



ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

<http://eclass.aueb.gr/courses/INF511/>

Γλώσσες Προγραμματισμού (ΚΕΦΑΛΑΙΟ 6)

Αλκμήνη Σγουρίτσα

Κοδριγκτώνος 12, 2^{ος} όροφος

E-mail: alkmini@aeub.gr

ΚΕΦΑΛΑΙΟ 6: Γλώσσες Προγραμματισμού

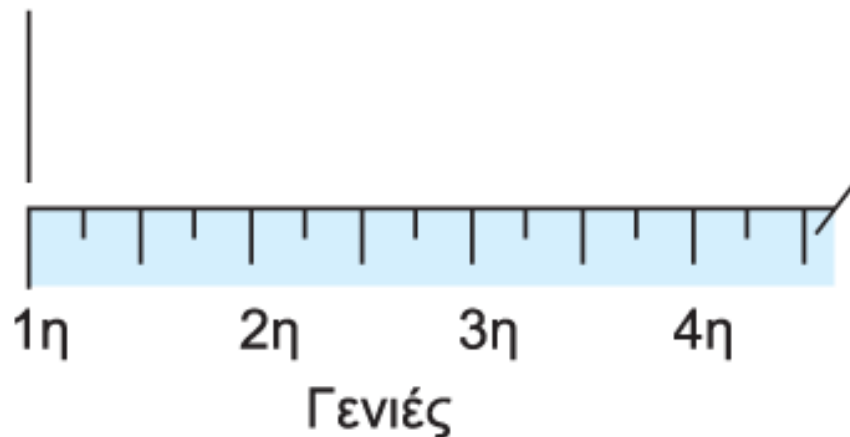
- Ιστορική εξέλιξη Γλωσσών Προγραμματισμού
- Προγραμματιστικά μοντέλα
 - Διαδικαστικό
 - Συναρτησιακό
 - Δηλωτικό
 - Αντικειμενοστραφές
- Σύνταξη γλώσσας και γραμματική
 - Συντακτικά δέντρα

Γλώσσες προγραμματισμού

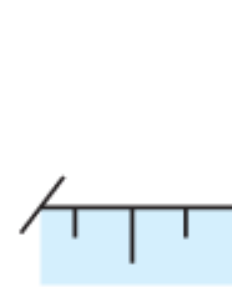
Γλώσσα Προγραμματισμού: Διεπαφή ανθρώπου-υπολογιστή

- **προσιτό στον χρήστη** (για αναπαράσταση αλγορίθμων)
- **εύκολα μετατρέψιμο σε γλώσσα μηχανής** που είναι κατανοητή από τον Η/Υ

Τα προβλήματα λύνονται σε ένα περιβάλλον όπου ο άνθρωπος πρέπει να συμμορφώνεται με τα χαρακτηριστικά της μηχανής



Τα προβλήματα λύνονται σε ένα περιβάλλον όπου η μηχανή "συμμορφώνεται" με τα χαρακτηριστικά του ανθρώπου



Πρώτη Γενιά: Γλώσσα Μηχανής

- Γλώσσα μηχανής: οι εντολές αναπαρίστανται σε δυαδικό σύστημα (θυμηθείτε Κεφ.2)
- Κάθε εντολή αναπαρίσταται ως ακολουθία από δυαδικά ψηφία
- Το πρόγραμμα είναι μια σειρά από εντολές, π.χ.

```
156C  
166D  
5056  
306E  
C000
```

- Εξαιρετικά δύσκολο το γράψιμο και η κατανόηση προγραμμάτων
- Συχνά σφάλματα, δύσκολη αποσφαλμάτωση (debugging)

Δεύτερη γενιά: γλώσσα Assembly

- Μνημονικό σύστημα για την αναπαράσταση προγραμμάτων
- Χρήση **μνημονικών** ονομάτων για αναπαράσταση εντολών
- **1-1** αντιστοιχία μεταξύ των εντολών της γλώσσας μηχανής και εντολών της γλώσσας Assembly

Γλώσσα μηχανής

156C

166D

5056

30CE

C000

Γλώσσα assembly

LD R5, Price

LD R6, ShippingCharge

ADD R0, R5 R6

ST R0, TotalCost

HLT

Μειονεκτήματα της Assembly

- Ο προγραμματιστής πρέπει να σκέφτεται **πάλι** σε γλώσσα μηχανής (απλά αλλάζει η αναπαράσταση και σύνταξη των εντολών)
- Αρχέτυπα (εντολές) **χαμηλού** επιπέδου (πολύ κοντά στο hardware)
- **Δεν διευκολύνει στον μακροσκοπικό γενικό σχεδιασμό** ενός προγράμματος (γιατί αποτελείται από «μικροεντολές» που συνδέονται με μικρολειτουργίες της CPU)
- Η Assembly είναι εγγενώς εξαρτημένη από τη μηχανή
 - **Δύσκολη ή αδύνατη φορητότητα** (portability) (μεταφορά του προγράμματος σε άλλη μηχανή)

Τρίτη γενιά γλωσσών

- Βασίζεται σε μικρό αριθμό από αρχέτυπα «υψηλού επιπέδου»
π.χ.
identifier ← expression
- **Ανεξάρτητη από την μηχανή** (τις περισσότερες φορές)
- Κάθε αρχέτυπο αντιστοιχεί σε μία μεγαλύτερη ακολουθία εντολών γλώσσας μηχανής
 - Π.χ. ανάθεσε στο TotalCost την τιμή Price + PostalCost
- Παραδείγματα: C++, Python, Java
- Μετάφραση σε γλώσσα μηχανής από ένα πρόγραμμα που λέγεται **compiler** (**μεταγλωττιστής**)
- Εναλλακτικά, αντί για compiler, **διερμηνευτής (interpreter)**
 - Ο διερμηνευτής εκτελεί εντολές καθώς τις μεταφράζει
 - ...ενώ ο compiler πρώτα μεταφράζει όλες τις εντολές του προγράμματος σε γλώσσα μηχανής και μετά τις εκτελεί

