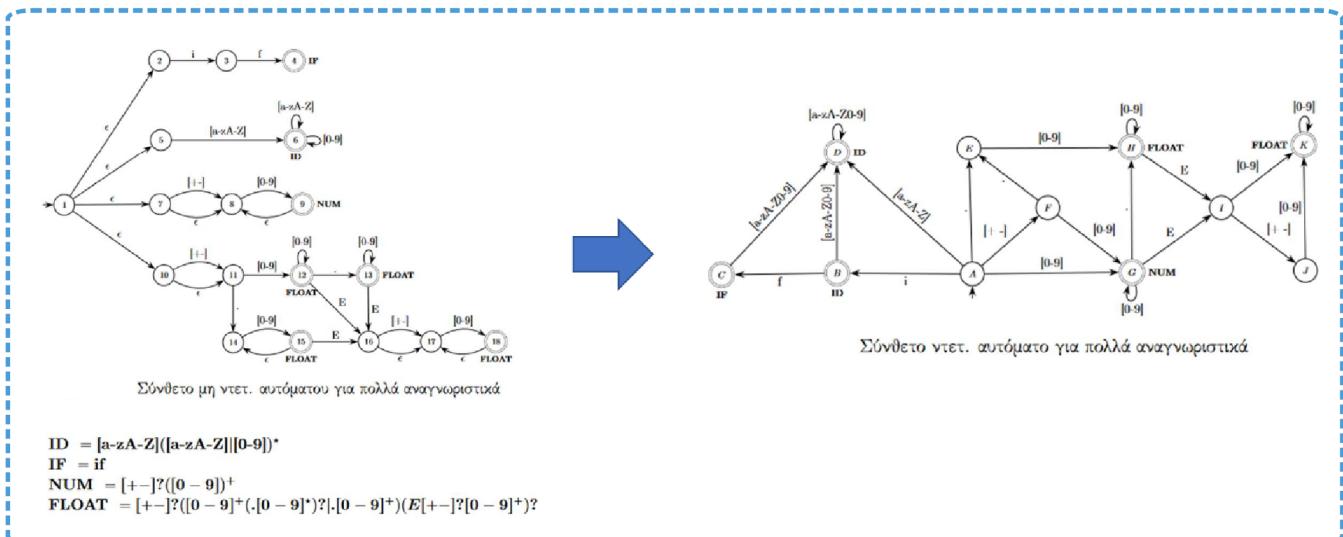
- **"Πραγματική" Προσέγγιση**

1. Δημιουργεία ΜΝΠΑ για κάθε ΚΕ αναγνωρηστικού
2. Επισημένωση των καταλήξεων κάθε ΜΝΠΑ με το όνομα του αναγνωρηστικού που δέχονται
3. Συνδιασμός των ΜΝΠΑ σε ένα ενιαίο ΜΝΠΑ, με την προσθήκη μίας νέας αρχικής κατάστασης και ε-μεταβάσεις στην αρχή κάθε επιμέρους ΜΝΠΑ
4. Μετατροπή του ενιαίου ΜΝΠΑ σε ΝΠΑ
5. Κάθε κατάληξη του ΝΠΑ θα αποτελείται από ένα σύνολο καταστάσεων του ΜΝΠΑ με τουλάχιστον μία κατάληξη από αυτές του βήματος 2. Η επισήμανση χρησιμοποιείται για την αναγνώρηση του τύπου του αναγνωρηστικού

- **Συγκρόύσεις**

- Από τη διαδικασία μετατροπής ΜΝΠΑ σε ΝΠΑ, μία κατάληξη του ΝΠΑ μπορεί να αντιστοιχεί σε πολλαπλές επισημανσένες καταλήξεις του ΜΝΠΑ. Στην περίπτωση που ο τύπος ενός αναγνωρηστικού είναι αμφίβολος, μπορούμε
  - Ο λεξικός αναλυτής να παράγει λάθος, ενώ προσπαθούμε οι ορισμοί να είναι αμοιβαίως αποκλειώμενοι, δλδ αν μην υπάρχουν συμβολοσειρές που αναγνωρίζονται με πάων από έναν τύπο
  - Ο χρήστης της γενήτριας να επιλέγει ποιό από τα αναγνωρηστικά προτιμάται
    - Συνήθως η επικάλυψη αντιμετοπίζεται συμερίζοντας την σειρά εμφάνισης των αναγνωρηστικών στην είσοδο της γενήτριας Λ.Α.

## Παράδειγμα



# Flex

## Κανονικές Εκφράσεις

Καν. έκφραση	Περιγραφή	Παράδειγμα
"..."	Αναγνωρίζεται η ακολουθία χαρακτήρων ανάμεσα στα "και "	"a+b"
[...]	Χαρακτήρας που αναφέρεται/περιλαμβάνεται στα [ και ]	[ A-Za-z0-9]
[ ^p ]	Χαρακτήρας που δεν αναφέρεται/περιλαμβάνεται στα [ και ]	[ ^ab]
.	Οποιοσδήποτε χαρακτήρας εκτός από τη νέα γραμμή	
p*	Μηδέν ή περισσότερες φορές συμβ/σειρά που εκφράζει η κ.ε. p	abc*
p q	Συμβ/σειρά που εκφράζει η κ.ε. p ή η κ.ε. q	a b
p+	Μία ή περισσότερες φορές συμβ/σειρά που εκφράζει η κ.ε. p	a(bc)+
p?	Προαιρετική εμφάνιση συμβ/σειράς που εκφράζει η κ.ε. p	a(bc)?
p{n,m}	Συμβ/σειρά που η κ.ε. p επαναλαμβάνεται από n μέχρι m φορές	a{1,5}
^p	Συμβ/σειρά της κ.ε. p μόνο μετά από αλλαγή γραμμής	
p\$	Συμβ/σειρά της p ακολουθούμενη από αλλαγή γραμμής	
p/q	Συμβ/σειρά της p ανν ακολουθείται από συμβ/σειρά της q	ab/cd
/c	Ο χαρακτήρας c χωρίς ειδική σημασία	

## Σύνταξη

```
%{
/* Κώδικας που πρέπει να περιληφθεί στη Λ.Α. */
%}
/* Ορισμοί κανονικών εκφράσεων */
%%%
/* Κανονικές εκφράσεις και ενέργειες */
    p1          { ενέργεια 1 }
    p2          { ενέργεια 2 }
    ...
    pn          { ενέργεια n }

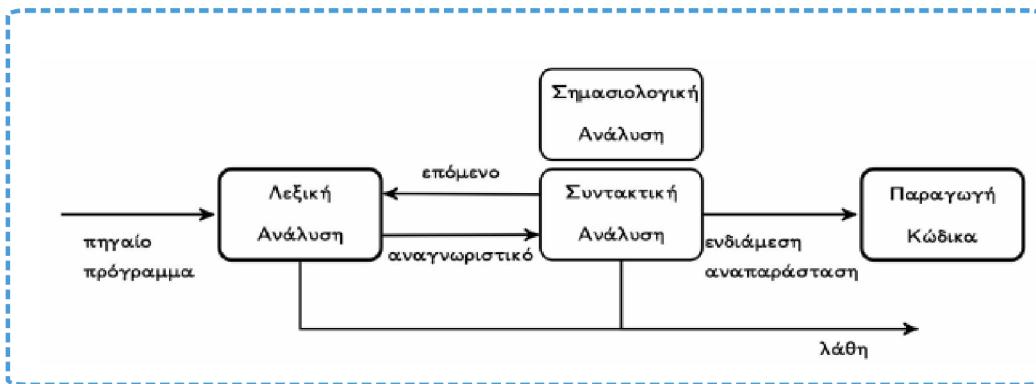
%%
/* Βοηθητικές συναρτήσεις */
```

### Παράδειγμα

```
%{
# include <stdio.h>
%
ends_with_a    .*a\n
begins_with_a  a.*\n
%%
{ends_with_a}      ECHO;
{begins_with_a}    ECHO;
.*\n;
%%
void main()
{
    yylex();
}
```

- Αν κάποια συμβολοσειρά ταυτίζεται με περισσότερους από έναν ορισμούς, , η Flex την ταυτίζει με τον νωρίτερα ορισμένο ορισμό
- Αν κάποια συμβολοσειρά παράγεται από μία Κ.Ε. και ταυτόχρονα είναι μέρος μεγαλύτερης λεξικής μονάδας, τότε η Flex την αναγνωρίζει ως μέρος της μεγαλύτερης λεξικής μονάδας

## Συντακτική Ανάλυση



**Η Συντακτική Ανάλυση - Σ.Α.** δέχεται αναγωνιστικά που συγκροτούν το πηγαίο πρόγραμμα, του οποίου τη συντακτική δομή θα αναγνωρίσει/καταγράψει

- Αν το πρόγραμμα **συμμορφώνεται** στον ορισμό σύνταξης της Γλ.Πρ., τότε η ΣΑ μπορεί να καταγράφει τη δομή του σε μορφή συντακτικού δέντρου, την ενδιάμεση αναπαράσταση (υπάρχουν και γραμμικές ενδιάμεσες αναπαραστάσεις)
- Αν το πρόγραμμα **δεν συμμορφώνεται** στον ορισμό σύνταξης της Γλ.Πρ., τότε ο αλγόριθμος ΣΑ, εντοπίζει ένα συντακτικό λάθος, και μετά εκτελεί μία διαδικασία ανάκαμψης, ώστε να συνεχίσει η ΣΑ μέχρι το τέλος του Προγράμματος
- Η ΣΑ κατά τη λειτουργεία της ενεργοποιεί τους ελέγχους που σχετίζονται με τη σημασία του προγράμματος - αντλούνται πληροφορίες από το συντακτικό δέντρο και τον πίνακα συμβόλων
- Το συντακτικό δέντρο αντικατοπτρίζει τη σειρά ανίχνευσης των αναγνωριστικών στη δομή του προγράμματος
- Μετάφραση σε ένα πέρασμα:** όταν μία μόνο διάσχιση του δέντρου αρκεί για την εκτέλεση σημασιολογικών ελέγχων και δεν αποθηκεύεται κάποια δομή δεδομένων στη μνήμη
- Μετάφραση σε πολλά περάσματα:** για πιο απαιτητικούς σημασιολογικούς ελέγχους, που αποθηκεύεται η δομή του συντακτικού δέντρου

# Ανάκαμψη από Λάθη

- Οι τεχνικές ανάκαμψης, που μπορούν να εφαρμοστούν, εξαρτώνται από τη μέθοδο ανάλυσης
- Γενικές αρχές
  - Ο εντοπισμός λάθους είναι **αποτελεσματικός**, όταν γίνεται **έγκαιρα**. Προτιμώνται αλγόριθμοι με μηχανισμούς πρόβλεψης του κανόνα που θα εφαρμοστεί
  - Ο μεταγλωττιστής θα πρέπει να εντοπίζει **πραγματικά λάθη** και όχι υποτιθέμενα λάθη εξαιτίας της διαδικασίας ανάκαμψης από το προηγούμενο λάθος (**διαδιδόμενα λάθη**)
  - Μετά το εντοπισμό λάθους πρέπει να επιλέγεται σημείο συνέχισης της ανάλυσης, δηλαδή εκεί που **ξαναενεργοποιείται** η δυνατότητα εντοπισμού λαθών
  - Θα πρέπει να αποκλείεται η πιθανότητα μετάπτωσης του αναλυτή σε **αέναους βρόγχους** επανάληψης, που δεν επιτρέπουν τη συνέχιση της ανάγνωσης του πηγαίου προγράμματος

## Στρατηγικές Ανάκαμψης από Λάθη

- Διακρίνουμ 4 βασικές στρατηγικές ανάκαμψης
  1. Ανάκαμψη σε κατάσταση "**Πανικού**"  
Μετά από τον εντοπισμό λάθους "αγνωούνται" λεξικές μονάδες, μέχρι να βρεθεί μία από την οποία μπορεί να συνεχίσει με ασφάλεια την ανάλυση (μονάδες συγχρονισμού πχ. επιλεγμένοι διαχωριστές)  
Πλεονεκτίματα: απουσία αέναων βρόγχων και διαδιδόμενων λαθών, για όσο αγνούνται λεξικές μονάδες μετά τον εντοπισμό λάθους
  2. Ανάκαμψη σε **επίπεδο φράσης**  
Μπορεί να αντικαθίσταται, μετά από λάθος, ένα πρόθεμα της εναπομείνασας εισόδου, ώστε να μπορεί να συνεχίσει την ανάλυση πχ. Αντικατάσταση κόμματος με ερωτηματικό, εισαγωγή ερωτηματικού εκεί που πρέπει να υπάρχει  
Χρειάζεται προσοχή στην επιλογή των επιτρεπτών διορθώσεων, ώστε να αποφεύγονται οι αέναιοι βρόγχοι. Δεν είναι πάντα εύκολη η εφαρμογή της
  3. **Κανόνες παραγωγής για λάθη:**  
Αν υπάρχει δυνατότητα πρόβλεψης των λαθών, μπορούν να διατυπωθούν επιπλέον κανόνες παραγωγής, που καθοδηγούν την ανάλυση, όταν αναγνωρίζονται αυτά τα λάθη  
Κάθε φορά που χρησιμοποιείται από τον αναλυτή ένας τέτοιος κανόνας, παράγεται και κατάλληλο διαγνωστικό μήνυμα
  4. **Συνολική Διόρθωση**  
Η υλοποίησή της επιπτυγχάνεται με την εφαρμογή κατάλληλου αλγορίθμου διόρθωσης λαθών, με όσο το δυνατόν μικρότερο αριθμό αλλαγών. Αυτοί οι αλγόριθμοι έχουν αποδειχθεί πολύ δαπανηροί στην πράξη, και είναι περισσότερο θεωρητικού, παρά πρακτικού ενδιαφέροντος

# Ορισμός Σύνταξης Γλώσσας Προγραμματισμού

- Για τον ορισμό της συντακτικής δομής των γλωσσών προγραμματισμού, χρησιμοποιούνται Γραμματικές Χωρίς Συμφρεζόμενα

- Τερματικά σύμβολα είναι τα αναγνωρηστικά (γράφονται με **bold**)
- Μη τερματικά σύμβολα εκφράζουν τις συντακτικές δομές της γλώσσας (assig-stmt, expr, term...) (γράφονται με *italics*)
- Η Αρχή της ΓΧΣ αντιστοιχεί σε ένα οποιοδήποτε πρόγραμμα (σλυμβολο στο αριστερό μέρος του πρώτου κανόνα)

Παράδειγμα

$assign-stmt \rightarrow var := expr$	(1)
$expr \rightarrow term$	(2)
$expr + term$	(3)
$expr - term$	(4)
$term \rightarrow prim Expr$	(5)
$term \times prim Expr$	(6)
$term / prim Expr$	(7)
$prim Expr \rightarrow var$	(8)
<b>NUM</b>	(9)
$( expr )$	(10)
$var \rightarrow ID$	(11)
$ID [sbscr-lst]$	(12)
$sbscr-lst \rightarrow expr$	(13)
$sbscr-lst, expr$	(14)

# Κατηγοριοποίηση Γραμματικών

## Το σύνολο FIRST

- Αν  $\bar{u}$  είναι συμβολοσειρά που αποτελείται από μη-τερματικά ή/και τερματικά σύμβολα μίας γραμματικής, τότε

$$FIRST(\bar{u}) = \{t | \bar{u} \Rightarrow^* t\bar{o}\}$$

το σύνολο των  $t$  (τερματικών συμβόλων ή  $\epsilon$ ) που μπορεί να εμφανίζονται στην αρχή όλων των συμβολοσειρών, που παράγονται από τη  $\bar{u}$  μέσω κανόνων

## Αλγόριθμος Υπολογισμού FIRST

- Είσοδος: Γραμματική  $G = (N, T, S, P)$  και μία προτασιακή μορφή  $\bar{u}$

- Έξοδος: το σύνολο  $FIRST(\bar{u})$

```

1: if ( $\bar{u} = x_1 \in N \cup T \cup \{\epsilon\}$ ) then
2:   για  $x_1 \in T$ , έχουμε  $FIRST(\bar{u}) = \{x_1\}$ 
3:   για  $x_1 = \epsilon$ , έχουμε  $FIRST(\bar{u}) = \{\epsilon\}$ 
4:   για  $x_1 = X \in N$ , με  $X \rightarrow \bar{u}_1 | \bar{u}_2 | \dots | \bar{u}_k$ 
      έχουμε  $FIRST(\bar{u}) = FIRST(\bar{u}_1) \cup FIRST(\bar{u}_2) \cup \dots \cup FIRST(\bar{u}_k)$ 
5: if ( $\bar{u} = x_1 x_2 \dots x_n$ , με  $x_i \in N \cup T$ ) then
6:    $FIRST(\bar{u}) = \{\}$ ; j=0;
7:   repeat
8:     j=j+1;
9:     προσθέτουμε στο  $FIRST(\bar{u})$  το  $FIRST(x_j) - \{\epsilon\}$ ;
10:    until  $x_j$  δεν είναι απαλείψιμο (δεν παράγει το  $\epsilon$ ) ή  $j = n$ 
11:    if (η  $x_1 x_2 \dots x_n$  είναι απαλείψιμη) then
12:      προσθέτουμε το  $\epsilon$  στο  $FIRST(\bar{u})$ 

```

- Αν το  $\bar{u}$  είναι τερματικό ή  $\epsilon$ , το  $FIRST(\bar{u}) = \{\bar{u}\}$
- Αν το  $\bar{u}$  είναι μη-τερματικό, το  $FIRST(\bar{u})$  είναι η ένωση των  $FIRST$  όλων των συμβολοσειρών που μπορούν να παραχθούν με έναν κανόνα από το  $\bar{u}$
- Αν το  $\bar{u}$  είναι σύζευξη από τερματικά ή/και μη τερματικά  $x_1 x_2 \dots x_n$ 
  - Παίρνουμε με τη σειρά κάθε επιμέρους σύμβολο  $x_j$  και προσθέτουμε στο  $FIRST(\bar{u})$ , το  $FIRST(x_j) - \{\epsilon\}$
  - Μέχρι να βρούμε  $x_j$  που να μην παράγει το  $\epsilon$ , ή να τελειώσουν τα σύμβολα
  - Αν η  $\bar{u}$  παράγει το  $\epsilon$ , το προσθέτουμε στη  $FIRST(\bar{u})$

## Το Σύνολο FOLLOW

- Αν  $X$  ένα μη-τερματικό σύμβολο, τότε  $FOLLOW(X)$  το σύνολο των τερματικών συμβόλων που μπορεί να εμφανιστούν αμέσως μετά από αυτό, σε όλες τις περιπτώσεις παραγώμενων συμβολοσειρών

## Αλγόριθμος Υπολογισμού

- Είσοδος: Γραμματική  $G = (N, T, S, P)$  και ένα μη-τερματικό  $X$
- Έξοδος: το σύνολο  $FOLLOW(X)$

```

1: if ( $X = S$ ) then
2:   $ ∈ FOLLOW(X), όπου $ συμβολίζει το τέλος συμβολοσειρών
3:   for all  $p \in P$ , κανόνας  $Y \rightarrow \bar{m}X\bar{o}$ , δηλ. με  $X$  στο δεξί του μέρος do
4:     if ( $\bar{o}$  αρχίζει από  $c \in T$ ) then
5:        $c \in FOLLOW(X)$ 
6:     if ( $\bar{o}$  αρχίζει από ένα  $Z \in N$ ) then
7:       προσθέτουμε στο FOLLOW(X) το FIRST( $\bar{o}$ ) - { $\epsilon$ };
8:     if ( $\bar{o} = \epsilon$  ή  $\bar{o} \Rightarrow^* \epsilon$ ) then
9:       προσθέτουμε στο FOLLOW(X) το FOLLOW(Y);

```

- Αν  $X$  είναι η αρχή της γραμματικής, τότε βάζουμε το  $$$  στο  $FOLLOW(X)$
- Για κάθε κανόνα της γραμματικής, μορφής  $Y \rightarrow \bar{m}X\bar{o}$  (με το  $X$  στο δεξί του μέρος)
  - αν το  $\bar{o}$  αρχίζει με τερματικό, βάλε αυτό το τερματικό στο  $FOLLOW(X)$
  - αν το  $\bar{o}$  αρχίζει με μη-τερματικό, βάλε το  $FIRST(\bar{o}) - \{\epsilon\}$  στο  $FOLLOW(X)$
  - αν το  $\bar{o}$  έιναι ή παράγει  $\epsilon$ , βάλε το  $FOLLOW(Y)$  στο  $FOLLOW(X)$

### Παράδειγμα

## Σύνολα FIRST και FOLLOW

$E$	$\rightarrow TT_r$	$FIRST(E) = \{‘, “αριθμός”\}$
$T_r$	$\rightarrow ‘+’ TT_r$	$FIRST(T) = \{‘, “αριθμός”\}$
	$  ‘-’ TT_r$	$FIRST(F) = \{‘, “αριθμός”\}$
	$  \epsilon$	$FIRST(T_r) = \{‘+’, ‘-’, \epsilon\}$
$T$	$\rightarrow FF_r$	$FIRST(F_r) = \{*, ‘/’, \epsilon\}$
$F_r$	$\rightarrow ‘*’ FF_r$	Υποδεικνύει το τέλος του προγράμματος.
	$  ‘/’ FF_r$	$FOLLOW(E) = \{$, ‘)’$\}$
	$  \epsilon$	$FOLLOW(T) = \{‘+’, ‘-’, $, ‘)’$\}$
$F$	$\rightarrow ‘(’ E ‘)’$	$FOLLOW(T_r) = \{$, ‘)’$\}$
	$  “αριθμός”$	$FOLLOW(F) = \{*, ‘/’, $, ‘)’, ‘+’, ‘-’, $, ‘)’$\}$
		$FOLLOW(F_r) = \{‘+’, ‘-’, $, ‘)’$\}$

19

## Γραμματικές LL(1)

- Μία γραμματική είναι  $LL(1)$  αν-ν
  - Για κάθε κανόνα  $X \rightarrow \overline{u_1}|\overline{u_2}| \dots |\overline{u_n}$  ισχύει

$$FIRST(\overline{u_i}) \cap FIRST(\overline{u_j}) = \emptyset$$

για όλα τα  $i, j = 1 \dots n$  με  $i \neq j$

- Για κάθε μη-τερματικό  $X$ , τέτοιο ώστε  $\epsilon \in FIRST(X)$  ισχύει

$$FIRST(X) \cap FOLLOW(X) = \emptyset$$

# Απομάκρυνση Αριστερής Αναδρομικότητας

## Άμεση Αριστερή Αναδρομικότητα

- Πρόβλημα με κανόνα παραγωγής του τύπου

$$X \rightarrow X\bar{u}_1|X\bar{u}_2|\dots|X\bar{u}_n|\bar{o}_1|\bar{o}_2|\dots|\bar{o}_m$$

- $\bar{u}_i$ : συμβολοσειρές από τερματικά ή/και μη-τερματικά
- $\bar{o}_j$ : συμβολοσειρές από τερματικά ή/και μη-τερματικά που δεν ξεκινούν από  $X$
- Η λύση δίνεται με την μετατροπή της αριστερής αναδρομής σε δεξιά ως εξής:

$$X \rightarrow \bar{o}_1 X' |\bar{o}_2 X' | \dots |\bar{o}_m X'$$

$$X' \rightarrow \bar{u}_1 X' |\bar{u}_2 X' | \dots |\bar{u}_n X' |\varepsilon$$

- όπου  $X'$  ένα νέο μη-τερματικό σύμβολο
- Η παραπάνω αντικατάσταση δεν επηρεάζει τη γλώσσα της γραμματικής

## Έμμεση Αριστερή Αναδρομικότητα

- Πρόβλημα με κανόνες παραγωγής του τύπου

$$X \rightarrow Y\bar{u}_2|\dots$$

$$Y \rightarrow X\bar{u}_1|\dots$$

- Για να εφαρμοστεί ο αναγκαίος μετασχηματισμός, προϋποθέτει η γραμματική
  - Να μην περιέχει κυκλικούς κανόνες παραγωγής
  - πχ  $M \rightarrow Q, Q \rightarrow R, R \rightarrow M$
  - Να μην περιέχει κανόνες-ε
  - Σύμφωνα με θεώρημα, όταν στη γραμματική υπάρχουν κανόνες-ε, είναι γενικά δυνατή η απομάκρυνσή τους με διατύπωση ισοδύναμης γραμματικής

**Είσοδος:** γραμματική  $G$  χωρίς κυκλικούς κανόνες παραγωγής και κανόνες-ε

**Έξοδος:** ισοδύναμη γραμματική με την  $G$  χωρίς αριστερή αναδρομή

- 1: έστω τα μη τερματικά  $X_1, X_2, \dots, X_n$  της  $G$  με δείκτες που αντιστοιχούν στη σειρά εμφάνισης τους κανόνα αντικατάστασης τους στη γραμματική
- 2: **for**  $i:=1$  **to**  $n$  **do**
- 3:   **for**  $j:=1$  **to**  $i-1$  **do**
- 4:     αντικατέστησε κάθε κανόνα της μορφής  

$$X_i \rightarrow X_j \bar{o}$$
 με τους κανόνες  

$$X_i \rightarrow \bar{u}_1 \bar{o} \mid \bar{u}_2 \bar{o} \mid \dots \mid \bar{u}_k \bar{o}$$
 για όλους τους κανόνες της μορφής  

$$X_j \rightarrow \bar{u}_1 \mid \bar{u}_2 \mid \dots \mid \bar{u}_k$$
- 5: απομάκρυνε την άμεση αριστερή αναδρομή στους κανόνες για το  $X_i$ ;

- Πάρε **ΜΕ ΤΗ ΣΕΙΡΑ** τα μη-τερματικά και

- Έλεγχε αν έχουν κανόνες, που το δεξιά τους μέρος, ξεκινά με μη-τερματικό, που έχει ήδη ελεγχθεί
- Αντικατέστησε κάθε κανόνα της μορφής

$$X_i \rightarrow X_j \bar{o}$$

με τους κανόνες

$$X_i \rightarrow \bar{u}_1 \bar{o} \mid \bar{u}_2 \bar{o} \mid \dots \mid \bar{u}_k \bar{o}$$

για κάθε κανόνα (του προηγούμενου μη-τερματικού) της μορφής

$$X_j \rightarrow \bar{u}_1 \mid \bar{u}_2 \mid \dots \mid \bar{u}_k$$

- Απομάκρυνε την άμεση αριστερή αναδρομή για τους κανόνες του  $X_i$

### Παράδειγμα

#### Απομάκρυνση έμμεσης και άμεσης αριστερής αναδρομής

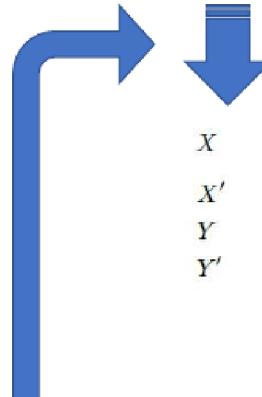
$$\begin{array}{l} X \rightarrow Y "p" \mid X "p" \mid "r" \\ Y \rightarrow Y "q" \mid X "q" \mid "s" \end{array}$$

 Απομάκρυνση άμεσης αριστερής αναδρομής κανόνα  $X \rightarrow X "p"$

$$\begin{array}{l} X \rightarrow Y "p" X' \mid "r" X' \\ X' \rightarrow "p" X' \mid \epsilon \\ Y \rightarrow Y "q" \mid X "q" \mid "s" \end{array}$$

 Απομάκρυνση έμμεσης αριστερής αναδρομής κανόνα  $Y \rightarrow X "q"$

$$\begin{array}{l} X \rightarrow Y "p" X' \mid "r" X' \\ X' \rightarrow "p" X' \mid \epsilon \\ Y \rightarrow Y "q" \mid Y "p" X' "q" \mid "r" X' "q" \mid "s" \end{array}$$



 Απομάκρυνση άμεσης αριστερής αναδρομής κανόνα  $Y \rightarrow Y "q"$  και  $Y \rightarrow Y "p" X' "q"$

$$\begin{array}{l} X \rightarrow Y "p" X' \mid "r" X' \\ X' \rightarrow "p" X' \mid \epsilon \\ Y \rightarrow "r" X' "q" Y' \mid "s" Y' \\ Y' \rightarrow "q" Y' \mid "p" X' "q" Y' \mid \epsilon \end{array}$$

7

### Μετασχηματισμός Γραμματικής σε LL(1) - Αριστερή Παραγοντοποίηση

- Υπάρχουν γραμματικές που εμφανίζουν σε έναν τουλάχιστον κανόνα της γραμματικής, δύο ή περισσότερες εναλλακτικές με το ίδιο πρόθεμα. Για να εκτελέσουμε προβλέπουσα ανάλυση, αυτοί οι κανόνες πρέπει να εξαλειφθούν

#### Αριστερή Παραγοντοποίηση Παράδειγμα

- Η

$$\begin{array}{l} C \rightarrow 'if' E 'then' C \\ \mid 'if' E 'then' C 'else' C \\ \mid '\text{άλλο}' \end{array}$$

- Γίνεται

$$\begin{array}{l} C \rightarrow 'if' E 'then' C C' \\ \mid '\text{άλλο}' \\ C' \rightarrow 'else' C \\ \mid \epsilon \end{array}$$

- δλδ Βρίσκουμε τα δεξιά μέρη με το κοινό πρόθεμα και τα αποσπούμε, αφήνοντας μόνο το κοινό πρόθεμα στον αρχικό κανόνα, και δημιουργόντας νέο μη-τερματικό για κάθε διαφορετικό επίθεμα, το οποίο μπορεί να γίνει είτε το επίθεμα, είτε  $\epsilon$

### Γραμματικές LR(1)

Για να είναι μία γραμματική  $LR(1)$  πρέπει:

1. Αν υπάρχει στοιχείο της μορφής  $[Y \rightarrow \bar{u} \cdot k\bar{o}, a] \in I_i$  στο σύνολο  $I_i$  για κάποιο τερματικό  $k$ , τότε δεν θα υπάρχει στο ίδιο σύνολο το συμπληρωμένο στοιχείο  $[X \rightarrow \bar{e}\cdot, k] \in I_i$  (shift-reduce conflict)
2. Ένα σύνολο στοιχείων  $I_i$  δεν μπορεί να περιέχει πανω από ένα συμπληρωμένα στοιχεία της μορφής  $[X \rightarrow \bar{e}\cdot, a]$  και  $[Z \rightarrow \bar{w}\cdot, a]$  (reduce-reduce conflict)

## Γραμματικές SLR(1)

- Οι γραμματικές  $SLR(1)$  είναι και (κανονική)  $LR(1)$  -  $SLR(1) \subset LR(1)$ , αλλά η ανάλυση  $LR(1)$  μπορεί να έχει περισσότερες καταστάσεις από την  $SLR(1)$
- Οι γραμματικές  $LR(1)$  δεν είναι απαραίτητα και  $SLR(1)$

## Αλγόριθμοι Συντακτικής Ανάλυσης

### • Καθοδική Συντακτική Ανάλυση

- *Ρίζα* (αρχή γραμματικής) προς φύλλα (αναγνωριστικά τερματικών)
- Προϋποθέτει μη-αριστερά αναδρομική γραμματική
- Αντιστοιχεί σε αριστερές παραγωγές της γραμματικής

### • Ανοδική Συντακτική Ανάλυση

- Φύλλα (αναγνωριστικά τερματικών) προς φύλλα (αρχή γραμματικής)
- Εφαρμόζεται σε ευρύτερη οικογένεια γραμματικών
- Αντιστοιχεί σε δεξιές παραγωγές της γραμματικής

### • Αλγόριθμοι Οπισθοδρόμησης

- Επιλεγεται εξ αρχής σειρά εφάρμοσης κανόνων γραμματικής και αν η ανάλυση φτάσει σε σημείο που δεν μπορεί να προχωρήσει
  - Ανατρέπεται ένας ή περισσότεροι κανόνες και εφαρμόζονται διαφορετικοί για τα ίδια σύμβολα
  - Υψηλό υπολογιστικό κόστος
  - Καθυστερημένη ανίχνευση λαθών
  - **DFS εφαρμογή κανόνων γραμματικής**

### • Αλγόριθμοι Πρόγνωσης

- Διαβάζονται ένα ή περισσότερα τερματικά σύμβολα και
  - με βάση αυτά αποφασίζεται ποιοός κανόνας θα χρησιμοποιηθεί, από αυτούς που αναφέρονται σε ένα συγκεκριμένο μη-τερματικό
- Συγκριτικά αποδοτικότερη μεταγλώττιση και πιο απλή ανάκαμψη από λάθη

## Ονοματολογία Αλγορίθμων Συντακτικής Ανάλυσης

Ένας αλγόριθμος ΣΑ παίρνει όνομα της μορφής

$$XY(n)$$

### • Όπου

- $X$  αναφέρεται στη φορά ανάγνωσης των συμβόλων, που αντιστοιχούν στα αναγνωριστικά συμβολοσειρών του πηγαίου προγράμματος
  - $X = L$  δηλώνει πως η είσοδος αναγνωριστικών στον αλγόριθμο γίνεται από τα **αριστερά προς τα δεξιά**
- $Y$  αναφέρεται στον τύπο παραγών που ακολουθεί ο αλγόριθμος
  - $Y = L$  δηλώνει **αριστερές παραγωγές** (χαρακτηριστικό της **καθοδικής ανάλυσης**)
  - $Y = R$  δηλώνει **δεξιές παραγωγές** (χαρακτηριστικό της **ανοδικής ανάλυσης**)
- $n$  αναφέρεται στον **αριθμό τερματικών συμβόλων** που διαβάζονται για να **αποφασιστεί** ο κανόνας της γραμματικής που θα χρησιμοποιηθεί

