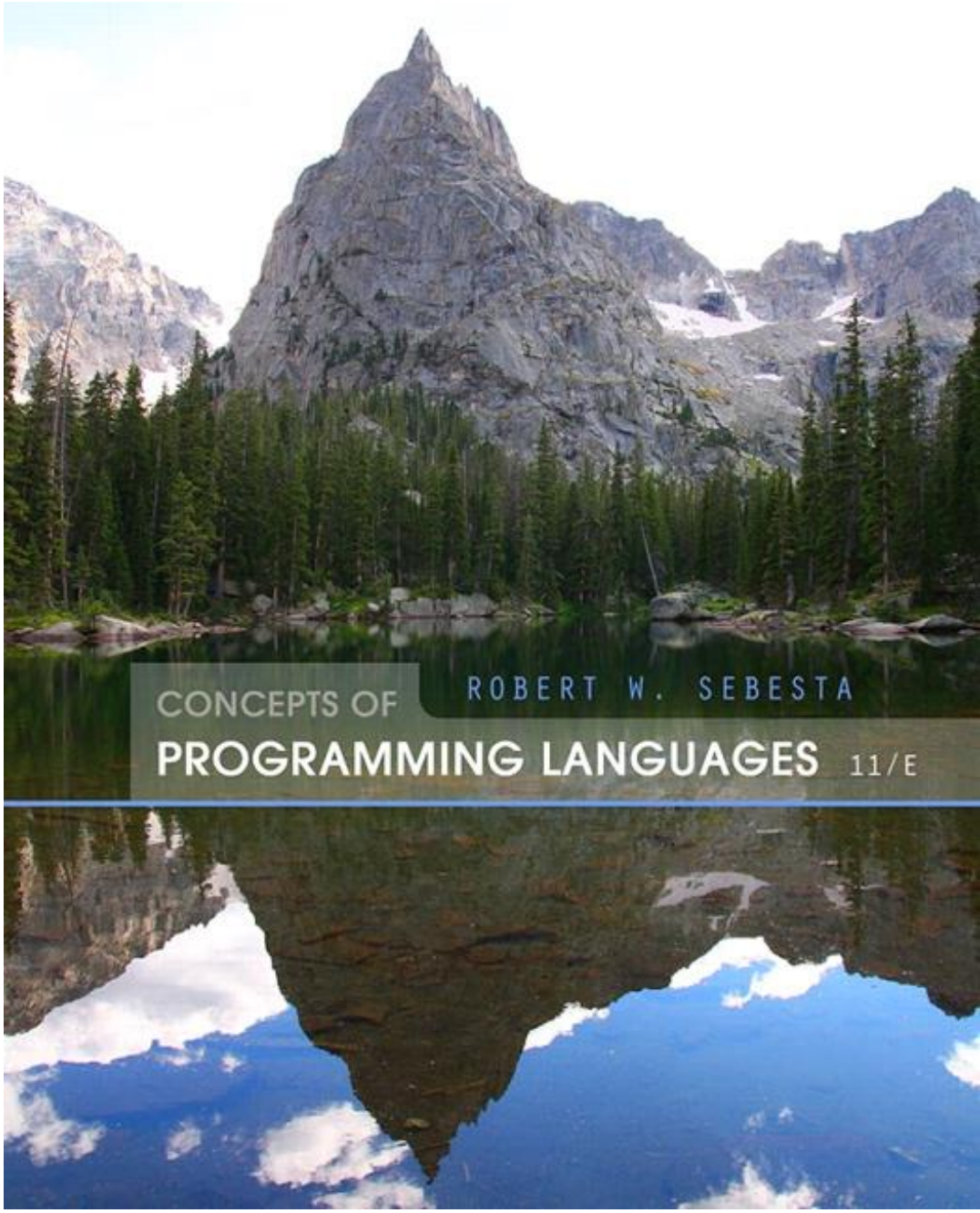
Έκδοση:
26/3/2025

Κεφάλαιο 6

Τύποι δεδομένων (Data Types)