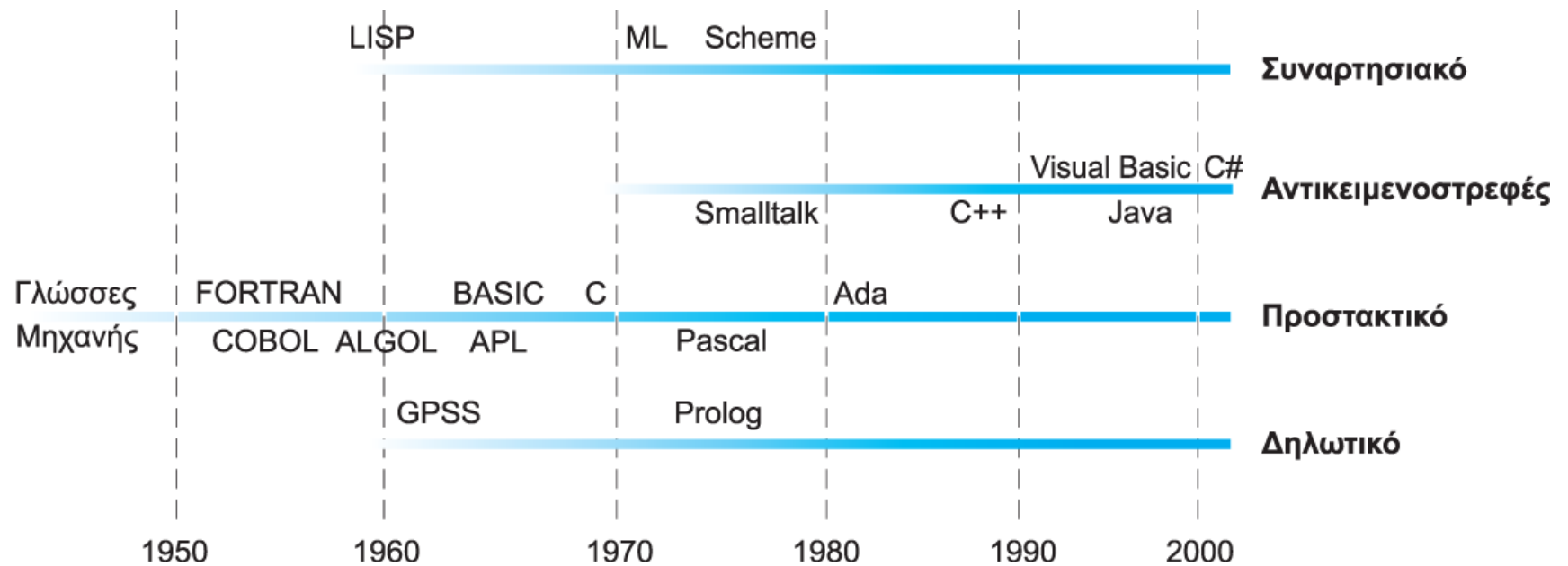
Υψηλότερες γενιές γλωσσών

- 4^η γενιά: εξυπηρετούν **συγκεκριμένους σκοπούς**, μία εντολή τους αντιστοιχεί σε πολλές εντολές σε γλώσσα 3^{ης} γενιάς.
 - Παράδειγμα: SQL για χρήση βάσεων δεδομένων, HTML για δημιουργία ιστοσελίδων
- 5^η γενιά: **το πρόβλημα λύνεται από τον υπολογιστή** και **όχι** από τον προγραμματιστή. Ο προγραμματιστής περιγράφει μόνο το πρόβλημα. Χρησιμοποιούνται κυρίως στην «**Τεχνητή Νοημοσύνη**».
 - Παραδείγματα: Mercury, Prolog
- 6^η γενιά: Είναι αποκομμένη από το hardware. **Δεν χρησιμοποιεί κώδικα**. Μπορεί να χρησιμοποιούν κάποιο **Natural Language Processing**.

Η εξέλιξη των Γλωσσών Προγραμματισμού

Προγραμματιστικά Μοντέλα

Προγραμματιστικό μοντέλο: Διαφορετική φιλοσοφία στον προγραμματισμό



Δείτε και το παρακάτω:

<http://www.inc.com/larry-kim/10-most-popular-programming-languages-today.html>

1^ο Προγραμματιστικό μοντέλο: Προστακτικό

- Προστακτικό (imperative) ή διαδικαστικό (procedural) μοντέλο: το πρόγραμμα είναι μια ακολουθία από εντολές, που όταν εκτελεστούν, επενεργούν στα δεδομένα και παράγουν το επιθυμητό αποτέλεσμα
- Πρέπει να βρεθεί ο αλγόριθμος για επίλυση του προβλήματος
- ... και μετά ο αλγόριθμος να αναπαρασταθεί ως ακολουθία από εντολές
- Το πιο παραδοσιακό μοντέλο
- Παραδείγματα:
 - FORTRAN (για αριθμητικούς υπολογισμούς)
 - COBOL (για επιχειρησιακές εφαρμογές)
 - BASIC, C, Pascal (γενικής χρήσης)
 - Ada (επίσημη γλώσσα ανάπτυξης στρατιωτικών εφαρμογών από το Υπ. Άμυνας των ΗΠΑ)

Η σύνθεση ενός τυπικού προστακτικού προγράμματος

Πρόγραμμα



Το πρώτο μέρος αποτελείται από εντολές δηλώσεων οι οποίες περιγράφουν τα δεδομένα που χειρίζεται το πρόγραμμα.
(Declarative statements)

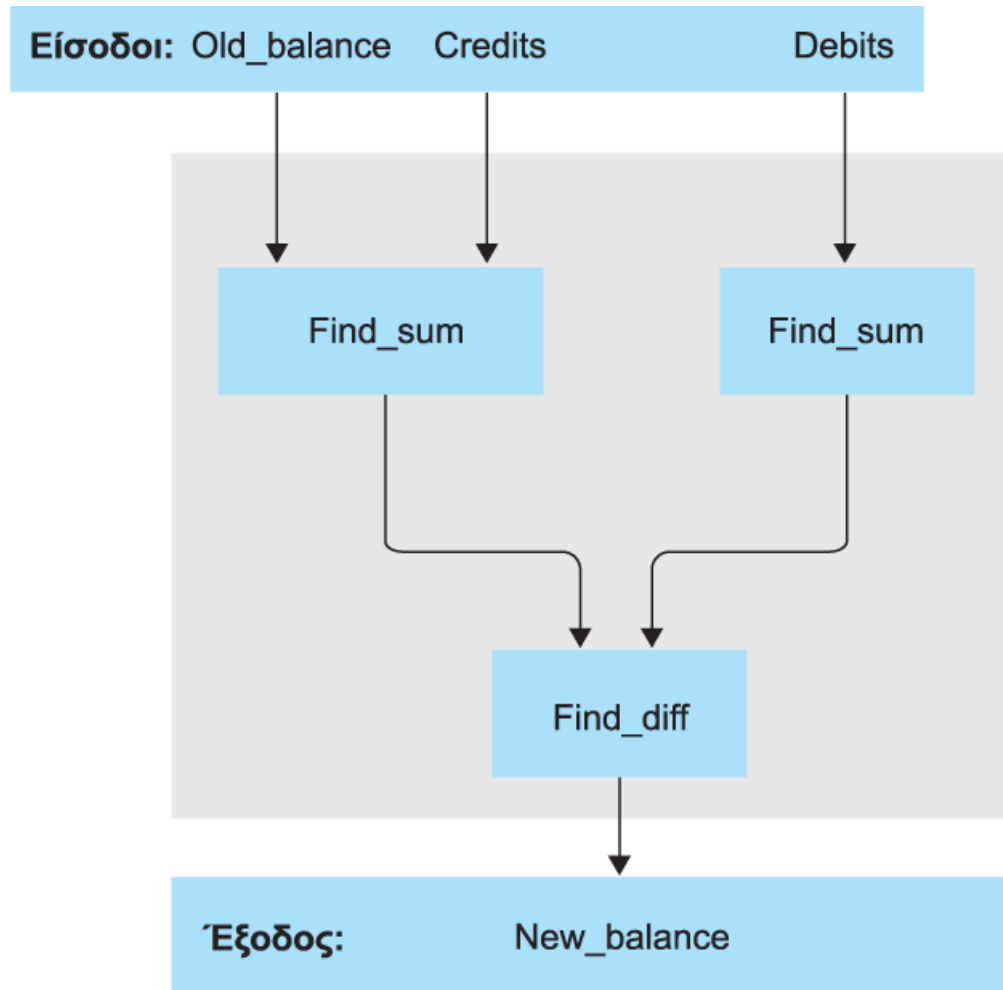
Το δεύτερο μέρος αποτελείται από προστακτικές εντολές οι οποίες περιγράφουν τις ενέργειες που πρέπει να πραγματοποιηθούν.
(imperative statements)

Σχόλια (comments) βελτιώνουν την αναγνωσιμότητα

2^ο Προγραμματιστικό μοντέλο: Συναρτησιακό

- **Συναρτησιακό (functional)** μοντέλο: Το πρόγραμμα είναι μια οντότητα που δέχεται εισόδους και βγάζει εξόδους (όπως η συνάρτηση)
 - Προγραμματισμός: σύνθεση αυτής της συνάρτησης από απλούστερες
 - Οι έξοδοι από μια συνάρτηση χρησιμοποιούνται ως είσοδοι σε άλλες συναρτήσεις
 - Παράδειγμα συναρτησιακής γλώσσας: LISP
 - **Πλεονέκτημα** σε σχέση με τις προστακτικές γλώσσες: δεν χρειάζονται πολλές μεταβλητές → καταναλώνεται **λιγότερος χώρος στη μνήμη**