# Καθοδική ΣΑ

## Καθοδική ΣΑ με Οπισθοδρόμηση

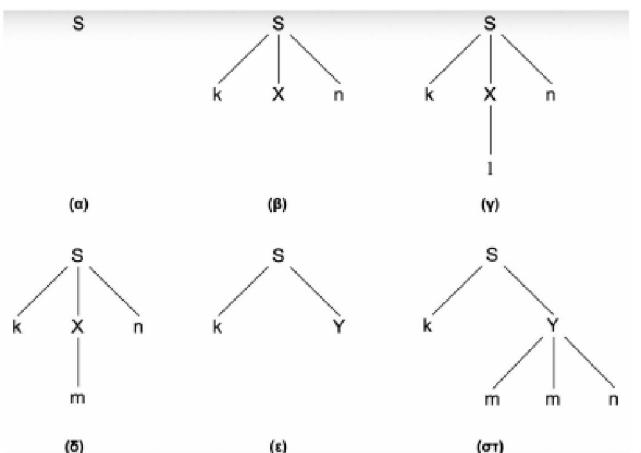
- Για ένα τερματικό σύμβολο, εφαρμόζεται ο πρώτος κανόνας που αναφέρεται σε αυτό
- Στη συμβολοσειρά που προκύπτει επιλέγεται το πρώτο από τα αριστερά μη τερματικό και εφαρμόζεται ο πρώτος κανόνας που αναφέρεται σε αυτό
- Επαναλαμβάνουμε το βήμα 2, για κάθε μη τερματικό σύμβολο που ακολουθεί μέχρι να παραχθεί
  - μία σειρά που αντιστοιχεί στη συμβολοσειρά εισόδου
  - μία σειρά τερματικών που διαφέρει από το αντίστοιχο τμήμα της εισόδου
 Όταν συμβεί αυτό αναφέρεται ο τελευταίος κανόνας που εφαρμόστηκε και επιλέγεται ο επόμενος για το αντίστοιχο μη τερματικό  
 Αν εξαντληθούν οι κανόνες για το συγκεκριμένο μη τερματικό, αναφέρεται ο επόμενος πιο πρόσφατος κανόνας  
 Οι αναφέσεις συνεχίζουν μέχρι να μπορεί να συνεχίσει η ανάλυση, ή να επιστρέψουμε στην αρχή της γραμματικής, που σημαίνει πως βρέθηκε συντακτικό λάθος
- Στην ουσία είναι **DFS** αναζήτηση για τη σωστή παραγωγή μίας συμβολοσειράς

### Παράδειγμα

## Καθοδική ΣΑ με Οπισθοδρόμηση

$S$	$\rightarrow$	"k" $X$ "n"
		"k" $Y$
$X$	$\rightarrow$	"l"         "m"
$Y$	$\rightarrow$	"mmn"
		"nnm"

- Παραγωγή του προγράμματος: **kmmn**
- $S \xrightarrow{(\beta)} kXn \xrightarrow{(\gamma)} kln$  αναίρεση τελευταίου κανόνα  
 $S \xrightarrow{(\beta)} kXn \xrightarrow{(\delta)} kmn$  αναίρεση προτελευταίου κανόνα  
 $S \xrightarrow{(\varepsilon)} kY \xrightarrow{(\sigma\tau)} kmmn$



11

- Εξαιρετικά αργός
- Ανίχνευση λάθους όταν έχει δοκιμαστεί κάθε πιθανή παραγωγή
- Τα περισσότερα (αν όχι όλα) χαρακτηριστικά που μπορεί να είναι απιθυμητά σε μία γλώσσα προγραμματισμού, μπορούν να περιγραφούν από γραμματικές με απαιτήσεις περιορισμένης ή ανύπαρκτης οπισθοδρόμησης

## Καθοδική ΣΑ Προβλεπουσας Αναδρομικής Κατάβασης

- Κάθε κανόνας της γραμματικής, για ένα μη τερματικό σύμβολο, εκφράζεται από μία διαδικασία (*function*)
  - Το αριστερό μέρος είναι το όνομα, το δεξί καθορίζει τον κώδικα
- Μία σειρά συμβόλων εκφράζεται από την κατά σειρά κλήση των αντίστοιχων συναρτήσεων
- Η ΣΑ προβλέπουσας αναδρομικής κατάβασης αποτελείται από
  - Μία καθολική μεταβλητή, με την τιμή του αναγνωριστικού που διαβάστηκε πιο πρόσφατα
  - Μία διαδικασία που ελέγχει αν το τρέχον αναγνωριστικό είναι ίδιο με κάποιο αναμενόμενο, και αν είναι καλεί τη διαδικασία ανάγνωσης του επόμενου αναγνωριστικού, που ενημερώνει την καθολική μεταβλητή
  - Τις διαδικασίες ανάλυσης για μη τερματικά σύμβολα της γραμματικής
  - Την κύρια συνάρτηση, που διαβάζει το πρώτο αναγνωριστικό, και έπειτα καλεί τη διαδικασία ανάλυσης για το μη τερματικό της αρχής της γραμματικής

### Παράδειγμα

Έστω η ΓΧΣ:

$$\begin{aligned} E &\rightarrow TT_r \\ T_r &\rightarrow '+' TT_r \\ &| '-' TT_r \\ &| \varepsilon \\ T &\rightarrow FF_r \\ F_r &\rightarrow '*' FF_r \\ &| '\' FF_r \\ &| \varepsilon \\ F &\rightarrow ('E')' \\ &| \text{'αριθμός'} \end{aligned}$$

Αυτό μεταφράζεται σε κώδικα

```
// Ena gia kathe tipo anagnwristikou
typedef enum {PLUS,MINUS,MULT,DIV,LPAR,RPAR,NUMBER,END_FILE} TokenType;
// H katholiki metavlth epomenou anagnoristikou
TokenType token;

void error() {
    printf("Syntax Error \n");
    exit(0);
}
//Synarthsh elegxou
void match(TokenType expexted_token) {
    if (token == expected_token)
        token = getToken(); //Lexikos analyths
    else
        error();
}

//Synarthseis analyhs mh termatikwn
void E();

void F() { // F →
    switch(token){
        case LPAR: //('E')
            token = getToken();
        case NUMBER:
            token = getToken();
        case END_FILE:
            token = getToken();
    }
}
```

```
        E();
        match(RPAR);
        break;
    case NUMBER: // 'arithmos'
        token = getToken();
        break;
    default:
        error();
}
}

void Fr() { //Fr →
    switch(token){
        case MULT: // '*'FFr
            token = getToken();
            F();
            Fr();
            break;
        case DIV: // '/'FFr
            token = getToken();
            F();
            Fr();
            break;
        default: // ε
            return;
    }
}

void T() { //T → FFr
    F();
    Fr();
}

void Tr() { //Tr →
    switch(token) {
        case PLUS: // '+'TTr
            token = getToken();
            T();
            Tr();
        case MINUS: // '-'TTr
            token = getToken();
            T();
            Tr();
        default: // ε
            return;
    }
}

void E() { //E → TTr
    T();
    Tr();
}
```

```

void main() {
    token = getToken();
    E();
    if (Token != END_FILE)
        return error();
}

```

- Όπως όλοι οι αλγόριθμοι καθοδικής ανάλυσης, δεν εφαρμόζεται σε αριστερά αναδρομικές γραμματικές
- χρειάζεται μετασχηματισμός της γραμματικής σε μη-αριστερά αναδρομική
- Η προβλεπουσα αναδρομική κατάβαση πρέπει να επιλέγει σε κάθε βήμα τον σωστό κανόνα μεταξύ των πιθανών επριπτώσεων (έτσι ώστε να αποφέυγεται η οπισθοδρόμηση)
  - η επιλογή του κανόνα γίνεται με βάση το τερματικό σύμβολο που βρίσκεται σε κάθε θέση
- Απαραίτητη προϋπόθεση για να εφαρμοστεί, είναι η γραμματική να είναι  $LL(1)$
- Αν δεν είναι τότε πρέπει να μετασχηματιστέσθετε σε  $LL(1)$
- Η γενική μορφή μιας συνάρτησης ανάλυσης είναι η εξής

Αν για το μη τερματικό σύμβολο  $X$  ορίζεται στη γραμματική ο κανόνας

$$X \rightarrow \bar{u}_1 | \bar{u}_2 | \dots | \bar{u}_k$$

και αν υποθέσουμε ότι η  $\bar{u}_i$  είναι απολείψιμη, τότε η διαδικασία που αντιστοιχεί στο μη τερματικό σύμβολο  $X$  έχει την μορφή:

```

1: X():
2: switch (token)
3: case c ∈ FIRST(ū₁):
4:   κλήση διαδικασιών για τα σύμβολα του ū₁
5: case c ∈ FIRST(ū₂):
6:   κλήση διαδικασιών για τα σύμβολα του ū₂
...
7: case c ∈ FIRST(ūᵢ) ∪ FOLLOW(X):
8:   κλήση διαδικασιών για τα σύμβολα του ūᵢ ή (αντίστοιχα) των διαδικασιών για τα σύμβολα που ακολουθούν μετά το X
9: default:
10: λάθος;
11: end switch
12: Τέλος X.

```

- Σε γενικούς όρους δουλεύει ως εξής
  - Όταν εκτελείται η διαδικασία, για ένα μη-τερματικό  $X$ , ελέγχεται αν το τρέχον αναγνωριστικό εμφανίζεται σε ένα  $FIRST$  για το δεξί μέρος κάποιου κανόνα του  $X$
  - Αν εμφανίζεται, τότε εφαρμόζεται αυτός ο κανόνας
  - Διαφορετικά υπάρχει συντακτικό λάθος, εκτός και αν περιέχεται παραλέιψιμη συμβολοσειρά στο δεξί μέρος του  $X$ , οπότε ελέγχεται αν το αναγνωριστικό βρίσκεται στο  $FOLLOW(X)$
- Στην προβλεπουσα αναδρομική κατάβαση, τα συντακτικά λάθη εντοπίζονται άμεσα, και αυτό διευκολύνει την ανάκαμψη, που είναι πιο κατάλληλη για κάθε περίπτωση
  - Όμως η απαίτηση να μην είναι αριστερά αναδρομική η γραμματική, αποκλειει την επιθυμητή ιδιότητα της αριστερής προσεταιριστικότητας
  - Το πρόβλημα αντιμετωπίζεται με προσεκτική σχεδίαση της αλληλεπίδρασης συναρτήσεων, που αντιστοιχούν στα μη τερματικά σύμβολα της γραμματικής

## Καθοδική ΣΑ Προβλεπουσας Αναδρομικής Κατάβασης με Ανάκαμψη σε Κατάσταση "Πανικού"

- Μετά από εντοπισμό λάθους, "αγνοούνται" λεξικές μονάδες μέχρι να βρεθεί μία από την οποία μπορεί να συνεχίσει με ασφάλεια η ανάλυση (**λεξικές μονάδες συγχρονισμού**)
- Πρέπει να επιλεγούν οι κατάλληλες λεξικές μονάδες συγχρονισμού για κάθε πιθανή κατάσταση από την οποία μπορεί να περάσει η ανάλυση
  - Υποψήφιες είναι οι λεξικές μονάδες του συνόλου *FOLLOW* του μη-τερματικού, στο δεξί μέρος ενός κανόνα παραγωγής
- Ιδιέταιρα χρήσιμα είναι και τα σύνολα *FIRST*, που λαμβάνονται υπόψη για να αποτρέπεται η προσπέραση λεξικών μονάδων με καίρια σημασία, εξαιτίας πιθανής απουσίας του τερματικού συμβόλου που αναμένεται (όπως το άνοιγμα μίας έφρασης, όταν λείπει το ";")

### Παράδειγμα

#### Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη σε κατάσταση «πανικού»

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#define MAXTOKENLEN 40
typedef enum {
    PLUS, MINUS,
    MULT, DIV,
    LPAR, RPAR,
    NUMBER, END_FILE
} TokenType;

TokenType token;
char tokenString[MAXTOKENLEN+1];
struct stoken_set {
    TokenType stoken;
    struct stoken_set *next;
    struct stoken_set *prev;
};
typedef struct stoken_set *STOKEN;
```

**H ignore προσπερνά έναν αριθμό λεξικών μονάδων μέχρι να διαβαστεί μια μονάδα συγχρονισμού ή το \$**

**// lookahead token // lookahead lexeme // set of tokens**

**H check ελέγχει αν η τιμή της token περιλαμβάνεται σε ένα σύνολο FIRST, που διαβιβάζεται ως παράμετρος**

```
void ignore (STOKEN first_stoken) {
    STOKEN temp;
    int found=0;
    while ((token!=END_FILE) && (found==0)) {
        temp=first_stoken;
        while (found==0
            && temp!=NULL && temp->next!=NULL)
        {
            temp=temp->next;
            if (temp->stoken==token)
                found=1;
        }
        if (found==0)
            token=getToken();
    }
}

void check ( STOKEN FIRST_first , STOKEN FOLLOW_first ) {
    STOKEN temp=FIRST_first;
    int found=0;
    while (temp->stoken!=token && temp->next!=NULL) {
        temp=temp->next;
    }
    if (temp->stoken==token) {
        printf("SYNTAX ERROR\n");
        temp->next=FOLLOW_first;
        ignore(FIRST_first);
    }
}
```

#### Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη σε κατάσταση «πανικού»

$$\begin{array}{ll}
 E & \rightarrow TT_r \\
 T_r & \rightarrow '+' TT_r \\
 & | '-' TT_r \\
 & | \epsilon \\
 T & \rightarrow FF_r \\
 F_r & \rightarrow '*' FF_r \\
 & | '/' FF_r \\
 & | \epsilon \\
 F & \rightarrow ('E') \\
 & | "αριθμός"
 \end{array}$$

```
void error() { //syntax error handling
    printf("SYNTAX ERROR\n");
}

void match(TokenType expected_token) {
    if (token==expected_token)
        token=getToken(); //lexical analysis
    else
        error();
}

int E(STOKEN, STOKEN);
```

**H check μπορεί να καλείται από τις συναρτήσεις μη τερματικών δύο φορές. Την πρώτη φορά ελέγχεται αν η token ανήκει στο σύνολο FIRST του μη τερματικού συμβόλου και αν δεν ανήκει, τότε έχουμε λάθος και ενεργοποιείται η διαδικασία ανάκαμψης.**

**Τη δεύτερη φορά ελέγχεται αν η token ανήκει στο σύνολο FOLLOW του μη τερματικού, κάτι που συμβαίνει είτε μετά από επιτυχή εφαρμογή του κανόνα, είτε μετά την ανάκαμψη από λάθος.**

## Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη σε κατάσταση «πανικού»

$$\begin{array}{l}
 E \rightarrow TT_r \\
 T_r \rightarrow '+' TT_r \\
 | \quad \neg TT_r \\
 | \quad \epsilon
 \end{array}$$

$$\begin{array}{l}
 T \rightarrow FF_r \\
 F_r \rightarrow '*' FF_r \\
 | \quad \neg FF_r \\
 | \quad \epsilon
 \end{array}$$

$$\begin{array}{l}
 F \rightarrow ('( E ') \\
 | \quad "αριθμός"
 \end{array}$$

Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη

```

int F( TOKEN synch_token_first,
       TOKEN synch_token_last) {
    int val=-999;
    int found=0;
    TOKEN temp1, temp2, temp;
    temp=synch_token_first;
    while(synch_token_last>temp) {
        if (temp->stoken!=token) {
            temp=temp->next;
        }
    }
}

Ta synch_token_first και synch_token_last προσαρμόζουν κατάλληλα το σύνολο των λεξικών μονάδων συγχρονισμού.

```

```

void F() {
    switch(token) {
        case LPAR: token=getToken();
                     E();
                     match(RPAR);
                     break;
        case NUMBER: token=getToken();
                      break;
        default: error();
    }
}

if (temp->stoken!=token) {
    switch(token) {
        case LPAR: token=getToken();
                     temp1->stoken=RPAR;
                     temp1->prev=NULL;
                     temp1->next=NULL;
                     val=E(temp1,temp1);
                     match(NUMBER);
                     break;
        case NUMBER: val=atoi(tokenString);
                      token=getToken();
                      break;
        default: error();
    }
}

temp1->stoken=LPAR;
temp2->stoken=NUMBER;
temp1->prev=NULL;
temp2->next=NULL;
temp1->next=temp2;
temp2->prev=temp1;
check(synch_token_first,synch_token_last,temp1);
return val;
}

```

10

## Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη σε κατάσταση «πανικού»

$$\begin{array}{l}
 E \rightarrow TT_r \\
 T_r \rightarrow '+' TT_r
 \end{array}$$

$$\begin{array}{l}
 | \quad \neg TT_r \\
 | \quad \epsilon
 \end{array}$$

$$\begin{array}{l}
 T \rightarrow FF_r \\
 F_r \rightarrow '*' FF_r
 \end{array}$$

$$\begin{array}{l}
 | \quad \neg FF_r \\
 | \quad \epsilon
 \end{array}$$

$$\begin{array}{l}
 F \rightarrow ('( E ') \\
 | \quad "αριθμός"
 \end{array}$$

Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη

Ta synch\_token\_first και synch\_token\_last προσαρμόζουν κατάλληλα το σύνολο των λεξικών μονάδων συγχρονισμού.

Προβλέπουσα αναδρομική κατάβαση χωρίς ανάκαμψη (από προηγούμενη εκδοχή)

```

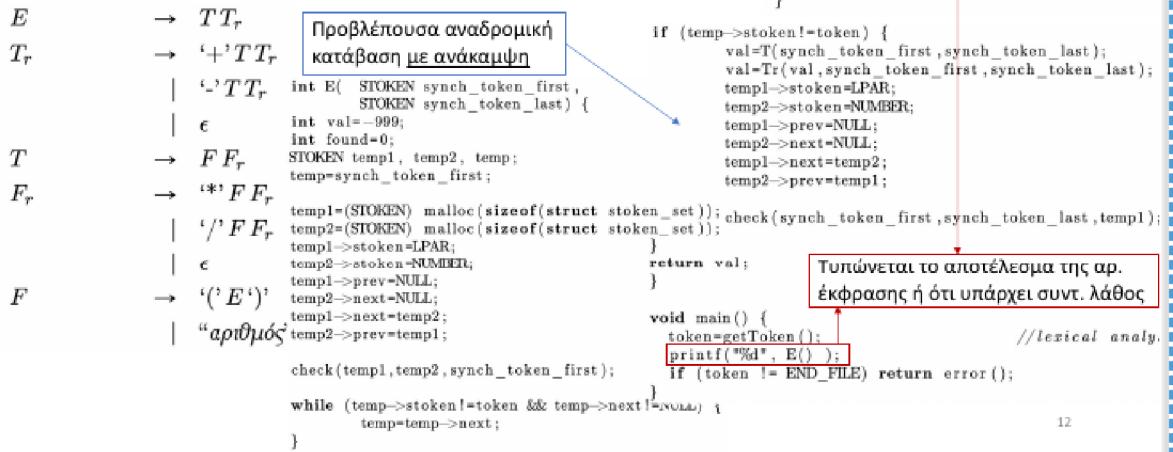
void Fr() {
    switch(token) {
        case MULT: token=getToken();
                     F();
                     Fr();
                     break;
        case DIV: token=getToken();
                     F();
                     Fr();
                     break;
        default: return;
    }
}

if (temp->stoken!=token) {
    switch(token) {
        case MULT: token=getToken();
                     val=val*M(synch_token_first,
                                synch_token_last);
                     val=Fr(val,synch_token_first,
                            synch_token_last);
                     break;
        case DIV: token=getToken();
                     val=val/V(synch_token_first,
                                synch_token_last);
                     val=Fr(val,synch_token_first,
                            synch_token_last);
                     break;
    }
}
return val;
}

```

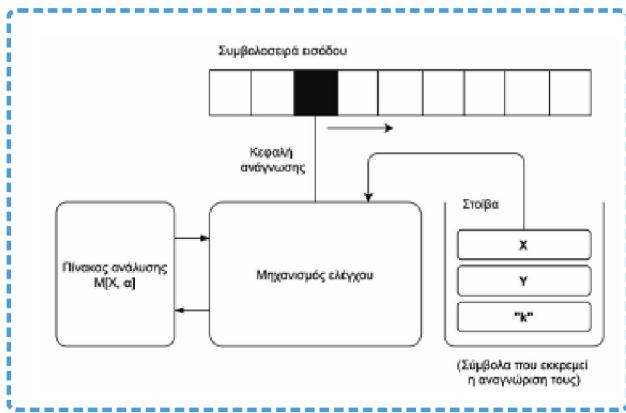
11

## Προβλέπουσα αναδρομική κατάβαση με ανάκαμψη σε κατάσταση «πανικού»



12

## Ανάλυση LL(1)



- Στην ανάλυση  $LL(1)$ , για κάθε συνδιασμό μη-τερματικού συμβόλου  $X$  και τερματικού συμβόλου  $a$ , ο αλγόριθμος αποφασίζει την ενέργεια που θα εφαρμώσει, με βάση έναν πίνακα  $M[X, a]$
- Ο πίνακας υπολογίζεται εφαρμόζοντας έναν αλγόριθμο στους κανόνες της γραμματικής
- Το μη τερματικό σύμβολο της επόμενης παραγωγής, δίνεται κάθε φορά από την κορυφή της στοίβας που συντηρείται από τον αλγόριθμο ανάλυσης
- Λειτουργεί ως εξής
  - Η στοίβα αρχικοποιείται με το σύμβολο  $\$$  και ακολουθείται από το σύμβολο αρχής της γραμματικής
    - Αν στην κορυφή της στοίβας έχουμε τερματικό και είναι ίδιο με αυτό στην κεφαλή ανάγνωσης (τρέχον αναγνωριστικό), αφαιρείται από τη στοίβα και διαβάζεται το επόμενο αναγνωριστικό. Αν τα δύο αναγνωριστικά δεν ταυτίζονται, τότε έχουμε λάθος
    - Αν στην κορυφή της στοίβας έχουμε μη-τερματικό, επιλέγεται από τον πίνακα ανάλυσης  $M$ , το δεξί μέρος του κανόνα που αντιστοιχεί στο μη-τερματικό και στο τερματικό που υποδεικνύει η κεφαλή ανάγνωσης
  - Αν η θέση του πίνακα έχει τιμή  $error$  (κενή θέση), τότε έχουμε λάθος. Διαφορετικά αντικαθίσταται το μη τερματικό της κορυφής της στοίβας με τα σύμβολα του δεξιού μέρους του κανόνα που επιλέχθηκε
  - Η ανάλυση ολοκληρώνεται με επιτυχία, όταν προσεγγιστεί το τέλος της συμβολοσειράς εισόδου, και η στοίβα περιέχει μόνο το  $\$$

### Παράδειγμα

#### Καθοδική ανάλυση $LL(1)$

Πίνακας Ανάλυσης $M$	$E$	$\rightarrow TT_r$							
	$T_r$	$\rightarrow + TT_r$							
			$- TT_r$						
			$\epsilon$						
	$T$	$\rightarrow FF_r$							
	$F_r$	$\rightarrow * FF_r$							
			$/ FF_r$						
			$\epsilon$						
	$F$	$\rightarrow ( E )$							
			"αριθμός"						

$E$	"αριθμός"	$+$	$-$	$*$	$/$	$($	$)$	$\$$
$TT_r$						$TT_r$		
		$+ TT_r$	$- TT_r$				$\epsilon$	$\epsilon$
$FF_r$						$FF_r$		
$( E )$				$* FF_r$	$/ FF_r$		$\epsilon$	$\epsilon$
"αριθμός"						$( E )$		

ΣΤΟΙΒΑ	ΣΥΜΒΟΛΟΣΕΙΡΑ	ΚΑΝΟΝΑΣ
$\$E$	$27 - 5 * \$\$$	
$\$T_r T$	$27 - 5 * \$\$$	$E \rightarrow TT_r$
$\$T_r F_r F$	$27 - 5 * \$\$$	$T \rightarrow F F_r$
$\$T_r F_r "αριθμός"$	$27 - 5 * \$\$$	$F \rightarrow "αριθμός"$
$\$T_r F_r$	$-5 * \$\$$	
$\$T_r$	$-5 * \$\$$	$F_r \rightarrow \epsilon$
$\$T_r T_r^{-1}$	$-5 * \$\$$	$T_r \rightarrow -1 TT_r$
$\$T_r T$	$5 * \$\$$	
$\$T_r F_r F$	$5 * \$\$$	$T \rightarrow F F_r$
$\$T_r F_r "αριθμός"$	$5 * \$\$$	$F \rightarrow "αριθμός"$
$\$T_r F_r$	$* \$\$$	
$\$T_r F_r F^{**}$	$* \$\$$	$F_r \rightarrow ** FF_r$
$\$T_r F_r F$	$\$\$$	
$\$T_r F_r "αριθμός"$	$\$\$$	$F \rightarrow "αριθμός"$
$\$T_r F_r$	$\$$	
$\$T_r$	$\$$	$F_r \rightarrow \epsilon$
$\$$	$\$$	$T_r \rightarrow \epsilon$

• Κανόνες ανάλυσης  $LL(1)$  – αριστερή παραγωγή της **27-5\*8**

$E \Rightarrow TT_r \Rightarrow FF_r T_r \Rightarrow "αριθμός" F_r T_r \Rightarrow "αριθμός" T_r \Rightarrow "αριθμός" - T_r \Rightarrow "αριθμός" - FF_r T_r \Rightarrow "αριθμός" - "αριθμός" F_r T_r \Rightarrow "αριθμός" - "αριθμός" - F_r T_r \Rightarrow "αριθμός" - "αριθμός" * F_r T_r \Rightarrow "αριθμός" - "αριθμός" * "αριθμός" F_r T_r \Rightarrow "αριθμός" - "αριθμός" * "αριθμός"$  28

## Αλγόριθμος Υπολογισμού Πίνακα Ανάλυσης

**Είσοδος:** μία γραμματική  $G$

**Έξοδος:** ο πίνακας  $M$  της ανάλυσης LL(1)