ISBN 0-321-49362-1

Περιεχόμενα Κεφαλαίου 6

- Εισαγωγή
- Πρωτογενείς τύποι δεδομένων
- Τύποι χαρακτήρα και συμβολοσειρών
- Τύποι απαρίθμησης (enumeration types)
- Τύποι διατάξεων (arrays)
- Συσχετιστικές διατάξεις (associative arrays)
- Τύποι εγγραφής (record types)
- Τύποι πλειάδας (tuple types)
- Τύποι λίστας (list types)
- Τύποι ένωσης (union types)
- Δείκτες και τύποι αναφοράς
- Έλεγχος τύπου
- Ισχυροί τύποι (strong typing)
- Ισοδυναμία τύπων
- Θεωρία και τύποι δεδομένων

Εισαγωγή

- Ένας τύπος δεδομένων ορίζει μια συλλογή από αντικείμενα δεδομένων και μια συλλογή από προκαθορισμένες λειτουργίες σε αυτά τα αντικείμενα
- Ένας *descriptor* (περιγραφέας) είναι η συλλογή των ιδιοτήτων μιας μεταβλητής
- Ένα *object* (αντικείμενο) αναπαριστά ένα στιγμιότυπο ενός τύπου ορισμένου από τον χρήστη (αφηρημένων δεδομένων)
- Ένα θέμα σχεδίασης για όλους τους τύπους δεδομένων: Ποιες είναι οι λειτουργίες του και πως αυτές καθορίζονται;

Πρωτογενείς Τύποι Δεδομένων

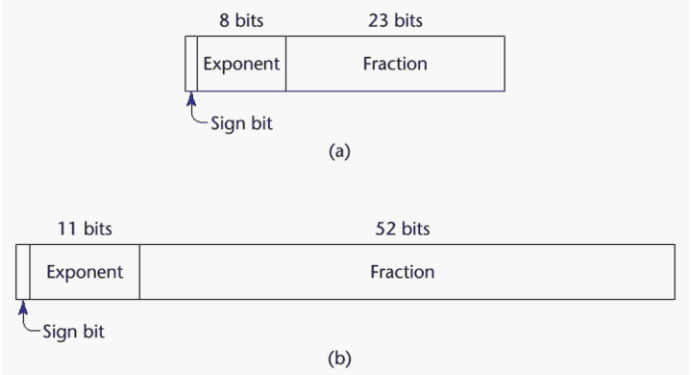
- Σχεδόν όλες οι γλώσσες προγραμματισμού παρέχουν ένα σύνολο από πρωτογενείς τύπους δεδομένων (primitive data types)
- Πρωτογενείς τύποι δεδομένων: Εκείνοι οι τύποι δεδομένων που δεν ορίζονται με όρους άλλων τύπων δεδομένων
- Ορισμένοι πρωτογενείς τύποι δεδομένων είναι απλές αντανακλάσεις του υλικού του Η/Υ
- Άλλοι χρειάζονται μικρή μόνο υποστήριξη μη-υλικού για την υλοποίησή τους

Πρωτογενείς Τύποι Δεδομένων: Ακέραιοι (Integer)

- Σχεδόν πάντα είναι μια ακριβής αναπαράσταση του υλικού ώστε η αντιστοίχιση να είναι απλή
- Μπορούν να υπάρχουν έως και οκτώ διαφορετικοί τύποι ακεραίων σε μια γλώσσα
- Οι προσημασμένοι ακέραιοι στη Java, με αυξανόμενο μέγεθος, είναι: **`byte`**, **`short`**, **`int`**, **`long`**

Πρωτογενείς Τύποι Δεδομένων: Κινητής Υποδιαστολής (Floating Point)

- Αναπαριστά τους πραγματικούς αριθμούς των μαθηματικών, αλλά μόνο ως προσεγγίσεις
- Οι γλώσσες για επιστημονικές εφαρμογές υποστηρίζουν τουλάχιστον δύο τύπους κινητής υποδιαστολής Languages for scientific (π.χ., **`float`** και **`double`**, μερικές φορές περισσότερους
- Συνήθως αναπαρίστανται ακριβώς όπως στο υλικό, αλλά όχι πάντα
- IEEE Floating-Point Standard 754



