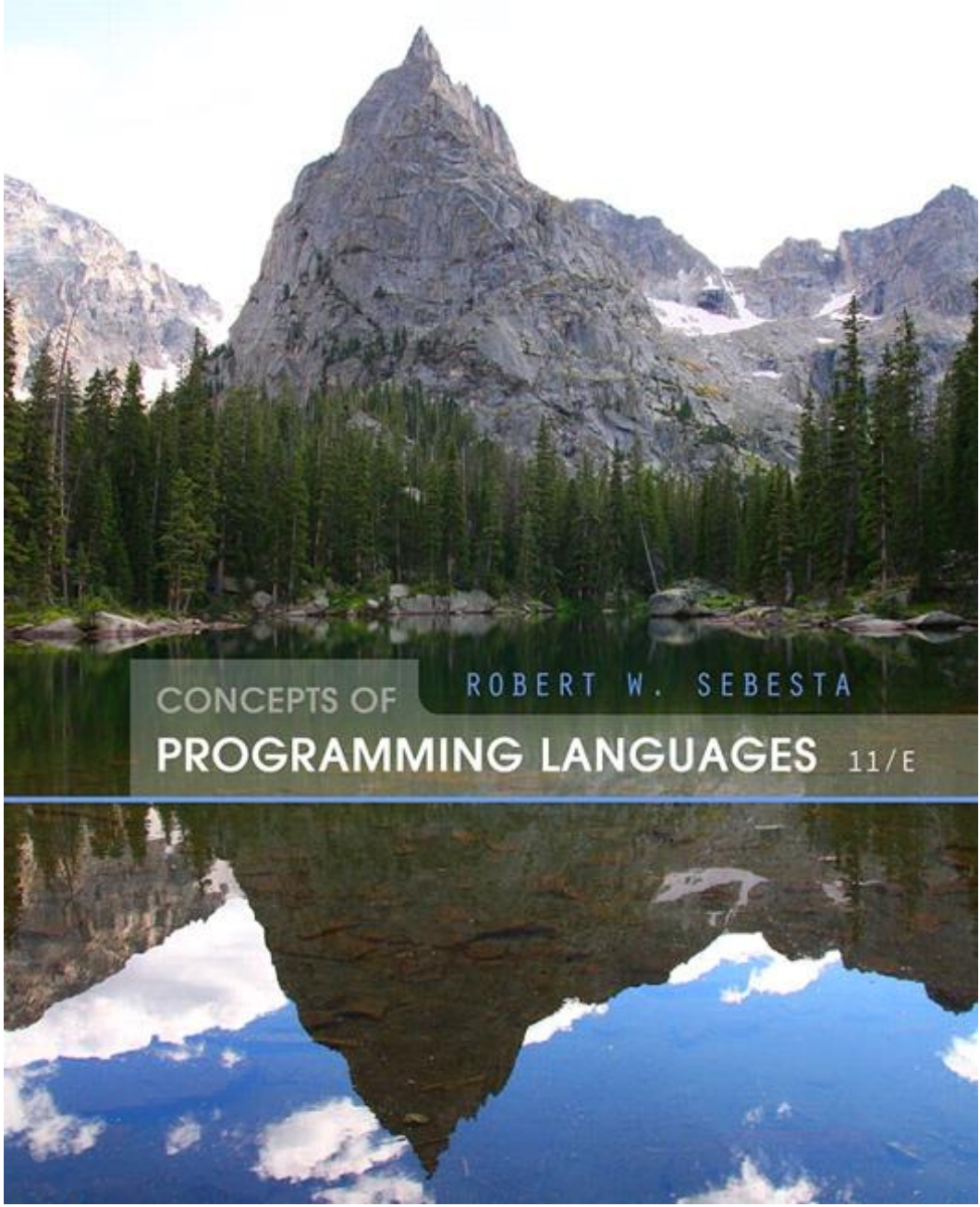
Έκδοση:
31/3/2025

Κεφάλαιο 7

Εκφράσεις και εντολές ανάθεσης



ISBN 0-321-49362-1

Θέματα Κεφαλαίου 7

- Εισαγωγή
- Αριθμητικές Εκφράσεις
- Υπερφορτωμένοι Τελεστές
- Μετατροπές Τύπων
- Σχεσιακές εκφράσεις και Λογικές εκφράσεις
- Εσπευσμένη Αποτίμηση Εκφράσεων
- Εντολές Ανάθεσης
- Αναθέσεις Μεικτής-Κατάστασης