Παράδειγμα Συναρτησιακού Μοντέλου



Σε μια προστακτική γλώσσα
Προγραμματισμού θα είχαμε

```
Temp_balance ← Old_balance + credits  
Total_debits ← sum of all Debits  
Balance ← Temp_balance - Total_debits
```

Σε LISP: (Find_diff (Find_sum Old_balance Credits) (Find_sum Debits))

Προγραμματιστικό μοντέλο (3): Δηλωτικό

- **Δηλωτικό (declarative)** μοντέλο: ο προγραμματιστής πρέπει να περιγράψει το πρόβλημα προς επίλυση, και όχι τον αλγόριθμο
 - Εφαρμόζεται ένας **γενικός αλγόριθμος επίλυσης** προβλημάτων
 - Ο προγραμματιστής **περιγράφει λεπτομερώς το πρόβλημα και όχι τον αλγόριθμο για την επίλυση του προβλήματος**
 - Χρήση για **προσομοίωση συστημάτων** (οικονομία, πολιτική, μετεωρολογία, περιβάλλον) π.χ. για έλεγχο υποθέσεων ή πρόγνωση
 - Ο αλγόριθμος πρόγνωσης υπάρχει στην γλώσσα
 - Παραδείγματα: Prolog (τομέας Τεχνητής Νοημοσύνης, δεκαετία 1970-80)
- Σημείωση: ο όρος «Τεχνητή Νοημοσύνη» στις μέρες μας δεν έχει να κάνει με Prolog...
- ...αλλά με Machine Learning!

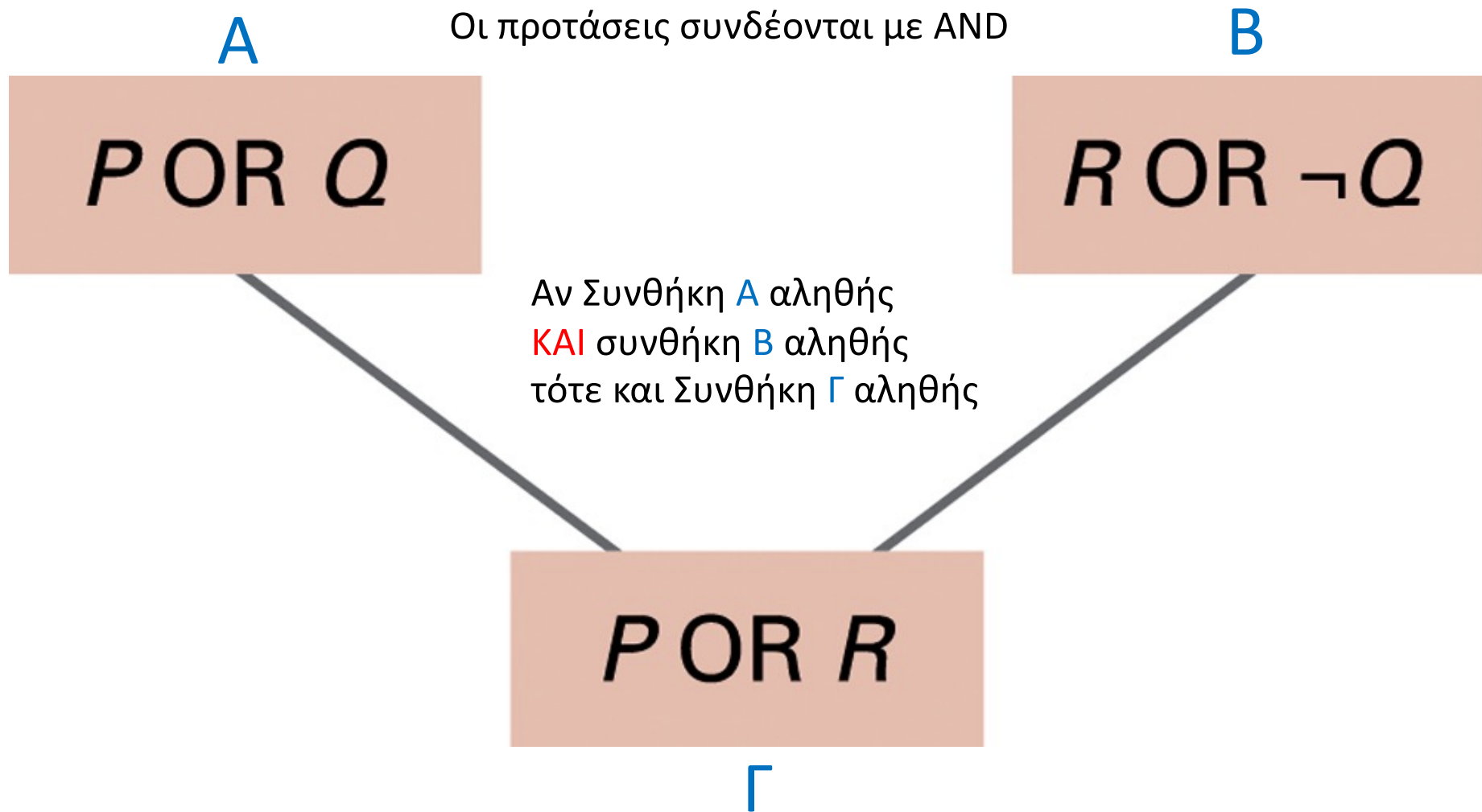
(Πολύ Σύντομη) Εισαγωγή στην Prolog

- **Κατηγορημα (predicate):** βασικό δομικό στοιχείο της γλώσσας
- **Γεγονός (Fact):** Κατηγορημα(όρισμα, όρισμα, ..)
 - Παραδείγματα: `parent(bill, mary).` `faster(rabbit, turtle).`
- **Κανόνας (Rule):** συμπέρασμα :- πρόταση
 - :- σημαίνει “αν”
 - Παράδειγμα: `wise(X) :- old(X).` Σημαίνει $old(X) \rightarrow wise(X)$
 - Παράδειγμα: `faster(X, Z) :- faster(X, Y), faster(Y, Z).`
Σημαίνει $faster(X, Y) \text{ AND } faster(Y, Z) \rightarrow faster(X, Z)$
- Όλες οι προτάσεις είναι είτε γεγονότα είτε κανόνες
 - Ο προγραμματιστής καθορίζει τα γεγονότα και τους κανόνες