Πρωτογενείς Τύποι Δεδομένων: Μιγαδικοί Αριθμοί (Complex)

- Μερικές γλώσσες προγραμματισμού υποστηρίζουν τον μιγαδικό τύπο, π.χ., C99, Fortran, και Python
- Κάθε μιγαδική τιμή αποτελείται από δύο floats, το πραγματικό τμήμα και το φανταστικό τμήμα του μιγαδικού αριθμού
- Παράδειγμα μιγαδικού αριθμού (σε Python):

  `(7 + 3j)`, όπου `7` είναι το πραγματικό τμήμα και `3` είναι το φανταστικό τμήμα

Πρωτογενείς Τύποι Δεδομένων: Δεκαδικοί (Decimal)

- Για επιχειρηματικές εφαρμογές (π.χ. χειρισμός χρηματικών ποσών)
  - Πρόκειται για βασικό τύπο δεδομένων στην COBOL
  - Και η C# διαθέτει τύπο δεδομένων decimal
- Αποθηκεύει έναν σταθερό αριθμό δεκαδικών ψηφίων, σε κάποια μορφή κωδικοποίησης (BCD)
- *Πλεονέκτημα*: ακρίβεια πράξεων
- *Μειονεκτήματα*: περιορισμένο εύρος, σπατάλη μνήμης

Πρωτογενείς Τύποι Δεδομένων: Λογικός (Boolean)

- Πρόκειται για τον απλούστερο τύπο δεδομένων
- Ο λογικός τύπος διαθέτει δύο στοιχεία, το “true” και το “false”
- Για τις λογικές τιμές μπορούν να χρησιμοποιηθούν bits, αλλά συχνά χρησιμοποιούνται bytes για καλύτερη αναγνωσιμότητα

Πρωτογενείς Τύποι Δεδομένων: Χαρακτήρας (Character)

- Κάθε χαρακτήρας αποθηκεύεται ως μια αριθμητική κωδικοποίηση
- Η πλέον κοινή κωδικοποίηση χαρακτήρων: ASCII – χρησιμοποιεί 8-bits
- Μια εναλλακτική, κωδικοποίηση 16-bit: Unicode (UCS-2)
  - Περιέχει χαρακτήρες από τις περισσότερες φυσικές γλώσσες που χρησιμοποιούν οι άνθρωποι
  - Αρχικά χρησιμοποιήθηκε στην Java
  - Η C# και η JavaScript επίσης υποστηρίζουν την κωδικοποίηση Unicode
- 32-bit Unicode (UCS-4)
  - Υποστηρίζεται από την Fortran, ξεκινώντας από το 2003

Τύποι Συμβολοσειράς

- Οι τιμές είναι ακολουθίες χαρακτήρων
- Θέματα σχεδίασης:
  - Θα είναι οι συμβολοσειρές (strings) πρωτογενής (primitive) τύπος ή απλά μια ειδική περίπτωση ενός πίνακα;
  - Θα πρέπει το μήκος των συμβολοσειρών να είναι στατικό ή δυναμικό;

Λειτουργίες σε Συμβολοσειρές

- Τυπικές λειτουργίες που εφαρμόζονται σε συμβολοσειρές:
  - Ανάθεση
  - Αντιγραφή
  - Σύγκριση (=, >, κ.λπ.)
  - Συνένωση (catenation)
  - Αναφορά σε υποσυμβολοσειρά μιας συμβολοσειράς
  - Pattern Matching (ταίριασμα προτύπων)

Ο Τύπος Συμβολοσειράς σε Διάφορες Γλώσσες

- C και C++
  - Δεν είναι πρωτογενής (ωστόσο, στην C++ υπάρχει ο πρωτογενής τύπος δεδομένων string)
  - Χρησιμοποιούνται πίνακες **`char`** και μια βιβλιοθήκη συναρτήσεων που παρέχει λειτουργίες
- SNOBOL4 (πρόκειται για μια γλώσσα προγραμματισμού που από την κατασκευή της είναι προσανατολισμένη στο χειρισμό συμβολοσειρών)
  - Πρωτογενής τύπος
  - Πολλές λειτουργίες, συμπεριλαμβανομένου περίτεχνου και με πολλές δυνατότητες pattern matching
- Fortran and Python
  - Πρωτογενής τύπος με ανάθεση και πολλές λειτουργίες
- Java
  - Πρωτογενής μέσω της κλάσης `String`
- Perl, JavaScript, Ruby, and PHP