```

1: for all κανόνες  $X \rightarrow \bar{u}$  do
2:   for all τερματικά  $b \in FIRST(\bar{u})$  do
3:     προσθέτουμε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, b]$ ;
4:     if  $\epsilon \in FIRST(\bar{u})$  then
5:       for all τερματικά  $c \in FOLLOW(X)$  do
6:         προσθέτουμε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, c]$ ;
7:         if  $\$ \in FOLLOW(X)$  then
8:           προσθέτουμε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, \$]$ ;
9:     στις θέσεις του  $M$  που δεν έχει οριστεί τιμή, εγκωμαρείται η τιμή error;
```

- Για κάθε κανόνα της γραμματικής  $X \rightarrow \bar{u}$ 
  - για κάθε τερματικό  $b \in FIRST(\bar{u})$ 
    - προσθεσε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, b]$ 
      - αν  $\epsilon \in FIRST(\bar{u})$ , βάλε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, c]$ , για κάθε τερματικό  $c \in FOLLOW(X)$
      - αν  $\$ \in FOLLOW(X)$ , βάλε τον κανόνα  $X \rightarrow \bar{u}$  στη θέση  $M[X, \$]$
- Στις θέσεις που μένουν κενές εννοείται πως γράφουν **error**

### Παράδειγμα

$$\begin{array}{lcl}
E & \rightarrow & TT_r \\
T_r & \rightarrow & '+' TT_r \\
& | & '-' TT_r \\
& | & \epsilon \\
T & \rightarrow & FF_r \\
F_r & \rightarrow & '*' FF_r \\
& | & '/' FF_r \\
& | & \epsilon \\
F & \rightarrow & '(' E ')' \\
& | & "αριθμός"
\end{array}$$

$$\begin{array}{ll}
FIRST(E) = \{ '(', "αριθμός" \} & FOLLOW(E) = \{ \$, ')' \} \\
FIRST(T) = \{ '(', "αριθμός" \} & FOLLOW(T) = \{ '+', '−', \$, ')' \} \\
FIRST(F) = \{ '(', "αριθμός" \} & FOLLOW(F_r) = \{ \$, ')' \} \\
FIRST(T_r) = \{ '+', '−', \epsilon \} & FOLLOW(F) = \{ '*', '/', '+', '−', \$, ')' \} \\
FIRST(F_r) = \{ '*', '/', \epsilon \} & FOLLOW(F_r) = \{ '+', '−', \$, ')' \}
\end{array}$$

	"αριθμός"	'+'	'−'	'*'	'/'	'('	')'	\$
E	TT <sub>r</sub>					TT <sub>r</sub>		
T <sub>r</sub>		+' TT <sub>r</sub>	−' TT <sub>r</sub>					ε
T	FF <sub>r</sub>					FF <sub>r</sub>		
F <sub>r</sub>	ε	ε		*' FF <sub>r</sub>	/' FF <sub>r</sub>		ε	ε
F	"αριθμός"					'(' E ')'		

#### **Ανάλυση LL(1) με Ανάκαμψη σε Κατάσταση "Πανικού"**

- Στην  $LL(1)$  εντοπίζεται λάθος όταν στην κορυφή της στοίβας έχουμε ένα μη-τερματικό  $X$ , τέτοιο ώστε το τρέχον σύμβολο εισόδου να μην ανήκει στο  $FIRST(X)$  και αν σε αυτό συμβαίνει να περιέχεται το  $\epsilon$ , ούτε στο σύνολο  $FOLLOW(X)$  (κενές θέσεις του πίνακα)
  - Η ανάκαμψη σε κατάσταση "πανικού" υλοποιείται προσδιορίζοντας την κατάλληλη ενέργεια για κάθε περίπτωση κενής θέσης στον πίνακα ανάλυσης
    - **pop**: Αφαίρεση του  $X$  από τη στοίβα
    - **scan**: προσπέρασμα συμβόλων στην είσοδο μέχρι τον εντοπισμό λεξικής μονάδας που μπορεί να χρησιμοποιηθεί για την επανεκκινηση της ανάλυσης
    - Εισαγωγή ενός νέου μη-τερματικού στη στοίβα

# Αλγόριθμος

1. Ξεκινώντας από τον απλό πίνακα ανάλυσης  $LL(1)$
  2. Σε κάθε θέση  $M[X, a]$  (που είναι κενή), με  $a \in FOLLOW(X) \cup \{\$\}$ , βάλε την ενέργεια **pop**
  3. Σε κάθε θέση  $M[X, a]$  (που είναι κενή), με  $a \notin FIRST(X) \cup FOLLOW(X) \cup \{\$\}$ , βάλε την ενέργεια **scan**
  4. Αν αδειάσει η στοίβα, ενώ δεν έχει ολοκληρωθεί ακόμα η ανάγνωση της συμβολοσειράς εισόδου, εισάγεται στη στοίβα το σύμβολο της αρχής, και αγνοούνται όλα τα επόμενα, μέχρι να αναγνωστεί ένα σύμβολο που να ανήκει στο  $FIRST$  της αρχής

## Παράδειγμα

Ανάκαμψη σε κατάσταση «πανικού» σε ανάλυση LL(1)

$E$	$\rightarrow TT_r$	«πανικού» σε ανάλυση LL(1)						
$T_r$	$\rightarrow '+' TT_r$							
	$\cdot' TT_r$	$FIRST(E) = \{ ., "αριθμός" \}$	$FOLLOW(E) = \{ \$, . \}$					
	$\epsilon$	$FIRST(T) = \{ ., "αριθμός" \}$	$FOLLOW(T) = \{ +, -, \$, . \}$					
		$FIRST(F) = \{ ., "αριθμός" \}$	$FOLLOW(F) = \{ \$, . \}$					
$T$	$\rightarrow FF_r$	$FIRST(T_r) = \{ +, -, \epsilon \}$	$FOLLOW(T_r) = \{ \$, . \}$					
$F_r$	$\rightarrow {*} FF_r$	$FIRST(F_r) = \{ *, /, \epsilon \}$	$FOLLOW(F_r) = \{ +, -, \$, . \}$					
	$/' FF_r$							
	$\epsilon$							
$F$	$\rightarrow (' E ')$							
	$"αριθμός"$							

Εκτέλεση

Παράδειγμα

$\text{"պաթիսօ՞ն"}$	$\text{"+/-"}$	$\text{"*?"}$	$\text{"**?"}$	$\text{"?/?"}$	$\text{"(')"}$	$\text{"')?"}$	$\text{"\$"}$
$T$	$T_{\bar{r}}$	scan	scan	scan	scan	$T T_{\bar{r}}$	pop
$T_{\bar{r}}$	scan	$\text{"-1" } T T_{\bar{r}}$	$\text{"-1" } T T_{\bar{r}}$	scan	scan	scan	$\epsilon$
$F$	$F_{\bar{r}}$	pop	pop	scan	scan	$F F_{\bar{r}}$	pop
$F_{\bar{r}}$	scan	$\epsilon$	$\epsilon$	$\text{"*?" } F_{\bar{r}}$	$\text{"?/?" } F_{\bar{r}}$	scan	$\epsilon$
$F$	$\text{"պաթիսօ՞ն"}$	non	non	non	non	$\text{"! } (F \text{ ?)?"}$	non

- Η ανάλυση της συμβολοσειράς (27-\*) ολοκληρώνεται μετά από την ανάγνωση και του τελευταίου χαρακτήρα, ενώ αν δεν υπήρχαν στον πίνακα ανάλυσης οι ενέργειες ανάκαμψης από λάθος αυτό δε θα μπορούσα να γίνειν.
  - Η ανάλυση επανέρχεται σε κανονική λειτουργία (κελιά με ενέργειες ανάλυσης) μετά από την εκτέλεση δύο ενεργειών ανάκαμψης.
  - Για να αποφευχθύνουν τα διαγνωστικά μηνύματα από διαδιδόμενα λάθη, θα πρέπει να επιτρέπεται η παραγωγή τους μόνο μετά από την επανάληψη της ανάλυσης σε κανονική λειτουργία και την εκτέλεση μιας ή δύο ενεργειών ανάλυσης.

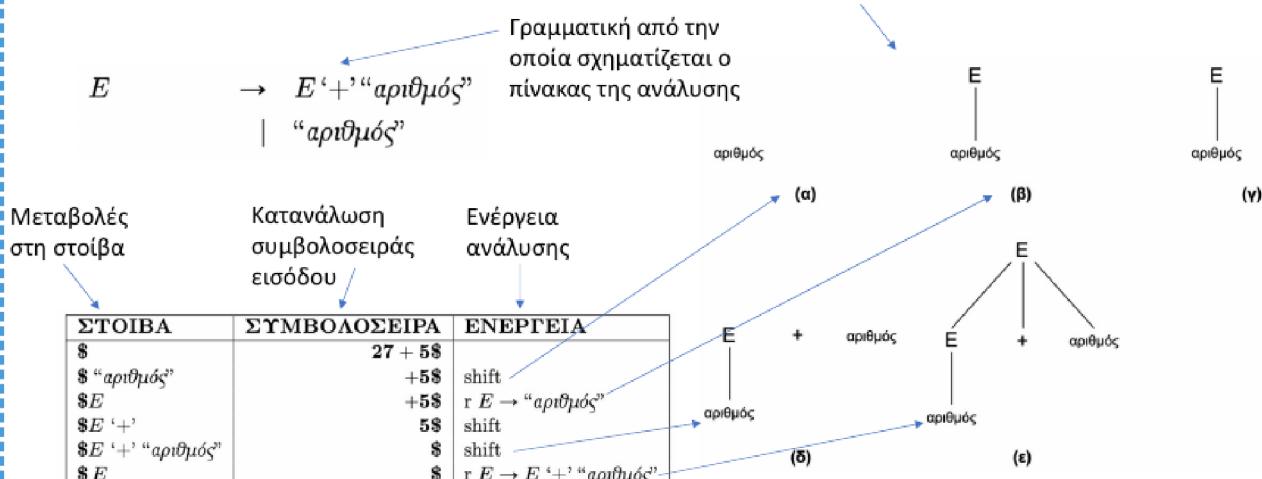
ΣΤΟΙΒΑ	ΣΤΜΒΟΛΟΣΕΙΡΑ	ΚΑΝΟΝΑΣ
\$E	(27 - *)\$	
\$T, T	(27 - *)\$	$E \rightarrow TT_r$
\$T, F, F	(27 - *)\$	$T \rightarrow FF_r$
\$T, F_r )' E(	(27 - *)\$	$F \rightarrow (^{'} E)^{'}$
\$T, F_r )' E	27 - *)\$	
\$T, F_r )' T, T	27 - *)\$	$E \rightarrow TT_r$
\$T, F_r )' T, F_r F	27 - *)\$	$T \rightarrow FF_r$
\$T, F_r )' T, F_r "αριθμός"	27 - *)\$	$F \rightarrow "αριθμός"$
\$T, F_r )' T_r	-*)\$	
\$T, F_r )' T_r	-*)\$	$F_r \rightarrow \epsilon$
\$T, F_r )' T_r T, :	-*)\$	$T_r \rightarrow \cdot \cdot \cdot TT_r$
\$T, F_r )' T_r T	*)\$	
\$T, F_r )' T_r T	*)\$	scan
\$T, F_r )' T_r	*)\$	pop
\$T, F_r )'	*)\$	$T_r \rightarrow \epsilon$
\$T, F_r	\$	
\$T_r	\$	$F_r \rightarrow \epsilon$
\$	\$	$T_r \rightarrow \epsilon$

## Ανοδική ΣΑ

- Η συμβολοσειρά εισόδου διαβάζεται από τα αριστερά προς τα δεξιά, και στην πορεία απλοποιείται προς το σύμβολο της αρχής της γραμματικής, με εφαρμογή των κανόνων της
- Το παραγόμενο δέντρο αναπτύσσεται σταδιακά, από τα αριστερά προς τα δεξιά και από τα φύλλα προς τη ρίζα
- Η ανοδική ΣΑ μπορεί να ταυτιστεί με την ανάλυση LR

Παράδειγμα

### Παράδειγμα ανοδικής ΣΑ



- Αντιστροφή σειράς εφαρμογής κανόνων, κατά την εκτέλεση ενεργειών reduce (r)

$E \Rightarrow E + "αριθμός" \Rightarrow "αριθμός" + "αριθμός" \rightarrow \text{δεξιά παραγωγή συμ/ράς εισόδου}$

# Ανάλυση LR

## Βασικές Έννοιες

- Σε κάθε βήμα της ανοδικής ΣΑ εκτελείται μία από τις εξής **Ενέργειες**
  - Απλοποίηση (reduce)**: σύμφωνα με κάποιον κανόνα της γραμματικής (πχ  $X \rightarrow u_1u_2\dots u_n$ ) αντικαθιστούνται τα σύμβολα του δεξιού μέρους, από την κορυφή της στοίβας, με το μητερματικό του δεξιού
  - Εισαγωγή Αναγνωιστικού (shift)** στη στοίβα της ανάλυσης
  - Αποδοχή (accept)** της συμβολοσειράς εισόδου
- Όταν εκτελείται απολοποίηση με κανόνα-ε ( $X \rightarrow \epsilon$ ), τότε το  $X$  εισάγεται στη στοίβα
- Η ανοδική ΣΑ είναι μία αλληλουχία ενεργειών *shift* και *reduce*, μέχρι να εμφανιστέι λάθος, ή να προσεγγιστεί κατάσταση αποδοχής (*accept*)
- Ενεργό Πρόθεμα**: τα σύμβολα που περιέχονται στη στοίβα, σε κάποιο βήμα της εκτέλεσης της ανοδικής ΣΑ
- Λαβή απλοποίησης**: μία ακολουθία συμβόλων στην κορυφή της στοίβας, που ταιριάζει στο δεξιό μέρος ενός κανόνα, και η εφαρμογή του αποδίδει ένα βήμα δεξιάς παραγωγής
  - Η ταύτιση με το δεξιό μέρος κανόνα δεν αρκεί για να είναι μία συμβολοσειρά λαβή
  - Μόνο όταν ο κανόνας δημιουργεί δεξιά παραγωγή με την εφαρμογή του, θεωρούμε τη συμβολοσειρά, λαβή
- Ο κύρια λειτουργία ενός αλγορίθμου ανοδικής ΣΑ, είναι η **αναγνώριση της επόμενης λαβής**
- Σε μία ανοδική ΣΑ, ο αλγόριθμος μπορεί να βρεθεί σε κατάσταση, που η επόμενη ενέργεια δεν είναι ξεκάθαρη, και πρέπει να επιλαγεί ανάμεσα σε πολλές υποψήφιες
  - Όταν η επιλογή είναι ανάμεσα σε ενέργειες *shift* και *reduce*, έχουμε **σύγκρουση εισαγωγής-απλοποίησης (shift-reduce conflict)**
  - Πιο σπάνια έχουμε και **σύγκρουση απλοποίησης-απλοποίησης (reduce-reduce conflict)**
- Σε περιπτώσεις *shift-reduce conflict* προτιμάται η *shift*, χωρίς να είναι εγκυημένα η σωστή απάντηση
- Σε περιπτώσεις *reduce-reduce conflict* προτιμάται ο πρώτος από τους δύο κανόνες, χωρίς παλι να είναι εγκυημένα ο σωστός

## Γενική Λειτουργία

- Παράγεται ο **Πίνακας Ενέργειών**
  - (Κατάσταση - Επόμενο Σύμβολο) -> Επόμενη Ενέργεια
- Παράγεται ο **Πίνακας Μεταβάσεων**
  - (Κατάσταση - Σύμβολο) -> Κατάσταση
- Ξεκινάμε με κατάσταση 0
- Επόμενη Ενέργεια** (πιν. Ενεργ.) <-  
πιο δεξιά κατάσταση + επόμενο σύμβολο
- Shift**:
  - Προσθέτουμε **νέα ρίζα** (με όνομα το σύμβολο εισαγωγής)
    - Κατάσταση νέας ρίζας (πιν. Μετ.) <-  
πιο δεξιά κατάσταση + σύμβολο εισαγωγής
- Reduce**  $X \rightarrow \bar{u}$ :
  - νέος κόμβος για το  $X$** , συνδέεται ως πρόγονος στις πιο δεξιές ρίζες που αντιστοιχούν στο  $\bar{u}$ 
    - Κατάσταση ρίζας  $X$  (πιν. Μετ.) <-  
αμέσως αριστ. κατάσταση από συνδεδεμένους κόμβους + σύμβολο εισαγωγής
  - Στη στοίβα διατηρείται μαζί με κάθε σύμβολο (ρίζα) και η αντίστοιχη κατάστασή
    - Σε ενέργεια *shift*, καταχωρείται στην κορυφή της στοίβας, πίσω από το σύμβολο που εισάγεται

- Σε ενέργεια *reduce* η λαβή που έχει σχηματιστέι στην κορυφή της στοίβας, αντικαθιστάται αό το μη-τερματικό στο αριστερό μέρος του κανόνα, μαζί με τη νέα κατάσταση
- Είναι απαραίτητη μία λειτουργεία ανάκαμψης από λάθη

### Παράδειγμα

## Ανάλυση LR(1)

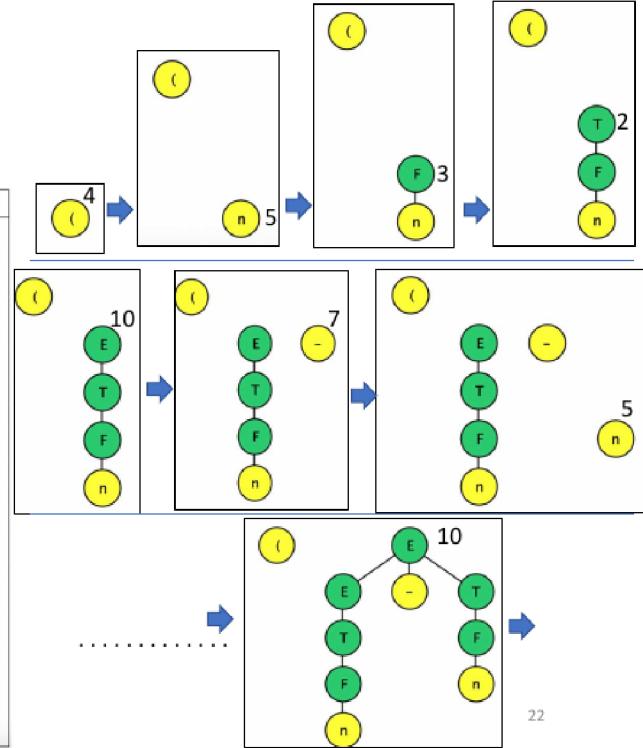
	Πίνακας ενεργειών							
	" <i>αριθμός</i> "	$\epsilon +$	$\epsilon \cdot$	$\epsilon^{*}$	$\epsilon /$	$\epsilon ($	$) \epsilon ^*$	$\$$
state 0	s							
state 1		s	s					accept
state 2		r 3	r 3	s				r 3 r 3
state 3		r 6	r 6	r 6	r 6			r 6 r 6
state 4	s					s		
state 5		r 8	r 8	r 8	r 8		r 8	r 8
state 6	s					s		
state 7	s					s		
state 8	s					s		
state 9	s					s		
state 10		s	s				s	
state 11		r 1	r 1	s	s	r 1	r 1	
state 12		r 2	r 2	s	s	r 2	r 2	
state 13		r 4	r 4	r 4	r 4	r 4	r 4	
state 14		r 5	r 5	r 5	r 5	r 5	r 5	
state 15		r 7	r 7	r 7	r 7	r 7	r 7	

(1) $E$	$\rightarrow E \cdot +'T$
(2)	$E \cdot -'T$
(3)	$T$
(4) $T$	$\rightarrow T \cdot ^{*'}F$
(5)	$T \cdot ^{+'}F$
(6)	$F$
(7) $F$	$\rightarrow ('E \cdot )'$
(8)	"αριθμός"

	Πίνακας μεταβάσεων									
	" <i>αριθμός</i> "	$\epsilon +$	$\epsilon \cdot$	$\epsilon^{*}$	$\epsilon /$	$\epsilon ($	$) \epsilon ^*$	$E$	$T$	$F$
state 0	5							4	1	2
state 1		6	7							
state 2			8	9						
state 3										
state 4	5							4	10	2
state 5										
state 6	5							4	11	3
state 7	5							4	12	3
state 8	5							4		13
state 9	5							4		14
state 10		6	7							15
state 11			8	9						
state 12			8	9						
state 13										
state 14										
state 15										

## Ανάλυση LR(1)

ΣΤΟΙΒΑ	ΣΥΜΒΟΛΟΣΕΙΡΑ	ΕΝΕΡΓΕΙΑ
\$0	$(27 - 5) * 8\$$	
\$0 '4	$27 - 5) * 8\$$	shift
\$0 '4 "αριθμός"5	$-5) * 8\$$	shift
\$0 '4 F3	$-5) * 8\$$	r 8
\$0 '4 T2	$-5) * 8\$$	r 6
\$0 '4 E10	$-5) * 8\$$	r 3
\$0 '4 E10 '7	$5) * 8\$$	shift
\$0 '4 E10 '7 "αριθμός"5	$) * 8\$$	shift
\$0 '4 E10 '7 F3	$) * 8\$$	r 8
\$0 '4 E10 '7 T12	$) * 8\$$	r 6
\$0 '4 E10	$) * 8\$$	r 2
\$0 '4 E10 ')15	$*8\$$	shift
\$0 F3	$*8\$$	r 7
\$0 T2	$*8\$$	r 6
\$0 T2 '*8	$8\$$	shift
\$0 T2 '*8 "αριθμός"5	$\$$	shift
\$0 T2 '*8 F13	$\$$	r 8
\$0 T2	$\$$	r 4
\$0 E1	$\$$	r 3
\$0 E1	$\$$	accept



## Κατασκευή Πινάκων Ανάλυσης LR

### Σύνολο Καταστάσεων

- Για κάθε παραλλαγή της ανάλυσης LR πρέπει να υπολογιστεί ένα σύνολο καταστάσεων, από το οποίο προκύπτουν οι πίνακες

### LR(0)

- Στοιχείο  $LR(0)$  (Ορισμός)**
  - Κάθε στοιχείο του  $LR(0)$  ορίζεται από έναν κανόνα παραγωγής και μία τελεία σε κάποια θέση του δεξιού μέρους του κανόνα, που καταγράφει ένα ενδιάμεσο βήμα στην αναγνώριση των συμβόλων που αυτό περιλαμβάνει
  - Έστω ο κανόνας  $X \rightarrow uvz$ 
    - με  $u, v, z$  οποιαδήποτε τερματικά ή μη τερματικά σύμβολα της γραμματικής
  - Τα στοιχεία  $LR(0)$  που αναφέρονται στον συγκεκριμένο κανόνα είναι
    - $[X \rightarrow \cdot uvz]$  <- αρχικό στοιχείο  $LR(0)$  - τελεία δεξιά
    - $[X \rightarrow u \cdot vz]$
    - $[X \rightarrow uv \cdot z]$
    - $[X \rightarrow uvz \cdot]$  <- συμπληρωμένο στοιχείο  $LR(0)$  - τελεία αριστερά
- πχ. το 2ο στοιχείο  $LR(0)$  περιγράγει την κατάσταση που έχει ήδη αναγνωριστεί στην είσοδο, συμβολοσειρά που παράγεται από το  $u$  και υπάρχει πιθανότητα αναγνώρισης στην είσοδο, συμβολοσειράς που παράγεται από το  $vz$
- Ο όρος  $LR(0)$  εκφράζει ότι κάθε στοιχείο δεν εξαρτάται από την τιμή κάποιου συμβόλου εισόδου
- Ο κανόνας  $X \rightarrow \epsilon$  παράγει το  $LR(0)$  σύνολο με μόνο στοιχείο το  $[X \rightarrow \cdot]$
- Ξεκινάμε επεκτείνοντας την γραμματική με ένα κανόνα (0) που ορίζεται ως αρχή της γραμματικής και παίρνει τη μορφή

$$(0)S \rightarrow E$$

, με  $E$  την παλιά αρχή της γραμματικής. Μία συμβολοσειρά εισόδου γίνεται αποδεκτή, όταν ο αναλυτής θα μπορούσε να εκτελέσει απλοποίηση για τον κανόνα (0)

- Αντιστοιχίζουμε την αρχική κατάσταση (state 0) το στοιχείο

$$[S \rightarrow \cdot E]$$

που εκφράζει ότι δεν έχει αναγνωριστεί κάποιο μη-κενό ενεργό πρόθεμα

- Εφαρμόζουμε τους εξής κανόνες για κάθε σύνολο στο εξής:
  - Επέκταση μη-τερματικών (ε-κλείσιμο)** : για κάθε στοιχείο  $[Y \rightarrow \bar{u} \cdot X\bar{o}]$  (με  $\cdot$  πριν μη-τερματικό), προσθέτουμε στο ίδιο σύνολο και καθε στοιχείο της μορφής  $[X \rightarrow \cdot \bar{u}]$  για κάθε κανόνα της μορφής  $X \rightarrow \bar{u}$ , δηλαδή κάθε κανόνα με το μη-τερματικό  $X$  στο αριστερό του μέρος, με την  $\cdot$  στην αρχή
  - Δημιουργία νέων συνόλων (ανάγνωση)**: για κάθε σύμβολο (τερματικό ή μη) με  $\cdot$  πριν από αυτό σε κάποιο στοιχείο του συνόλου, δημιουργησε ένα νέο σύνολο, και αρχικοποίησέ το με, βάζοντας την  $\cdot$  μετά από αυτό, για κάθε στοιχείο που εμφανίζεται
    - Αυτό μπορεί να γραφτεί και ως  $\alpha_1\gamma_1\eta(I_i, X) = I_j$ , όπου
      - $I_i$ : το παρόν σύνολο
      - $X$ : το σύμβολο που "πυροδώτησε" τη δημιουργία νέου συνόλου
      - $I_j$ : το νέο σύνολο
    - Σημείωσε τις παραπάνω πληροφορίες γιατί θα χρειαστούν αργότερα

**Παράδειγμα**

(0)	$S$	$\rightarrow E$
(1)	$E$	$\rightarrow E '+' T$
(2)		$  \quad E '-' T$
(3)		$  \quad T$
(4)	$T$	$\rightarrow T '*' F$
(5)		$  \quad T '/' F$
(6)		$  \quad F$
(7)	$F$	$\rightarrow (' E ')$
(8)		$  \quad "αριθμός"$

Η κατασκευή συνεχίζεται, οπότε με τις πράξεις ανάγνωση συμβόλου και ε-κλείσιμο, παίρνουμε τα σύνολα στοιχείων:

<b>state 0</b> επόμενα σύμβολα: $E$ - state 1 $T$ - state 2 $F$ - state 3 $($ - state 4 <b>αριθμός</b> - state 5	$[S \rightarrow \cdot E]$ $[E \rightarrow \cdot E + T]$ $[E \rightarrow \cdot E - T]$ $[E \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[F \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot αριθμός]$	<b>state 4</b> επόμενα σύμβολα: $E$ - state 10 $T$ - state 2 $F$ - state 3 $($ - state 4 <b>αριθμός</b> - state 5	$[F \rightarrow (\cdot E)]$ $[E \rightarrow \cdot E + T]$ $[E \rightarrow \cdot E - T]$ $[E \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot αριθμός]$	<b>state 8</b> επόμενα σύμβολα: $F$ - state 13 $($ - state 4 <b>αριθμός</b> - state 5	$[T \rightarrow T * \cdot F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot αριθμός]$	<b>state 12</b> επόμενα σύμβολα: $*$ - state 8 $/$ - state 9	$[E \rightarrow E - T \cdot]$ $[T \rightarrow T \cdot * F]$ $[T \rightarrow T \cdot / F]$
<b>state 1</b> επόμενα σύμβολα: $+$ - state 6 $-$ - state 7	$[S \rightarrow E \cdot]$ $[E \rightarrow E \cdot + T]$ $[E \rightarrow E \cdot - T]$	<b>state 5</b>	$[F \rightarrow αριθμός]$	<b>state 9</b> επόμενα σύμβολα: $F$ - state 14 $($ - state 4 <b>αριθμός</b> - state 5	$[T \rightarrow T \cdot / F]$ $[F \rightarrow (\cdot E)]$ $[F \rightarrow \cdot αριθμός]$	<b>state 13</b>	$[T \rightarrow T \cdot F]$
<b>state 2</b> επόμενα σύμβολα: $*$ - state 8 $/$ - state 9	$[E \rightarrow T \cdot]$ $[T \rightarrow T \cdot * F]$ $[T \rightarrow T \cdot / F]$	<b>state 6</b> επόμενα σύμβολα: $T$ - state 11 $F$ - state 3 $($ - state 4 <b>αριθμός</b> - state 5	$[E \rightarrow E + \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot αριθμός]$	<b>state 10</b> επόμενα σύμβολα: $F$ - state 15 $($ - state 6 $-$ - state 7	$[F \rightarrow (E \cdot)]$ $[E \rightarrow E \cdot + T]$ $[E \rightarrow E \cdot - T]$	<b>state 14</b>	$[T \rightarrow T / F]$
<b>state 3</b>	$[T \rightarrow F \cdot]$	<b>state 7</b> επόμενα σύμβολα: $T$ - state 12 $F$ - state 3 $($ - state 4 <b>αριθμός</b> - state 5	$[E \rightarrow E - T \cdot]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot αριθμός]$	<b>state 11</b> επόμενα σύμβολα: $*$ - state 8 $/$ - state 9	$[E \rightarrow E + T \cdot]$ $[T \rightarrow T \cdot * F]$ $[T \rightarrow T \cdot / F]$	<b>state 15</b>	$[F \rightarrow (E \cdot)]$

26

- Για να είναι μία γλώσσα  $LR(0)$  είναι αναγκαίο:
  - Να μην υπάρχει σύνολο στοιχείων που περιέχει ταυτόχρονα ένα συμπληρωμένο στοιχείο (· στο τέλος) και ένα στοιχείο της μορφής  $[Y \rightarrow \bar{u} \cdot k\bar{o}]$ , με  $k$  τερματικό
  - Κάθε σύνολο στοιχείων να περιέχει το πολύ ένα συμπληρωμένο στοιχείο

## LR(1)

- **Στοιχείο LR(1) (Ορισμός)**

- Κάθε στοιχείο  $LR(1)$  έχει τη γενική μορφή

$$[A \rightarrow X_1 \dots X_i \cdot X_{i+1} \dots X_j, a]$$

και περιγράφει ότι έχουν σχηματιστεί στη στοίβα τα  $X_1 \dots X_i$  και αναμένεται να σχηματιστούν τα  $X_{i+1} \dots X_j$  και μετά να απλοποιηθούν, στην περίπτωση που ακολουθεί το τερματικό  $a$  (*lookahead σύμβολο*) στην είσοδο

- Το lookahead symbol μπορεί να είναι οποιοδήποτε **τερματικό** ή  $\$$  και επηρεάζει μόνο όταν βρίσκεται σε  $LR(1)$  στοιχεία της μορφής  $[A \rightarrow X_1 \dots X_j, a]$ , (όταν  $\cdot$  είναι στο τέλος), όταν δηλαδή μπορεί να γίνει απλοποίηση μόνο με επόμενο στοιχείο εισόδου  $a$
- Μπορεί για ένα στοιχείο να έχουμε περισσότερα από ένα lookahead symbols, οπότε για ευκολεία τα απαριθμούμε σε μία λίστα, διαχωρίζοντάς τα με /
- δηλαδή τα  $[A \rightarrow \bar{u}\cdot, a]$ ,  $[A \rightarrow \bar{u}\cdot, b]$  και  $[A \rightarrow \bar{u}\cdot, c]$ , μπορουν να γραφτούν

$$[A \rightarrow \bar{u}\cdot, a/b/c]$$

- Το lookahead ενός στοιχείου  $LR(1)$  θα είναι πάντα ένα **υποσύνολο** του  $A$

Για να παράξουμε τα σύνολα στοιχείων  $LR(1)$  μίας γραμματικής

1. Προσθέτουμε κανόνα (0) της μορφής  $S \rightarrow E$ , με  $E$  την προηγούμενη αρχή της γραμματικής
2. Αρχικοποιούμε το σύνολο  $I_0$  με το στοιχείο  $[S \rightarrow \cdot A, \$]$
3. Επεκτείνουμε κάθε σύνολο και δημιουργούμε νέα σύνολα ακολουθώντας τους εξής κανόνες
  - **Επέκταση μη-τερματικών (ε-κλείσιμο)**: για κάθε στοιχείο της μορφής  $[Y \rightarrow \bar{u} \cdot X\bar{o}, a] \in I_i$  (με  $\cdot$  πριν από κάποιο μη-τερματικό  $X$ ) προσθέτουμε στο ίδιο σύνολο  $I_i$  ένα νέο στοιχείο για κάθε κανόνα  $X \rightarrow \bar{w}$  (με το  $X$  στο αριστερό του μέρος), τοποθετώντας την  $\cdot$  στην αρχή του αριστερού του μέρους και με lookahead όλα τα στοιχεία του  $FIRST(\bar{o}a)$ , δηλαδή

$$[X \rightarrow \cdot \bar{w}, a_1/a_2/\dots/a_n]$$

- όπου  $FIRST(\bar{o}a) = a_1, a_2, \dots, a_n$  (και  $\$$  όπου βγαίνει κενό)
- **Δημιουργία νέων συνόλων (ανάγνωση)**: για κάθε σύμβολο (τερματικό ή μη) με  $\cdot$  πριν από αυτό σε κάποιο στοιχείο του συνόλου, δημιουργούμε ένα νέο σύνολο, και αρχικοποίησέ το με, βάζοντας την  $\cdot$  μετά από αυτό, για κάθε στοιχείο που εμφανίζεται
  - Αυτό μπορεί να γραφτεί και ως  $ανάγνωση(I_i, X) = I_j$ , όπου
    - $I_i$ : το παρόν σύνολο
    - $X$ : το σύμβολο που "πυροδώτησε" τη δημιουργία νέου συνόλου
    - $I_j$ : το νέο σύνολο
  - Σημείωσε τις παραπάνω πληροφορίες γιατί θα χρειαστούν αργότερα

### Παράδειγμα

(0)	$S$	$\rightarrow$	$A$
(1)	$A$	$\rightarrow$	$X X$
(2)	$X$	$\rightarrow$	"a" $X$
(3)	$X$	$\rightarrow$	"b"

<b>state 0</b> επόμενα σύμβολα: $A$ - state 1 $X$ - state 2 $a$ - state 3 $b$ - state 4	$[S \rightarrow \cdot A, \$]$ $[A \rightarrow \cdot XX, \$]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 5</b>	$[A \rightarrow XX\cdot, \$]$
<b>state 1</b>	$[S \rightarrow A\cdot, \$]$	<b>state 6</b> επόμενα σύμβολα: $X$ - state 9 $a$ - state 6 $b$ - state 7	$[X \rightarrow a \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$
<b>state 2</b> επόμενα σύμβολα: $X$ - state 5 $a$ - state 6 $b$ - state 7	$[A \rightarrow X \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$	<b>state 7</b>	$[X \rightarrow b\cdot, \$]$
<b>state 3</b> επόμενα σύμβολα: $X$ - state 8 $a$ - state 3 $b$ - state 4	$[X \rightarrow a \cdot X, a/b]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 8</b>	$[X \rightarrow aX\cdot, a/b]$
<b>state 4</b>	$[X \rightarrow b\cdot, a/b]$	<b>state 9</b>	$[X \rightarrow aX\cdot, \$]$

## Πίνακες

### LR(0)

**Είσοδος:** μία γραμματική  $G$  με αρχή το σύμβολο  $S$   
**Έξοδος:** πίνακας ενεργειών  $M$  και μεταβάσεων  $N$  ανάλυσης  $LR(0)$

- 1: Δημιουργείται το σύνολο  $I = \{I_0, I_1, \dots, I_n\}$ ;
- 2: **for**  $i:=0$  **to**  $n$  **do**
- 3:     **if**  $([X \rightarrow \bar{e}\cdot] \in I_i \text{ και } X \neq S)$  **then**
- 4:         **for all** σύμβολο εισόδου  $\alpha$  **do**
- 5:              $M[i, \alpha] := \text{reduce } X \rightarrow \bar{e};$
- 6:         **if**  $([S \rightarrow \bar{e}\cdot] \in I_i)$  **then**
- 7:              $M[i, \$] := \text{accept};$
- 8:         **for all**  $([Y \rightarrow \bar{u} \cdot k \bar{o}] \in I_i \text{ για } k \text{ τερματικό})$  **do**
- 9:             **if**  $(\text{ανάγνωση}(I_i, k) = I_j)$  **then**
- 10:                  $M[i, k] := \text{shift};$
- 11:                  $N[i, k] := j;$
- 12:         **for all** μη τερματικό σύμβολο  $X$  **do**
- 13:             **if**  $(\text{ανάγνωση}(I_i, X) = I_j)$  **then**
- 14:                  $N[i, X] := j;$

- Για τον πίνακα **Μεταβάσεων**  $N$ :

- Για κάθε σύνολο  $I_i$  οι μεταβάσεις του είναι οι **αναγνώσεις ανάγνωση**  $\text{ανάγνωση}(I_i, X) = I_j$  μέσω των συμβόλων  $X$  στα σύνολα  $I_j$
- Για τον πίνακα **Ενεργειών**  $M$ :

  - Για κάθε στοιχείο του  $N$  που δεν είναι κενό, βάλε στην αντίστοιχη θέση του  $M$  το *shift*
  - Αν ένα σύνολο  $I_i$  έχει στοιχείο της μορφής  $[X \rightarrow \bar{e}\cdot]$  (με την  $\cdot$  στο τέλος), αν το  $X$  είναι το η αρχή της γραμματικής,  $S$ , βάλε *accept* στο σύμβολο  $\$$ , διαφορετικά, βάλε *reduce*  $X \rightarrow \bar{e}$  σε όλα τα σύμβολα (που δεν έχουν ήδη *shift*)

Παράδειγμα (στον πίνακα ενεργειών σκέψου όπου υπάρχει μία ενέργεια γ να επεκτείνεται σε όλη τη γραμμή)

<b>state 0</b>	$[S \rightarrow \cdot E]$ επόμενα σύμβολα: $E \rightarrow E + T$ $E \rightarrow E - T$	<b>state 4</b>	$[F \rightarrow \cdot \cdot E)]$ επόμενα σύμβολα: $E \rightarrow E + T$ $E \rightarrow E - T$	<b>state 8</b>	$[T \rightarrow T \cdot \cdot F]$ επόμενα σύμβολα: $F \rightarrow state 13$ $(\cdot - state 4$ αριθμός - state 5	<b>state 12</b>	$[E \rightarrow E - T \cdot]$ επόμενα σύμβολα: $T \rightarrow T \cdot \cdot F$ $\cdot - state 8$ $/ - state 9$
$T - state 2$	$[E \rightarrow T]$	$T - state 2$	$[E \rightarrow T]$				
$F - state 3$	$[T \rightarrow T \cdot F]$	$F - state 3$	$[T \rightarrow T \cdot F]$	<b>state 9</b>	$[T \rightarrow T \cdot / F]$ επόμενα σύμβολα: $F \rightarrow \cdot (E)$	<b>state 13</b>	$[T \rightarrow T \cdot * F]$
$(\cdot - state 4$	$[T \rightarrow \cdot T / F]$	$(\cdot - state 4$	$[T \rightarrow \cdot T / F]$	<b>state 10</b>	$[F \rightarrow (E) \cdot]$ επόμενα σύμβολα: $E \rightarrow E + T$	<b>state 14</b>	$[T \rightarrow T / F]$
<b>αριθμός - state 5</b>	$[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \alphaριθμός]$	<b>αριθμός - state 5</b>	$[T \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \alphaριθμός]$	<b>state 11</b>	$[E \rightarrow E + T \cdot]$ επόμενα σύμβολα: $T \rightarrow T \cdot \cdot F$	<b>state 15</b>	$[F \rightarrow (E) \cdot]$
<b>state 1</b>	$[S \rightarrow E]$ επόμενα σύμβολα: $E \rightarrow E + T$	<b>state 5</b>	$[F \rightarrow \alphaριθμός]$				
$+ - state 6$	$[E \rightarrow E - T]$	<b>state 6</b>	$[E \rightarrow E + T]$				
$-- state 7$		<b>state 7</b>	$[T \rightarrow \cdot T * F]$				
<b>state 2</b>	$[E \rightarrow T \cdot]$ επόμενα σύμβολα: $T \rightarrow T \cdot * F$	$T - state 11$	$[T \rightarrow T / F]$				
$* - state 8$	$[T \rightarrow T \cdot / F]$	$F - state 3$	$[T \rightarrow \cdot F]$				
$/ - state 9$		$(\cdot - state 4$	$[F \rightarrow \cdot (E)]$				
<b>state 3</b>	$[T \rightarrow F]$	<b>αριθμός - state 5</b>	$[F \rightarrow \alphaριθμός]$				

26

	$\text{accept}$	$+^*$	$-^*$	$*^*$	$/^*$	$(^*$	$)^*$	\$
state 0	s					s		
state 1		s	s					accept
state 2	r 3	r 3	s			r 3	r 3	
state 3	r 6	r 6	r 6	r 6		r 6	r 6	
state 4	s					s		
state 5		r 8	r 8	r 8	r 8		r 8	r 8
state 6	s					s		
state 7	s					s		
state 8	s					s		
state 9	s					s		
state 10		s	s				s	
state 11		r 1	r 1	s	s		r 1	r 1
state 12		r 2	r 2	s	s		r 2	r 2
state 13		r 4	r 4	r 4	r 4		r 4	r 4
state 14		r 5	r 5	r 5	r 5		r 5	r 5
state 15		r 7	r 7	r 7	r 7		r 7	r 7

## SLR(1)

**Είσοδος:** μία γραμματική  $G$  με αρχή το σύμβολο  $S$

**Έξοδος:** πίνακες ενεργειών  $M$  και μεταβάσεων  $N$  ανάλυσης  $LR(0)$