Εισαγωγή στην Λογική για Δηλωτικό Προγραμματισμό

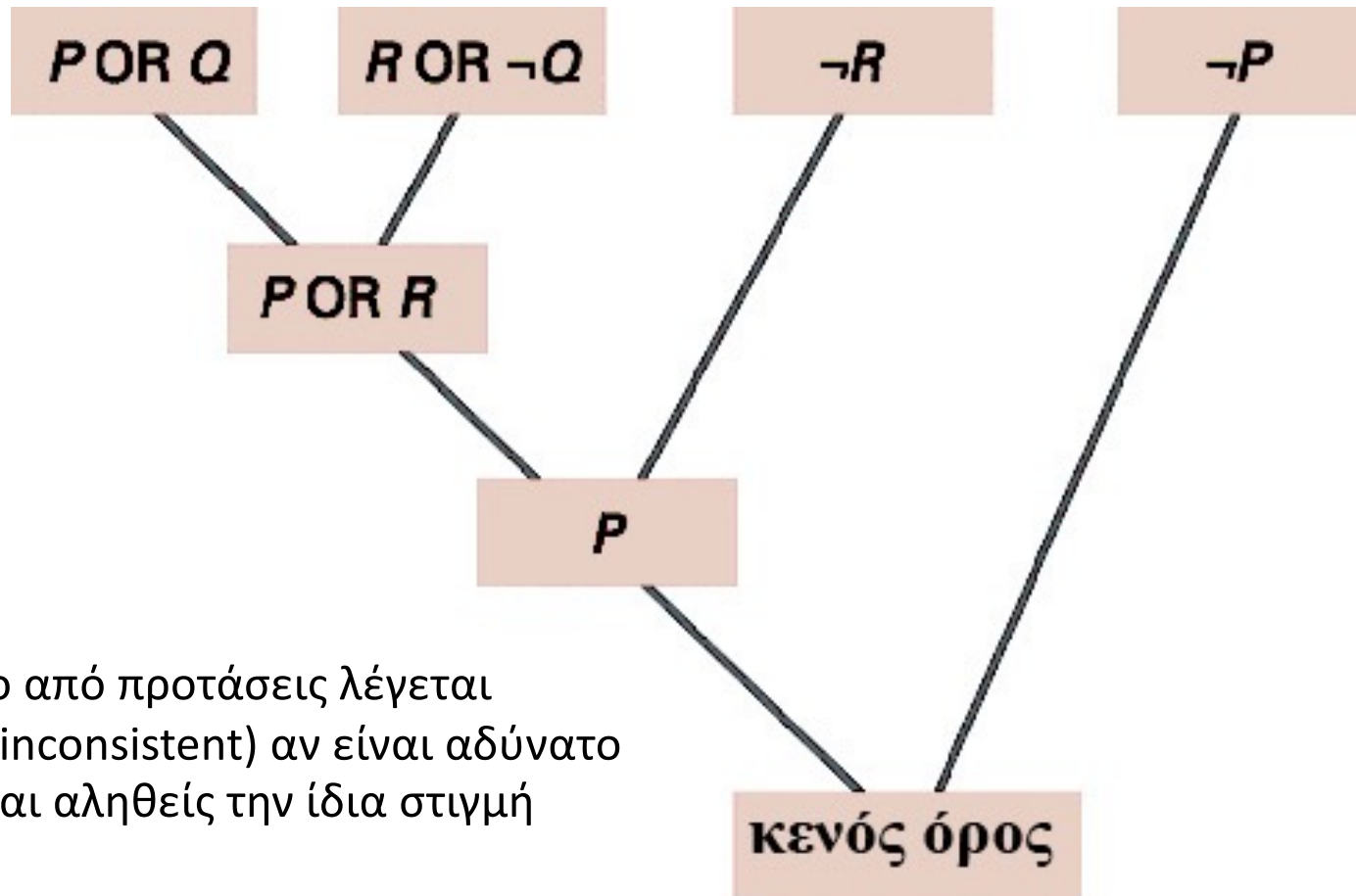
- **Επίλυση (resolution)**: συνδυασμός δύο ή περισσότερων προτάσεων για παραγωγή μίας νέας πρότασης (που είναι αληθής αν οι αρχικές προτάσεις είναι αληθείς)
 - **Επιλυθέν (resolvent)**: η νέα πρόταση που σχηματίζεται από την επίλυση
 - Παράδειγμα: $(P \text{ OR } Q) \text{ AND } (R \text{ OR } \neg Q)$ δίνει ως αποτέλεσμα (\rightarrow) την $(P \text{ OR } R)$ (Σημείωση $\neg Q = \text{NOT } Q$)
 - Σημαίνει ότι αν η αρχική πρόταση, π.χ. $(P \text{ OR } Q) \text{ AND } (R \text{ OR } \neg Q)$, είναι αληθής, τότε και το αποτέλεσμα, $(P \text{ OR } R)$, είναι αληθές. Γιατί;
 - Αν Q αληθής, τότε R πρέπει να είναι αληθής. Αν Q ψευδής, τότε η P πρέπει να είναι αληθής. Άρα $P \text{ OR } R$ είναι αληθής, ανεξάρτητα του τι είναι το Q .

Παράδειγμα: Επίλυση των προτάσεων $(P \text{ OR } Q)$ και $(R \text{ OR } \neg Q)$ ώστε να παραχθεί η $(P \text{ OR } R)$



Άσκηση: Επίλυση των προτάσεων $(P \text{ OR } Q)$, $(R \text{ OR } \neg Q)$, $\neg R$, και $\neg P$

Οι προτάσεις συνδέονται με AND



Ένα σύνολο από προτάσεις λέγεται **ασυνεπές** (inconsistent) αν είναι αδύνατο όλες να είναι αληθείς την ίδια στιγμή

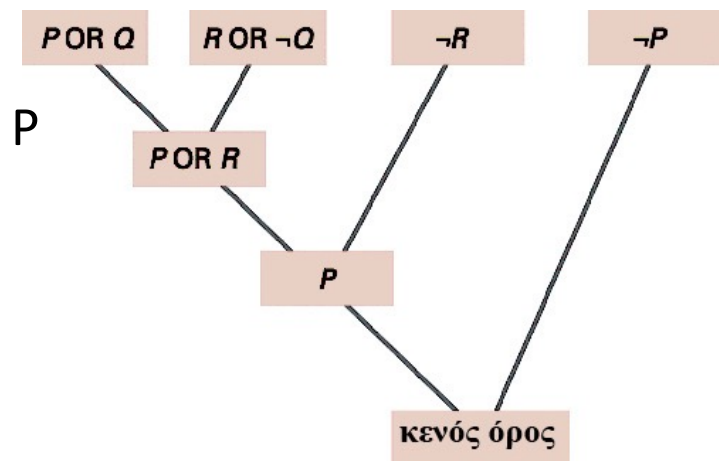
Επαναληπτική επίλυση για να βρεθεί αν το σύνολο προτάσεων είναι ασυνεπές ή όχι

Χρήση ασυνεπούς συνόλου προτάσεων για την επίλυση $W \rightarrow P$

Πώς μπορούμε να αποδείξουμε $W \rightarrow P$ με τη χρήση των παραπάνω;

- Πώς μπορούμε να εκφράσουμε το $W \rightarrow P$ σαν πρόταση (\cdot OR \cdot);
 $W \rightarrow P$ είναι ισοδύναμο με $(P$ OR $\neg W)$
- Εναλλακτικά: πότε η $W \rightarrow P$ **δεν ισχύει**;
Όταν η **W** ισχύει και η **P** δεν ισχύει
Ή αλλιώς όταν ικανοποιείται η $(W$ AND $\neg P)$
- Συμπέρασμα: $W \rightarrow P$ ισχύει όταν $(W$ AND $\neg P)$ δεν ισχύει, δηλαδή όταν το σύνολο των προτάσεων $W, \neg P$ είναι ασυνεπές.

- Παράδειγμα: $((P$ OR $Q)$ AND $(R$ OR $\neg Q)$ AND $\neg R) \rightarrow P$
- Εδώ $W = (P$ OR $Q)$ AND $(R$ OR $\neg Q)$ AND $\neg R$
- Η $W \rightarrow P$ ισχύει όταν το σύνολο των προτάσεων $(P$ OR $Q), (R$ OR $\neg Q), \neg R, \neg P$ είναι ασυνεπές