Εισαγωγή

- Οι εκφράσεις (expressions) αποτελούν τον θεμελιώδη τρόπο περιγραφής υπολογισμών στις γλώσσες προγραμματισμού
- Για να είναι κατανοητή η αποτίμηση των εκφράσεων, θα πρέπει κανείς να γνωρίζει τις προτεραιότητες των τελεστών (operators) και της αποτίμησης των τελεστέων (operands)
- Οι εντολές ανάθεσης παίζουν βασικό ρόλο στις προστακτικές γλώσσες

Αριθμητικές Εκφράσεις

- Η πραγματοποίηση αριθμητικών πράξεων ήταν από τα βασικά κίνητρα της ανάπτυξης των πρώτων γλωσσών προγραμματισμού
- Οι αριθμητικές εκφράσεις αποτελούνται από τελετές, τελεστέους, παρενθέσεις και κλήσεις συναρτήσεων

Αριθμητικές εκφράσεις: Θέματα Σχεδιασμού

Θέματα σχεδιασμού για αριθμητικές εκφράσεις

- Κανόνες προτεραιότητας τελεστών
- Κανόνες προσεταιριστικότητας τελεστών
- Σειρά αποτίμησης τελεστέων
- Παρενέργειες (side-effects) αποτίμησης τελεστέων
- Υπερφόρτωση τελεστών
- Ανάμειξη τύπων σε εκφράσεις

Αριθμητικές Εκφράσεις: Τελεστές

- Ένας μοναδιαίος (unary) τελεστής έχει έναν τελεστέο
- Ένας δυαδικός (binary) τελεστής έχει δύο τελεστέους
- Ένας τριαδικός (ternary) τελεστής έχει τρεις τελεστέους

Αριθμητικές Εκφράσεις: Κανόνες Προτεραιότητας Τελεστών

- Οι *κανόνες προτεραιότητας τελεστών* ορίζουν τη σειρά με την οποία, σε μια έκφραση, αποτιμώνται γειτονικοί τελεστές που έχουν διαφορετικά επίπεδα προτεραιότητας
- Τυπικά επίπεδα προτεραιότητας
  1. παρενθέσεις
  2. μοναδιαίοι τελεστές
  3. ** (αν η γλώσσα το υποστηρίζει)
  4. *, /
  5. +, -

Αριθμητικές Εκφράσεις: Κανόνες Προσεταιριστικότητας Τελεστών

- Οι κανόνες προσεταιριστικότητας (associativity) τελεστών καθορίζουν τη σειρά με την οποία αποτιμώνται, σε μια έκφραση, γειτονικοί τελεστές με την ίδια προτεραιότητα
- Τυπικοί κανόνες προσεταιριστικότητας:
  - Από αριστερά προς τα δεξιά, με εξαίρεση το **, που είναι από δεξιά προς τα αριστερά
  - Σε ορισμένες περιπτώσεις γλωσσών οι μοναδιαίοι τελεστές αποτιμώνται από δεξιά προς τα αριστερά (π.χ., στη FORTRAN)
- Η APL είναι διαφορετική, όλοι οι τελεστές έχουν την ίδια προτεραιότητα και η προσεταιριστικότητα ισχύει από δεξιά προς τα αριστερά
- Οι κανόνες προτεραιότητας και προσεταιριστικότητας μπορούν να παρακαμφθούν με παρενθέσεις

Εκφράσεις στην Ruby και στην Scheme

- Ruby
  - Όλοι οι αριθμητικοί τελεστές, οι σχεσιακοί τελεστές, ο τελεστής ανάθεσης καθώς και η δεικτοδότηση πινάκων, οι ολισθήσεις και οι τελεστές δυαδικών ψηφίων, υλοποιούνται ως μέθοδοι
  - Ως αποτέλεσμα, όλοι αυτοί οι τελεστές μπορούν να επαναοριστούν (να γίνουν override) από τον προγραμματιστή
- Scheme (και Common Lisp)
  - Όλες οι αριθμητικές και λογικές λειτουργίες πραγματοποιούνται με ρητή κλήση υποπρογραμμάτων
  - `a + b * c` κωδικοποιείται ως `(+ a (* b c))`