```

1: Δημιουργείται το σύνολο  $I = \{I_0, I_1, \dots, I_n\}$ ;
2: for i:=0 to n do
3:   for all  $([X \rightarrow \bar{e}] \in I_i \text{ με } X \neq S)$  do
4:     for all σύμβολο εισόδου  $\alpha \in FOLLOW(X)$  do
5:        $M[i, \alpha] := \text{reduce } X \rightarrow \bar{e};$ 
6:     if  $([S \rightarrow \bar{e}] \in I_i)$  then
7:        $M[i, \$] := \text{accept};$ 
8:     for all  $([Y \rightarrow \bar{u} \cdot k \bar{o}] \in I_i \text{ για } k \text{ τερματικό})$  do
9:       if  $(\text{ανάγνωση}(I_i, k) = I_j)$  then
10:         $M[i, k] := \text{shift};$ 
11:         $N[i, k] := j;$ 
12:      for all μη τερματικό σύμβολο  $X$  do
13:        if  $(\text{ανάγνωση}(I_i, X) = I_j)$  then
14:           $N[i, X] := j;$ 

```

βασική διαφορά σε σχέση με τον αλγόριθμο για την κατασκευή των πινάκων της ανάλυσης  $LR(0)$

31

- Για τον πίνακα **Μεταβάσεων  $N$ :**

- Για κάθε σύνολο  $I_i$  οι μεταβάσεις του είναι οι αναγνώσεις ανάγνωση( $I_i, X$ ) =  $I_j$  μέσω των συμβόλων  $X$  στα σύνολα  $I_j$

- Για τον πίνακα **Ενεργειών  $M$ :**

- Για κάθε στοιχείο του  $N$  που δεν είναι κενό, βάλε στην αντίστοιχη θέση του  $M$  το *shift*
- Αν ένα σύνολο  $I_i$  έχει στοιχείο της μορφής  $[X \rightarrow \bar{e}]$  (με την  $\cdot$  στο τέλος), αν το  $X$  είναι το η αρχή της γραμματικής,  $S$ , βάλε *accept* στο σύμβολο  $\$$ , διαφορετικά, βάλε *reduce*  $X \rightarrow \bar{e}$  σε όλα τα σύμβολα του  $FOLLOW(X)$  (που δεν έχουν ήδη *shift*)

### Παράδειγμα

#### Οτοριζόντων.

state 0 επόμενα σύμβολα: $[E \rightarrow \cdot E]$ $E - \text{state 1}$ $T - \text{state 2}$ $F - \text{state 3}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[S \rightarrow \cdot E]$ $[E \rightarrow \cdot E + T]$ $[E \rightarrow \cdot E - T]$ $[T \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[F \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \text{αριθμός}]$	state 4 επόμενα σύμβολα: $[E \rightarrow \cdot E + T]$ $E - \text{state 10}$ $T - \text{state 2}$ $F - \text{state 3}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[F \rightarrow (\cdot E)]$ $[E \rightarrow \cdot E + T]$ $[E \rightarrow \cdot E - T]$ $[T \rightarrow \cdot T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[F \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \text{αριθμός}]$	state 8 επόμενα σύμβολα: $F - \text{state 13}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[T \rightarrow T * \cdot F]$ $[T \rightarrow \cdot F]$ $[F \rightarrow \cdot \text{αριθμός}]$	state 12 επόμενα σύμβολα: $* - \text{state 8}$ $/ - \text{state 9}$	$[E \rightarrow E - T \cdot]$ $[T \rightarrow T \cdot * F]$ $[T \rightarrow T \cdot / F]$
state 1 επόμενα σύμβολα: $[S \rightarrow E]$ $+ - \text{state 6}$ $- - \text{state 7}$	$[E \rightarrow E \cdot + T]$ $[E \rightarrow E \cdot - T]$	state 5 επόμενα σύμβολα: $[T \rightarrow E \cdot + T]$ $T - \text{state 11}$ $F - \text{state 3}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[F \rightarrow \cdot \text{αριθμός}]$	state 9 επόμενα σύμβολα: $F - \text{state 14}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[T \rightarrow T / \cdot F]$ $[F \rightarrow \cdot (E)]$ $F - \text{state 14}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	state 13 $[T \rightarrow T \cdot * F]$	
state 2 επόμενα σύμβολα: $[T \rightarrow E \cdot + T]$ $* - \text{state 8}$ $/ - \text{state 9}$	$[T \rightarrow T \cdot * F]$ $[T \rightarrow T \cdot / F]$	state 6 επόμενα σύμβολα: $[T \rightarrow E \cdot - T]$ $T - \text{state 12}$ $F - \text{state 3}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[E \rightarrow E \cdot + T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[F \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \text{αριθμός}]$	state 10 επόμενα σύμβολα: $) - \text{state 15}$ $+ - \text{state 6}$ $- - \text{state 7}$	$[F \rightarrow (E \cdot)]$ $[E \rightarrow E \cdot + T]$ $) - \text{state 15}$ $+ - \text{state 6}$ $- - \text{state 7}$	state 14 $[T \rightarrow T \cdot / F]$	
state 3	$[T \rightarrow F \cdot]$	state 7 επόμενα σύμβολα: $[T \rightarrow E \cdot - T]$ $E - \text{state 12}$ $F - \text{state 3}$ $( - \text{state 4}$ $\text{αριθμός} - \text{state 5}$	$[E \rightarrow E \cdot - T]$ $[T \rightarrow \cdot T * F]$ $[T \rightarrow \cdot T / F]$ $[F \rightarrow \cdot F]$ $[F \rightarrow \cdot (E)]$ $[F \rightarrow \cdot \text{αριθμός}]$	state 11 επόμενα σύμβολα: $* - \text{state 8}$ $/ - \text{state 9}$	$[E \rightarrow E + T \cdot]$ $[T \rightarrow \cdot * F]$ $[T \rightarrow \cdot / F]$	state 15 $[F \rightarrow (E \cdot)]$	

26

	$"\alpha\rho\theta\mu\delta\$"$		$\epsilon_+$	$\epsilon_{-}$	$\epsilon^{*}$	$\epsilon/\epsilon$	$\epsilon(\cdot)$	$\epsilon^{\cdot}$	\$
state 0	s					s			
state 1		s	s						accept
state 2	r 3	r 3	s			r 3	r 3		
state 3	r 6	r 6	r 6	r 6		r 6	r 6		
state 4	s				s				
state 5		r 8	r 8	r 8	r 8	r 8	r 8		
state 6	s				s				
state 7	s				s				
state 8	s				s				
state 9	s				s				
state 10		s	s			s			
state 11	r 1	r 1	s	s		r 1	r 1		
state 12	r 2	r 2	s	s		r 2	r 2		
state 13	r 4	r 4	r 4	r 4		r 4	r 4		
state 14	r 5	r 5	r 5	r 5		r 5	r 5		
state 15	r 7	r 7	r 7	r 7		r 7	r 7		

	$"\alpha\rho\theta\mu\delta\$"$		$\epsilon_+$	$\epsilon_{-}$	$\epsilon^{*}$	$\epsilon/\epsilon$	$\epsilon(\cdot)$	$\epsilon^{\cdot}$	E	T	F	
state 0	5							4		1	2	3
state 1		6	7									
state 2				8	9							
state 3												
state 4	5							4		10	2	3
state 5												
state 6	5							4		11	3	
state 7	5							4		12	3	
state 8	5							4			13	
state 9	5							4			14	
state 10		6	7						15			
state 11				8	9							
state 12				8	9							
state 13												
state 14												
state 15												

### (Κανονική) LR(1)

**Είσοδος:** μία γραμματική  $G$  με αρχή το σύμβολο  $S$

**Έξοδος:** πίνακας ενεργειών  $M$  και μεταβάσεων  $N$  ανάλυσης  $LR(1)$

```

1: Δημιουργείται το σύνολο  $I = \{I_0, I_1, \dots, I_n\}; /$ 
2: for i:=0 to n do
3:   for all ( $[X \rightarrow \bar{e}\cdot, a] \in I_i$  με  $X \neq S$ ) do
4:      $M[i, a] := \text{reduce } X \rightarrow \bar{e};$ 
5:     if ( $[S \rightarrow \bar{e}\cdot, \$] \in I_i$ ) then
6:        $M[i, \$] := \text{accept};$ 
7:     for all ( $[Y \rightarrow \bar{u} \cdot k \bar{o}, b] \in I_i$  για  $k$  τερματικό) do
8:       if ( $\alpha\gamma\eta\omega\sigma\eta(I_i, k) = I_j$ ) then
9:          $M[i, k] := \text{shift};$ 
10:         $N[i, k] := j;$ 
11:      for all μη τερματικό σύμβολο  $X$  do
12:        if ( $\alpha\gamma\eta\omega\sigma\eta(I_i, X) = I_j$ ) then
13:           $N[i, X] := j;$ 

```

- Για τον πίνακα **Μεταβάσεων**  $N$

- Για κάθε σύνολο  $I_i$  οι μεταβάσεις του είναι οι αναγνώσεις αινάγμωση( $I_i, X$ ) =  $I_j$  μέσω των συμβόλων  $X$  στα σύνολα  $I_j$

- Για τον πίνακα **Ενεργειών**  $M$

- Για κάθε στοιχείο του  $N$  που δεν είναι κενό, βάλε στην αντίστοιχη θέση του  $M$  το *shift*
- Αν ένα σύνολο  $I_i$  έχει στοιχείο της μορφής  $[X \rightarrow \bar{e}\cdot, a]$  (με την · στο τέλος), αν το  $X$  είναι το η αρχή της γραμματικής,  $S$  και το  $a = \$$ , βάλε *accept* στο σύμβολο  $\$$ , διαφορετικά, βάλε *reduce*  $X \rightarrow \bar{e}$  σε όλα τα σύμβολα  $a$  (που δεν έχουν ήδη *shift*)

**Παράδειγμα**

<b>state 0</b> επόμενα σύμβολα: $A$ - state 1 $X$ - state 2 $a$ - state 3 $b$ - state 4	$[S \rightarrow \cdot A, \$]$ $[A \rightarrow \cdot XX, \$]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 5</b>	$[A \rightarrow XX\cdot, \$]$
<b>state 1</b>	$[S \rightarrow A\cdot, \$]$	<b>state 6</b> επόμενα σύμβολα: $X$ - state 9 $a$ - state 6 $b$ - state 7	$[X \rightarrow a \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$
<b>state 2</b> επόμενα σύμβολα: $X$ - state 5 $a$ - state 6 $b$ - state 7	$[A \rightarrow X \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$	<b>state 7</b>	$[X \rightarrow b, \$]$
<b>state 3</b> επόμενα σύμβολα: $X$ - state 8 $a$ - state 3 $b$ - state 4	$[X \rightarrow a \cdot X, a/b]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 8</b>	$[X \rightarrow aX\cdot, a/b]$
<b>state 4</b>	$[X \rightarrow b\cdot, a/b]$	<b>state 9</b>	$[X \rightarrow aX\cdot, \$]$

	“a”	“b”	\$
state 0	s	s	
state 1			accept
state 2	s	s	
state 3	s	s	
state 4	r 3	r 3	
state 5			r 1
state 6	s	s	
state 7			r 3
state 8	r 2	r 2	
state 9			r 2

	“a”	“b”	$A$	$X$
state 0	3	4	1	2
state 1				
state 2	6	7		5
state 3	3	4		8
state 4				
state 5				
state 6	6	7		9
state 7				
state 8				
state 9				

## LALR(1)

Παράδειγμα

### Ανάλυση LALR(1)

#### LR(1)

<b>state 0</b> επόμενα σύμβολα: $A$ - state 1 $X$ - state 2 $a$ - state 3 $b$ - state 4	$[S \rightarrow \cdot A, \$]$ $[A \rightarrow \cdot XX, \$]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 5</b>	$[A \rightarrow XX\cdot, \$]$
<b>state 1</b>	$[S \rightarrow A\cdot, \$]$	<b>state 6</b> επόμενα σύμβολα: $X$ - state 9 $a$ - state 6 $b$ - state 7	$[X \rightarrow a \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$
<b>state 2</b> επόμενα σύμβολα: $X$ - state 5 $a$ - state 6 $b$ - state 7	$[A \rightarrow X \cdot X, \$]$ $[X \rightarrow \cdot aX, \$]$ $[X \rightarrow \cdot b, \$]$	<b>state 7</b>	$[X \rightarrow b\cdot, \$]$
<b>state 3</b> επόμενα σύμβολα: $X$ - state 8 $a$ - state 3 $b$ - state 4	$[X \rightarrow a \cdot X, a/b]$ $[X \rightarrow \cdot aX, a/b]$ $[X \rightarrow \cdot b, a/b]$	<b>state 8</b>	$[X \rightarrow aX\cdot, a/b]$
<b>state 4</b>	$[X \rightarrow b\cdot, a/b]$	<b>state 9</b>	$[X \rightarrow aX\cdot, \$]$

- Παρατηρούμε ότι κάποιες από τις καταστάσεις της LR(1) ανάλυσης έχουν σημαντικές ομοιότητες μεταξύ τους, π.χ. οι 3 και 6. Αυτές οι καταστάσεις διαφέρουν μόνο ως προς τα σύνολα των συμβόλων lookahead. Το ίδιο επίσης ισχύει για τις καταστάσεις 4 & 7, καθώς και για 8 & 9.
- Στην συγκεκριμένη γραμματική με την ανάλυση LALR(1) έχουμε πολύ πιο μικρό χώρο καταστάσεων για ισοδύναμη λειτουργία.
- Χρειάζεται όμως προσοχή γιατί το παραπάνω δεν ισχύει πάντα.

#### LALR(1)

<b>state 3-6</b>	$[X \rightarrow a \cdot X, a/b/\$]$ $[X \rightarrow \cdot aX, a/b/\$]$ $[X \rightarrow \cdot b, a/b/\$]$
<b>state 4-7</b>	$[X \rightarrow b\cdot, a/b/\$]$
<b>state 8-9</b>	$[X \rightarrow aX\cdot, a/b/\$]$

40

## Συστήματα Τύπων (Επισκόπηση Θεωρίας)

- Η πιο στοιχειώδης αναπαράσταση δεδομένων σε ηλεκτρονικό υπολογιστή είναι οι ακολουθίες bits
- Οι γλώσσες προγραμματισμού έχουν αφαιρετικές αναπαραστάσεις (**τύπους**) για απλά δεδομένα
  - ακέραιοι
  - πραγματικοί αριθμοί
  - boolean τιμές
- Επίσης υπάρχουν μηχανισμοί σύνθεσης νέων μορφών αναπαράστασης δεδομένων, από ήδη υπάρχουσες
- **Πλεονεκτήματα** βελτιωμένη αναγνωσιμότητα, διευκόλυνση συγγραφής προγραμμάτων, αξιοπιστία και ανεξαρτησία από την μηχανή εκτέλεσης
- **Ζητήματα** στη χρήση τύπων δεδομένων
  - η υλοποίηση τύπων δεδομένων έχει εξαρτήσεις από τη μηχανή εκτέλεσης, που μπορεί να μην λαμβάνονται υπόψη στον ορισμό της γλώσσας
  - ασυμφωνία ως προς κατά πόσο πρέπει να γίνεται ρητή χρήση πληροφοριών τύπων (άδηλοι vs ρητοί τύποι) και ποιό ρόλο πρέπει να έχουν στην επαλήθευση ορθότητας των προγραμμάτων (καμία vs αυστηρός έλεγχος) κατά τη μεταγλώττιση
- Το **σύστημα τύπων** μιας γλώσσας ορίζει το πως οι τύποι συσχετίζονται σε υπολογισμούς με εκφράσεις, που διατυπώνονται με βάση τη σύνταξη της γλώσσας

## Σχεδίαση Συστήματος Τύπων

1. Ορίζεται η σύνταξη των εντολών και των εκφράσεων της γλώσσας, καθώς και οι τύποι που αυτή θα υποστηρίζει
2. Ορίζονται οι κανόνες εμβέλειας, που αντιστοιχίζουν την κάθε εμφάνιση ονόματος στο σημείο που αυτό δηλώνεται
3. Ορίζονται οι κανόνες του συστήματος τύπων της γλώσσας, που περιγράφουν μία σχέση  $t : \tau$ , που αντιστοιχίζει εντολές ή εκφράσεις  $t$  σε τύπους  $\tau$ , και ακόμα δύο σχέσεις:
  1. της  $\tau_A \leq \tau_B$ , που εκφράζει ότι ο τύπος  $\tau_A$  είναι **μετατρέψιμος** σε τύπο  $\tau_B$
  2. της  $\tau_A = \tau_B$  που εκφράζει την ισοδυναμία δύο τύπων
    - Οι κανόνες αυτοί ορίζονται πάντα ως προς ένα σύνολο πληροφοριών, το **στατικό περιβάλλον τύπων** που βρίσκονται στον πίνακα συμβόλων
    - Αν η  $t : \tau$  συνδέεται με το περιβάλλον τύπων  $\Gamma$ , με πληροφορίες για τις ελεύθερες μεταβλητές στην έκφραση  $t$ , τότε η σχέση  $\Gamma \vdash t : \tau$ , σημαίνει ότι η  $t$  έχει τύπο  $\tau$  στο περιβάλλον  $\Gamma$
4. Ορίζεται η σημασία της γλώσσας (3 προσεγγίσεις) ως μία σχέση που αντιστοιχίζει εντολές και εκφράσεις στα αποτελέσματα της αποτίμησής τους
  - Η σημασία και το σύστημα τύπων είναι αλληλένδετοι ορισμοί
  - Οι τύποι μίας έκφρασης ή μίας εντολής και του αποτελέσματος αυτής, πρέπει να είναι *ίδιοι* ή να συσχετίζονται μέσω κατάλληλης σχέσης
- Ένα ολοκληρωμένο σύστημα τύπων θα πρέπει να συσχετιστεί με τη σημασία της γλώσσας, καθώς οι εγγυήσεις που αυτό εξασφαλίζει μπορούν να ισχύουν μόνο για μία συγκεκριμένη σημασία της γλώσσας

## Τύποι Δεδομένων

- Ο τύπος είναι μία *ιδιότητα* στοιχείων σύνταξης, όπως οι μεταβλητές, συνατήσεις, εκφράσεις κα.
- **Τύπος**
  - Ένας τύπος ορίζει ένα σύνολο (*εύρος*) πιθανών τιμών και ένα σύνολο λειτουργειών (*πράξεων*), που έχει νόημα για αυτές τις τιμές
  - Όλες οι τιμές των δεδομένων, που επιτρέπει μία γλώσσα, πρέπει να ανήκουν σε έναν τύπο, ακόμα και αν ο τύπος δε δηλώνεται ρητά από τον προγραμματισμό

## Γλώσσες Προγραμματισμού ως Προς τα Συστήματα Τύπων τους

### Ρητοί και Άδηλοι Τύποι

- **Ρητοί και Άδηλοι Τύποι**
  - Μία γλώσσα προγραμματισμού, που η σύνταξή της απαιτεί τη δήλωση τύπων, λέμε ότι διαθέτει **ρητούς τύπους**, ενώ σε διαφορετική περίπτωση, λέμε ότι η γλώσσα έχει **άδηλους τύπους**

### Χρησιμότητα Τύπων

- Δυνατότητα εντοπισμού λαθών στη χρήση της μνήμης προγράμματος πχ
  - απόποειρα ανάθεσης ακέραιας τιμής, στη θέση ενός δείκτη
  - αναφορά ενός πεδίου `private` εκτός του κώδικα της κλάσης (τύπου) που αυτό ορίζεται
- Υποβοήθηση της μετάφρασης
  - Σε κάποιες γλώσσες (C), όλες οι μεταβλητές έχουν τύπο με εκ των προτέρων γνωστή αναπαράσταση στη γλώσσα μηχανής (**στατικός τύπος**)
  - Άλλες (python) δεν έχουν τύπο οι μεταβλητές, αλλά οι τιμές, που είναι αντικείμενα του χρόνου εκτέλεσης (**δυναμικός τύπος**)
- Ο κώδικας που παράγεται σε γλώσσα μηχανής από στατικούς τύπους είναι πιο μικρός και γρήγορος, αφού η αναπαράσταση τύπων είναι ήδη γνωστή από τη μεταγλώττιση
- Για τους δυναμικούς τύπους, ο κώδικας που παράγεται πρέπει να είναι πιο γενικός

### Ορθότητα Τύπων και Ασφάλεια Τύπων

- Ο στατικός τύπος είναι ιδιότητα που συνδέεται σε χρόνο μεταγλώττισης και αναφέρεται σε όλους τους πιθανούς δυναμικούς τύπους, που θα μπορούσαν να δοθούν σε μία έκφραση
- **Θεώρημα Ορθότητας ή Ασφάλειας Τύπων**
  - Αν ένα πρόγραμμα μπορεί να επαληθευτεί ως προς τους τύπους του, τότε θα πρέπει εγγυημένα να έχει **καλώς ορισμένη συμπεριφορά** κατά την εκτέλεσή του: η τιμή (ή ο δυναμικός τύπος) που μπορεί να πάρει οποιαδήποτε έκφραση στο πρόγραμμα, θα είναι μέλος του στατικού τύπου έκφρασης
  - Η **ασφάλεια τύπων** επιβάλλεται απορρίπτοντας όλα τα προγράμματα που πιθανώς δεν είναι ασφαλή
- Αν ένα πρόγραμμα επαληθεύεται ως προς τους τύπους του, αποκλείονται τα **λάθη τύπων**
- **Λάθη Τύπων**
  - Κάθε λάθος που προκευύπτει από την εφαρμογή μίας λειτουργείας (*πράξης*) σε έναν τύπο δεδομένων, για τον οποίο αυτή δεν ορίζεται
- Δύο κατηγορίες λαθών σε χρόνο εκτέλεσης
  - **Λάθη κατάρρευσης** που προκαλούν διακοπή της εκτέλεσης του προγράμματος (πχ διαίρεση με το μηδέν)
  - **Λάθη ασταθούς συμπεριφοράς** που μπορεί να περνούν απαρατήρητα, αλλά εκτρέπουν την εκτέλεση του προγράμματος σε ανακόλουθη συμπεριφορά

- **Ασφαλές** Θεωρείται κάθε πρόγραμμα, που δεν έχει λάθη ασταθούς συμπεριφοράς. Συνήθως η ασφάλεια τύπων σε μία γλώσσα, έχει στόχο να αποτρέψει τα λάθη αυτά, μαζί με ένα υποσύνολο των λαθών κατάρρευσης (*απαγορευμένα λάθη*)
- **Ισχυρά ελεγχόμενη γλ.πρ. (Strongly Typed)**
  - Μία γλώσσα προγραμματισμού της οποίας όλα τα προγράμματα, που επιτρέπεται η εκτέλεσή τους, επιδυκνείουν καλή συμπεριφορά (δεν εκδηλώνουν απαγορευμένα λάθη)
  - Σε μία ισχυρά ελεγχόμενη γλ.πρ., τα προγράμματα είναι ασφαλή και αποτρέπονται τα απαγορευμένα λάθη
  - Η αποφυγή λαθών κατάρρευσης, που επιτρέπει ο μεταγλωττιστής, είναι τελικά ευθύνη του προγραμματιστή
  - Οι γλώσσες με στατικούς τύπους, που δεν αποτρέπουν όλα τα λάθη ασταθούς συμπεριφοράς, λέμε ότι δεν είναι *ισχυρά ελεγχόμενες* (*weakly typed*)
- **Έλεγχος Τύπων** (απορρίπτει προγράμματα που δεν επιδυκνείουν καλή συμπεριφορά)
  - Για δοθείσα έκφραση  $e$  και τύπο  $T$ , ο έλεγχος τύπων (*type checking*) είναι ένας αλγόριθμος που αποφαίνεται αν πράγματι η  $e$  είναι τύπου  $T$ . Ένα πρόγραμμα που περνά από έλεγχο τύπων, λέμε ότι έχει *καλώς ορισμένους τύπους*
  - Κατά τη μεταγλώττιση μπορεί να γίνεται *στατικός έλεγχος τύπων*: οι πληροφορίες για τον αλγόριθμο προέρχονται από δηλώσεις τύπων των μεταβλητών, των συναρτήσεων και άλλων στοιχείων σύνταξης, από τον πίνακα συμβόλων. Αυτές οι πληροφορίες χρησιμοποιούνται για να ελεγχθούν οι τύποι, που σχετίζονται με πράξεις που περιέχει το πρόγραμμα
  - Κατά την εκτέλεση μπορεί να εφαρμοστεί *δυναμικός έλεγχος τύπων*
    - είναι απαραίτητος ακόμα και σε γλώσσες με στατικούς τύπους
- **Συναγωγή Τύπων (type inference)**
  - Η διαδικασία συναγωγής πληροφοριών τύπων, που λείπουν από ένα πρόγραμμα. Για δοθείσα έκφραση  $e$  που δεν γνωρίζουμε τον τύπο της, η συναγωγή απαντά στο πρόβλημα προσδιορισμού του τύπου της  $e$

## Σύστημα Τύπων

- **Σύστημα τύπων (type system)**
  - Ένα σύνολο κανόνων, που αποδίδουν τύπους στα δομικά στοιχεία σύνταξης μίας γλώσσας, και καθορίζουν ποιές λειτουργείες είναι έγκυρες για κάθε τύπο
- **Εύρωστο** λέμε ένα σύστημα τύπων, στο οποίο τα προγράμματα με καλώς ορισμένους τύπους αποκλείεται να προκαλέσουν λάθη τύπων
- Ένα σύστημα τύπων πρέπει επίσης
  - Να καθιστά *αποφασίσιμη* την επαλίθευση προγραμμάτων
  - Να διακρίνεται εύκολα από τον προγραμματιστή αν ένα πρόγραμμα περνά από τον έλεγχο τύπων
  - Να επιβάλλει την εφαρμογή του ως εξής
    1. Έλεγχος δηλώσεων τύπων στατικά ή δυναμικά όταν δεν είναι εφικτό
    2. Επαλήθευση της συνέπειεας των προγραμμάτων με τις δηλώσεις

## Πλεονεκτήματα και Μειωνεκτήματα Συστήματος με Στατικό Έλεγχο Τύπων

- Συλλαμβάνονται πολλά προγραμματιστικά λάθη στη φάση της μεταγλωττισης, ενώ ο χρόνος απόκρισης του προγράμματος κατά την εκτέλεση δεν επιβαρύνεται από την επεξεργασία έλεγχου τύπων.
- Είναι πιθανό να απορρίπτει μερικά ορθά προγράμματα, καθώς οι τιμές εισόδου ή οι πιθανές συμπεριφορές του προγράμματος κατά την εκτέλεση δεν μπορούν να προβλεφθούν με απόλυτη ακρίβεια.

- Μπορεί να είναι αρκετά περιοριστικά για τους προγραμματιστές, δηλαδή να απαιτείται περισσότερη δουλειά για να υλοποιηθούν λειτουργίες, που σε μία γλώσσα με δυναμικό έλεγχο τύπων θα υλοποιούνταν πιο εύκολα και πιο γρήγορα.
  - Οι γλώσσες με στατικό έλεγχο τύπων συνήθως διαθέτουν κάποιους μηχανισμούς επανάκτησης της χαμένης ευελιξίας στον προγραμματισμό, όπως οι υποτύποι, οι μετατροπές τύπων και η παραμετροποίηση τύπου.

## Συστήματα Τύπων (Αναλυτική Θεωρία)

### Σύστημα Τύπων

- **Σύστημα Τύπων:** μία αποδοτικά υπολογίσιμη μέθοδος κανόνων συμπερασμού, με βάση τη σύνταξη, για να αποδεικνύεται ότι τα προγράμματα είναι απαλλαγμένα από συγκεκριμένες συμπεριφορές, μέσω κατηγοριοποίησης των φράσεων σύμφωνα με τα είδη τιμών, που αυτές υπολογίζουν.
  - Ένα σύστημα τύπων είναι **εύρωστο**, αν τα προγράμματα με καλώς ορισμένους - σύμφωνα με το σύστημα - τύπους δεν υπάρχει περίπτωση να προκαλέσουν λάθος τύπων.
    - Ιδανικά, θα πρέπει να αποδεικνύεται ότι ένα σύστημα τύπων είναι πράγματι εύρωστο.
  - Πολλές γλώσσες με στατικό έλεγχο τύπων υποστηρίζουν χαρακτηριστικά, όπως η μετατροπή σε υποτύπο (downcasting), που δε γίνεται να οριστούν κανόνες σε σύστημα τύπων με εγγυήσεις ευρωστίας. Σ' αυτή την περίπτωση, η ασφάλεια τύπων επιβάλλεται με δυναμικό έλεγχο της κάθε χρήσης των συγκεκριμένων χαρακτηριστικών.

### Έλεγχος Τύπων

- Ο **έλεγχος τύπων** επαληθεύει ότι όλες οι εκφράσεις σε ένα πρόγραμμα έχουν νόημα ως προς τα προβλεπόμενα από το σύστημα τύπων, κάτι βέβαια που εξαρτάται από τους τύπους των σταθερών, των μεταβλητών και των διαδικασιών, που περιέχουν
- Ο έλεγχος τύπων στηρίζεται στην εφαρμογή ενός **αλγορίθμου ισοδυναμίας τύπων**
- Ο έλεγχος τύπων γίνεται **στατικά**, δυναμικά ή με ένα συνδυασμό των δύο προσεγγίσεων. Σε μία ισχυρά ελεγχόμενη γλώσσα, όλα τα λάθη τύπων πρέπει να εντοπιστούν πριν από την εκτέλεση του προγράμματος, αλλά ο στατικός έλεγχος δεν μπορεί από μόνος του να αποτρέψει όλα τα λάθη κατάρρευσης.

#### Παραδείγματα

- Για παράδειγμα, αν `a` και `b` είναι μεταβλητές τύπου `int` και έχουν ανατεθεί σ' αυτές πολύ μεγάλες τιμές, τότε η έκφραση `a * b` μπορεί να υπερβαίνει το αποδεκτό εύρος τιμών των ακεραίων
- Δεύτερο παράδειγμα είναι μία απόπειρα υπολογισμού του λόγου μεταξύ δύο ακεραίων μπορεί να προκαλέσει διαίρεση με το 0.
- Αυτού του είδους τα λάθη κατάρρευσης δεν μπορούν να συλληφθούν από τον έλεγχο τύπων σε χρόνο μεταγλώττισης.

#### Παράδειγμα 8 (Έλεγχος τύπων ιδιοκτησίας στη Rust)

- Στη γλώσσα Rust, ένας δεσμευμένος πόρος μπορεί να μεταφέρεται από έναν ιδιοκτήτη σε έναν άλλο με εντολές, όπως η ανάθεση τιμής ή η κλήση συνάρτησης με παραμέτρους. Μετά τη μεταφορά της ιδιοκτησίας, ο αρχικός ιδιοκτήτης χάνει την πρόσβαση στον πόρο.

### Παράδειγμα 8 (Έλεγχος τύπων ιδιοκτησίας στη Rust)

- Στο παρακάτω πρόγραμμα ορίζεται η μεταβλητή `x` και ανατίθεται σε αυτήν ένα `struct String` (γραμμή 2), που περιέχει ένα δείκτη σε συμβολοσειρά και άλλες σχετικές πληροφορίες. Ιδιοκτήτης της συμβολοσειράς γίνεται η μεταβλητή `x`

```

1 fn main() {
2     let x = String::from("Hello Rust!");
3     let y = x;
4     println!("{} {}", x);
5 }
```

- Με την εντολή ανάθεσης `y=x`, η ιδιοκτησία του πόρου μεταφέρεται στην `y`. Ετσι από το σημείο της ανάθεσης η `x` δεν μπορεί να προσπελάσει τον πόρο και η εντολή στη γραμμή 4 οδηγεί σε λάθος μεταγλώττισης με το εξής μήνυμα:

```

>> error: borrow of moved value: 'x'
|
| let x = String::from("Hello Rust!");
|   -- move occurs because 'x' has type 'String',
|   which does not implement the 'Copy' trait
```

31

### Παράδειγμα 8 (Έλεγχος τύπων ιδιοκτησίας στη Rust)

- Το λάθος είναι αποτέλεσμα του ελέγχου τύπων, που βρίσκει ότι το `String` δεν υλοποιεί το απαιτούμενο trait (interface), για να συνεχίσει η `x` να χρησιμοποιεί τον πόρο μετά από την ανάθεση τιμής.
- Αν μετά την εντολή `y=x`, η `x` εξακολουθούσε να χρησιμοποιεί το `String`, τότε δε θα μπορούσε η Rust να εγγυηθεί την ασφαλή χρήση της μνήμης. Αυτό συμβαίνει επειδή οι `x` και `y` θα δείχνανε στην ίδια θέση μνήμης, με αποτέλεσμα όταν η ροή ελέγχου εξέρχεται από την εμβέλεια της δήλωσης των μεταβλητών ο πόρος θα αποδεσμεύταν δύο φορές (double free error)

## Συμβατοί Τύποι

- Δύο διαφορετικοί τύποι, που όμως μπορούν να συνδυαστούν με συγκεκριμένους τρόπους λέγονται **συμβατοί τύποι**.
- Η συμβατότητα μεταξύ τύπων είναι ένα κριτήριο, που δίνει μεγαλύτερη ευελιξία στον έλεγχο τύπων, έτσι ώστε να μην είναι απαραίτητο οι τύποι, που ελέγχονται να είναι ακριβώς ίδιοι μ' αυτούς που προβλέπονται από τον αλγόριθμο ισοδυναμίας τύπων.
- Για παράδειγμα, η έκφραση `e1 + e2` μπορεί να έχει νόημα ακόμη και αν οι τύποι των `e1` και `e2` δεν είναι ακριβώς ίδιοι μεταξύ τους και με τους προβλεπόμενους από το σύστημα τύπων της γλώσσας.
- Ένας άλλος σχετικός όρος είναι η **συμβατότητα ανάθεσης**, που χρησιμοποιείται για την επαλήθευση ορθότητας της ανάθεσης `x = e`.
- Η συμβατότητα τύπων των `x` και `e` περιπλέκεται από το γεγονός ότι στο αριστερό μέρος της ανάθεσης πρέπει να βρίσκεται μία *l-τιμή* ή μία αναφορά, ενώ στο δεξιό μέρος πρέπει να έχουμε μία *r-τιμή*.
- Η συμβατότητα ανάθεσης μπορεί να επεκταθεί, όπως στην περίπτωση της συμβατότητας τύπων, για να καλύψει περιπτώσεις, που δεν έχουμε τους ίδιους ακριβώς τύπους στα δύο μέρη της ανάθεσης. Π.χ. στη Java, η ανάθεση `x = e` είναι επιτρεπτή, όταν η `e` έχει αριθμητικό τύπο, που η τιμή του μπορεί να μετατραπεί στον τύπο της `x` χωρίς απώλεια πληροφορίας (π.χ. από `int` σε `long`)

## Μετατροπή Τύπων

- **Μετατροπή τύπου** έχουμε όταν ο προγραμματιστής χρησιμοποιεί μία τιμή ενός τύπου σε ένα πλαισιο χρήσης, στο οποίο αναμένεται ένας άλλος τύπος (ρητή μετατροπή)
  - Γενικά, μία μετατροπή τύπου μπορεί να επιφέρει μία αλλαγή τιμής ή αλλαγή αναπαράστασης
- Οι μετατροπές τύπων όμως δεν είναι αποκλειστικά πράξεις, που γίνονται μόνο με ρητή δήλωση, αλλά μπορεί να συμβαίνουν και **άρρητα** (αυτόματα). Οι άδηλες μετατροπές τύπων, που καλούνται επίσης και **εξαναγκασμός** τύπων (**type coercion**) είναι πράξεις, που, αν χρειάζεται, εκτελεί ο μεταγλωττιστής.

### Παραδείγματα

Στη Java επιτρέπεται να γράψουμε τους ορισμούς:

```
int x = 'c' ;
float y = x ;
```

αλλά η σχέση μεταξύ του τύπου `char` και του τύπου `int`, όπως επίσης και η σχέση του `int` με τον τύπο `float` δεν είναι μία σχέση υποτύπου (δηλ. κληρονομικότητας) αλλά ονομάζεται μετατροπή (ή εξαναγκασμός) τύπου.

### Κανόνες μετατροπής τύπων σε εκφράσεις της C

- Πρώτα, οι τελεστέοι τύπου `char` ή `short` μετατρέπονται σε `int`, και οι τύπου `float` μετατρέπονται σε `double`. Στη συνέχεια, αν κάποιος από τους τελεστέους είναι `double`, τότε ο άλλος μετατρέπεται σε `double` και αυτός είναι ο τύπος του αποτελέσματος.
- Αλλιώς, αν κάποιος από τους τελεστέους είναι `long`, τότε ο άλλος μετατρέπεται σε `long` και αυτός είναι ο τύπος του αποτελέσματος.
- Αλλιώς, αν κάποιος από τους τελεστέους είναι `unsigned`, τότε ο άλλος μετατρέπεται σε `unsigned` και αυτός είναι ο τύπος του αποτελέσματος.
- Αλλιώς, όλοι οι τελεστέοι θα πρέπει να είναι `int` και αυτός είναι ο τύπος του αποτελέσματος.

- Οι μετατροπές διακρίνονται επίσης σε

- **Μετατροπές διεύρυνσης (widening conversions)**: μετατρέπουν «μικρότερους» τύπους σε «μεγαλύτερους», δηλαδή σε τύπους που οι τιμές τους είναι υπερσύνολα.
- **Μετατροπές περιορισμού (narrowing conversions)**: ακολουθούν την αντίθετη κατεύθυνση και υπάρχει κίνδυνος απώλειας πληροφορίας.

### Παράδειγμα

#### Παράδειγμα 9

- Ας υποθέσουμε ότι έχουμε τους παρακάτω ορισμούς μεταβλητών σε ένα πρόγραμμα Java.
- Οι αναθέσεις τιμών `x=y`; και `a=b`; επιτρέπονται
- Άλλα οι `y=x`; και `b=a`; θα προκαλούσαν λάθος.
- Επίσης, οι ρητές μετατροπές τύπου:

```
x = (Object) y;
a = (int) b;
```

```
Object x=...;
String y=...;
int a=...;
short b=42;
```

είναι και αυτές ασφαλείς, αλλά η `y=(String)x`; μπορεί να προκαλέσει εξαίρεση, ενώ η `b=(short)a`; μπορεί να προκαλέσει απώλεια πληροφορίας

# Συστήματα Τύπων (Πρακτικά Ζητήματα)

## Δομητές Τύπων

- **Απλοί τύποι:** είναι κυρίως απλοί τύποι που δεν έχουν καμία άλλη δομή εκτός από την εγγενή αριθμητική και ακολουθιακή τους δομή
  - κάθε γλώσσα προγραμματισμού διαθέτει ένα σύνολο προκαθορισμένων τύπων με βάση το οποίο δομούνται όλοι οι υπόλοιποι
  - Υπάρχουν και απλοί τύποι που δεν είναι προκαθορισμένοι
- **Κανονικοί Τύποι:** τύποι με διακριτή διάταξη των στοιχείων που περιέχουν

### Παράδειγμα

Τύποι Απαρίθμησης

```
enum Day { Monday, Tuesday, Wednesday, Thursday,
           Friday, Saturday, Sunday};
```

- **Δομητές τύπων:** είναι οι τρόποι με τους οποίους μπορούν να συνδυαστούν οι απλοί τύποι για να δημιουργήσουν νέους
- **Καρτεσιανό γινόμενο:**  $Z \times Y = \{(z, y) | z \in Z, y \in Y\}$ 
  - με  $Z$  και  $Y$  να είναι (κανονικοί) τύποι
  - ουσιαστικά η διαδικασία που δημιουργεί structs και classes
- **Ένωση:** ένας τύπος που μπορεί να παίρνει τιμές από το εύρος άλλων (κανονικών) τύπων
  - **Διακριτή Ένωση:** προσθέτει μία ετκικέτα για κάθε στοιχείο, ώστε να διακρίνεται από ποιό σύνολο προέρχεται
  - **Μη-Διακριτή Ένωση:** κάνει το σύστημα τύπων μη-ασφαλές
- **Υποσύνολο:** Δηλώνει πως ένας σύνθετος τύπος κληρονομεί τις πράξεις του από ένα άλλο (κανονικό) τύπο, αλλά δεν μπορεί να πάρει τιμές από όλο το εύρος του.
  - Παράδειγμα Ada

```
sybtype IntDigit_Type is integer range 0 ... 9;
```

- **Πίνακες και Συναρτήσεις:**  $f: Z \rightarrow Y$ 
  - με  $Z$  και  $Y$  κανονικούς τύπους
  - $f$  ένα πίνακα με δείκτες τύπου  $Z$  και τιμές τύπου  $Y$ 
    - μία αντιστοίχιση τιμών  $Z$  σε τιμές  $Y$
  - Παράδειγμα C

```
typedef int (*IntFunction) (int);
```

- **Δείκτες και Αναδρομικοί Τύποι:** δημιουργούν το σύνολο όλων των διευθύνσεων, για τον τύπο που προσδιορίζεται
  - Παράδειγμα C

```
typedef int* IntPtr;
```

## Ισοδυναμία Τύπων

- Πότε μπορούμε να πούμε ότι δύο μεταβκητές έχουν τον ίδιο τύπο;
- **Δομική Ισοδυναμία:** αν έχουν ίδια δομή, δηλαδή έχουν κατασκευαστεί από τους ίδιους απλούς τύπους, με τους ίδιους δομητές, τότε δύο τύποι είναι ισοδύναμοι
  - **C** σε περιπτώσεις που δεν είναι *struct* ή *union*
  - **Java** σε πίνακες (με ειδικούς κανόνες)
- **Όνομαστική Ισοδυναμία:** αν δύο τύποι έχουν το ίδιο όνομα, τότε είναι ισοδύναμοι
  - **C** σε *structs* και *unions*
  - **Ada**
  - **Java** σε *classes* και *interfaces*

# Θέσεις και Κανόνες Τύπων

## Περιβάλλον

- (απλή) **Θέση Τύπου** ονομάζεται μία παράσταση της μορφής

$$e : \tau$$

- και σημαίνει ότι η **έκφραση**  $e$  αποτιμάται σε αποτέλεσμα **τύπου**  $\tau$
- και έμμεσα ότι η  $e$  είναι καλώς ορισμένου τύπου (περνά από τον έλεγχο τύπων)

- **Περιβάλλον:** ένα σύνολο από απλές δηλώσεις τύπων, της μορφής

$$\Gamma = x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n$$

- Αν μία μεταβλητή  $x$  ορίζεται ως τύπου  $\tau$  σε ένα περιβάλλον  $\Gamma$ , μπορούμε να γράψουμε  $\Gamma \vdash x : \tau$

## Θέση Τύπου

- **Θέση Τύπου:** μία παράσταση της μορφής

$$\Gamma \vdash e : \tau$$

- που σημαίνει πως αν όλες οι ελεύθερες μεταβλητές της **έκφρασης**  $e$  ορίζονται στο περιβάλλον  $\Gamma$ , τότε η **έκφραση**  $e$  αποτιμάται σε τιμή τύπου  $\tau$

## Κανόνας Τύπων

- **Κανόνας Τύπων:** μία παράσταση της μορφής

$$\frac{\text{υπόθεση}_1 \dots \text{υπόθεση}_n}{\text{συμπέρασμα}}$$

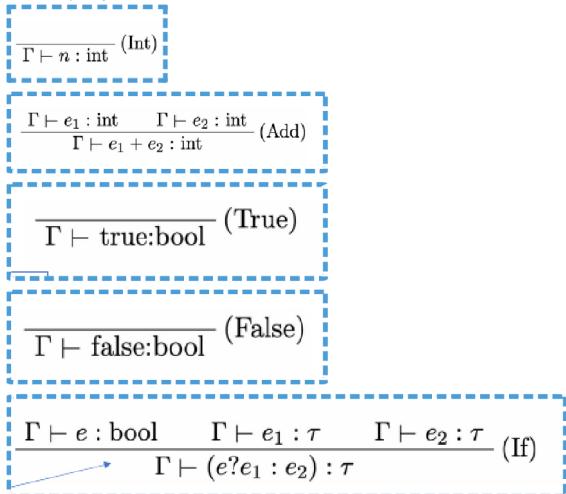
- που σημαίνει πως αν **μπορούν να αποδειχθούν οι υποθέσεις** τότε θα **ισχύει το συμπέρασμα**
- Ο πιο βασικός κανόνας τύπων είναι το

$$\frac{}{\Gamma \vdash id : t} (Id)$$

- Μπορούμε να τον χρησιμοποιούμε αν  $id : \tau \in \Gamma$
- λέγεται **(Id)**
- και ανήκει σε μία ειδική κατηγορία κανόνων που λέγονται **Αξιώματα**, γιατί δεν έχουν **υποθέσεις**

### Παραδείγματα

το **Αξίωμα**



## Επέκταση Περιβάλλοντος

- Σε ένα κανόνα τύπων, μπορεί αντί για περιβάλλον  $\Gamma$  να δούμε ένα επεκταμένο περιβάλλον

$$\Gamma, x_1 : \tau_1, \dots, x_n : \tau_n$$

που είναι το περιβάλλον  $\Gamma$  μαζί με τους κανόνες  $x_i : \tau_i$

- Πιο συγκεκριμένα η έκφραση  $\Gamma, x : \tau$  σημαίνει
  - $\Gamma \cup \{x : \tau\}$  όταν  $\exists \tau' \neq \tau | x : \tau' \in \Gamma$
  - $\Gamma - \{x : \tau'\} \cup \{x : \tau\}$  όταν  $\exists \tau' \neq \tau | x : \tau' \in \Gamma$
- Με άλλα λόγια ο κανόνας που εισάγεται υπερισχύει αν υπάρχει άλλος κανόνας για την μεταβλητή  $x$  στο  $\Gamma$

### Παράδειγμα

$$\frac{\Gamma, \text{id} : \tau \vdash e : \tau}{\Gamma, \text{id} : \tau \vdash \text{id} = e : \tau} (\text{Var} - \text{assign})$$

$$\frac{\Gamma \vdash e_3 : \tau \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_1 : \text{array}[\tau]}{\Gamma \vdash e_1[e_2] = e_3 : \tau} (\text{Array} - \text{assign})$$

## Κανόνες Συνάρτησης

- Μία συνάρτηση ορίζεται  $\tau_n f(\tau_1 id_1, \dots, \tau_n id_n)$
- Ο κανόνας που ελέγχει την εγκυρότητα της δήλωσης μίας συνάρτησης είναι ο
 
$$\frac{\Gamma, \text{id}_1 : \tau_1, \dots, \text{id}_k : \tau_k \vdash e : \tau_r}{\Gamma \vdash (\lambda[id_i : \tau_i]_{i=1,\dots,k}. e) : \tau_1 \times \tau_2 \dots \times \tau_k \rightarrow \tau_r} (\text{Fun} - \text{body})$$
- 'Όπου
  - Η  $(\lambda[id_i : \tau_i]_{i=1,\dots,k}. e)$  είναι μία  $\lambda$ -έκφραση, δηλαδή μία ανώνυμη συνάρτηση με παραμέτρους  $id_i$  τύπων  $\tau_i$  και με σώμα  $e$
  - Ο  $\tau_1 \times \tau_2 \dots \times \tau_k \rightarrow \tau_r$  είναι μία αντιστοίχιση (συνάρτηση) από το καρτεσιανό γινόμενο τύπων  $\tau_i$  σε τύπο  $\tau_r$
- Ο κανόνας που ελέγχει την εγκυρότητα της κλήσης μίας συνάρτησης είναι ο
 
$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \dots \times \tau_k \rightarrow \tau_r \quad \Gamma \vdash [e_i : \tau_i]_{i=1,\dots,k}}{\Gamma \vdash e(e_1, \dots, e_k) : \tau_r} (\text{Fun} - \text{call})$$

## Τύπος void

- Υπάρχουν σε κάθε γλώσσα εκφράσεις που δεν έχουν τιμή, αυτές λέμε ότι είναι τύπου *void* και το συμβολίζουμε με

$$s : \text{void}$$

- Ο τύπος *void* χρησιμοποιείται σε κανόνες που κάνουν έλεγχο τύπου σε εκφράσεις που δεν χαρακτηρίζονται απαραίτητα από την τιμή που επιστρέφουν

**Παράδειγμα**

$$\frac{\Gamma \vdash e: \text{bool} \quad \Gamma \vdash s: \tau}{\Gamma \vdash \text{while } (e) s : \text{void}} \text{ (While)}$$

$$\frac{\Gamma, \text{return} : \tau_r \vdash e: \tau_r}{\Gamma, \text{return}: \tau_r \vdash \text{return } e: \text{void}} \text{ (Return)}$$

- Ελέγχει πως στην έκφραση "return e" θα το e θα είναι ίδιου τύπου με τον ορισμό της συνάρτησης, αλλά η ίδια η έκφραση δεν αποτιμάται σε κάποιον τύπο

- Κάθε ακολουθία εντολών που συγκροτεί ένα πρόγραμμα θα πρέπει να περνά από έλεγχο τύπων. Θα πρέπει λοιπόν η πρώτη εντολή να είναι καλώς ορισμένου τύπου, καθώς επίσης και οι υπόλοιπες εντολές της ακολουθίας:

$$\frac{\Gamma \vdash s_1: \tau_1 \quad \Gamma \vdash (s_2; \dots; s_k): \tau_k}{\Gamma \vdash (s_1; s_2; \dots; s_k): \tau_k} \text{ (Seq)}$$

- Δεν καταλαβαίνω τι σημαίνει αυτό

## Κλάσεις

- Όταν μία γλώσσα επιτρέπει την δημιουργία κλάσεων, τότε για κάθε κλάση ορίζεται ένα (τοπικό) περιβάλλον

**Παράδειγμα**

Για την κλάση

```
class C {
    int x, y;
    int get_x() {return x;}
}
```

το περιβάλλον είναι

$$\Gamma_C = x: \text{int}, y: \text{int}, \text{get\_x}: \text{void} \rightarrow \text{int}$$

- Οι αντικειμενοστρεφείς γλώσσες επίσης διαθέτουν την λέξη - κλειδί this, που μπορεί να χρησιμοποιείται μέσα στην εμβέλεια του ορισμού της κάθε κλάσης.  
Όταν το περιβάλλον  $\Gamma$  περιέχει τις συνδέσεις με εμβέλεια μία κλάση  $\tau$ , τότε ισχύει το αξίωμα:

$$\frac{}{\Gamma \vdash \text{this} : \tau} \text{ (This)}$$

- Το null είναι η προκαθορισμένη τιμή για κάθε μεταβλητή κλάσης, που δεν έχει αρχικοποιηθεί στο σημείο που δηλώνεται.

Αφού το null είναι μετατρέψιμο σε οποιονδήποτε τύπο κλάσης, θα πρέπει να συμπεριληφθεί στη σχέση υποτύπου. Άρα θα υπάρχει «τύπος» null, που κληρονομεί από όλους τους τύπους κλάσης.

Με δεδομένο ότι ο «τύπος»  $\tau_{\text{null}}$  κληρονομεί από όλους τους τύπους κλάσης, ισχύει το αξίωμα:

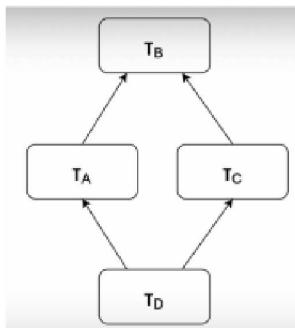
$$\frac{}{\Gamma \vdash \text{null} : \tau_{\text{null}}} \text{ (Null)}$$

## Σχέση Υποτύπων

- Επίσης λαμβάνεται υπόψη και ιεραρχία των κλάσεων. Με τη σχέση υποτύπου

$$\tau_A \leq \tau_B$$

ορίζουμε ότι η κλάση  $\tau_A$  κληρονομεί από την  $\tau_B$



Η σχέση υποτύπου έχει τις ιδιότητες:

- ανακλαστική:  $\tau_A \leq \tau_A$
- μεταβατική: αν  $\tau_D \leq \tau_A$  και  $\tau_A \leq \tau_B$ , τότε  $\tau_D \leq \tau_B$
- αντισυμμετρική: αν  $\tau_K \leq \tau_L$  και  $\tau_L \leq \tau_K$ , τότε  $\tau_K = \tau_L$

δηλ. είναι μία σχέση μερικής διάταξης.

- Αυτός ο ορισμός για τη σχέση υποτύπων μπορεί να επεκταθεί σε όλους τους τύπους της γλώσσας (μαζί με τους απλούς)
- Αν  $\tau_A, \tau_B$  είναι τύποι της γλώσσας, και  $\tau_A$  είναι Απλός Τύπος ή Τύπος Πίνακα, τότε
  - $\tau_A \leq \tau_B \Rightarrow \tau_A = \tau_B$
  - $\tau_B \leq \tau_A \Rightarrow \tau_A = \tau_B$

### Παράδειγμα

$$\frac{\Gamma, \text{id} : \tau_1 \vdash e : \tau_2 \quad \tau_2 \leq \tau_1}{\Gamma, \text{id} : \tau_1 \vdash \text{id} = e : \tau_1} (\text{Var-assign})$$

$$\frac{}{\Gamma \vdash \text{new } \tau : \tau} (\text{Con})$$

$$\frac{\Gamma \vdash e : \tau \quad (\text{id}, \tau') \in \Gamma_C}{\Gamma \vdash e.\text{id} : \tau'} (\text{Field-acc})$$

## Συμπερασμοί Τύπων

- Όλοι οι κανόνες που ορίζουμε χρησιμοποιούνται για να αποδείξουμε ότι οι τύποι των διαφορετικών εκφράσεων μέσα σε ένα πρόγραμμα είναι έγκυροι

### Παράδειγμα

$$\frac{}{\Gamma \vdash n : \text{int}} (\text{Int}) \quad \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}} (\text{Add})$$

$$\frac{}{\Gamma \vdash \text{true}: \text{bool}} (\text{True}) \quad \frac{}{\Gamma \vdash \text{false}: \text{bool}} (\text{False})$$

όπως  
διαφορετικών  
εκφράσεων

$$\frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e?e_1 : e_2) : \tau} (\text{If})$$

Με τους κανόνες, που έχουμε ορίσει, αν

$$\Gamma = b: \text{bool}, x: \text{int}$$

αποδεικνύεται ότι  $\Gamma \vdash (b?2 + 1:x): \text{int}$

$$\frac{}{\Gamma \vdash b : \text{bool}} \text{(Id)} \quad \frac{\Gamma \vdash 2 : \text{int} \quad \text{(Int)}}{\Gamma \vdash 2 + 1 : \text{int}} \quad \frac{\Gamma \vdash 1 : \text{int} \quad \text{(Int)}}{\Gamma \vdash 2 + 1 : \text{int}} \text{(Add)} \quad \frac{\Gamma \vdash x : \text{int} \quad \text{(Id)}}{\Gamma \vdash (b?2 + 1 : x) : \text{int}} \text{(If)}$$

δηλ. εφαρμόζεται ο κανόνας (If) για  $\tau = \text{int}$ .

## Σημασιολογική Ανάλυση

- **Σημασιολογική Ανάλυση:** επεξεργασία και υπολογισμός πληροφοριών και εκτέλεση ελέγχων που δεν μπορούν να οριστούν σε μία ΓΧΣ
  - Κανόνες που εγγυώνται την ορθότητα και την απρόσκοπτη εκτέλεση του πηγαίου προγράμματος

## Στατική Σημασία

- **Στατική Σημασία Προγράμματος:** επεξεργασία που σχετίζεται με πληροφορίες που είναι διαθέσιμες κατά τη μεταγλώττιση

## Δυναμική Σημασία

- **Δυναμική Σημασία Προγράμματος:** επεξεργασία που σχετίζεται με πληροφορίες που είναι διαθέσιμες κατά την εκτέλεση του προγράμματος

## Σημασιολογικοί Έλεγχοι

- **Πίνακας Συμβόλων:** πληροφορίες που κεataγράφονται σχετικά με την σημασία των ονομάτων, χρησιμοποιούνται μεταξύ άλλων στον έλεγχο τύπων
- Εξετάζονται
  - **Δηλώσεις:** για κάθε όνομα πρέπει να υπάρχει μία και μόνο μία δήλωση
  - **Έλεγχοι Ακολουθιών Ροής**
  - **Έλεγχοι Ορίων:** εξασφαλίζουν την προσπέλαση στις τιμές των στοιχείων ενός πίνακα μέσα στα επιτρεπτά όρια μεγέθους του
  - **Απλές Βελτιστοποιήσεις** όπως η αναδίπλωση σταθερών
- Υλοποιούνται λειτουργίες όπως
  - **Ρητή ή Άρρητη Μετατροπή Τύπων**
  - **Υπερφόρτωση Τελεστών και Ονομάτων**
  - **Χρήση Πολυμορφικών Συναρτήσεων**

## Γραμματικές Ιδιοτήτων

- **Γραμματική Ιδιοτήτων** είναι μία ΓΧΣ, της οποίας οι κανόνες  $X \rightarrow \bar{u}$  συνοδεύονται από σημασιολογικούς κανόνες της μορφής

$$b = f(c_1, c_2, \dots, c_n)$$

- $f$  μία συνάρτηση
  - που μπορεί να έχει τη μορφή έκφρασης
  - να περιλαμβάνει κλήσεις συναρτήσεων που ορίζονται αλλού
  - να περιλαμβάνει εκφράσεις τύπου *if ... then ... else ...*
  - και ότι άλλο κρίνεται λογικό για τον ορισμό της στατικής σημασίας
- $b$  μία ιδιότητα, η οποία μπορεί να είναι
  - **Συνθέσιμη** (η τιμή της μπορεί να υπολογιστεί με βάση τις τιμές των απογόνων της στο σημασιολογικό δέντρο) του  $X$ , με τα  $c_1, c_2, \dots, c_n$  να είναι ιδιότητες συμβόλων του  $\bar{u}$
  - **Κληρονομήσιμη** (η τιμή της μπορεί να υπολογιστεί με βάση τους προγόνους και τους αδελφικούς της κόμβους στο σημασιολογικό δέντρο) ενός συμβόλου του  $\bar{u}$ , με τα  $c_1, c_2, \dots, c_n$  να είναι ιδιότητες του  $X$  ή/και συμβόλων του  $\bar{u}$

## S-Γραμματική Ιδιοτήτων

- **S-Γραμματική Ιδιοτήτων** ονομάζεται μία γραμματική ιδιοτήτων που χρησιμοποιεί **αποκλειστικά συνθέσιμες ιδιότητες**

## L-Γραμματική Ιδιοτήτων

- **L-Γραμματική Ιδιοτήτων** μία γραμματική ιδιοτήτων που οι ιδιότητές της μπορούν να υπολογιστούν με μία μόνο διάσχιση του συντακτικού δέντρου, από τα αριστερά προς τα δεξιά με **DFS**
  - Χρησιμοποιεί συνθέσιμες ιδιότητες, αλλά επιτρέπει και κληρονομήσιμες μόνο από σύμβολα, που βρίσκονται στα αριστέρα του συμβόλου από το οποίο κληρονομούν

## Παραδείγματα

1o

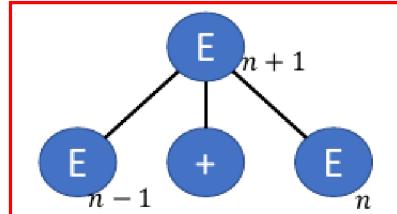
Παρακάτω ορίζεται μία απλή γραμματική ιδιοτήτων για τον υπολογισμό της τιμής αριθμητικών εκφράσεων:

- |   |   |
|---|---|
| (0) $E_{n+1} \rightarrow E_{n-1}' +' E_n$ | $[E_{n+1}.val = E_{n-1}.val + E_n.val]$ |
| (1) $E_{n+1} \rightarrow E_{n-1}' *' E_n$ | $[E_{n+1}.val = E_{n-1}.val * E_n.val]$ |
| (2) $E_{n+1} \rightarrow (' E_n')$        | $[E_{n+1}.val = E_n.val]$               |
| (3) $E_{n+1} \rightarrow "id"$            | $[E_{n+1}.val = \text{id}.val]$         |

οι  $E_i$  μπορούν να θεωρηθούν ως προσωρινές μεταβλητές, για το ίδιο σύμβολο της γραμματικής  $E$ , με τους δείκτες να καθορίζονται από τη σειρά, που δημιουργούνται οι μεταβλητές.

Όλες οι  $E_i.val$  είναι συνθέσιμες ιδιότητες.

συνθέσιμες ιδιότητες



Ανοδική ανάλυση και σημασία της  $a+b*c$  για  $a=1$ ,  $b=2$ ,  $c=3$ :

- (0)  $E_{n+1} \rightarrow E_{n-1}' +' E_n$
- (1)  $E_{n+1} \rightarrow E_{n-1}' *' E_n$
- (2)  $E_{n+1} \rightarrow (' E_n')$
- (3)  $E_{n+1} \rightarrow "id"$

ΣΤΟΙΒΑ	ΣΥΜΒΟΛΟΣΕΙΡΑ	ΣΥΝ. ΑΝΑΛΥΣΗ	ΕΝΕΡΓΕΙΑ
\$	$a + b * c \$$		
\$ "id"	$+b * c \$$	shift	
\$E	$+b * c \$$	r 3	
\$E +''	$b * c \$$	shift	
\$E +' id"	$*c \$$	shift	
\$E +' E	$*c \$$	r 3	
\$E +' E **'	$c \$$	shift	
\$E +' E ** id"	$\$$	shift	
\$E +' E ** E	$\$$	r 3	
\$E +' E	$\$$	r 1	
\$E	$\$$	r 0	
\$E	$\$$	accept	

Στη δεξιά στήλη εμφανίζεται η σειρά εκτέλεσης των ενεργειών, που σχετίζονται με τη στατική σημασία, δηλαδή με τον υπολογισμό της τιμής της έκφρασης  $a+b*c$ .

Η σειρά εξαρτάται από την εναλλαγή των shift και reduce της συντακτικής ανάλυσης.

2o

Η γραμματική ιδιοτήτων παρακάτω, που είναι επέκταση της γραμματικής χωρίς συμφραζόμενα για δηλώσεις μεταβλητών, είναι ένα παράδειγμα με κληρονομήσιμη ιδιότητα:

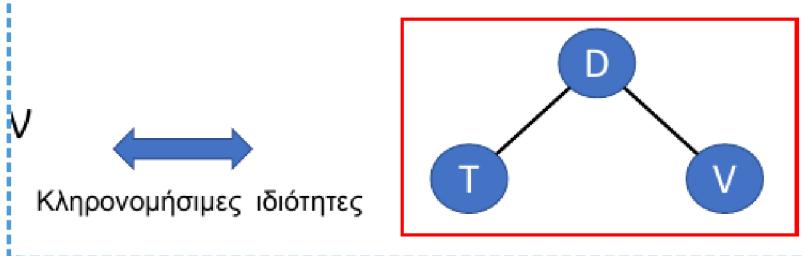
(0)	$D \rightarrow T \ V$	$[V.in = T.name]$
(1)	$T \rightarrow \text{"int"}$	$[T.name = \text{"integer"}]$
(2)	$  \quad \text{"float"}$	$[T.name = \text{"float"}]$
(3)	$V \rightarrow \text{"id"}$	$[\text{settype(id.entry}, V.in)]$
(4)	$  \quad V_R \ , \ \text{"id"}$	$[V_R.in = V.in, \text{settype(id.entry}, V.in)]$

Η συνάρτηση  
settype(x,y)  
ορίζει τον  
τύπο της  
μεταβλητής x  
ως y

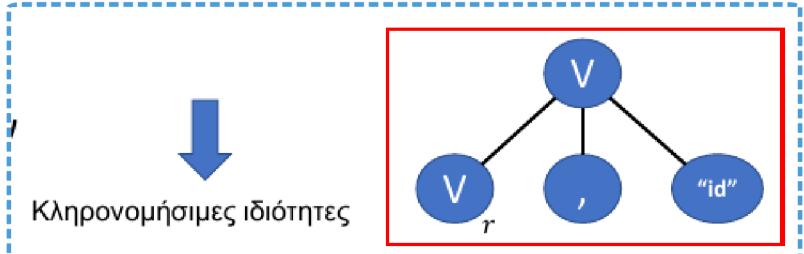
Ο δείκτης R στο σύμβολο  $V_R$  διακρίνει την εμφάνιση του συμβόλου V στο δεξί μέρος του κανόνα, από την εμφάνισή του ίδιου συμβόλου στο αριστερό μέρος.

Με χρήση της συνάρτησης settype έχουμε μεταφορά πληροφορίας, που αναφέρεται στο σύμβολο id (η id.entry προκύπτει με αναζήτηση στον πίνακα συμβόλων), από σύμβολο του αριστερού μέρους του κανόνα (ιδιότητα  $V.in$  που κληρονομείται)

(0)



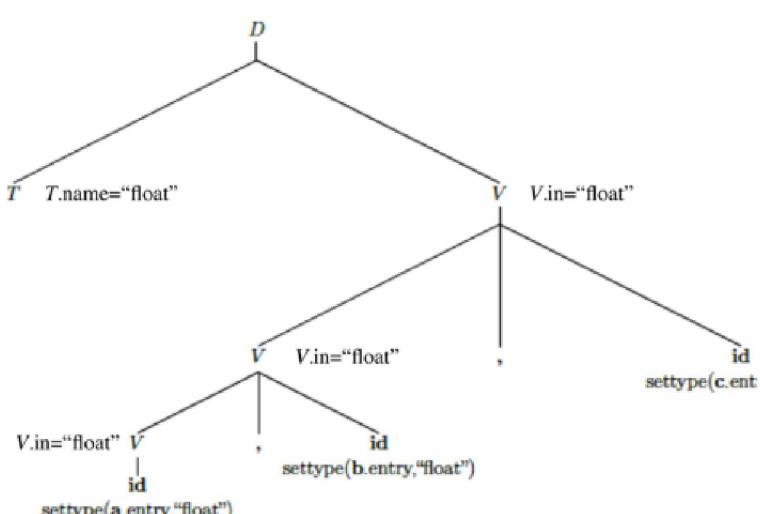
(4)



### Παράδειγμα 2 (συνέχεια)

Οι ενέργειες της σημασιολογικής ανάλυσης για μία συμβολοσειρά συχνά εκφράζεται καταγράφοντας τις ιδιότητες ή τα βήματα υπολογισμού τους στους κόμβους του συντακτικού δέντρου. Ένα τέτοιο δέντρο ονομάζεται δέντρο με σχόλια.

Στο σχήμα απεικονίζεται το συντακτικό δέντρο της συμ/ράς float a, b, c



3o

Οι γραμματικές ιδιοτήτων χρησιμοποιούνται επίσης για την περιγραφή της διαδικασίας ανάπτυξης του συντακτικού δέντρου. Ορίζουμε τις παρακάτω ιδιότητες και συναρτήσεις:

- $E_{nptr}$  αναφέρεται στον κόμβο του δέντρου
- $id.entry$  αναφέρεται στην εγγραφή ενός ονόματος του πίνακα συμβόλων
- $MkNode$  συνάρτηση για την δημιουργία νέου κόμβου και σύνδεση του με απογόνους
- $MkLeaf$  συνάρτηση για την δημιουργία νέου φύλλου

$$\begin{array}{lll}
 (0) & E_{n+1} \rightarrow E_{n-1} '+' E_n & [E_{n+1}.nptr = MkNode('+', E_{n-1}.nptr, E_n.nptr)] \\
 (1) & E_{n+1} \rightarrow E_{n-1} '*' E_n & [E_{n+1}.nptr = MkNode('*', E_{n-1}.nptr, E_n.nptr)] \\
 (2) & E_{n+1} \rightarrow (' E_n ') & [E_{n+1}.nptr = E_n.nptr] \\
 (3) & E_{n+1} \rightarrow "id" & [E_{n+1}.nptr = MkLeaf("id", id.entry)]
 \end{array}$$

4o

Οι γραμματικές ιδιοτήτων χρησιμοποιούνται για την περιγραφή της μετάφρασης, δηλαδή της παραγωγής κώδικα στη γλώσσα - στόχο.

**Παράδειγμα 4 (Σημασιολογικοί κανόνες για τον υπολογισμό της δεκαδικής τιμής ενός δυαδικού αριθμού με πρόσημο)**

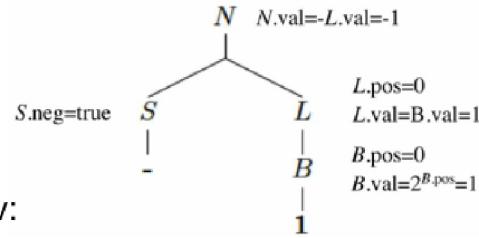
Μη τερματικό σύμβολο  $N$  για τον αριθμό,  $S$  για το πρόσημο του,  $B$  για κάθε bit, που μπορεί να είναι 1 ή 0 και  $L$  για την λίστα των bits από τα οποία αποτελείται ο αριθμός.

$$\begin{array}{llll}
 (0) & N & \rightarrow & S \ L \quad [L.pos = 0, \\
 & & & \text{if}(S.neg) \text{ then} \\
 & & & \quad N.val = -L.val \\
 & & & \text{else} \\
 & & & \quad N.val = L.val] \\
 (1) & S & \rightarrow & '+' \quad [S.neg = \text{false}] \\
 (2) & & | & '-' \quad [S.neg = \text{true}] \\
 (3) & L_n & \rightarrow & L_{n+1} \ B \quad [L_{n+1}.pos = L_n.pos + 1, \\
 & & & \quad B.pos = L_n.pos, \\
 & & & \quad L_n.val = L_{n+1}.val + B.val] \\
 (4) & & | & B \quad [B.pos = L_n.pos, \\
 & & & \quad L_n.val = B.val] \\
 (5) & B & \rightarrow & 0 \quad [B.val = 0] \\
 (6) & & | & 1 \quad [B.val = 2^{B.pos}] \quad 16
 \end{array}$$

Σύμβολο	Ιδιότητες
$B$	$B.val$ (χλευονομήση): η δεκαδική τιμή ενός bit ανάλογα με τη θέση του $B.pos$ (χλευονομήση): η θέση ενός bit, με το δεξιότερο στη θέση 0
$S$	$S.neg$ (συνθέση): true αν ο αριθμός έχει αρνητικό πρόσημο
$L$	$L.val$ (συνθέση): η δεκαδική τιμή της λίστας των bits που διαβάστηκαν $L.pos$ (χλρ.): η θέση του αριστερότερου bit της λίστας που διαβάστηκε
$N$	$N.val$ (συνθέση): η υπολογιζόμενη δεκαδική τιμή του αριθμού

Αν θεωρήσουμε τη συμβολοσειρά εισόδου -1, μία δεξιά παραγωγή σύμφωνα με τους κανόνες της γραμματικής είναι η εξής:

$$\begin{aligned}
 N &\Rightarrow SL \quad [0: N \rightarrow SL] \\
 &\xrightarrow{rm} S B \quad [4: L \rightarrow B] \\
 &\xrightarrow{rm} S 1 \quad [6: B \rightarrow "1"] \\
 &\xrightarrow{rm} -1 \quad [2: S \rightarrow '-' 1] \\
 &\xrightarrow{rm}
 \end{aligned}$$



Μία πιθανή σειρά εκτέλεσης των υπολογισμών:

$$L.\text{pos} \rightarrow S.\text{neg} \rightarrow B.\text{pos} \rightarrow B.\text{val} \rightarrow L.\text{val} \rightarrow N.\text{val}$$

Όχι εφικτός ο υπολογισμός με μία μόνο διάσχιση του δένδρου. Η σειρά εκτέλεσης των υπολογισμών πρέπει να είναι συμβατή με το γράφο εξάρτησης

Θα μπορούσαν να υπάρξουν και άλλες περιπτώσεις εκτέλεσης των υπολογισμών

17

## Γραμματικές Ιδιοτήτων

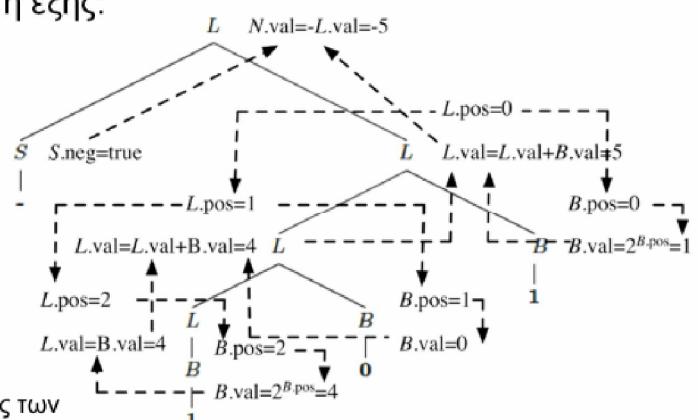
### Παράδειγμα 4 (συνέχεια)

Μερικές εξαρτήσεις έχουν διεύθυνση από πάνω προς τα κάτω (ή από το πλάι) → κληρονομήσιμες ιδιότητες

Μερικές εξαρτήσεις έχουν διεύθυνση από κάτω προς τα πάνω → συνθέσιμες ιδιότητες

Αν θεωρήσουμε τη συμβολοσειρά εισόδου -101, μία δεξιά παραγωγή σύμφωνα με τους κανόνες της γραμματικής είναι η εξής:

$$\begin{aligned}
 N &\Rightarrow SL \quad [0: N \rightarrow SL] \\
 &\xrightarrow{rm} SLB \quad [3: L \rightarrow LB] \\
 &\xrightarrow{rm} SL1 \quad [6: B \rightarrow "1"] \\
 &\xrightarrow{rm} SLB1 \quad [3: L \rightarrow LB] \\
 &\xrightarrow{rm} SL01 \quad [5: B \rightarrow "0"] \\
 &\xrightarrow{rm} SB01 \quad [4: L \rightarrow B] \\
 &\xrightarrow{rm} S101 \quad [6: B \rightarrow "1"] \\
 &\xrightarrow{rm} -101 \quad [2: S \rightarrow '-' 1]
 \end{aligned}$$



Με διακεκομμένη γραμμή εμφανίζονται οι εξαρτήσεις των ιδιοτήτων, που επηρεάζουν τη σειρά εκτέλεσης των πράξεων για τον υπολογισμό τους.

18

## Σειρά Εκτέλεσης Σημασιολογικών Κανόνων

- Έστω ότι όλοι οι σημασιολογικοί κανόνες βρίσκονται (αν δεν βρίσκονται, μπορούν να μετατραπούν) στη μορφή

$$b = f(c_1, c_2, \dots, c_k)$$

, με

- $b$  και  $c_i, i = 1 \dots k$  ιδιότητες της γραμματικής
- Ακόμα και κανόνες που περιγράφουν την κλήση συνάρτησης, μπορούν να μετασχηματιστούν στην παραπάνω μορφή, δημιουργώντας μία νέα συνθέσιμη ιδιότητα  $b$
- Για μία γραμματική ιδιοτήτων, οπου ορίζεται με βάση ΓΧΣ με σύνολο κανόνων  $P$ , κάθε κανόνας  $\pi \in P$  ορίζει ένα γράφημα εξάρτησης πιμών ιδιοτήτων, που αποτελείται από
  - έναν κόμβο, για κάθε ιδιότητα του κανόνα και
  - τόξα  $x \rightarrow y$ , με  $x, y$  ιδιότητες συμβόλων του κανόνα, αν-ν  $y = f(\dots, x, \dots)$ , δηλαδή η ιδιότητα  $y$  εξαρτάται από την  $x$

- Για κάθε συμβολοσειρά που παράγεται από τη γραμματική της γλώσσας, το γράφημα εξάρτησης τιμών ιδιοτήτων του συντακτικού δέντρου, σχηματίζεται από τη συνένωση των γραφημάτων εξάρτησης των κανόνων, για τα σύμβολα των εσωτερικών κόμβων του δέντρου

**Είσοδος:** το συντακτικό δέντρο της ανάλυσης μιας συμβολοσειράς  
**Έξοδος:** το γράφημα εξάρτησης τιμών ιδιοτήτων συμβόλων του δέντρου

- 1: **for all**  $n$  κόμβο του συντακτικού δέντρου **do**
- 2:   **for all**  $c$  ιδιότητα του συμβόλου του κόμβου  $n$  **do**
- 3:     δημιουργείται στο γράφημα ένας νέος κόμβος για την ιδιότητα  $c$ ;
- 4: **for all**  $n$  κόμβο του συντακτικού δέντρου **do**
- 5:   **for all** σημασιολογικό κανόνα  $b = f(c_1, c_2, \dots, c_k)$  για τον κόμβο  $n$  **do**
- 6:     **for all** ιδιότητα  $c_i$  του σημασιολογικού κανόνα **do**
- 7:         δημιουργείται ένα τόξο από τον κόμβο  $c_i$  στον κόμβο  $b$ ;

### Παράδειγμα

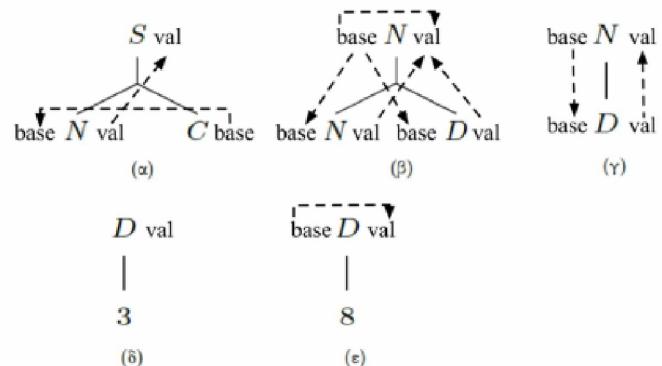
- Γραμματική για αναπαράσταση οκταδικών και δεκαδικών αριθμών, στην μορφή 527ο ή 527d ανάλογα με το αν είναι δεκαδική ή οκταδική αναπαράσταση

(0) $S \rightarrow N C$	$[S.val = N.val, N.base = C.base]$	(5) $D \rightarrow "0"$	$[D.val = 0]$
(1) $C \rightarrow "o"$	$[C.base = 8]$	(6) $  "1"$	$[D.val = 1]$
(2) $  "d"$	$[C.base = 10]$	...	...
(3) $N \rightarrow D$	$[N.val = D.val, D.base = N.base]$	(13) $  "8"$	$[D.val =$ if( $D.base = 8$ ) then error else 8]
(4) $  N_R D$	$[N.val =$ if( $D.val = \text{error} \wedge N_R.val = \text{error}$ ) then error else $N_R.val * N.base + D.val,$ $N_R.base = N.base,$ $D.base = N.base]$	(14) $  "9"$	$[D.val =$ if( $D.base = 8$ ) then error else 9]

12

- Γραφήματα εξάρτησης τιμών ιδιοτήτων, για τους κανόνες:

- (0)  $S \rightarrow N C$  (γράφημα α)
- (4)  $N \rightarrow N D$  (γράφημα β)
- (3)  $N \rightarrow D$  (γράφημα γ)
- (8)  $D \rightarrow "3"$  (γράφημα δ)
- (13)  $D \rightarrow "8"$  (γράφημα ε)



- Αν μπορούμε να αποδείξουμε ότι για κάθε επιμέρους κανόνα της γραμματικής, κάθε πιθανή συμβολοσειρά ακολουθεί του κανόνες των επιμέρους γραφημάτων τότε μπορούμε να κατασκευάσουμε αλγόριθμο βασισμένο σε κανόνες.

13

## Γράφημα Εξάρτησης Τιμών Ιδιοτήτων

- Το **Γράφημα Εξάρτησης Τιμών Ιδιοτήτων** απεικονίζει τη ροή της πληροφορίας μεταξύ των ιδιοτήτων των συμβόλων μίας γραμματικής σε ένα συντακτικό δέντρο
  - Ένα τόξο από μία ιδιότητα προς μία άλλη σημαίνει πως για να υπολογιστεί η τιμή της δεύτερης ιδιότητας, πρέπει πρώτα να έχει υπολογιστεί η τιμή της πρώτης
- Η σειρά εκτέλεσης των ενεργειών σημασιολογικής ανάλυσης μπορεί να καθορίζεται
  - Από τη σειρά με την οποία εφαρμόζονται οι κανόνες της ΓΧΣ κατά τη συντακτική ανάλυση
  - Αν η γραμματική ιδιοτήτων δημιουργεί σύνθετες εξαρτήσεις μεταξύ ιδιοτήτων, θα πρέπει να λαμβάνεται υπόψη το γράφημα εξάρτησης τιμών ιδιοτήτων

Παράδειγμα

### Γραμματικές Ιδιοτήτων

#### Παράδειγμα 4 (συνέχεια)

Αν θεωρήσουμε τη συμβολοσειρά εισόδου  $-101$ , μία δεξιά παραγωγή σύμφωνα με τους κανόνες της γραμματικής είναι η εξής:

```

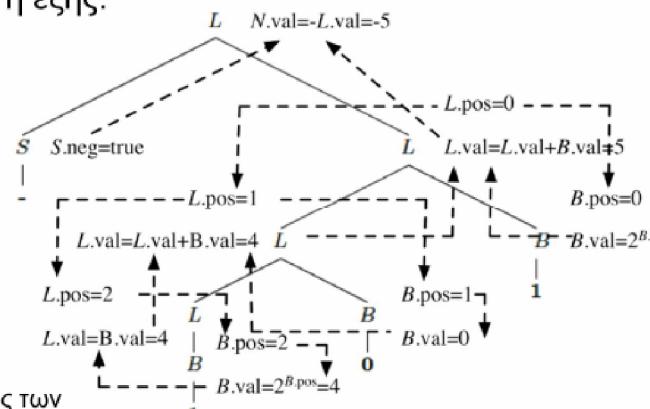
 $N \Rightarrow SL \quad [0: N \rightarrow SL]$ 
 $\text{rm} \Rightarrow S LB \quad [3: L \rightarrow LB]$ 
 $\text{rm} \Rightarrow SL1 \quad [6: B \rightarrow "1"]$ 
 $\text{rm} \Rightarrow SLB1 \quad [3: L \rightarrow LB]$ 
 $\text{rm} \Rightarrow SL01 \quad [5: B \rightarrow "0"]$ 
 $\text{rm} \Rightarrow SB01 \quad [4: L \rightarrow B]$ 
 $\text{rm} \Rightarrow S101 \quad [6: B \rightarrow "1"]$ 
 $\text{rm} \Rightarrow -101 \quad [2: S \rightarrow '-' 1]$ 
 $\text{rm}$ 

```

Με διακεκομμένη γραμμή εμφανίζονται οι εξαρτήσεις των ιδιοτήτων, που επηρεάζουν τη σειρά εκτέλεσης των πράξεων για τον υπολογισμό τους.

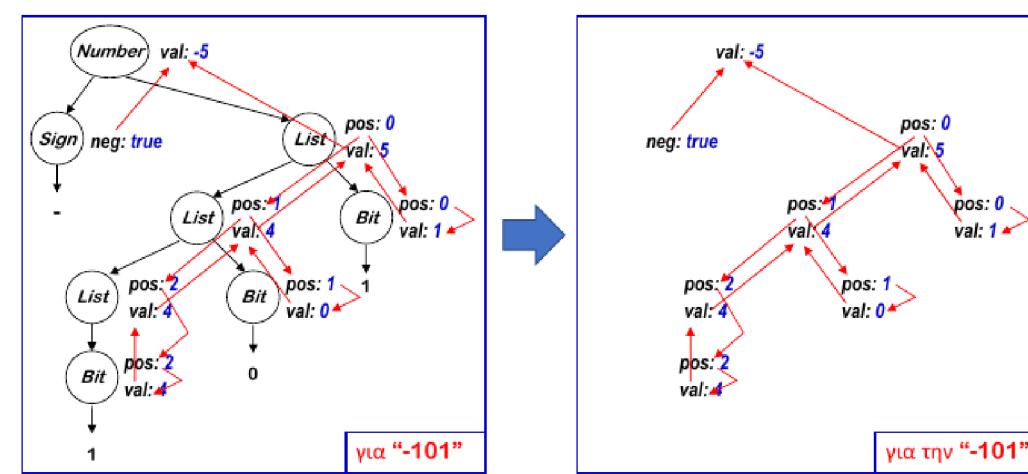
Μερικές εξαρτήσεις έχουν διεύθυνση από πάνω προς τα κάτω (ή από το πλάι) → κληρονομήσμες ιδιότητες

Μερικές εξαρτήσεις έχουν διεύθυνση από κάτω προς τα πάνω → συνθέσιμες ιδιότητες



18

Αν από το προηγούμενο παράδειγμα αφαιρέσουμε το παράγωγο δένδρο ...



## Σχήματα Μετάφρασης

- Ένα **Σχήμα Μετάφρασης** (όπως μία γραμματική ιδιοτήτων) ορίζει σημασιολογικές ενέργειες που επισυνάπτονται στο δεξί μέρος κάθε κανόνα ανάμεσα σε { και }
- Οι ενέργειες εκτελούνται όταν εφαρμόζονται οι αντίστοιχοι κανόνες παραγογής κατά τη συντακτική ανάλυση
- Σε αντίθεση με τις γραμματικές ιδιοτήτων, όπου οι τιμές ιδιοτήτων μπορούν να υπολογιστούν και σε διαφορετικές στιγμές
- Σε ένα σχήμα μετάφρασης πρέπει λοιπόν να διασφαλίζεται ότι οι τιμές ιδιοτήτων υπολογίζονται μόνο με βάση άλλες τιμές που υπολογίστηκαν σε προηγούμενα βήματα
  - Αν πρόκειται για υπολογισμό συνθέσιμης ιδιότητας, τότε εκχωρείται τιμή στην ιδιότητα, στο πλάισιο μίας ενέργειας που γράφεται ανάμεσα σε { και } στο τέλος του δεξιού μέρους ενός κανόνα της γραμματικής
  - Αν πρόκειται για κληρονομήσιμες ιδιότητες πρέπει
    - Κληρονομήσιμη ιδιότητα συμβόλου στο δεξί μέρος κανόνα -> πρέπει να υπολογίζεται σε σημασιολογική ενέργεια που προηγείται του συγκεκριμένου συμβόλου
    - Συνθέσιμη ιδιότητα συμβόλου στο αριστερό μέρος κανόνα -> μπορεί να υπολογιστεί μόνο αφού υπολογιστούν όλες οι ιδιότητες στις οποίες αναφέρεται. Η ενέργεια τοποθετείται στο τέλος του δεξιού μέρους του κανόνα
    - Στον υπολογισμό μίας ιδιότητας πρέπει να γίνεται αναφορά σε συνθέσιμη ιδιότητα συμβόλου, που βρίσκεται στα δεξιά της σημασιολογικής ενέργειας

### Παράδειγμα

#### Παράδειγμα 5 (συνέχεια)

Έχουμε:

τις συνθέσιμες ιδιότητες **num.val**, **id.lexeme**, που αναφέρονται στις τιμές των αριθμών και στα ονόματα των μεταβλητών της έκφρασης,

τις κληρονομήσιμες ιδιότητες **'+'**.lexeme, **'-'**.lexeme, **'\*' .lexeme**, **'/' .lexeme**, που επιστρέφουν τις λεξικές μονάδες των αντίστοιχων τερματικών.

(0)	$E$	$\rightarrow$	$TT_r$
(1)	$T_r$	$\rightarrow$	$'+' T \{print('+' .lexeme)\} T_r$
(2)			$'-' T \{print('-' .lexeme)\} T_r$
(3)			$\epsilon$
(4)	$T$	$\rightarrow$	$FF_r$
(5)	$F_r$	$\rightarrow$	$'*' F \{print('*' .lexeme)\} F_r$
(6)			$'/' F \{print('/' .lexeme)\} F_r$
(7)			$\epsilon$
(8)	$F$	$\rightarrow$	$(' E ')$
(9)			$"num" \{print(num.val)\}$
(10)			$"id" \{print(id.lexeme)\}$

Το σχήμα μετάφρασης του παραδείγματος μετατρέπει αριθμητικές εκφράσεις ένθετης μορφής (infix notation) στην αντίστοιχη επιθεματική μορφή (postfix). Η έκφραση

$$((a+b)^*c)/(d-e^*f)+3$$

δίνει ως αποτέλεσμα την

$$ab+c^*def^*- /3+$$

# Μέθοδοι Σημασιολογικού Ελέγχου

## Ανοδική Ανάλυση

- Οι *S-Γραμματικές Ιδιοτήτων* μπορούν να εφαρμοστούν άμεσα κατά την ανοδική ανάλυση, όπου οι υπολογισμοί των ιδιοτήτων γίνονται κατά τις απλοποιήσεις
  - Για την αποθήκευση και προσπέλαση των τιμών ιδιοτήτων, μπορεί να χρησιμοποιηθεί και η στοίβα ανάλυσης

## DFS

- Οι *L-Γραμματικές Ιδιοτήτων* επιτρέπουν τη σημασιολογική ανάλυση εκτελώντας την παρακάτω παραλλαγή του αλγόριθμου DFS στο συντακτικό δέντρο

**Είσοδος:** το συντακτικό δέντρο της ανάλυσης μιας συμβολοσειράς

**Έξοδος:** οι τιμές των ιδιοτήτων στους κόμβους του δέντρου

Καλείται η DFSVISIT για τη ρίζα του δέντρου.

**procedure** DFSVISIT (*n* : κόμβος δέντρου)

1: **for all** *m* απόγονο του *n* από τ' αριστερά προς τα δεξιά **do**

2: υπολόγισε τις κληρονομήσιμες ιδιότητες του *m*;

3: DFSVISIT(*m*);

4: υπολόγισε τις συνθέσιμες ιδιότητες του *n*;

**end procedure**

## Μετάφραση Προσανατολισμένη στη Σύνταξη

- Κατά τη μετάφραση προσανατολισμένη στη σύνταξη, η μετάφραση της πηγαίας γλώσσας καθοδηγείται εξ' ολοκλήρου από τη συντακτική ανάλυση
- Εκτελείται σε τρία στάδια



αλλά όχι πάντα με τη σειρά που φαίνεται

## Πίνακας Συμβόλων

- Ο προγραμματιστής επιλέγει ονόματα για διάφορες οντότητες, που πρόκειται να επεξεργαστεί ο Μεταγλωττιστής
  - σταθερές
  - τύποι
  - ετικέτες εντολών
  - συναρτήσεις
  - μεταβλητές (περιοχή δεδομένων που περιέχει μία απλή ή δομημένη τιμή)
  - αρχεία, συσκευές
  - μακροεντολές
- Οι οντότητες αυτές περιγράφονται από ένα σύνολο ιδιοτήτων όπως
  - όνομα
  - εμβέλεια
  - τύπος
  - τιμή
  - μέγεθος (χώρος μνήμης)
  - κα.

- Ο μεταγλωττιστής συνδέει τα ονόματα των οντοτήτων με τις ιδιότητές τους μέσω μίας δομής που ονομάζεται κόμβος συμβόλου
  - **Κόμβος Συμβόλου:** μία εγγραφή που ως πεδία έχει τις ιδιότητες της οντότητας

## Γενική Δομή και Λειτουργείες

- Σχηματίζεται κατά την συντακτική ανάλυση και χρησιμοποιείται ξανά για το στατικό σημασιολογικό έλεγχο, τη δέσμευση χώρου μνήμης και τη δημιουργία κώδικα
  - Παρμένει στην κύρια μνήμη για να χρησιμοπιθεί σε κάθε μία από αυτές τις επεξεργασίες
  - Κάθε αναφορά σε όνομα του πηγαίου κώδικα έχει ως αποτέλεσμα την προσπέλασή του, για την εύρεση του κόμβου συμβόλου που αντιστοιχεί στο όνομα, με σκοπό την καταχώρηση νέας τιμής ή την ανάκτηση της υπάρχουσας
  - Πρέπει να επιτρέπει την **ταχεία προσπέλαση** των καταχωρήσεων
- Η οργάνωσή του ακολουθεί τη δομή *block* των προγραμμάτων
  - Αυτό εξηπηρετεί τη γρήγορη εύρεση της εγγραφής της δήλωσης που αντιστοιχεί σε ένεα όνομα, όταν αυτό συναντιέται μέσα σε ένα πρόγραμμα
  - Για να υλοποιηθεί αυτή η λειτουργεία, ο πίνακας συμβόλων υποστηρίζει τις **εξής συναρτήσεις**
    - **create\_table()**: δημιουργεί άδειο πίνακα συμβόλων
    - **bind(id)**: συνδέει το *id* με μία εγγραφή στον πίνακα. Αν το *id* υπάρχει, τότε δίνεται προτεραιότητα στη νέα σύνδεση ως προς την υπάρχουσα
    - **lookup(id)**: αναζητεί το *id* στον πίνακα και επιστρέφει την εγγραφή του αν υπάρχει, αλλιώς επιστρέφει *NULL*
    - **enter\_block()**: εγγράφει την είσοδο σε νέο *block*
    - **exit\_block()**: αποκαθιστά τον πίνακα στην κατάσταση της τελευταίας κλήσης της *enter\_block()*, δλδ. διαγράφει τις εγγραφές, για τις δηλώσεις του *block* που κλείνει. Έτσι κάθε στιγμή ο πίνακας συμβόλων περιέχει τις εγγραφές δηλώσεων όλων των ανοικτών *block* που δεν έχουν κλείσει ακόμα
- Όταν κατά την ανάλυση ονόματος προσεγγίζεται μία χρήση ονόματος στο πρόγραμμα, τότε αναζητείται στον πίνακα συμβόλων η εγγραφή για την αντίστοιχη δήλωσή του, σύμφωνα με τους κανόνες εμβέλειας και ορατότητας της γλώσσας
  - Με την εύρεση της εγγραφής ανασύρονται όλες οι πληροφορίες για τις ιδιότητες που έχουν συνδεθεί στη δήλωση του ονόματος
- Η υλοποίηση του πίνακα συμβόλων πρέπει να εγγυάται ότι η *lookup(id)* επιλέγει κάθε φορά τη σωστή εγγραφή για το *id*, σύμφωνα με τους κανόνες εμβέλειας.

## Υλοποίηση Στοίβας για Block

- Μία δυνατότητα είναι να διατηρούνται όλα τα ανοικτά blocks σε μία στοίβα. Για κάθε block, που εισέρχεται η ανάλυση του προγράμματος, υπάρχει μία δομή αντιστοίχισης, που αντιστοιχίζει τα ονόματα που βρίσκει στην εγγραφή της σωστής δήλωσης. Η συνάρτηση `enter_block()` εισάγει στη στοίβα μία νέα (κενή) δομή αντιστοίχισης, ενώ η `exit_block()` αφαιρεί από τη στοίβα τη δομή αντιστοίχισης που βρίσκεται στην κορυφή της

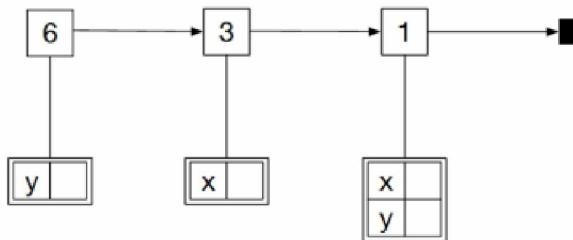
## Παράδειγμα

```
1 double x;
2 double y;

4 void p(){
5     int x;
6     ...
7     { int y[100];
8         ...
9     }
10    ...
11 }

13 void q(){
14     char y;
15     ...
16 }

18 main(){
19     int x;
20     ...
21 }
```



**Η look up (id )** αναζητά το id στη δομή αντιστοίχισης, που βρίσκεται στην κορυφή της στοίβας. Αν δε βρεθεί εκεί, τότε η αναζήτηση συνεχίζεται στη δομή αντιστοίχισης, που βρίσκεται από κάτω, για το block στο οποίο περικλείεται το ένθετο block. Η διαδικασία αναζήτησης μπορεί να συνεχίζεται μέχρι να προσεγγιστεί η δομή αντιστοίχισης για το εξωτερικό block και αποτυγχάνει, μόνο αν δε βρεθεί εγγραφή για το id ούτε εκεί. Έτσι υλοποιείται ο κανόνας εμβέλειας του πλησιέστερου block.

- Η είσοδος και η έξοδος από ένα block δεν είναι κρίσιμες για την αποδοτικότητα της δομής του πίνακα συμβόλων. Από την άλλη, η αποδοτικότητα της *lookup(id)* εξαρτάται από τις δομές, που επιλέγονται για την υλοποίηση των δομών αντιστοίχισης. Συνήθως χρησιμοποιούνται:
    - **Συνδεδεμένες Λίστες:** ο χρόνος εκτέλεσης της *lookup(id)* μέσα σε κάθε block αυξάνεται γραμμικά με τον αριθμό των ονομάτων, που δηλώνονται σ' αυτό
    - **Δεντρα Δυαδικής Αναζήτησης:** ο χρόνος εκτέλεσης αυξάνεται λογαριθμικά με τον αριθμό των ονομάτων
    - **Hash Maps:** οι χρόνοι αναζήτησης είναι θεωρητικά σταθεροί

## Υλοποίηση Στοίβας για Όνομα

- Ανεξάρτητα από τη δομή αντιστοίχισης, ο χρόνος εκτέλεσης της `lookup(id)` αυξάνεται γραμμικά ως προς το βάθος ένθεσης των `blocks`.
  - Για μία γλώσσα προγραμματισμού που στα προγράμματα της έχουμε ένθεση σε πολλά επίπεδα, τότε είναι προτιμότερη μία υλοποίηση πίνακα συμβόλων, που ο χρόνος εκτέλεσης της `lookup(id)` είναι ανεξάρτητος από το βάθος ένθεσης.
  - Μία εναλλακτική υλοποίηση οργανώνει τον πίνακα συμβόλων έτσι ώστε να αναθέτει σε κάθε όνομα μία στοίβα από εγγραφές για αντίστοιχες δηλώσεις. Η στοίβα για το όνομα `x`, περιέχει σε μία δεδομένη στιγμή τις εγγραφές για όλες τις έγκυρες δηλώσεις του `x` στα ένθετα `blocks` του προγράμματος. Για κάθε νέα δήλωση του `x` δημιουργείται νέα εγγραφή, που εισάγεται στη στοίβα. Έτσι, η αναζήτηση εγγραφής για την τελευταία δήλωση του `x` έχει σταθερό κόστος σε χρόνο.
  - Όμως αυξάνει η πολυπλοκότητα της `exit_block()`: πρέπει να αφαιρεθούν από τον πίνακα όλες οι εγγραφές για δηλώσεις σ' αυτό το `block`. Αυτό υλοποιείται εύκολα αν για κάθε `block` έχουμε σε ξεχωριστή λίστα όλα τα ονόματα, που δηλώνονται σ' αυτό.

## Παράδειγμα

- Πίνακας συμβόλων με μία στοίβα ανά όνομα για το παρακάτω πρόγραμμα (γραμμή 7)

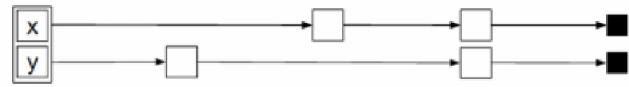
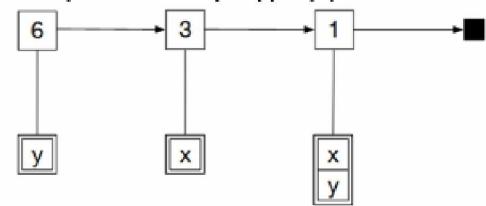
```

1 double x;
2 double y;

4 void p(){
5     int x;
6     ...
7     { int y[100];
8     ...
9     }
10 ...
11 }

13 void q(){
14     char y;
15     ...
16 }

18 main(){
19     int x;
20     ...
21 }
```



Με την κλήση της `enter_block()` δημιουργείται μία νέα κενή λίστα στην κορυφή της στοίβας. Όταν στη συνέχεια καλείται η `bind(x)` καταγράφεται το όνομα `x` στη λίστα του τρέχοντος block και εισάγονται οι συνδέσεις της δήλωσης για το `x` στη στοίβα με τις εγγραφές για το `x`. Η κλήση της `exit_block()` διατρέχει την λίστα όλων των ονομάτων, που δηλώνονται στο τρέχον block και αφαιρεί για το κάθε όνομα την εγγραφή της δήλωσης από την<sub>21</sub> αντίστοιχη στοίβα

- Πίνακας συμβόλων με μία στοίβα ανά όνομα για το παρακάτω πρόγραμμα (γραμμή 7)

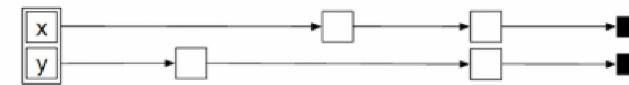
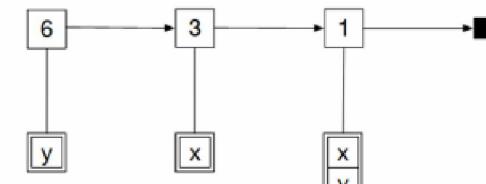
```

1 double x;
2 double y;

4 void p(){
5     int x;
6     ...
7     { int y[100];
8     ...
9     }
10 ...
11 }

13 void q(){
14     char y;
15     ...
16 }

18 main(){
19     int x;
20     ...
21 }
```



Τα επιπλέον κόστη για τη διατήρηση της στοίβας των blocks μπορούν να κατανεμηθούν στις επιμέρους κλήσεις της `bind()` με συνέπεια μία (μικρή μόνο) αύξηση κατά μία σταθερή ποσότητα χρόνου.

22

- Η κλήση της `exit_block()` διατρέχει την λίστα όλων των ονομάτων, που δηλώνονται στο τρέχον block και αφαιρεί για το κάθε όνομα την εγγραφή της δήλωσης από την αντίστοιχη στοίβα.
- Μετά αφαιρείται η λίστα των ονομάτων από τη στοίβα των blocks.
- Τα επιπλέον κόστη για τη διατήρηση της στοίβας των blocks μπορούν να κατανεμηθούν στις επιμέρους κλήσεις της `bind()` με συνέπεια μία (μικρή μόνο) αύξηση κατά μία σταθερή ποσότητα χρόνου.

## Σημασιολογική Ανάλυση για Έλεγχο Τύπων

### Παράδειμα

- Η σημασιολογική ανάλυση μπορεί να χρησιμοποιηθεί και για τον έλεγχο τύπων. Έστω η (ασαφής) γραμματική :
- Κάθε πρόγραμμα αποτελείται από μία σειρά δηλώσεων συναρτήσεων (κανόνες 0, 1 και 2), που είναι αναδρομικές μεταξύ τους.
- Κάθε δήλωση συνάρτησης προσδιορίζει τον τύπο του αποτελέσματός της, καθώς επίσης τους τύπους και τα ονόματα των παραμέτρων (κανόνας 3).
- Οι πιθανοί τύποι (κανόνες 4, 5) είναι int και bool (boolean).
- Το σώμα μιας συνάρτησης είναι μία έκφραση (κανόνας 3), που μπορεί να είναι ακέραιη σταθερά, το όνομα μιας μεταβλητής, έκφραση αθροίσματος, σύγκριση, υποθετική εντολή, κλήση συνάρτησης ή έκφραση με μία τοπική δήλωση (κανόνες 8 - 14).

(0)	prog	→	flist
(1)	flist	→	funflist
(2)			fun
(3)	fun	→	decl `(` dlist `)` `(` expr `)`
(4)	decl	→	“int” “id”
(5)			“bool” “id”
(6)	dlist	→	decl `(` , `)` dlist
(7)			decl
(8)	expr	→	“num”
(9)			“id”
(10)			expr `+` expr
(11)			expr `<` expr
(12)			“if” expr “then” expr “else” expr
(13)			“id” `(` elist `)`
(14)			“let” “id” `=’ expr “in” expr
(15)	elist	→	expr
(16)			expr `;` elist

24

- Για τον έλεγχο τύπων, χρειαζόμαστε δύο πίνακες συμβόλων, που θα διατηρούν τις συνδέσεις τύπων των μεταβλητών και των συναρτήσεων.
- Σε μία μεταβλητή μπορεί να συνδέεται ένας από τους δύο πιθανούς τύπους (int ή bool). Ο τύπος που συνδέεται σε ένα όνομα συνάρτησης αποτελείται από την λίστα των τύπων των παραμέτρων της και από τον τύπο του αποτελέσματός της.
- Για παράδειγμα, ο τύπος συνάρτησης (int,bool) →int σημαίνει ότι δέχεται δύο παραμέτρους (τύπου int και bool αντίστοιχα) και επιστρέφει έναν ακέραιο.
- Ο πίνακας συμβόλων με τα ονόματα των μεταβλητών αναφέρεται ως **vtable**, ενώ ο πίνακας συμβόλων με τα ονόματα των συναρτήσεων είναι ο **ftable**. Όταν εντοπίζεται λάθος τύπων καλείται η συνάρτηση **error()**, που το αναφέρει, αλλά επιστρέφει τον έλεγχο με μία return, για να μπορεί να συνεχιστεί ο έλεγχος τύπων.

## Γραμματική Ιδιοτήτων για Έλεγχο Τύπων

### Παράδειγμα

- Έκφραση για αριθμό: η ιδιότητα **expr.type** παίρνει την τιμή int (κανόνας 8).

$$(8) \text{expr} \rightarrow \text{“num”} \quad [\text{expr.type} = \text{int};]$$

- Έκφραση για όνομα (κανόνας 9): Γίνεται αναζήτηση του ονόματος στον πίνακα vtable. Αν δε βρεθεί, τότε επιστρέφεται η τιμή unknown, που σημαίνει ότι δεν έχει δηλωθεί μεταβλητή με το συγκεκριμένο όνομα. Αναφέρεται λάθος και ο έλεγχος τύπων συνεχίζει την ανάλυση αφού δώσει στην **expr.type** (αυθαίρετα) την τιμή int. Αν το όνομα βρεθεί, τότε η **expr.type** παίρνει την τιμή, που αντιστοιχεί στον τύπο της μεταβλητής.

$$(9) \text{expr} \rightarrow \text{“id”}$$