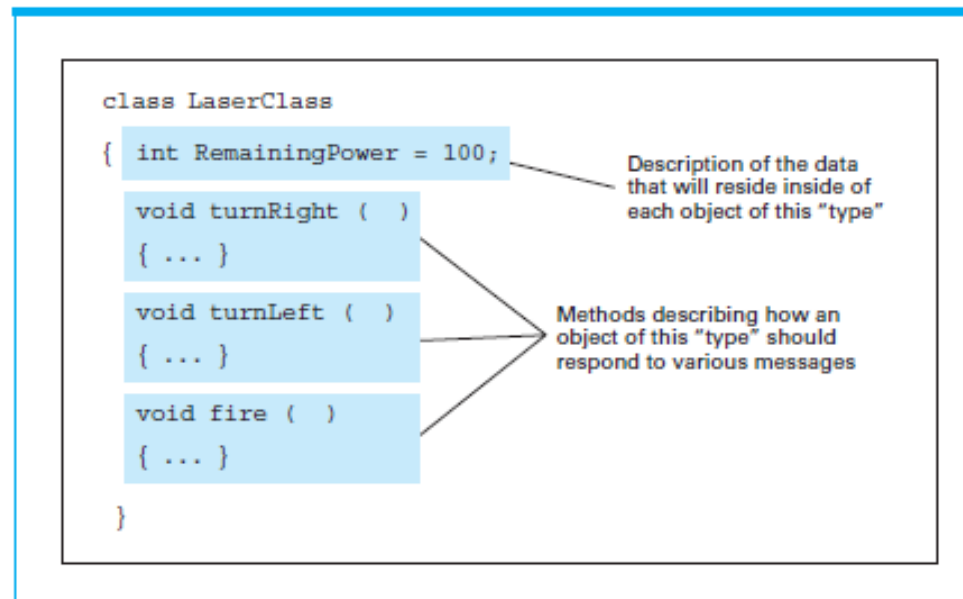


Προγραμματιστικό μοντέλο (4): Αντικειμενοστραφές

- **Αντικειμενοστραφές (object-oriented) /Object-oriented Programming(OOP)**
 - Κατά βάση προστακτικό
 - Μονάδες δεδομένων: ενεργά **αντικείμενα**
 - Ένα αντικείμενο μπορεί να αποτελείται από **δεδομένα** και **διαδικασίες χειρισμού** (εισαγωγή/ αφαίρεση στοιχείου στη λίστα, ταξινόμηση, αναζήτηση)
- **Κλάση (class) αντικειμένων**: σύνολο ιδιοτήτων που χαρακτηρίζουν ίδια αντικείμενα (δηλ. που ανήκουν στην ίδια κλάση)
- Ορίζεται μια κλάση αντικειμένων π.χ. Εικονίδιο, Δορυφόρος, Σταθμός
 - Έπειτα ορίζονται **αντικείμενα** π.χ. icon1, icon2, Satellite1, Satellite2 ως **εκδοχές (instances) της κλάσης**
- Παραδείγματα αντικειμενοστραφών γλωσσών: C++, Visual Basic, Java, C#
- Η python ακολουθεί το **προστακτικό**, το **συναρτησιακό** και το **αντικειμενοστρεφές** μοντέλο (συνδυασμός, με διαφορετικά στοιχεία από το καθένα μοντέλο)
 - Γρήγορο online python tutorial: <https://www.w3schools.com/python/default.asp>

Αντικειμενοστραφής προγραμματισμός

- **Αντικείμενο** (object): αυτόνομη προγραμματιστική μονάδα που περιέχει δεδομένα και διαδικασίες
- **Κλάση** (class): όλα τα αντικείμενα του ίδιου τύπου
- Ένα αντικείμενο ονομάζεται **στιγμιότυπο** (instance) της κλάσης.



- **Ορισμός αντικειμένων** στην C++: `LaserClass Laser1, Laser2, Laser3;`
στην Java: `LaserClass Laser1 = new LaserClass();`

Στοιχεία ενός αντικειμένου

- **Μεταβλητή στιγμιοτύπου (instance variable):** μεταβλητή που βρίσκεται στο εσωτερικό ενός αντικειμένου.
 - Π.χ. `RemainingPower`
- **Μέθοδος (method):** Διαδικασία στο εσωτερικό του αντικειμένου
 - Μπορεί να χειριστεί τις μεταβλητές στιγμιοτύπου του αντικειμένου
 - Π.χ. `turnRight()`, `turnLeft()`, `fire()`

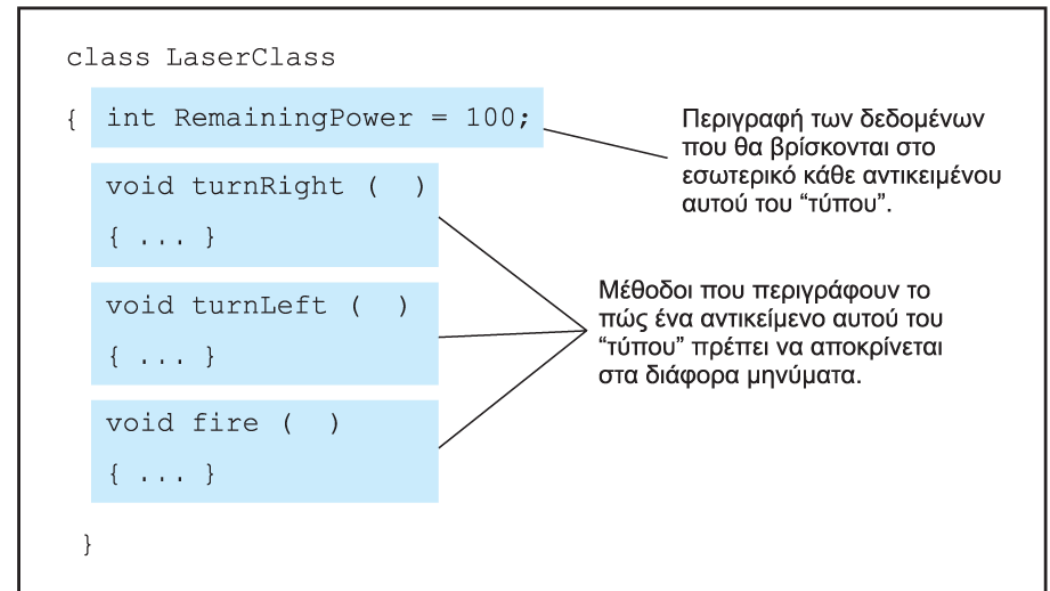
```
Laser1.fire();
```

```
Laser2.turnLeft();
```

- **Κληρονομικότητα (inheritance):** επιτρέπει σε μία κλάση να περιλαμβάνει ιδιότητες κάποιας άλλης ορισμένης κλάσης

```
class RechargeableLaser extends LaserClass  
{  
    .  
    .  
}
```

```
RechargeableLaser Laser3, Laser4;
```



Μέθοδος Κατασκευής

- **Μέθοδος κατασκευής** (constructor): εκτελείται **αυτόματα κατά τη δημιουργία** ενός αντικειμένου από την κλάση
 - Έχει το **ίδιο όνομα** με το αντικείμενο
 - Βοηθά στην **διαφορετική αρχικοποίηση** ή ανάθεση διαφορετικών παραμέτρων σε αντικείμενα της ίδιας κλάσης

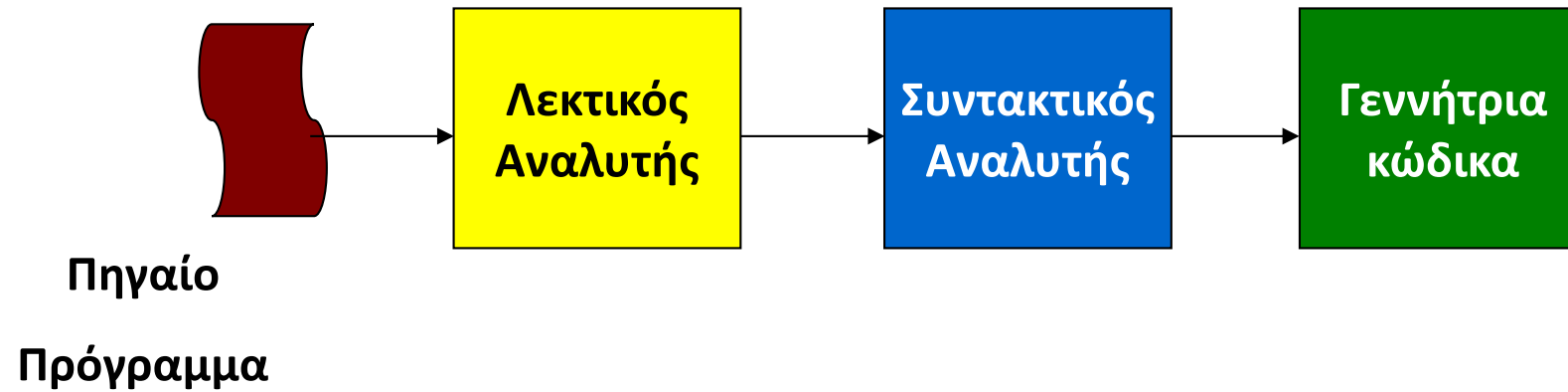
```
LaserClass Laser1(50), Laser2(100);
```

```
LaserClass Laser1 = new LaserClass(50);  
LaserClass Laser2 = new LaserClass(100);
```

```
class LaserClass  
{ int RemainingPower;  
  
  { LaserClass (InitialPower)  
  { RemainingPower = InitialPower;  
  }  
  
  void turnRight ( )  
  { ... }  
  
  void turnLeft ( )  
  { ... }  
  
  void fire ( )  
  { ... }  
  
}
```