  – Παρέχουν ενσωματωμένο στη γλώσσα (built-in) pattern matching, με χρήση κανονικών εκφράσεων

Επιλογές για το Μήκος του Τύπου Συμβολοσειράς

- Στατικό: COBOL, η κλάση `String` της Java
- *Περιορισμένο δυναμικό μήκος*: C and C++
  - Σε αυτές τις γλώσσες ένας ειδικός χαρακτήρας χρησιμοποιείται για να υποδηλώσει το τέλος της συμβολοσειράς, αντί να διατηρείται το μήκος της
- *Δυναμικό* (χωρίς περιορισμό μήκους): SNOBOL4, Perl, JavaScript

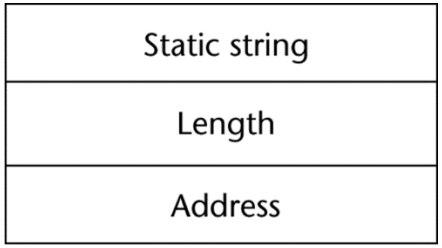
Αποτίμηση Χρησιμότητας Τύπου Συμβολοσειράς

- Η ύπαρξη του τύπου συμβολοσειράς βοηθά στη συγγραφή προγραμμάτων
- Ως πρωτογενής τύπος με στατικό μήκος, η λήψη της τιμής των συμβολοσειρών γίνεται χωρίς σημαντικό υπολογιστικό κόστος -- γιατί να μην είναι σε όλες τις γλώσσες έτσι;
- Το δυναμικό μήκος συμβολοσειρών είναι βολικό αλλά αξίζει να υπάρχει στις γλώσσες;

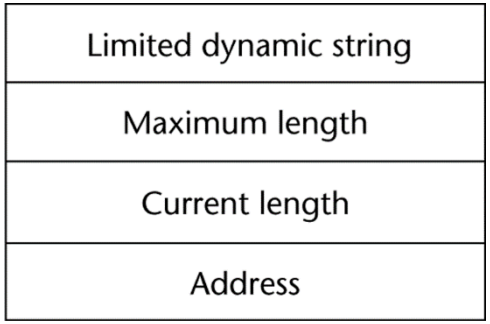
Υλοποίηση Συμβολοσειράς Χαρακτήρων

- Στατικό μήκος: compile-time descriptor
- Περιορισμένο δυναμικό μήκος: μπορεί να χρειάζεται έναν run-time descriptor για το μήκος (όχι όμως στην C και στην C++)
- Δυναμικό μήκος: απαιτεί έναν run-time descriptor, η δέσμευση / αποδέσμευση μνήμης είναι το μεγαλύτερο από το προβλήματα που πρέπει να αντιμετωπίσει η υλοποίηση σε αυτή την περίπτωση

Compile-Time Descriptors και Run-Time Descriptors



Compile-time descriptor για στατικά strings



Run-time descriptor για δυναμικά strings, περιορισμένου μεγέθους

Ordinal Τύποι, Ορισμένοι από τον Χρήστη

- Ένας ordinal (διατακτικός) τύπος είναι ένα τύπος που το εύρος των πιθανών του τιμών μπορεί αντιστοιχιστεί σε ένα υποσύνολο των θετικών ακεραίων
- Παραδείγματα πρωτογενών ordinal τύπων στην Java
  - **`integer`**
  - **`char`**
  - **`boolean`**

Τύποι Απαρίθμησης (Enumeration Types)

- Όλες οι πιθανές τιμές που μπορεί να λάβει μια μεταβλητή τύπου απαρίθμησης, είναι ονοματισμένες σταθερές που παρέχονται κατά τον ορισμό του τύπου απαρίθμησης
- Παράδειγμα στη C#

```
enum days {mon, tue, wed, thu, fri, sat, sun};
```