Αριθμητικές Εκφράσεις: Εκφράσεις υπό Συνθήκη

- Εκφράσεις υπό συνθήκη
  - C-based γλώσσες (π.χ., C, C++)
  - Ένα παράδειγμα:

```
average = (count == 0)? 0 : sum / count
```

  - Αποτιμάται σαν να έχει γραφεί ως εξής:

```
if (count == 0)
   average = 0
else
   average = sum / count
```

Αριθμητικές Εκφράσεις: Σειρά Αποτίμησης Τελεστέων

Σειρά αποτίμησης τελεστέων

1. Μεταβλητές: ανάκληση της τιμής από τη μνήμη
2. Σταθερές: σε κάποιες περιπτώσεις γίνεται ανάκληση από τη μνήμη, σε άλλες η σταθερά βρίσκεται σε εντολή γλώσσας μηχανής
3. Εκφράσεις σε παρενθέσεις: αποτίμηση κατά προτεραιότητα
4. Η πλέον ενδιαφέρουσα περίπτωση είναι όταν ο τελεστέος είναι μια κλήση συνάρτησης

Αριθμητικές Εκφράσεις: Πιθανότητα Παρενεργειών

- *Παρενέργειες συναρτήσεων (functional side effects):* όταν μια συνάρτηση αλλάζει μια παράμετρο δύο δρόμων (εισόδου και εξόδου) ή μια μη-τοπική μεταβλητή
- Πρόβλημα με τις παρενέργειες συναρτήσεων:
  - Όταν μια συνάρτηση που αναφέρεται σε μια έκφραση τροποποιεί έναν άλλο τελεστέο της έκφρασης, π.χ.:

```
a = 10;
/* υποθέστε ότι η συνάρτηση fun αλλάζει την
παράμετρό της */
b = a + fun(&a);
```

Παρενέργειες συναρτήσεων

Δύο πιθανές λύσεις στο πρόβλημα:

1. Ο ορισμός της γλώσσας να μην επιτρέπει να συμβούν παρενέργειες συναρτήσεων
   - Να μην επιτρέπονται παράμετροι δύο δρόμων στις συναρτήσεις
   - Να μην επιτρέπονται μη τοπικές αναφορές στις συναρτήσεις
   - **Πλεονέκτημα:** δεν υπάρχουν παρενέργειες συναρτήσεων
   - **Μειονέκτημα:** έλλειψη ευελιξίας λόγω ύπαρξης μόνο παραμέτρων ενός δρόμου και απουσίας μη-τοπικών μεταβλητών
2. Ο ορισμός της γλώσσας να γραφεί με την απαίτηση η σειρά αποτίμησης των τελεστέων να γίνεται με προκαθορισμένο τρόπο
   - **Μειονέκτημα**: Ο μεταγλωττιστής τότε μπορεί να επιτελέσει περιορισμένες βελτιστοποιήσεις
   - Η Java ορίζει ότι οι τελεστέοι μιας έκφρασης αποτιμώνται από αριστερά προς τα δεξιά

Αναφορική Διαφάνεια (Referential Transparency)

- Ένα πρόγραμμα έχει την ιδιότητα της αναφορικής διαφάνειας *(referential transparency),* αν οποιεσδήποτε δύο εκφράσεις του προγράμματος που έχουν την ίδια τιμή μπορούν να υποκαταστήσουν η μια την άλλη οπουδήποτε στο πρόγραμμα, χωρίς να επηρεάζεται η λειτουργία του προγράμματος

```
result1 = (fun(a) + b) / (fun(a) - c);
temp = fun(a);
result2 = (temp + b) / (temp - c);
```

Αν η `fun` δεν έχει παρενέργειες, `result1 = result2`

Αν η `fun` έχει παρενέργειες, τότε παραβιάζεται η αναφορική διαφάνεια

Αναφορική Διαφάνεια (συνέχεια)