Constructor assigns a value to Remaining Power when an object is created.

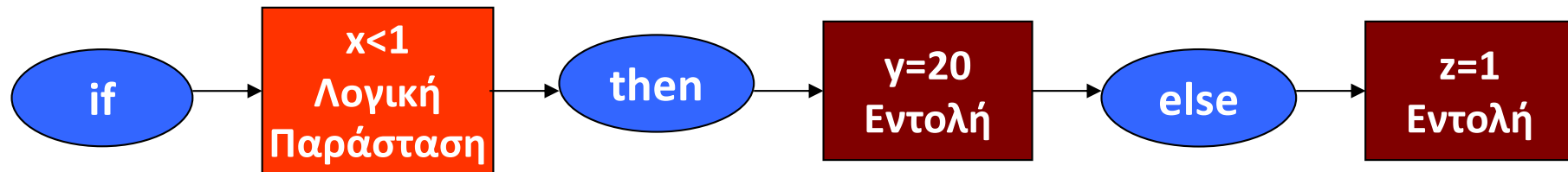
Η διαδικασία της Μετάφρασης



Πηγαίο πρόγραμμα: π.χ. `if x<1 then y=20 else z=1`

Λεκτικός αναλυτής: `'if' 'x' '<' '1' 'then' 'y' '=' '20' 'else' 'z' '=' '1'`

Συντακτικός αναλυτής: `if, then, else`: reserved words

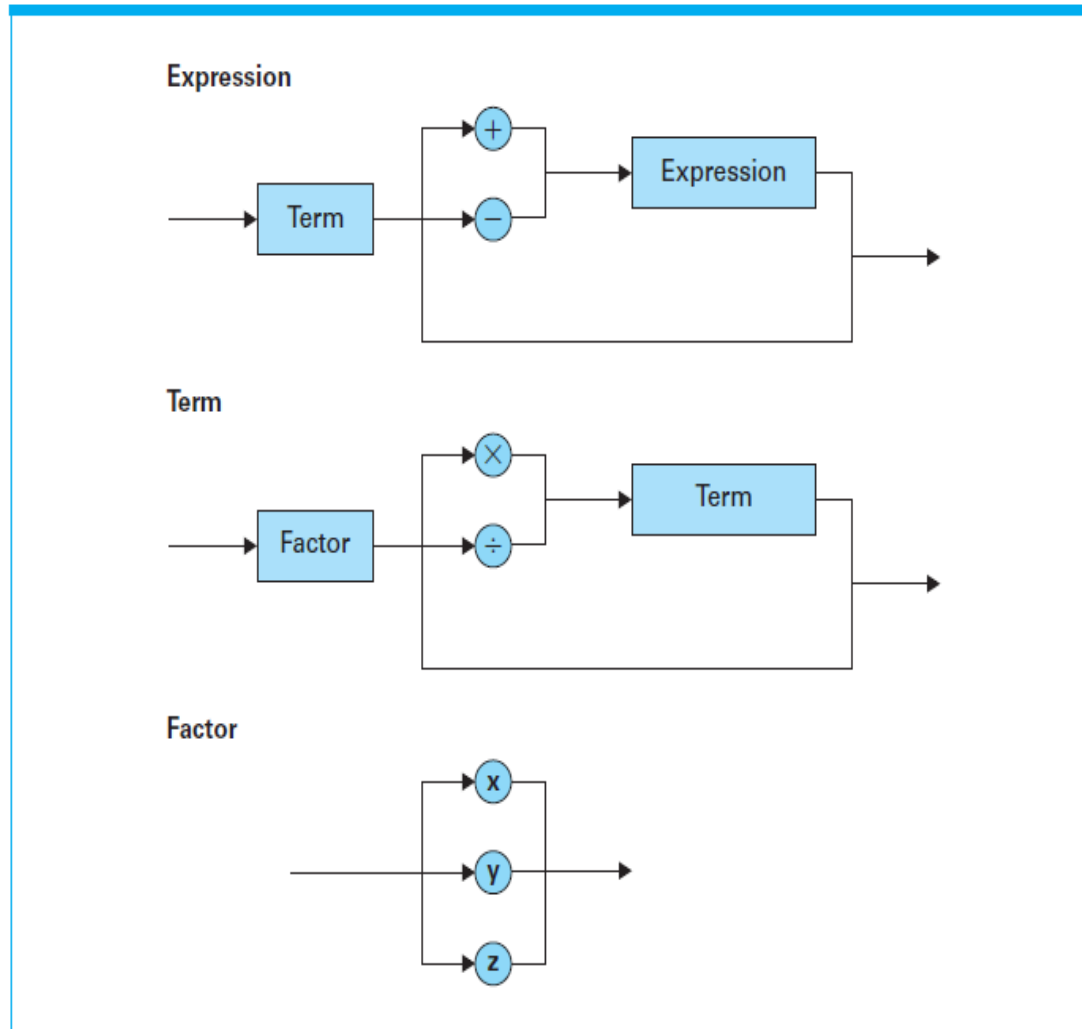


Γεννήτρια κώδικα: Μετάφραση σε γλώσσα μηχανής

Μετάφραση

- **Λεκτικός αναλυτής (lexical analyzer):** διαβάσει το πρόγραμμα χαρακτήρα προς χαρακτήρα, αναγνωρίζει συμβολικές οντότητες (tokens)
- **Συντακτικός αναλυτής (parser):** συντάσσει τα tokens σε προτάσεις και statements
 - Αναγνώριση γραμματικής δομής του προγράμματος
 - Αναγνώριση ρόλου κάθε token
 - Δεσμευμένες λέξεις (reserved words)
 - Ο parser χρησιμοποιεί την Γραμματική (grammar) της γλώσσας
 - **Γραμματική: σύνολο από συντακτικούς κανόνες**
- **Γεννήτρια κώδικα (code generator):** Κατασκευή των εντολών γλώσσας μηχανής.
 - Μπορεί να χρησιμοποιεί και **βελτιστοποίηση κώδικα**
 - Π.χ. $x \leftarrow y + z$, $w \leftarrow x + y$. Αφού εκτελεστεί η 1^η εντολή, **οι τιμές των x και y βρίσκονται ήδη στην ΚΜΕ**, οπότε **δεν χρειάζεται να φορτωθούν ξανά.**

Παράδειγμα: Γραμματική αριθμητικής παράστασης



Κανόνες BNF (Backus-Naur Form):