```
[t = lookup(vtable, id.entry),
expr.type =
if(t = unknown) then
    error(); int
else
    t ]
```

27

- Έκφραση αθροίσματος (κανόνας 10): περίπτωση για την οποία απαιτείται οι δύο τελεστέοι να είναι int και αν αυτό συμβαίνει, η expr.type γίνεται επίσης int.

$$(10) \text{expr} \rightarrow \text{expr}_1 + \text{expr}_2 \quad [\text{expr.type} = \\ \text{if}(\text{expr}_1.\text{type} = \text{int} \text{ και} \\ \text{expr}_2.\text{type} = \text{int}) \text{then} \\ \text{int} \\ \text{else} \\ \text{error()}; \text{int}]$$

- Έκφραση σύγκρισης (κανόνας 11): απαιτείται και οι δύο τελεστέοι να είναι του ίδιου τύπου ή αλλιώς εντοπίζεται λάθος τύπων. Σε κάθε περίπτωση, η expr.type γίνεται bool.

$$(12) \text{expr} \rightarrow \text{"if"} \text{ } \text{expr}_1 \text{ "then"} \text{ } \text{expr}_2 \text{ "else"} \text{ } \text{expr}_3 \quad [\text{expr.type} = \\ \text{if}(\text{expr}_1.\text{type} = \text{bool} \text{ και} \\ \text{expr}_2.\text{type} = \text{expr}_3.\text{type}) \text{then} \\ \text{expr}_2.\text{type} \\ \text{else} \\ \text{error()}; \text{expr}_2.\text{type}]$$

28

- Στις δηλώσεις συναρτήσεων ορίζονται ρητά οι τύποι των παραμέτρων τους. Η πληροφορία αυτή χρησιμοποιείται, για να κατασκευαστεί ένας πίνακας συμβόλων για μεταβλητές, στον οποίο στηρίζεται ο έλεγχος τύπων του σώματος κάθε συνάρτησης. Στη γραμματική ιδιοτήτων, που ακολουθεί, η Check<sub>dlist</sub>(dlist) κατασκευάζει τον πίνακα vtable, για την λίστα των παραμέτρων dlist και ελέγχει για κάθε όνομα ότι δεν έχει προηγηθεί καταχώρησή του στον πίνακα. Ο τύπος του σώματος της συνάρτησης πρέπει να ταιριάζει με τον τύπο αποτελέσματος, που δηλώθηκε για τη συνάρτηση

$$(3) \text{fun} \rightarrow \text{decl} ('( \text{dlist} ')') \quad [((f, t_0) = (\text{decl.name}, \text{decl.type}), \\ \{' \text{expr}'\} \quad \text{vtable} = \text{Check}_{\text{dlist}}(\text{dlist}), \\ \quad t_1 = \text{expr.type}, \\ \quad \text{if}(t_0 \neq t_1) \text{then} \\ \quad \text{error()})]$$

## Διαδικασίες και Περιβάλλον Εκτέλεσης

### Περιβάλλον Εκτέλεσης

- Οι διαδικασίες και οι συναρτήσεις είναι αφαιρετικές προγραμματιστικές δομές, που μπορεί να διαφέρουν σημαντικά από γλώσσα σε γλώσσα. Αυτές οι υψηλού επιπέδου προγραμματιστικές αφαιρέσεις απεικονίζονται σε δομές δεδομένων, που τις υλοποιούν, στο επίπεδο της μηχανής εκτέλεσης.
- Πολλές γλώσσες προγραμματισμού διακρίνουν σε επίπεδο σύνταξης τις διαδικασίες από τις συναρτήσεις:
  - οι **συναρτήσεις** θα πρέπει να υπολογίζουν και να επιστρέφουν μόνο μία τιμή
  - ενώ οι **διαδικασίες** δεν επιστρέφουν τιμή στο σημείο που έχουν κληθεί
- Έτσι, εκτός από τις συνδέσεις των ονομάτων ενός προγράμματος, είναι επίσης απαραίτητη η χρήση ενός συνόλου δομών δεδομένων (π.χ. στοίβας, στατικών δεδομένων κ.α.), για την εκτέλεση των διαδικασιών και των συναρτήσεων.
- Περιβάλλον Εκτέλεσης:**ένα σύνολο δομών δεδομένων, που συντηρείται κατά την εκτέλεση ενός προγράμματος, για να υλοποιεί υψηλού επιπέδου προγραμματιστικές αφαιρέσεις.
  - Το περιβάλλον εκτέλεσης αναλαμβάνει την οργάνωση (οριοθέτηση) και τη διαχείριση της μνήμης του προγράμματος, την επικοινωνία με άλλες διεπαφές όπως το λειτουργικό σύστημα, τον μηχανισμό πρόσβασης του κώδικα στις μεταβλητές και γενικά την πιστή εκτέλεση του προγράμματος.

- Οι διαφορές στην έννοια της διαδικασίας, μεταξύ των γλωσσών προγραμματισμού, έχουν σημαντικό αντίκτυπο στη δομή και στην πολυπλοκότητα του περιβάλλοντος εκτέλεσης.
- Σε σχέση με το περιβάλλον εκτέλεσης, η κλήση μιας διαδικασίας/συνάρτησης σχετίζεται με την έννοια της εγγραφής ενεργοποίησης.
- **Εγγραφή Ενεργοποίησης:** Η ενεργοποίηση μιας διαδικασίας/συνάρτησης αντιστοιχεί σε μία κλήση της. Η εγγραφή ενεργοποίησης (activation record) είναι μία δομή δεδομένων, στην οποία διατηρούνται όλα τα απαραίτητα δεδομένα για την κλήση, καθώς και όλες οι πληροφορίες ροής ελέγχου εκτέλεσης

### Παραδείγματα

#### Παράδειγμα κλήσης των διαδικασιών σε μία υποθετική γλώσσα προγραμματισμού

- Μέσα στο block  $b_1$  ορίζεται η τοπική μεταβλητή  $Z$  και η διαδικασία  $P$  (δήλωση και σώμα) με παραμέτρους τα  $X$  και  $Y$ . Ο ορισμός της  $P$  οριοθετεί το συγκείμενο  $CE_P$  (block  $b_1$ ) μέσα στο οποίο παίρνουν τιμές οι ελεύθερες μεταβλητές της διαδικασίας  $P$ , δηλαδή  $CE_P = \{Z \rightarrow 1\}$
- Η κλήση της διαδικασίας  $P$  μέσα στο block  $b_2$  δημιουργεί την εγγραφή ενεργοποίησης που περιέχει:

- τις συνδέσεις με τις ιδιότητες, που περιέχονται στο  $CE_P$
- τις συνδέσεις των τυπικών παραμέτρων (formal parameters) της  $P$  ( $X, Y$ ) με τις παραμέτρους της κλήσης (actual parameters), που στο πρόγραμμα είναι οι  $A$  και  $B$

```

 $b_0: begin [local P]$ 
 $b_1: begin [local Z]$ 
 $Z=1;$ 
 $proc P (X Y)$ 
 $begin$ 
 $Y=X+Z;$ 
 $end$ 
 $end b_1$ 

 $b_2: begin [local A B]$ 
 $A=3;$ 
 $P (A B);$ 
 $end b_2$ 
 $end b_0$ 

```

#### Παράδειγμα κλήσης των διαδικασιών σε μία υποθετική γλώσσα προγραμματισμού

- Η εγγραφή ενεργοποίησης της  $P$  είναι η  $E_P = \{X \rightarrow 3, Y \rightarrow B\} \cup CE_P$
- Η ροή εκτέλεσης του προγράμματος μεταφέρεται στην πρώτη εντολή του σώματος της διαδικασίας και επιστρέφει στο σημείο της κλήσης της στο block  $b_2$ , όταν ολοκληρωθεί η εκτέλεσή της. Τότε το  $B$  παίρνει την τιμή που έχει υπολογιστεί στο σώμα της  $P$ , που είναι 4.
- Είναι επίσης πιθανό η ροή εκτέλεσης του προγράμματος να επιστρέψει στο σημείο, που κλήθηκε η διαδικασία, πριν ακόμη ολοκληρωθούν όλες οι εντολές του σώματός
- Αυτό γίνεται με χρήση εντολών που δηλώνουν την επιστροφή στο σημείο κλήσης. Για παράδειγμα, στη C++, χρησιμοποιείται η εντολή `return`

```

 $b_0: begin [local P]$ 
 $b_1: begin [local Z]$ 
 $Z=1;$ 
 $proc P (X Y)$ 
 $begin$ 
 $Y=X+Z;$ 
 $end$ 
 $end b_1$ 

 $b_2: begin [local A B]$ 
 $A=3;$ 
 $P (A B);$ 
 $end b_2$ 
 $end b_0$ 

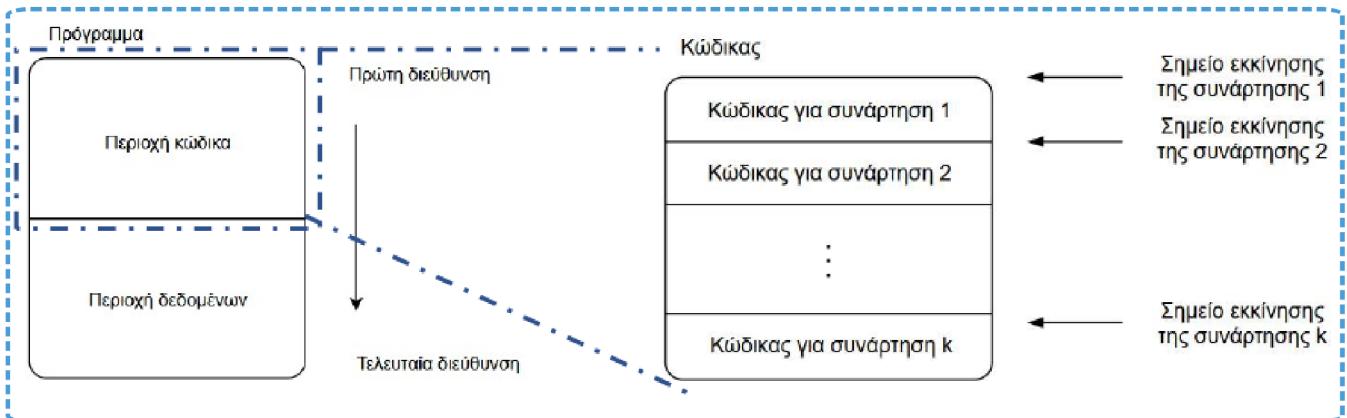
```

- Το περιβάλλον εκτέλεσης αναλαμβάνει τη διαχείριση του κώδικα, που παράγεται από τον μεταγλωττιστή, ο οποίος κατανέμεται σε συναρτήσεις και το μέγεθος του είναι γνωστό πριν την εκτέλεση του προγράμματος.
- Το περιβάλλον εκτέλεσης διαχειρίζεται και τα δεδομένα του προγράμματος, όπως είναι οι καθολικές σταθερές, οι τοπικές μεταβλητές, οι μεταβλητές που δημιουργούνται δυναμικά, καθώς και τις ενεργοποίήσεις των διαδικασιών και των συναρτήσεων.
- Όταν ένα πρόγραμμα φορτώνεται για να εκτελεστεί, συνήθως το λειτουργικό σύστημα είναι αυτό, που δεσμεύει χώρο στην μνήμη για την εκτέλεσή του. Στη συνέχεια, ο έλεγχος της εκτέλεσης μεταφέρεται στην πρώτη εντολή του προγράμματος (π.χ. στη συνάρτηση `main`).

- Οι εντολές του κώδικα που παράγει ο μεταγλωττιστής έχουν κατά την εκτέλεση έχουν τις δικές τους λογικές διευθύνσεις μνήμης (logical addresses) και η μηχανή εκτέλεσης αναλαμβάνει την αντιστοίχισή τους σε συγκεκριμένες φυσικές διευθύνσεις μνήμης (physical addresses) του συστήματος

## Γενική Απεικόνιση Μνήμης Προγράμματος

- Η γενική οργάνωση της μνήμης ενός προγράμματος, που χωρίζεται σε δύο κύριες κατηγορίες, την περιοχή του παραγόμενου κώδικα και την περιοχή των δεδομένων.
- Στο σχήμα φαίνεται η μνήμη που δεσμεύεται για το πρόγραμμα να αποτελείται από διαδοχικά bytes, αυτό όμως δεν είναι απαραίτητο να συμβαίνει στην πραγματικότητα

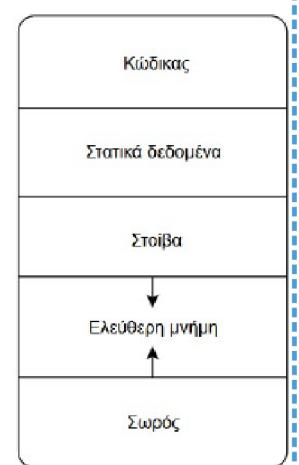


## Σχετική Διεύθυνση

- το μέγεθος του κώδικα είναι συγκεκριμένο και γνωστό πριν από την εκτέλεση του προγράμματος: μπορεί να γίνει ανάθεση σχετικών διευθύνσεων στις εντολές του κώδικα κατά την μεταγλωττιση και διασύνδεση.
- **Σχετική διεύθυνση (symbolic address)**: μία έκφραση που αποτελείται από μία διεύθυνση μνήμης (βάση) και έναν αριθμό offset, που εκφράζει την απόσταση από αυτήν. Ο υπολογισμός της διεύθυνσης από τη βάση και την απόσταση δίνει ως αποτέλεσμα μία λογική διεύθυνση μνήμης.
  - Έτσι, ανεξαρτήτως από τη θέση που τοποθετείται το πρόγραμμα στην μνήμη, μπορούν να υπολογιστούν όλες οι πραγματικές τιμές, που αντιστοιχούν στις σχετικές διευθύνσεις (relocatability)

## Περιοχή Δεδομένων

- Η περιοχή δεδομένων ξεκινάει από την περιοχή των στατικών δεδομένων, όπου αποθηκεύονται αντικείμενα όπως καθολικές σταθερές και άλλες πληροφορίες του μεταγλωττιστή, των οποίων το μέγεθος τους είναι γνωστό πριν από την εκτέλεση του κώδικα.
- Για τα δυναμικά δεδομένα που δημιουργούνται, χρησιμοποιείται η στοίβα και ο σωρός, δύο υποπεριοχές που το μέγεθος τους μπορεί να αλλάζει, καθώς εκτελείται το πρόγραμμα.
- Η στοίβα χρησιμοποιείται κατά την ενεργοποίηση των διαδικασιών και των συναρτήσεων, ενώ στο σωρό δεσμεύεται χώρος, όταν δημιουργείται ένα αντικείμενο και αποδεσμεύεται, όταν αυτό πλέον δεν είναι προσπελάσιμο.
- Η διαχείριση της μνήμης του σωρού διαφέρει από γλώσσα σε γλώσσα, για παράδειγμα στη C η δέσμευση και η αποδέσμευση μνήμης επιτυγχάνεται με τη ρητή κλήση συναρτήσεων (malloc και free, αντίστοιχα), ενώ στη Java, η αποδέσμευση της μνήμης πραγματοποιείται από το συλλέκτη απορριμμάτων.



## Διαδικασίες σε Περιβάλλοντα με Στοίβες

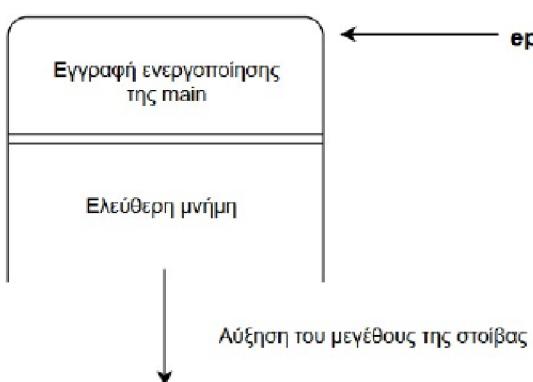
- Με την ενεργοποίηση μιας διαδικασίας/συνάρτησης, εισάγεται στη στοίβα ένα πλαίσιο και παραμένει εκεί μέχρι της ολοκλήρωση της εκτέλεσής της και άρα ο χρόνος ζωής μιας τοπικής μεταβλητής ισούται με τη διάρκεια εκτέλεσης της διαδικασίας/συνάρτησης.
- Όταν όμως γίνεται χρήση αναδρομής, χρειάζεται και ένας δείκτης για την τρέχουσα ενεργοποίηση, αφού για την κάθε διαδικασία το πλαίσιο στοίβας δε βρίσκεται σε μία σταθερή θέση και η θέση του μπορεί να μεταβάλλεται κατά την εκτέλεση του προγράμματος. Ο δείκτης της τρέχουσας ενεργοποίησης διατηρείται σε μία σταθερή θέση συνήθως μέσα σε ένα μητρώο και ονομάζεται δείκτης περιβάλλοντος (environment pointer ερ).
- Επιπλέον, χρειάζεται ένας ακόμη δείκτης στο πλαίσιο στοίβας του block από το οποίο προέκυψε η τρέχουσα ενεργοποίηση. Όταν αυτή τερματίζεται, το αντίστοιχο πλαίσιο πρέπει να αφαιρεθεί από την στοίβα, που σημαίνει ότι το ερ θα πρέπει να αποκατασταθεί, για να δείχνει στο προηγούμενο πλαίσιο.

### Παράδειγμα

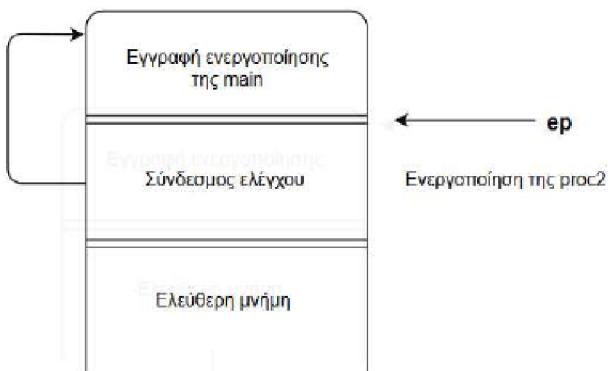
```

1 void proc1(void)
2 { ... }
3
4 void proc2(void)
5 {
6     proc1();
7     ...
8 }
9
10 main()
11 {
12     proc2();
13     ...
14 }
```

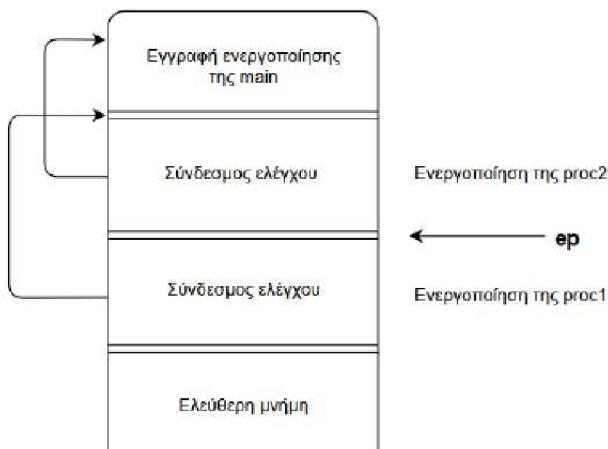
στην αρχή της εκτέλεσης του προγράμματος υπάρχει μόνο μία εγγραφή ενεργοποίησης για την `main` και ο ερ δείχνει σ' αυτήν.



Με την εκτέλεση της συνάρτησης proc2 έχουμε ένα νέο πλαίσιο, το οποίο εισάγεται στη στοίβα του περιβάλλοντος εκτέλεσης,



Με την εκτέλεση της proc1, προστίθεται ένα επιπλέον πλαίσιο στοίβας. Τα δεδομένα της στοίβας ολοένα και αυξάνονται, ενώ εμπλουτίζονται με νέες πληροφορίες που αφορούν δεδομένα ενεργοποίησης



## Μηχανισμοί Μετάδοσης Παραμέτρων

- Οι γλώσσες προγραμματισμού διαφέρουν μεταξύ τους στα είδη των μηχανισμών μετάδοσης παραμέτρων που υποστηρίζουν και το εύρος των επιτρεπτών αποτελεσμάτων, που ενδέχεται να προκύψουν από την εφαρμογή τους.
- Στις γλώσσες προγραμματισμού, που διατίθεται μόνο ένας μηχανισμός μετάδοσης παραμέτρων, χρησιμοποιούνται τεχνικές έμμεσης μετάδοσης παραμέτρων για να «μιμηθούν» ή να προσομοιώσουν τους άλλους μηχανισμούς.

## Μετάδοση με Τιμή

- Αποτελεί τον πιο συνηθισμένο μηχανισμό για την μετάδοση παραμέτρων
- Τα ορίσματα είναι εκφράσεις που υπολογίζονται τη στιγμή της κλήσης, και οι τιμές των εκφράσεων γίνονται οι τιμές των παραμέτρων της διαδικασίας.

### Παράδειγμα μετάδοσης παραμέτρων με τιμή

- Η έξοδος του προγράμματος είναι:  $x1 = 50$   
 $x2 = 10$
- Συνήθως ο μοναδικός μηχανισμός μετάδοσης παραμέτρων σε συναρτησιακές γλώσσες. Επιπλέον, είναι ο προεπιλεγμένος μηχανισμός στις γλώσσες C++, Pascal, C και Java

```
#include<iostream>
using namespace std;

void my_function(int x) {
    x = 50;
    cout << "x1=" << x << endl;
}

main() {
    int x = 10;
    my_function(x);
    cout << "x2=" << x
}
```

9

## Μετάδοση με Αναφορά

- Αντί να μεταβιβάζεται η τιμή της μεταβλητής, η μετάδοση με αναφορά περνά τη θέση της μεταβλητής στην μνήμη
- Έτσι η παράμετρος γίνεται ψευδώνυμο της μεταβλητής και τυχόν αλλαγές που γίνονται στην παράμετρο ενημερώνουν ταυτόχρονα το όρισμα.
- Στην γλώσσα προγραμματισμού FORTRAN, η μετάδοση με αναφορά είναι ο μόνος μηχανισμός που περνά παραμέτρους. Στην C++ η μετάδοση με αναφορά μπορεί να γίνει χρησιμοποιώντας το σύμβολο (&) μετά τον τύπο δεδομένων

```
1 void proc1(int& x) // C++
2 { x++; }
```

10

```
1 #include<iostream>
2 using namespace std;
3
4 void my_function(int &x) {
5     x = 50;
6     cout << "x1=" << x << endl;
7 }
8
9 main() {
10     int x = 10;
11     my_function(x);
12     cout << "x2=" << x;
13 }
```

- Η έξοδος του προγράμματος με βάση τον μηχανισμό μετάδοσης με αναφορά είναι:

x1=50

x2=50

11

## Μετάδοση με Αποτέλεσμα Τιμής

- Παρόμοιο αποτέλεσμα με την μετάδοση παραμέτρων με αναφορά με την μόνη διαφορά ότι δεν καθορίζεται ψευδώνυμο. Κατά την κλήση υποπρογράμματος η τιμή του ορίσματος αντιγράφεται και χρησιμοποιείται στη διαδικασία και όταν η διαδικασία ολοκληρωθεί η τελική τιμή της παραμέτρου αντιγράφεται στο όρισμα.

```

1 void proc_value_return(int x, int y) {
2     x++;
3     y++;
4 }
5
6 main()
7 {
8     int a = 5;
9     proc_value_return(a,a);
10 ...
11 }
```

Στον κώδικα του προγράμματος, εάν χρησιμοποιούμε την μέθοδο μετάδοσης με αποτέλεσμα τιμής, μετά την εκτέλεση της εντολής 9 και στην ουσία την ολοκλήρωση της εκτέλεσης της διαδικασίας, η μεταβλητή a θα έχει την τιμή 6.

12

## Μετάδοση με αποτέλεσμα τιμής

```

1 void proc_value_return(int x, int y) {
2     x++;
3     y++;
4 }
5
6 main()
7 {
8     int a = 5;
9     proc_value_return(a,a);
10 ...
11 }
```

	main	proc_value_return()	
	πρόγραμμα	x	y
8	5		
9 → 1		5	5
2		6	
3			6
4	6 (a=x)		
4	6 (a=y)		
10			

Τιμές μεταβλητών με μετάδοση με αποτέλεσμα τιμής

	main	proc_value_return()	
	πρόγραμμα	x	y
8	5		
9 → 1		5	5
2	6	6	6
3	7	7	7
4			
10			

Τιμές μεταβλητών με μετάδοση με αναφορά

## Μετάδοση με Όνομα

- Σκοπός του μηχανισμού είναι η μεταφορά παραμέτρων με μορφή κειμένου και αντικατάστασή τους την ώρα των εκφράσεων και μόνο τότε.
- Στην πορεία της ανάπτυξης των γλωσσών προγραμματισμού αποδείχθηκε ότι είναι δύσκολο να εφαρμοστεί γιατί έχει πολύπλοκες αλληλεπιδράσεις με άλλες δομές όπως οι πίνακες. Γενικά είναι μία τεχνική που δεν υιοθετήθηκε από πολλές γλώσσες προγραμματισμού.
- Όμως έχει ιδιαίτερο ενδιαφέρον ο μηχανισμός από θεωρητική άποψη.
- Χρησιμοποιήθηκε για πρώτη φορά στη γλώσσα προγραμματισμού Algol60

## Μετάδοση με όνομα

### Παράδειγμα χρήσης μηχανισμού μετάδοσης με όνομα

- Ο κώδικας έχει ως αποτέλεσμα να αλλάξει την τιμή του `a[2]` και να αφήσει την `a[1]` ανέπαφη
- Παρατηρούμε ότι όταν γίνεται κλήση της `proc1` στην `x` αποθηκεύεται το `a[i]`, που θα αντικατασταθεί αυτούσιο για να γίνουν οι πράξεις στις εκφράσεις που εμφανίζεται η `x`.
- Αυτό γίνεται στην γραμμή 6 του προγράμματος. Η έκφραση `x++` αντικαθίσταται με την `a[i]++` όπου θα ξεκινήσει να γίνονται αντικαταστάσεις των τιμών των μεταβλητών.

```

1 int i;
2 int a[5];
3
4 void proc1(int x){
5     i++;
6     x++;
7 }
8
9 main() {
10    i = 1;
11    a[1] = 7;
12    a[2] = 12;
13    proc1(a[1]);
14
15 }
```

main πρόγραμμα			proc1()
i	a[1]	a[2]	x
10	1		
11		7	
12			12
13 → 4			a[i]
5	2		
6			13
			x++ → a[i]++ → a[2]++

# Μετάδοση με όνομα

## 2<sup>o</sup> παράδειγμα χρήσης μηχανισμού μετάδοσης με όνομα

- Στην γραμμή 12 του προγράμματος γίνεται η κλήση της συνάρτησης  $p(i+j)$ . Αυτό έχει ως αποτέλεσμα στην παράμετρο  $y$  στην συνάρτηση (εντολή 3) να έχει την τιμή  $y=i+j$ .
- Στην γραμμή 4 η τοπική μεταβλητή  $j$  θα πάρει την τιμή:  $j=y=i+j=0+2=2$ , κατόπιν η καθολική μεταβλητή  $i$  θα αλλάξει τιμή σε 1 και στην επιστροφή θα υπολογιστεί η παράσταση:  $j+y=2+i+j=2+1+2=5$  όπου είναι η τιμή που θα εκτυπωθεί στην γραμμή 12.

```

1 int i;
2
3 int p(int y){
4     int j = y;
5     i++;
6     return j+y;
7 }
8
9 void q(void) {
10    int j = 2;
11    i = 0;
12    printf("%d\n", p(i + j));
13 }
14
15 main(){
16    q();
17    return 0;
18 }
```

## Άλλοι Μηχανισμοί Μετάδοσης

- Άλλες γλώσσες προγραμματισμού έχουν δικές τους προδιαγραφές για την μετάδοση παραμέτρων
- Π.χ. η Ada διαθέτει τις παραμέτρους-μέσα (in parameters) και έξω-παραμέτρους (out parameters). Οποιαδήποτε παράμετρος μπορεί να δηλωθεί μέσα ή έξω ή ακόμη και μέσα-έξω. Μία παράμετρος in καθορίζει ότι αυτή αντιπροσωπεύει μόνο μία εισερχόμενη τιμή. Μία παράμετρος out καθορίζει μόνο μία εξερχόμενη τιμή και μία παράμετρος in-out αναφέρεται σε μία εισερχόμενη και μία εξερχόμενη τιμή.
- Στην περίπτωση των αυστηρά ελεγχόμενων γλωσσών με μετάδοση παραμέτρων με αναφορά, οι παράμετροι συνήθως πρέπει να έχουν τον ίδιο τύπο, αλλά στην περίπτωση της μετάδοσης παραμέτρων με τιμή, αυτός ο κανόνας μπορεί να χαλαρώσει στη συμβατότητα ανάθεσης και μπορεί να επιτρέψει μετατροπές, όπως γίνεται στις γλώσσες προγραμματισμού C, C++ και Java.

## Παράδειγμα μετάδοσης με τιμή, απ. τιμής & αναφορά

- Η έξοδος του προγράμματος με βάση τον μηχανισμό μετάδοσης είναι:

μετάδοση με τιμή: 1 1

μετάδοση με αποτέλεσμα τιμής: 1 4

μετάδοση με αναφορά: 5 5