- Πλεονέκτημα της αναφορικής διαφάνειας
  - Η σημασιολογία του προγράμματος είναι ευκολότερο να γίνει κατανοητή
- Οι καθαρές συναρτησιακές γλώσσες δεν διαθέτουν μεταβλητές και συνεπώς έχουν αναφορική διαφάνεια
  - Οι συναρτήσεις δεν μπορούν να έχουν κατάσταση, που θα αποθηκεύονταν σε τοπικές μεταβλητές
  - Αν μια συνάρτηση χρησιμοποιεί μια εξωτερική τιμή, θα πρέπει η τιμή αυτή να είναι σταθερή (δεν υπάρχουν μεταβλητές). Συνεπώς, η τιμή μιας συνάρτησης εξαρτάται μόνο από τις παραμέτρους της

Υπερφορτωμένοι Τελεστές

- Η χρήση ενός τελεστή για περισσότερους από έναν σκοπούς, ονομάζεται υπερφόρτωση τελεστή *(operator overloading)*
- Ορισμένοι τελεστές μπορούν να είναι κοινοί (π.χ., ο τελεστής + είναι κοινός τόσο για **`int`** όσο και για **`float`**)
- Ορισμένοι τελεστές μπορούν να προκαλέσουν προβλήματα λόγω της υπερφόρτωσης (π.χ., ο τελεστής * στην C και στην C++)
  - Απώλεια ανίχνευσης λαθών από το μεταγλωττιστή (η παράλειψη ενός τελεστέου θα πρέπει να είναι ανιχνεύσιμο λάθος)
  - Υποβάθμιση αναγνωσιμότητας

Υπερφορτωμένοι Τελεστές (συνέχεια)

- Η C++, η C#, και η F# επιτρέπουν υπερφόρτωση τελεστών από τον προγραμματιστή
  - Όταν χρησιμοποιούνται με μέτρο, οι υπερφορτωμένοι τελεστές μπορούν να βελτιώσουν την αναγνωσιμότητα (αποφυγή κλήσης μεθόδων, φυσική εμφάνιση εκφράσεων)
  - Πιθανά προβλήματα:
    - Οι χρήστες μπορούν να ορίσουν τελεστές χωρίς νόημα
    - Η αναγνωσιμότητα μπορεί να υποβαθμιστεί, ακόμα και αν οι τελεστές έχουν νόημα

Μετατροπές Τύπων

- Μια *narrowing* μετατροπή μετατρέπει ένα αντικείμενο σε έναν τύπο που δεν μπορεί να περιέχει όλες τις τιμές του αρχικού τύπου π.χ., `float` σε `int`
- Μια *widening* μετατροπή είναι μια μετατροπή που ένα αντικείμενο μετατρέπεται σε έναν τύπο που μπορεί να περιλαμβάνει κατ' ελάχιστο προσεγγίσεις τιμών για όλες τις τιμές του αρχικού τύπου π.χ., `int` σε `float`

Μετατροπές Τύπων: Μεικτή Κατάσταση

- Μια έκφραση μεικτής κατάστασης *(mixed-mode expression)* έχει τελεστέους διαφορετικών τύπων
- Ένα *coercion* (εξαναγκασμός) είναι μια υπονοούμενη μετατροπή τύπων
- Μειονεκτήματα των *coercions* :
  - Μειώνουν την ικανότητα ανίχνευσης λαθών σε τύπους από τον μεταγλωττιστή
- Στις περισσότερες γλώσσες, όλοι οι αριθμητικοί τύποι που συναντώνται σε εκφράσεις γίνονται coerce χρησιμοποιώντας widening μετατροπές
- Στην ML και στην F#, δεν γίνονται coercions εκφράσεων

Ρητές Μετατροπές Τύπων

- Κλήση *casting* σε C-based γλώσσες
- Παραδείγματα
  - `C: (int)angle`
  - `F#: float(sum)`

Παρατηρήστε ότι το συντακτικό της F# είναι παρόμοιο με την κλήση συναρτήσεων

Λάθη σε Εκφράσεις

- Αιτίες
  - Υφιστάμενοι περιορισμοί της αριθμητικής π.χ., διαίρεση με το μηδέν
  - Περιορισμοί αριθμητικής υπολογιστών π.χ., υπερχείλιση (overflow)
- Συχνά, τα παραπάνω προβλήματα, δεν εντοπίζονται κατά το χρόνο εκτέλεσης