$E \rightarrow T + E \mid T - E \mid T$

$T \rightarrow F * T \mid F / T \mid F$

$F \rightarrow x \mid y \mid z$

Δημιουργία $x + y * z$:

factor1 = x, factor2 = y, factor3 = z

term1 = factor3 = z

term2 = factor2 * term1 = y * z

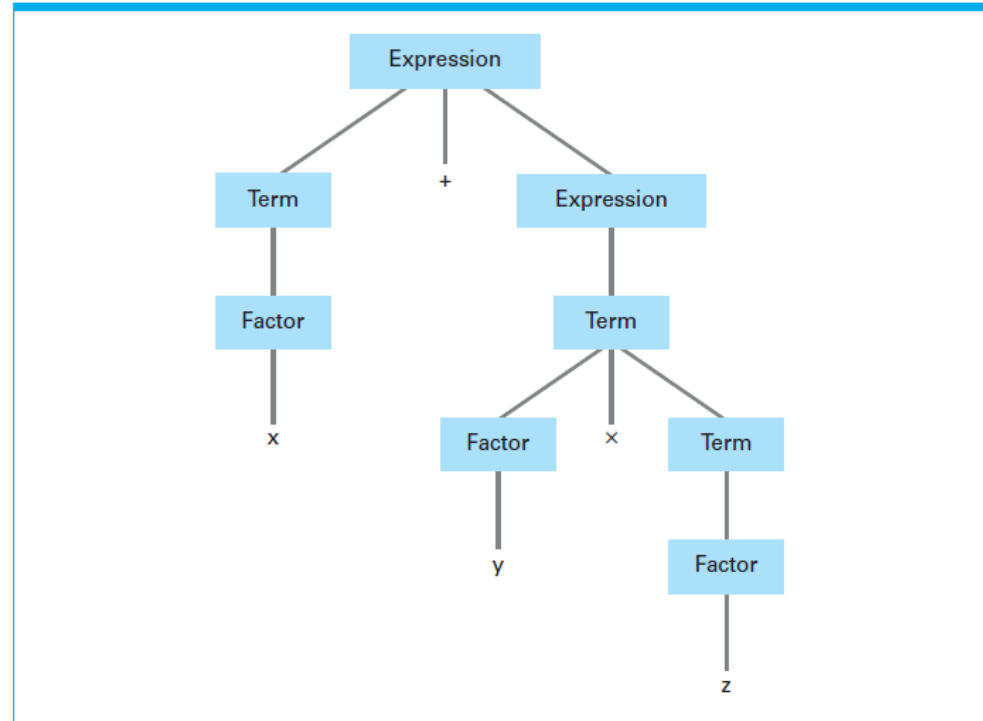
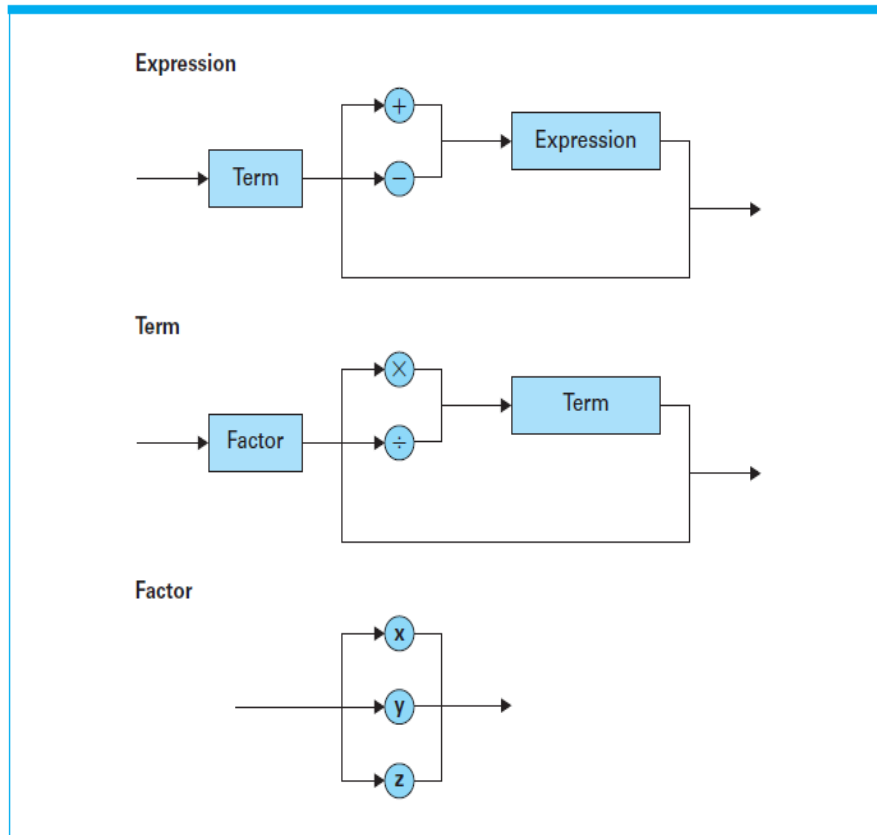
term3 = factor1 = x

expression1 = term2 = y * z

expression2 = term3 + expression1

= x + y * z

Δέντρο συντακτικής ανάλυσης για την παράσταση $x+y*z$



Ερώτηση: Είναι η $x+y*z$ συντακτικά σωστή;
Δηλ. είναι το $x+y*z$ Expression;

Απάντηση: Ναι, αν

- υπάρχει ένα συντακτικό δέντρο που να την απεικονίζει
- Αυτό το συντακτικό δέντρο έχει ως φύλλα τα $x, +, y, *, z$

$E \rightarrow T+E \mid T-E \mid T$
 $T \rightarrow F*T \mid F/T \mid F$
 $F \rightarrow x \mid y \mid z$

Παράδειγμα γλώσσας

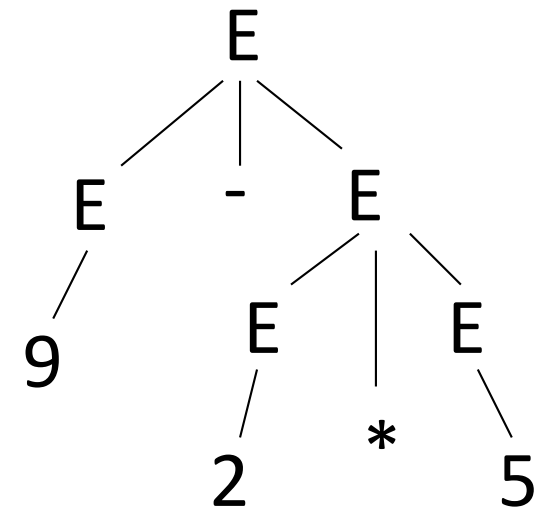
Γλώσσα = απλές αριθμητικές εκφράσεις με 0,...,9 και +,-,*,/

$E \rightarrow E + E \mid E - E \mid E * E \mid E / E$

$E \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

9-2*5 είναι σωστή πρόταση;

6+*4 είναι σωστή πρόταση;



συντακτικό δέντρο

Σωστή (συντακτικά) πρόταση: αυτή η πρόταση για την οποία υπάρχει ένα δέντρο που να την δικαιολογεί

Ασάφεια στην Γραμματική

Και αν υπάρχουν 2 συντακτικά δέντρα; Για παράδειγμα, έστω η γλώσσα:

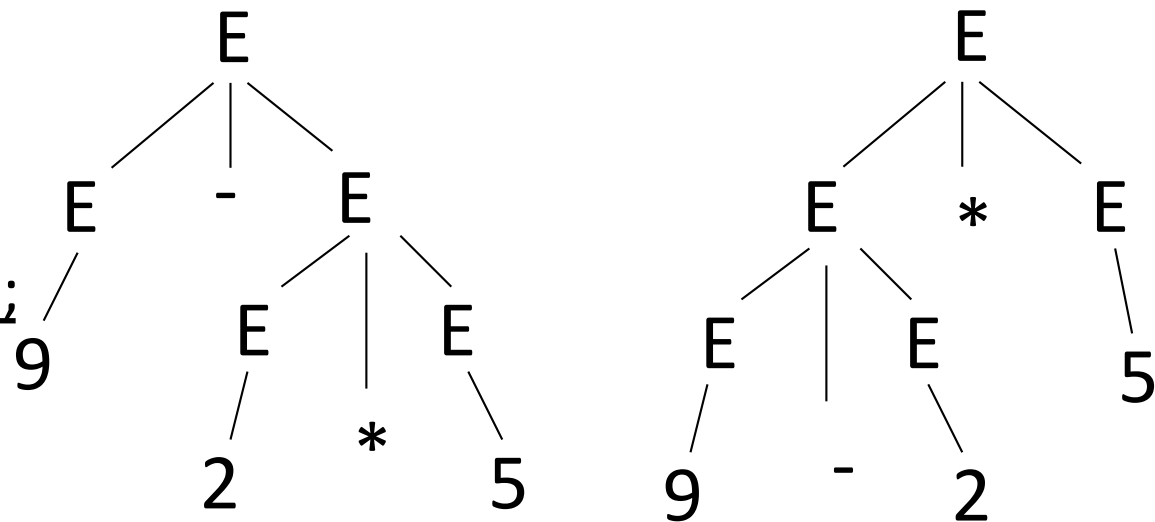
$E \rightarrow E + E \mid E - E \mid E * E \mid E / E$

$E \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

τι σημαίνει $9-2*5$;

2 συντακτικά δέντρα!

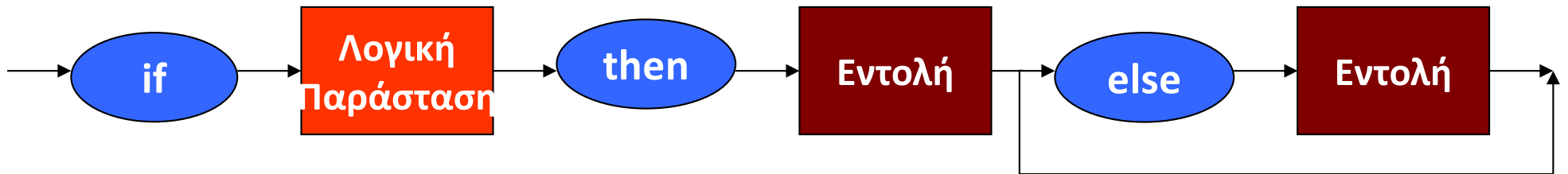
ποιο από τα δύο εννοούμε;



Ασαφής (συντακτικά) πρόταση: αυτή η πρόταση για την οποία υπάρχουν δύο ή περισσότερα προτασιακά δέντρα που να την δικαιολογούν

Περιγραφή Σύνταξης

A. Διαγράμματα σύνταξης



B. Κανόνες BNF

ifstmt -> **if** *log_exp* **then** *stmt* | **if** *log_exp* **then** *stmt* **else** *stmt*

If $x < 1$ then $y = 2$ else $z = 1$

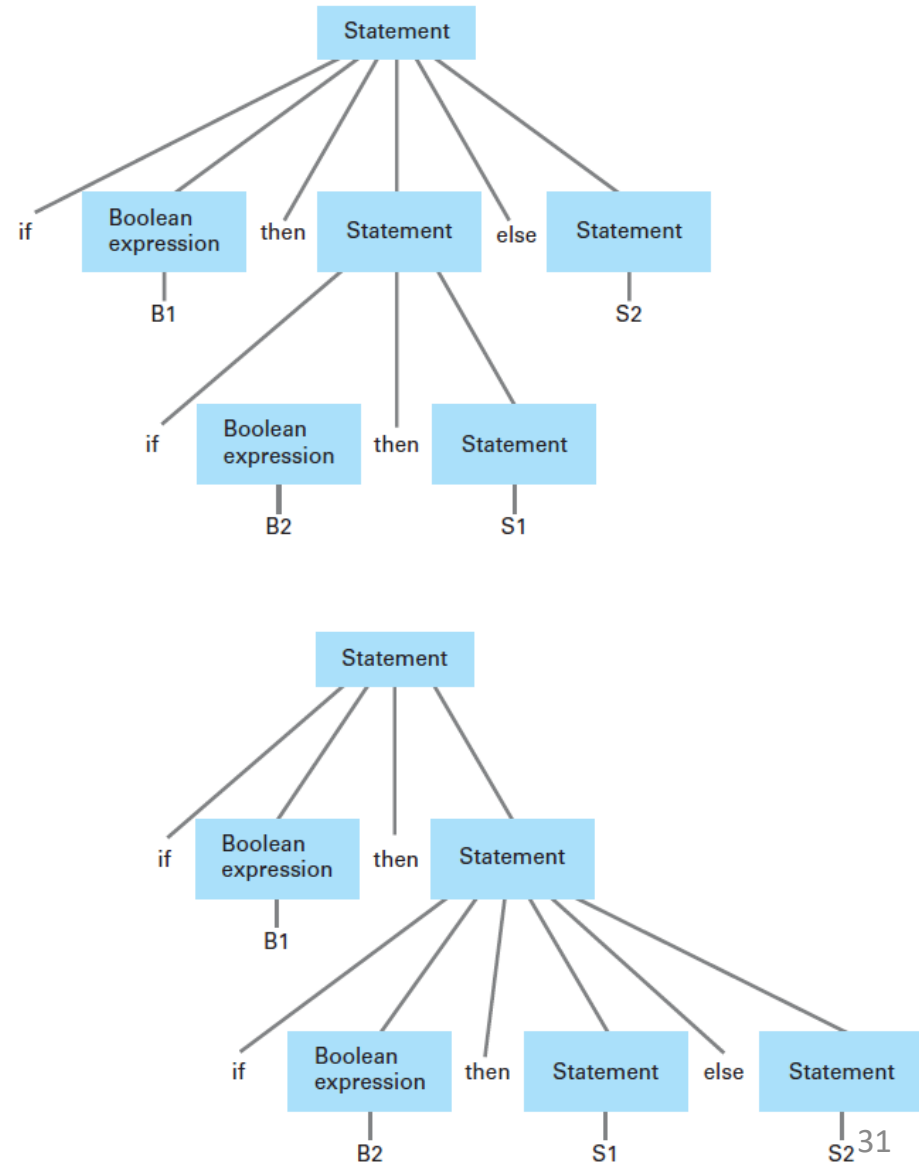
Παράδειγμα ασαφούς γραμματικής

- Θεωρείστε την εντολή:
`if B1 then if B2 then S1 else S2`
- Μπορεί να ερμηνευτεί και με τα **δυο** δέντρα
 - **Ασάφεια** (ambiguity)
 - Σε ποιο if αναφέρεται το else;
 - Στον ψευδοκώδικα αποφεύγουμε το μπέρδεμα ως εξής:

```
if B1
  then (if B2 then S1)
  else S2
```

and

```
if B1
  then (if B2 then S1
        else S2)
```



Για όποιον/α ενδιαφέρεται να ψάξει περισσότερο

- Τι είναι η javascript και η rhr και ποιες οι εφαρμογές τους;
- Γράψτε ένα πολύ απλό πρόγραμμα (π.χ. 5 γραμμές κώδικα) σε javascript και ένα σε rhr και αναφέρετε τι κάνει
- Γράψτε ένα μικρό πρόγραμμα σε Prolog
- Τι παραπάνω έχουν οι C++ και C# σε σχέση με την C;

Τέλος Κεφαλαίου 6

- Μιλήσαμε για προγραμματιστικά μοντέλα και σύνταξη γλώσσας και γραμματική.
- Στο επόμενο κεφάλαιο θα ασχοληθούμε με αλγορίθμους: έννοιες, βασικοί αλγόριθμοι και πολυπλοκότητα.