```

1 begin
2 integer n;
3 procedure p(k: integer);
4 begin
5     n := n+1;
6     k := k+4;
7     print(n);
8 end;
9 n := 0;
10 p(n);
11 print(n);
12 end;
```

## Μετάδοση με Όλους τους Τρόπους

1	<b>function</b> func2( <b>int</b> a, <b>int</b> b, <b>int</b> c)		
2	<b>begin</b>	μετάδοση με τιμή:	10 39 25
3	b := b + 5;		10 15
4	b := a + c + 4;		
5	print a, b, c;		
6	<b>end</b>	μετάδοση με αναφορά:	44 44 25
7			44 15
8	<b>function</b> main		
9	<b>begin</b>		
10	<b>int</b> j := 10;	μετάδοση με αποτέλεσμα τιμής:	10 39 25
11	<b>int</b> k := 15;		39 15
12	func2(j, j, j + k);		
13	print j, k;		
14	<b>end</b>	μετάδοση με όνομα:	49 49 64
			49 15

## Ενδιάμεση Αναπαράσταση

- **Ενδιάμεση Αναπαράσταση:** η εσωτερική αναπαράσταση του πηγαίου προγράμματος, με σκοπό την αποσύνθεση του προβλήματος της μετάφρασης σε απλούστερα προβλήματα, όπως η ανάλυση ροής ελέγχου, η ανάλυση δεδομένων και των εξαρτήσεών τους
- Χωρίζονται σε
  - **Γραμμικές ενδιάμεσες αναπαραστάσεις:** Είναι εντολές σε μορφή πλειάδων, που ανάλογα με το επίπεδο αφαίρεσης των εντολών, είτε διατηρούν τη δομή του κώδικα, είτε είναι εγγύτερα προς τη γλώσσα-στόχο.
  - **Δομημένες ενδιάμεσες αναπαραστάσεις:** Απεικονίζουν τον κώδικα σε δομές δεδομένων τύπου γραφήματος (π.χ. δομές δέντρων). Χρησιμοποιούνται στην ανάλυση του κειμένου των προγραμμάτων ή για την μετάφραση από έναν πηγαίο κώδικα σε άλλη μορφή.
- Μπορεί να εκφράζει τη σημασία εκτέλεσης των εντολών σε ένα υψηλό επίπεδο αφαίρεσης, όπως το αφαιρετικό συντακτικό δέντρο ή να μοιάζει περισσότερο με κώδικα στη γλώσσα-στόχο.
- Μπορεί να αξιοποιεί χαρακτηριστικά της μηχανής ή του περιβάλλοντος εκτέλεσης (π.χ. τα μεγέθη των τύπων δεδομένων, τις θέσεις των μεταβλητών και τη διαθεσιμότητα καταχωρητών).
- Μπορεί να ενσωματώνει πληροφορίες, που προέρχονται από τον πίνακα συμβόλων, όπως π.χ. εμβέλεια, blocks κ.α. Αν υπάρχει η σύνθεση κώδικα στη γλώσσα-στόχο στηρίζεται αποκλειστικά στην ενδιάμεση αναπαράσταση, διαφορετικά επιβάλλεται η χρήση του πίνακα συμβόλων
- Αν δεν εξαρτάται από την μηχανή εκτέλεσης, είναι το κλειδί της ανάπτυξης μεταφέρσιμων μεταγλωττιστών. Για την μεταφορά ενός μεταγλωττιστή σε άλλη μηχανή εκτέλεσης, αρκεί να αντικατασταθεί το τμήμα της μετάφρασης από τον ενδιάμεσο κώδικα σε κώδικα στη γλώσσα-στόχο (τελική επεξεργασία ή backend).
- Η επιλογή της κατάλληλης ενδιάμεσης αναπαράστασης για κάθε φάση επεξεργασίας επηρεάζει την ταχύτητα και την αποδοτικότητα της μετάφρασης

## Γραμμικές Ενδιάμεσες Αναπαραστάσεις

- Διακρίνονται σε τρεις κατηγορίες ανάλογα με το επίπεδο αφαίρεσής τους:
  1. **Υψηλού επιπέδου γραμμικές αναπαραστάσεις** λέγονται αυτές, που οι τελεστές τους είναι στο ίδιο επίπεδο με τους τελεστές του πηγαίου κώδικα, δεν μεταβάλουν τη δομή του προγράμματος και δεν εξαρτώνται από καταχωρητές και από το περιβάλλον εκτέλεσης.
  2. **Οι μεσαίου επιπέδου γραμμικές αναπαραστάσεις** διασπούν τους τελεστέους σε απλούστερες οντότητες, δε διατηρούν τη δομή του κώδικα και δεν εξαρτώνται από τη γλώσσα του πηγαίου προγράμματος και την αρχιτεκτονική της μηχανής.
  3. **Οι χαμηλού επιπέδου γραμμικές αναπαραστάσεις** εξαρτώνται από την αρχιτεκτονική της μηχανής εκτέλεσης, αντιμετωπίζουν ζητήματα που σχετίζονται π.χ. με διαχείριση μνήμης και δε διαφέρουν σημαντικά από τη γλώσσα-στόχο
- Γενικά για τις ενδιάμεσες παραστάσεις
  - Υψηλού-επιπέδου ενδιάμεση αναπαράσταση επιτρέπει τους βελτιστοποιητές να εξάγουν πληροφορίες σχετικά με τον πηγαίο κώδικα του προγράμματος. Από την άλλη, μια χαμηλού-επιπέδου ενδιάμεση αναπαράσταση επιτρέπει στον μεταγλωττιστή να παράγει πιο εύκολα κώδικα για την μηχανή εκτέλεσης.
  - Όσο πιο πολλές πληροφορίες ενσωματώνονται στην ενδιάμεση αναπαράσταση σχετικά με την γλώσσα-στόχο, τόσο περισσότερα είδη βελτιστοποιήσεων που αφορούν την αρχιτεκτονική της μηχανής εκτέλεσης μπορούν να ενεργοποιηθούν. Παρόλα αυτά, όσο πιο λεπτομερέστερη γίνεται μια ενδιάμεση αναπαράσταση σχετικά με μια συγκεκριμένη αρχιτεκτονική, τόσο πιο δύσκολη καθίσταται η παραγωγή κώδικα για άλλες αρχιτεκτονικές με διαφορετικά χαρακτηριστικά.
  - Αυτό έχει οδηγήσει σε μεταγλωττιστές που δεν υποστηρίζουν παραγωγή κώδικα για πολλές αρχιτεκτονικές αλλά εστιάζουν μόνο σε μία, χρησιμοποιώντας συγκεκριμένες ενδιάμεσες αναπαραστάσεις που είναι πολύ κοντά στη γλώσσα-στόχο και εξάγοντας έτσι αποδοτικότερα εκτελέσιμα προγράμματα. Όμως, στην περίπτωση που θέλουμε να υποστηρίξουμε πολλές αρχιτεκτονικές, η ανάπτυξη αποδοτικών μεταγλωττιστών για κάθε μια ξεχωριστά είναι μια μη εφικτή λύση.

## Κώδικας Τριών Διευθύνσεων

- Ο κώδικας τριών διευθύνσεων (three address code) είναι μία ακολουθία εντολών της μορφής  $x = y$  ορ  $z$ , με  $x, y, z$  ονόματα, σταθερές ή προσωρινές μεταβλητές  $t_1, t_2, \dots$ , που παράγει ο μεταγλωττιστής και ορ  $\epsilon$ ναν αριθμητικό ή λογικό τελεστή. Δεν επιτρέπεται η χρήση παρενθέσεων.
- Κάθε εντολή του κώδικα τριών διευθύνσεων μπορεί να αναπαρασταθεί ως τετράδα της μορφής  $(op, arg1, arg2, result)$ .
- Οδηγεί στη δημιουργία μεγάλου αριθμού προσωρινών μεταβλητών, που τελικά περιπλέκει την εφαρμογή μετασχηματισμών βελτιστοποίησης.

Η έκφραση  $a = (b+c) * d[i]$  αντιστοιχεί στον κώδικα τριών διευθύνσεων:

$$\begin{aligned} t_1 &= b + c \\ t_2 &= d[i] \\ t_3 &= t_1 * t_2 \\ a &= t_3 \end{aligned}$$



Διατύπωση σε τετράδες

+ , b , c , t1  
[] = , d , i , t2  
\* , t1 , t2 , t3  
= , t3 , , a

## Κώδικας Τριών Διευθύνσεων για Εκφράσεις

- Η παρακάτω γραμματική ιδιοτήτων, περιγράφει τη διαδικασία παραγωγής κώδικα τριών διευθύνσεων για εκφράσεις, που συνδυάζουν τους τελεστές της πρόσθεσης, της αφαίρεσης και της ανάθεσης τιμής

- (1)  $S \rightarrow id = E$  [  $S.code = E.code$   
                  || codegen(lookup(id.name), "=", E.addr)]
- (2)  $E \rightarrow E_1 + V$  [  $E.addr = new Temp()$ ,  
                   $E.code = E_1.code || V.code$   
                  || codegen(E.addr, "=", E\_1.addr, "+", V.addr)]
- (3)  $E \rightarrow E_1 - V$  [  $E.addr = new Temp()$ ,  
                   $E.code = E_1.code || V.code$   
                  || codegen(E.addr, "=", E\_1.addr, "-", V.addr)]
- (4)  $E \rightarrow V$  [  $E.addr = V.addr$ ,  
                   $E.code = V.code$ ]
- (5)  $V \rightarrow (E)$  [  $V.addr = E.addr$ ,  
                   $V.code = E.code$ ]
- (6)  $V \rightarrow id$  [  $V.addr = lookup(id.name)$ ,  
                   $V.code = " "$ ]
- (7)  $V \rightarrow num$  [  $V.addr = num.value$ ,  
                   $V.code = " "$ ]

Η συνάρτηση codegen δέχεται ως είσοδο μία ακολουθία συμβολοσειρών και επιστρέφει μία εντολή τριών διευθύνσεων, που σχηματίζεται από την παράθεσή τους (||), π.χ. η codegen ( $x, "=", y, "+", z$ ) επιστρέφει την εντολή  $x=y+z$ .

Η lookup (id.name) αναζητά στον πίνακα συμβόλων την καταχώρηση, που αντιστοιχεί (αν υπάρχει) στην λεξική μονάδα του id

Η ιδιότητα  $X.code$ ,  $X \in \{S, E, V\}$ , εκφράζει τον κώδικα τριών διευθύνσεων, για τα τερματικά σύμβολα, που απλοποιούνται σε X

19

## Κώδικας Δύο Διευθύνσεων

- Εναλλακτικά, υπάρχει η δυνατότητα χρήσης κώδικα δύο διευθύνσεων (τριάδες), στον οποίο παραλείπεται από κάθε εντολή το πεδίο του αποτελέσματός της.
- Το αποτέλεσμα ανακαλείται με αναφορά στην αρίθμηση της εντολής, αφού στον κώδικα δύο διευθύνσεων οι τριάδες αριθμούνται.

Η έκφραση  $a = (b+c) * d[i]$  αντιστοιχεί στον κώδικα τριών διευθύνσεων:

$$\begin{aligned} t_1 &= b + c \\ t_2 &= d[i] \\ t_3 &= t_1 * t_2 \\ a &= t_3 \end{aligned}$$



Διατύπωση σε κώδικα δύο διευθύνσεων

$$\begin{aligned} 1 &+ , b , c \\ 2 &[] = , d , i \\ 3 &* , d , (1) \\ 4 &= , (3) , a \end{aligned}$$

## Κώδικας Στοίβας

- Ο κώδικας στοίβας (stack-based code) είναι μία γραμμική ΕΑ που χρησιμοποιεί μία βοηθητική στοίβα για την εκτέλεση υπολογισμών. Διακρίνουμε τους εξής τύπους εντολών
  - Εντολές που φορτώνουν την τιμή μεταβλητής ή δημιουργούν και αποθηκεύουν μία σταθερά στη στοίβα (π.χ. LOAD X, LOAD\_CONST C) ή αποθηκεύουν το αποτέλεσμα μιας πράξης σε μεταβλητή (π.χ. STORE X).
  - Τελεστές δημιουργίας νέων αντικειμένων ή άλλων δομών (π.χ. NEW\_OBJ T, NEW\_ARR T κ.ο.κ.).
  - Εντολές-τελεστών που αφαιρούν στοιχεία από τη στοίβα, εκτελούν μία πράξη και εισάγουν το αποτέλεσμα στη στοίβα (π.χ. ADD, CMP, DIV, MUL, AND, OR κ.ο.κ.).
  - Εντολές αλλαγής ροής ελέγχου (π.χ. JUMP L, IF\_LT L, IF\_NULL L, IF\_GE L, CALL F κ.ο.κ.).
  - Εντολές μετατροπής τύπου (π.χ. I\_TO\_D, S\_TO\_I κ.ο.κ.).

## Παράδειγμα

- Έστω ο παρακάτω κώδικας:

```

1 float x;
2 int y;
3 ...
4 y = (x + 5.0) / 2.0;

```

- Ο κώδικας στοίβας της εντολής στη γραμμή 4 έχει ως εξής:

```

LOAD X
LOAD_CONST 5.0
ADD
LOAD_CONST 2.0
DIV
F_TO_I
STORE Y

```

- Αρχικά φορτώνονται στη στοίβα η τιμή της μεταβλητής  $x$  και η σταθερά 5.0.
- Με την ADD αφαιρούνται τα δύο στοιχεία από τη στοίβα, αθροίζονται και το αποτέλεσμα αποθηκεύεται πάλι στη στοίβα. Στη συνέχεια, εισάγεται στη στοίβα η σταθερά 2.0 και διαιρέται η τιμή του αθροίσματος (κάτω από την κορυφή) με το 2.0.
- Το αποτέλεσμα αφαιρείται από τη στοίβα, μετατρέπεται με την F\_TO\_I σε ακέραιο και αποθηκεύεται πάλι στη στοίβα. Με την STORE Y, αφαιρείται από τη στοίβα η τιμή που υπολογίστηκε και αποθηκεύεται στην μεταβλητή  $y$ .

## Κώδικας Στοίβας για Εκφράσεις

- Στην γραμματική ιδιοτήτων, που ακολουθεί, έχουμε τη σύνθεση κώδικα μηχανής στοίβας για εκφράσεις

- (1)  $S \rightarrow id = E$  [codegen("STORE", id.name)]
- (2)  $E \rightarrow E_1 + V$  [codegen("ADD")]
- (3)  $E \rightarrow E_1 - V$  [codegen("SUB")]
- (4)  $E \rightarrow V$
- (5)  $V \rightarrow (E)$
- (6)  $V \rightarrow id$  [codegen("LOAD", id.name)]
- (7)  $V \rightarrow num$  [codegen("CONST", num.value)]

- Παράδειγμα για έκφραση  $a = (b+5)+(e-d)$ :

```

LOAD b
CONST 5
ADD
LOAD e
LOAD d
SUB
ADD
STORE a

```

18

## Δομημένες Ενδιάμεσες Αναπαραστάσεις

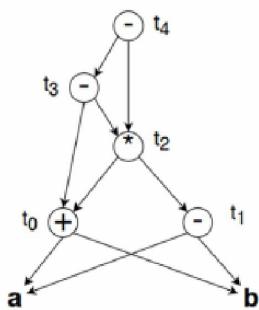
- Οι δομημένες ενδιάμεσες αναπαραστάσεις (**structured intermediate representation**) είναι δομές γραφημάτων, που αποτελούνται από κόμβους και ακμές. Διαφέρουν μεταξύ τους ως προς το επίπεδο αφαίρεσης της αναπαράστασης, το πώς συσχετίζονται με τον κώδικα που απεικονίζουν, καθώς και ως προς τη δομή του γραφήματος.
- Οι αλγόριθμοι επεξεργασίας των ενδιάμεσων αναπαραστάσεων αναφέρονται σε συστατικά στοιχεία των γραφημάτων: κόμβους, ακμές, λίστες ή δέντρα.
- Μια ενδιάμεση αναπαράσταση με δομή δέντρου μπορεί να διαφέρει, ανάλογα με τη χρήση της. Συνήθως όμως αναφερόμαστε στο αφαιρετικό συντακτικό δέντρο.

## Κατευθυνόμενα Ακυκλικά Γραφήματα

- Αναπαράσταση, που χρησιμοποιείται στην επεξεργασία εκφράσεων, που περιέχουν κοινές υποεκφράσεις. Ένα Κ.Α.Γ. έχει σε εσωτερικούς κόμβους τους τελεστές πράξεων με φύλλα τους τελεστέους των πράξεων.
- Η διαφορά με τις αναπαραστάσεις δέντρων είναι ότι ένας κόμβος σε Κ.Α.Γ. μπορεί να έχει περισσότερους από έναν γονείς και αυτό συμβαίνει, όταν το στοιχείο του κόμβου αναφέρεται μέσα στην έκφραση πάνω από μία φορές. Έτσι, αποφεύγεται η δημιουργία κόμβων για υποεκφράσεις που είναι κοινές και αυτό αξιοποιείται για τη σύνθεση αποδοτικότερου κώδικα.
- Η κατασκευή του Κ.Α.Γ. με βάση μία γραμματική χωρίς συμφραζόμενα διαφέρει από την κατασκευή του συντακτικού δέντρου, στο ότι πριν από τη δημιουργία νέου κόμβου, πρέπει να ελέγχεται αν ήδη υπάρχει κόμβος στο γράφημα, με τον οποίο μπορεί να ταυτιστεί.

### Παράδειγμα

- Ακολουθεί το Κ.Α.Γ. για την έκφραση  $((a + b) - ((a + b) * (a - b))) - ((a + b) * (a - b))$



Τα φύλλα  $a$  και  $b$  έχουν δύο γονείς γιατί εμφανίζονται στις υποεκφράσεις  $a+b$  και  $a-b$ . Η πρώτη υποέκφραση εμφανίζεται σε 2 εκφράσεις και γι'αυτό τον λόγο δημιουργούνται δύο γονείς για τον κόμβο  $+$ . Η υποέκφραση  $a-b$  εμφανίζεται μόνο σε 1 έκφραση και άρα ο κόμβος  $-$  - έχει ως μοναδικό γονέα τον κόμβο  $*$ . Τέλος, επειδή η έκφραση  $(a + b) * (a - b)$  εμφανίζεται δύο φορές, ο κόμβος  $*$  συνδέεται και αυτός με δύο γονείς.

Ο κώδικας τριών διευθύνσεων για την έκφραση του παραδείγματος είναι:

$$\begin{aligned} t_0 &= a + b \\ t_1 &= a - b \\ t_2 &= t_0 * t_1 \\ t_3 &= t_0 - t_2 \\ t_4 &= t_3 - t_2 \end{aligned}$$

## Γραφήματα Ροής Ελέγχου

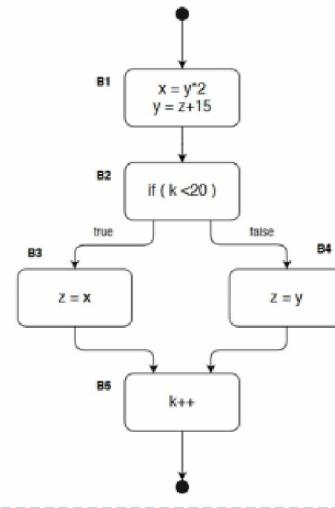
- Διασπούν τον κώδικα σε βασικά block, για να διευκολύνουν την εφαρμογή τεχνικών βελτιστοποίησης και τον εντοπισμό μη προσεγγίσιμου κώδικα.
- Βασικό block είναι μία ακολουθία εντολών μέγιστου μήκους χωρίς διακλάδωση, δηλαδή τέτοια ώστε η ροή ελέγχου να εισέρχεται στο block από την πρώτη εντολή του και εξέρχεται από την τελευταία. Οι εντολές ενός βασικού block εκτελούνται πάντα μαζί, εκτός και αν μία απ' αυτές προκαλέσει εξαίρεση.
- Μπορεί να εφαρμοστεί είτε σε υψηλού επιπέδου κώδικα (π.χ. τον πηγαίο), είτε σε κώδικα χαμηλού επιπέδου (π.χ. κώδικα τριών διευθύνσεων) και περιλαμβάνει τα εξής:
  - Εντοπισμό κύριων εντολών, δηλαδή των εντολών, που αναφέρονται πρώτες σε ένα βασικό block.
  - Δημιουργία ενός βασικού block  $B_i$ , για κάθε κύρια εντολή, που θα περιλαμβάνει την εντολή αυτή και όλες όσες ακολουθούν μέχρι την εμφάνιση της επόμενης κύριας εντολής ή του τέλους του προγράμματος.
  - Σύνδεση των βασικών block με ακμές, ανάλογα με τη ροή ελέγχου του προγράμματος.

### Παράδειγμα

- Για τον κώδικα υψηλού επιπέδου παράγεται το ακόλουθο γράφημα ροής ελέγχου:

```

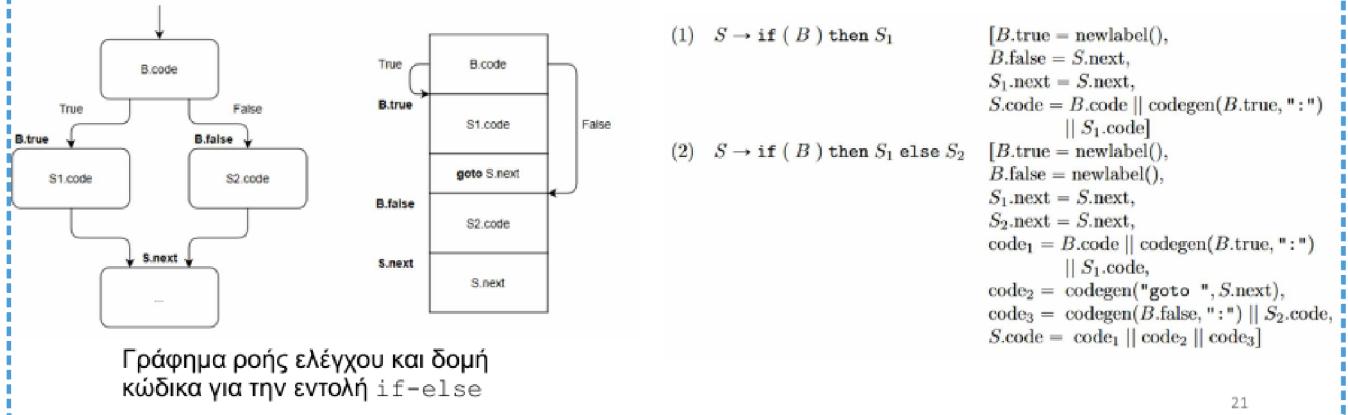
x = y*2;
y = z + 15;
if (k < 20) {
    z = x;
}
else {
    z = y;
}
k++;
  
```



15

## Κώδικας για Ροή Ελέγχου

- Η μετάφραση λογικών εκφράσεων είναι άρρηκτα συνδεδεμένη με τη μετάφραση εντολών αλλαγής της ροής του προγράμματος (εντολές if-else, while, for) σε κώδικα ενδιάμεσης αναπαράστασης



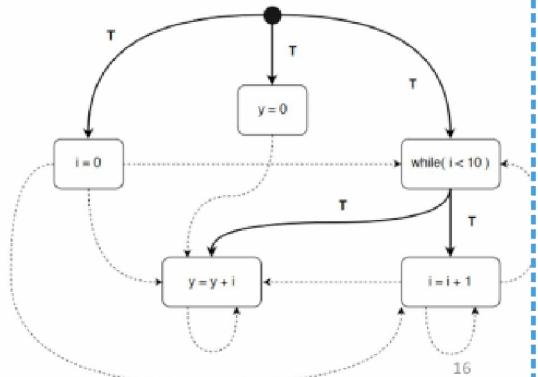
## Γραφήματα Εξαρτήσεων Προγράμματος

- Με το γράφημα εξαρτήσεων του προγράμματος μπορούμε να αναπαραστήσουμε τόσο τις εξαρτήσεις ροής ελέγχου μεταξύ τμημάτων κώδικα, όσο και τις εξαρτήσεις μεταξύ των δεδομένων τους. Τα δύο αυτά είδη εξαρτήσεων διακρίνονται στο γράφημα με δύο είδη ακμών.

Δύο κόμβοι ενώνονται με μη διακεκομμένες ακμές, αν η εκτέλεση του δεύτερου αποφασίζεται από τον πρώτο (εξάρτηση ροής ελέγχου).

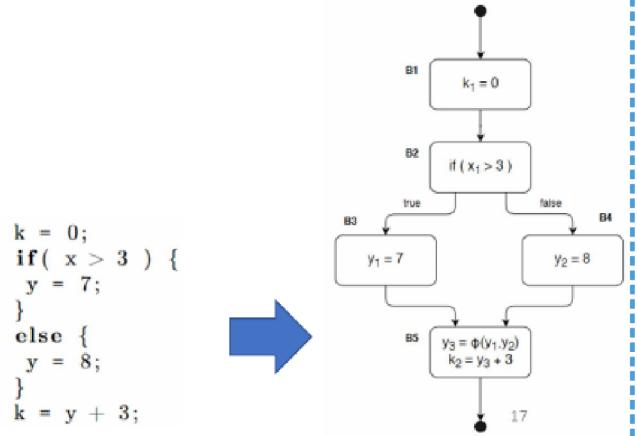
Οι διακεκομμένες ακμές εκφράζουν τις εξαρτήσεις δεδομένων.

```
i = 0;
y = 0;
while( i < 10 ) {
    y = y + i;
    i = i + 1;
}
```



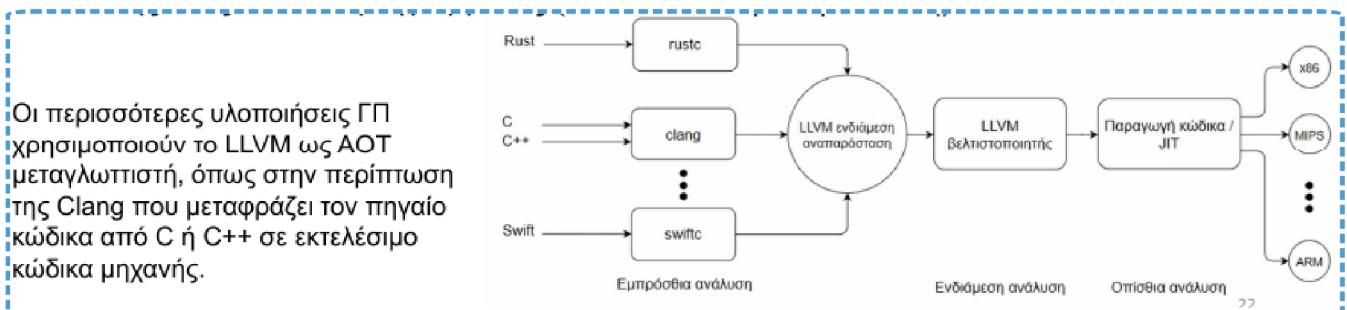
## Μορφή Στατικής Μοναδικής Εκχώρησης

- Αναπαριστά τον κώδικα με τέτοιο τρόπο, ώστε σε κάθε μεταβλητή ανατίθεται τιμή μία μόνο φορά και αυτή ορίζεται πριν από τη χρήση της. Όταν δύο διαφορετικά μονοπάτια εκτέλεσης αναθέτουν διαφορετική τιμή σε μία μεταβλητή  $x$ , τότε προστίθεται μία επιπλέον εντολή της μορφής  $x_i = \varphi(x_{i-1}, x_{i-2})$  που συνδυάζει τις δύο πιθανές τιμές, που μπορεί να αποδοθούν στην  $x_i$ .
- Η μορφή Σ.Μ.Ε. απλοποιεί την εφαρμογή κάποιων τεχνικών βελτιστοποίησης
- Επιτρέπει αποδοτικότερη χρήση των καταχωρητών λόγω του ότι είναι ευκολότερη η εκτίμηση του πλήθους των μεταβλητών που είναι ενεργές σε ένα οποιοδήποτε σημείο του προγράμματος.



## Ενδιάμεση Αναπαράσταση LLVM (Low Level Virtual Machine)

- Το Low Level Virtual Machine (LLVM) είναι μια συλλογή από βιβλιοθήκες για την ανάπτυξη μεταγλωττιστών και την επέκταση των ήδη υπαρχόντων με επιπλέον λειτουργίες.
- Το LLVM παρέχει μια γλώσσα ενδιάμεσης αναπαράστασης όπου οι μεταγλωττιστές άλλων γλωσσών παράγουν κώδικα σε αυτή τη μορφή. Στη συνέχεια, το LLVM διαβάζει και μεταγλωττίζει (Ahead-Of-Time μεταγλώττιση) ή μεταγλωττίζει τμήματα του σε κώδικα εκτέλεσης ενός άλλου προγράμματος (Just-In-Time μεταγλώττιση)



- Το LLVM υποστηρίζει ένα μεγάλο φάσμα τεχνικών βελτιστοποίησης με υψηλό βαθμό λεπτομέρειας (απαλοιφή μη προσεγγίσιμου κώδικα, ανάπτυξη βρόχων κ.λ.π.)
- Έτσι, πολλές υλοποιήσεις μεταγλωττιστών γλωσσών προγραμματισμού αφού αναλύσουν και εφαρμόσουν διάφορους ελέγχους, παράγουν ενδιάμεσο κώδικα για LLVM. Στη συνέχεια, εκτελούνται επαναληπτικά μια σειρά από βελτιστοποιήσεις πάνω στην ενδιάμεση αναπαράσταση και τέλος παράγεται ο κώδικας μηχανής.
- Το LLVM πέρα από την ενδιάμεση αναπαράσταση η οποία είναι ανεξάρτητη από την γλώσσα-στόχο, επιτρέπει τον χειρισμό χαμηλού επιπέδου προβλημάτων που σχετίζονται με την αρχιτεκτονική της μηχανής.

## Τύποι στο LLVM

- Το αυστηρό σύστημα τύπων του LLVM επιβάλλει στην γλώσσα της ενδιάμεσης αναπαράστασης να υπάρχουν τύποι για κάθε δομή δεδομένων.
- Έτσι το LLVM επιτρέπει ένα τεράστιο αριθμό από βελτιστοποιήσεις που εφαρμόζονται πάνω στον κώδικα απευθείας χωρίς να χρειάζεται άλλες αναλύσεις. Τύποι που υποστηρίζονται:
  - Κενός (void)
  - απλοί τύποι (i1, i8, i16, i32, i64, ... half, float, double, fp128)
  - πίνακες ([260 ^ i32] πίνακας 260 στοιχείων από ακέραιους μεγέθους 32 bits)
  - τύποι συναρτήσεων, i8 (i16, i32\*) (τύπος συνάρτησης που δέχεται ως παράμετρους έναν ακέραιο 16 bits και έναν δείκτη σε ακέραιο 32 bits και επιστρέφει έναν ακέραιο 8 bits)
  - δείκτες (- i32\*, float\*, [4 ^ i32]\*, i64 (i32\*)\*)
  - διανύσματα <5 ^ i32> (Διάνυσμα από 5 ακέραιους των 32 bits), <8 ^ float> (Διάνυσμα από 8 αριθμούς κινητής υποδιαστολής των 32 bits) – <6 ^ i64\*> (Διάνυσμα από 6 δείκτες σε ακέραιους των 64 bits) – <vscale ^ 5 ^ i32> (Κλιμακωτό διάνυσμα με μέγεθος πολλαπλάσιο των 5 ακεραίων των 32 bits) (το vscale είναι ένας θετικός ακέραιος αριθμός και είναι άγνωστος στη φάση της μεταγλώττισης της ενδιάμεσης αναπαράστασης σε γλώσσα-στόχο)

24

## Η εντολή getelementptr

- Το LLVM παρέχει την εντολή getelementptr η οποία χρησιμοποιείται για τον υπολογισμό της διεύθυνσης ενός πεδίου ενός σύνθετου τύπου (πίνακες, δομημένοι τύποι)

```
%Triangle = type { %Point, %Point, %Point }
%Point   = type { i32, i32 }
...
%temp = getelementptr %Triangle, %Triangle* %tr1,
        i64 0, i32 2, i32 1
```

ορίζονται δύο δομημένοι τύποι Triangle και Point. Ο πρώτος περιέχει τρία πεδία τύπου Point τα οποία αναπαριστούν τις συντεταγμένες ενός τριγώνου (περιέχει δύο πεδία ακεραίων των 32 bits που αναπαριστούν τις συντεταγμένες x, y ενός σημείου)

Θεωρούμε ότι έχει δημιουργηθεί μια μεταβλητή %tr1 τύπου Triangle.

25

ΤΥΠΟΙ)

Χρησιμοποιούμε την getelementptr για να αποκτήσουμε πρόσβαση (να υπολογίσουμε τη διεύθυνση) στη συντεταγμένη γ του τελευταίου σημείου του τριγώνου που είναι αποθηκευμένο στη μεταβλητή %tr1.

```
%Triangle = type { %Point, %Point, %Point }
%Point   = type { i32, i32 }
...
%temp = getelementptr %Triangle, %Triangle* %tr1,
        i64 0, i32 2, i32 1
```

Η πρώτη παράμετρος είναι ο τύπος της δομής που θα αποκτήσει πρόσβαση η εντολή.

Η δεύτερη είναι ο δείκτης ο οποίος δείχνει στη διεύθυνση μνήμης από την οποία θα ξεκινήσουν οι πράξεις υπολογισμού της θέσης του στοιχείου που θέλουμε να αποκτήσουμε πρόσβαση

**ΤΙΠΟΙ**

οι επόμενοι αριθμοί  
χρησιμοποιούνται ως  
ευρετήριο που υποδεικνύουν  
τη θέση του πεδίου που  
ενδιαφερόμαστε

```
%Triangle = type { %Point, %Point, %Point }
%Point = type { i32, i32 }
...
%temp = getelementptr %Triangle, %Triangle* %tr1,
```

Ο 1<sup>ος</sup> αριθμός δείχνει στο δείκτη που δίνεται ως  
δεύτερο όρισμα της εντολής (%Triangle\* %tr1).

Στην περίπτωση που έχουμε έναν πίνακα από  
δείκτες θα μπορούσε το ευρετήριο να ήταν μη-  
μηδενικός αριθμός. Επιστρέφει ως τύπο την δομή  
%Triangle={%Point,%Point,%Point}

Ο 2<sup>ος</sup> αριθμός δείχνει στο τρίτο  
πεδίο του τύπου και επιστρέφει τον  
τύπο %Point={i32,i32}

Ο 3<sup>ος</sup> αριθμός δείχνει στη δεύτερη  
τιμή του δομημένου τύπου Point  
και άρα επιστρέφει έναν ακέραιο  
τύπου i32

27

**LLVM και Μνήμη**

- Στην LLVM οι τιμές των καταχωρητών πρέπει να είναι σε μορφή Σ.Μ.Ε., ενώ οι μεταβλητές που αποθηκεύονται στη στοίβα δεν είναι απαραίτητο.
- Η δέσμευση χώρου για την αποθήκευση μεταβλητών γίνεται με την εντολή `alloca`, η οποία δεσμεύει χώρο στη στοίβα για κάθε τοπική ή προσωρινή μεταβλητή του πηγαίου προγράμματος και επιστρέφει έναν δείκτη προς αυτό. Όμως, η χρήση της είναι μη αποδοτική για απλές τοπικές μεταβλητές, καθώς η εκτέλεση εντολών που περιέχουν μεταβλητές που είναι στη στοίβα είναι πιο αργές σε σύγκριση με εκείνες που διαχειρίζονται τιμές σε καταχωρητές
  - Μια μεταβλητή που έχει δημιουργηθεί με χρήση της εντολής `alloca` μπορεί να αποθηκευτεί στους καταχωρητές αν η τιμή της δεν μεταφέρεται ποτέ στη μνήμη και δεν χρησιμοποιείται ως παράμετρος σε κλήση συνάρτησης. Σε αυτή τη περίπτωση μία δέσμευση μνήμης ονομάζεται προβιβάσιμη.
- Μια από τις διαδικασίες βελτιστοποίησης που λαμβάνουν χώρα στην ενδιάμεση ανάλυση είναι η εύρεση και η μεταφορά των μεταβλητών από τη στοίβα στους καταχωρητές, όποτε αυτό είναι εφικτό (`mem2reg`)
  - Το `mem2reg` εντοπίζει τις προβιβάσιμες δεσμεύεις μνήμης και τις αντικαθιστά με προσωρινές μεταβλητές σε μορφή Σ.Μ.Ε., τοποθετώντας συναρτήσεις φ σε συγκεκριμένα σημεία στον κώδικα

**Απλό πρόγραμμα με  
εντολή επιλογής.**

```
int x = ...
int y = ...
int z = ...
if (c) {
    x = y + 1;
}
else {
    x = y + 2;
}
z = x + 3;
```

Ο κώδικας έχει  
χωριστεί με ετικέτες,  
που χρησιμοποιούνται  
για να διαχωρίζονται  
τα βασικά blocks του  
κώδικα μεταξύ τους

Εντολή φ (phi): 1<sup>ο</sup> όρισμα των τύπων  
εισερχόμενων τιμών. 2<sup>ο</sup> όρισμα μια λίστα από  
ζεύγη μεταβλητών (val<sub>i</sub>) με ετικέτες (label<sub>i</sub>)

**Πρόγραμμα σε μορφή ενδιάμεσης  
αναπαράστασης LLVM**

```
entry:
  %x1 = ...
  %y1 = ...
  %z1 = ...
  %c = icmp ...
  br il %c, label %then, label %else
then:
  %x2 = add i32 %y1, 1
  br label %merge
else:
  %x3 = add i32 %y1, 2
merge:
  %x4 = phi i32 %x2, %then, %x3, %else
  %z2 = %add i32 %x4, 3
```

30