Σχεσιακές και Λογικές Εκφράσεις

- Σχεσιακές εκφράσεις
  - Χρήση σχεσιακών τελεστών και τελεστέων διαφόρων τύπων
  - Αποτιμώνται σε κάποια Boolean αναπαράσταση
  - Οι τελεστές που χρησιμοποιούνται μπορεί να διαφέρουν μεταξύ των γλωσσών (`!=`, `/=`, `~=`, `.NE.`, `<>`, `#`)
- Η JavaScript και η PHP έχουν δύο επιπλέον σχεσιακούς τελεστές, τον `===` και τον `!==`
  - Λειτουργούν παρόμοια με τους τελεστές, `==` και `!=`, με τη διαφορά ότι δεν εξαναγκάζουν σε εξαναγκασμένες μετατροπές (coerce) τους τελεστέους τους
  - Η Ruby χρησιμοποιεί το `==` για τελεστή ισότητας που χρησιμοποιεί coercions και το `eql?` ως τελεστή ισότητας που δεν χρησιμοποιεί coersions

Σχεσιακές και Λογικές Εκφράσεις

- Λογικές (Boolean) Εκφράσεις
  - Οι τελεστέοι είναι Boolean και το αποτέλεσμα είναι Boolean
- Η C89 δεν διαθέτει τύπο Boolean, αλλά χρησιμοποιεί τον τύπο `int` με το 0 για το false και οποιαδήποτε άλλη τιμή για true
- Ένα ιδιαίτερο χαρακτηριστικό των εκφράσεων της C:

  `a < b < c` είναι έγκυρη έκφραση, αλλά το αποτέλεσμα δεν είναι ίσως το αναμενόμενο:
  - Ο αριστερός τελεστής αποτιμάται, και επιστρέφει 0 ή 1
  - Το αποτέλεσμα της αποτίμησης στη συνέχεια συγκρίνεται με τον τρίτο τελεστέο (δλδ., το `c`)

Εσπευσμένη Αποτίμηση (Short Circuit evaluation)

- Πρόκειται για μια έκφραση στην οποία το αποτέλεσμα καθορίζεται χωρίς να αποτιμώνται όλοι οι τελεστέοι και οι τελεστές
- Παράδειγμα: `(13 * a) * (b / 13 - 1)`

  Αν το `a` είναι μηδέν, δεν υπάρχει λόγος να αποτιμηθεί το `(b / 13 - 1)`
- Το πρόβλημα της non-short-circuit αποτίμησης

```
index = 0;
while ((index < length) && (LIST[index] != value))
      index++;
```

  - Όταν `index == length`, θα προκληθεί σφάλμα στο `LIST[index]` λόγω της τιμής του δείκτη (υποθέτοντας ότι η `LIST` έχει μήκος `length`)

Εσπευσμένη Αποτίμηση (συνέχεια)

- C, C++, Java: χρησιμοποιούν εσπευσμένη αποτίμηση για τους συνηθισμένους Boolean τελεστές (`&&` και `||`), αλλά επιπλέον παρέχουν και bitwise Boolean τελεστές στους οποίους όμως δεν εφαρμόζεται εσπευσμένη αποτίμηση (`&` και `|`)
- Όλοι οι λογικοί τελεστές στις Ruby, Perl, ML, F#, και Python αποτιμώνται εσπευσμένα
- Η εσπευσμένη αποτίμηση αναδεικνύει τα πιθανά προβλήματα των παρενεργειών σε εκφράσεις
  π.χ., `(a > b) || (b++ / 3)`

Εντολές Ανάθεσης (Assignment Statements)

- Η γενική σύνταξη των εντολών ανάθεσης είναι η ακόλουθη:

  `<target_var> <assign_operator> <expression>`

- Ο τελεστής ανάθεσης
  - `=` Fortran, BASIC, και στις C-based γλώσσες
  - `:=` Ada
- `=` μπορεί να δημιουργεί προβλήματα όταν υπερφορτώνεται ο σχεσιακός τελεστής ισότητας (αυτός είναι ο λόγος για τον οποίο οι C-based γλώσσες χρησιμοποιούν το `==` ως σχεσιακό τελεστή ισότητας)

Εντολές Ανάθεσης: Στόχοι υπό Συνθήκη

- Στόχοι υπό συνθήκη (Perl)

```
($flag ? $total : $subtotal) = 0
```

Που είναι ισοδύναμο με:

```
if ($flag){
  $total = 0
} else {
  $subtotal = 0
}
```

Εντολές Ανάθεσης: Σύνθετοι Τελεστές Ανάθεσης

- Πρόκειται για μορφές ανάθεσης που συχνά συναντώνται στην πράξη
- Παρουσιάστηκαν για πρώτη φορά στην ALGOL, υιοθετήθηκε από τη C και τις C-based γλώσσες
  - Παράδειγμα:

```
a = a + b
```

μπορεί να γραφεί ως

```
a += b
```

Εντολές Ανάθεσης: Μοναδιαίοι Τελεστές Ανάθεσης

- Οι μοναδιαίοι τελεστές ανάθεσης (unary assignment operators) στις C-based γλώσσες συνδυάζουν λειτουργίες μοναδιαίας αύξησης και μοναδιαίας μείωσης με την ανάθεση
- Παραδείγματα:

  `sum = ++count` (το `count` αυξάνεται κατά ένα, και μετά ανατίθεται στο `sum`)

  `sum = count++` (το `count` ανατίθεται στο `sum`, και στη συνέχεια αυξάνεται κατά ένα)

  `count++` (το `count` αυξάνεται κατά ένα)

  `-count++` (το `count` αυξάνεται κατά ένα και μετά λαμβάνεται η αρνητική του τιμή)

  όταν δύο μοναδιαίοι τελεστές εφαρμόζονται στον ίδιο τελεστέο, οι τελεστές εφαρμόζονται από δεξιά προς τα αριστερά, δλδ. το -count++ είναι ισοδύναμο με -(count++) και όχι με (-count)++

Η Ανάθεση ως Έκφραση

- Στις C-based γλώσσες, όπως η Perl, και η JavaScript, η εντολή ανάθεσης παράγει ένα αποτέλεσμα που μπορεί να χρησιμοποιηθεί ως τελεστέος

```
while ((ch = getchar())!= EOF){...}
```

η `ch = getchar()` εκτελείται, το αποτέλεσμα ανατίθεται στο `ch` και χρησιμοποιείται ως τιμή συνθήκης στην εντολή **`while`**

- Μειονέκτημα: πρόκειται για ένα άλλο είδος παρενέργειας εκφράσεων

Πολλαπλές Αναθέσεις

- Η Perl, η Ruby, και η Lua επιτρέπουν αναθέσεις πολλαπλών στόχων, πολλαπλών πηγών

```
($first, $second, $third) = (20, 30, 40);
```

Επίσης, το ακόλουθο είναι έγκυρο και πραγματοποιεί αντιμετάθεση:

```
($first, $second) = ($second, $first);
```

Η Ανάθεση σε Συναρτησιακές Γλώσσες

- Τα αναγνωριστικά στις συναρτησιακές γλώσσες είναι μόνο ονόματα τιμών
- ML
  - Τα ονόματα προσδένονται σε τιμές με το **`val`**

```
val fruit = apples + oranges;
```

  - Αν ακολουθεί ένα άλλο val για το fruit, τότε είναι νέο και διαφορετικό όνομα
- F#
  - Η **`let`** της F# είναι παρόμοια με την **`val`** της ML's, με τη διαφορά ότι η **`let`** δημιουργεί επιπλέον μια νέα εμβέλεια

Αναθέσεις Μεικτού-Τύπου

- Οι εντολές ανάθεσης μπορούν επίσης να είναι μεικτού τύπου
- Στις Fortran, C, Perl, και C++, οποιοσδήποτε αριθμητικός τύπος μπορεί να ανατεθεί σε οποιαδήποτε μεταβλητή αριθμητικού τύπου
- Στην Java και στην C#, επιτρέπονται μόνο widening τύπου αναθέσεις με coercions
- Στην Ada, δεν επιτρέπεται καθόλου coercion κατά την ανάθεση

Σύνοψη

- Εκφράσεις
- Προτεραιότητα τελεστών και προσεταιριστικότητα
- Υπερφόρωση τελεστών
- Εκφράσεις μεικτών τύπων
- Διάφορες μορφές ανάθεσης