- Θέματα σχεδίασης
  - Επιτρέπεται σε μια σταθερά απαρίθμησης να εμφανίζεται σε περισσότερους από έναν τύπους απαρίθμησης, και αν ναι, πως ελέγχεται ο τύπος της σταθεράς, σε κάθε εμφάνιση της;
  - Εξαναγκάζονται (coerced) οι σταθερές απαρίθμησης να μετατραπούν σε ακεραίους;
  - Εξαναγκάζεται οποιοσδήποτε άλλος τύπος σε μετατροπή σε τύπο απαρίθμησης;

Αποτίμηση Χρησιμότητας Τύπου Απαρίθμησης

- Βοηθά στην αναγνωσιμότητα, π.χ., δεν υπάρχει ανάγκη, σε ένα πρόγραμμα, να κωδικοποιηθεί ένα χρώμα ως αριθμός
- Βοηθά στην αύξηση της αξιοπιστίας, π.χ., ο μεταγλωττιστής μπορεί να ελέγχει:
  - Επιτρεπτές λειτουργίες (να μην επιτρέπει σε χρώματα να προστεθούν σαν αριθμοί)
  - Δεν μπορεί να ανατεθεί σε μια μεταβλητή απαρίθμησης τιμή εκτός των τιμών που ορίζονται στην απαρίθμηση
  - Η C# και η Java 5.0 παρέχουν βελτιωμένη υποστήριξη για απαριθμήσεις σε σχέση με την C διότι οι μεταβλητές τύπου απαρίθμησης σε αυτές τις γλώσσες δεν μετατρέπονται σε ακέραιο τύπο

Τύποι Πινάκων (Array Types)

- Ένας πίνακας (array) ή διάταξη είναι μιας ομογενής συλλογή στοιχείων δεδομένων, στην οποία το κάθε μεμονωμένο στοιχείο προσδιορίζεται από τη θέση του στη συλλογή που καθορίζεται σε σχέση με το πρώτο στοιχείο της

Θέματα Σχεδίασης Πινάκων

- Ποιοι είναι έγκυροι τύποι δεικτών (subscripts);
- Ελέγχονται οι εκφράσεις που χρησιμοποιούνται ως δείκτες αναφορικά με το εύρος τιμών που λαμβάνουν;
- Πότε προσδένονται τα εύρη τιμών των δεικτών;
- Πότε γίνεται η κατανομή μνήμης;
- Επιτρέπονται οδοντωτοί (ragged) ή πολυδιάστατοι ορθογώνιοι πίνακες;
- Ποιος είναι ο μέγιστος επιτρεπτός αριθμός δεικτών που μπορεί αν χρησιμοποιηθεί;
- Μπορούν οι πίνακες να αρχικοποιηθούν;
- Υποστηρίζονται ο τεμαχισμός (slicing) πινάκων;

Δεικτοδότηση Πινάκων

- *Η δεικτοδότηση* (indexing ή subscripting) είναι μια αντιστοίχιση δεικτών σε στοιχεία

  array_name (index_value_list) → ένα στοιχείο
- Σύνταξη Δεικτών
  - Η Fortran και η Ada χρησιμοποιούν παρενθέσεις
    - Η Ada χρησιμοποιεί ρητά παρενθέσεις για να έχει ομοιομορφία μεταξύ των αναφορών πινάκων και των κλήσεων συναρτήσεων, καθώς και τα δύο είναι αντιστοιχίσεις (mappings)
  - Ωστόσο, οι περισσότερες γλώσσες χρησιμοποιούν αγκύλες αντί για παρενθέσεις για τη σύνταξη των δεικτών

Τύποι Δεικτών (subscript) για Πίνακες

- FORTRAN, C: μόνο ακεραίους
- Java: μόνο τύπους ακεραίων
- Έλεγχος επιτρεπτού εύρους δεικτών
  - Οι C, C++, Perl, Fortran δεν διαθέτουν έλεγχο εύρους δεικτών
  - Οι Java, ML, C# διαθέτουν έλεγχο εύρους δεικτών

Πρόσδεση Δεικτών (Subscript Binding) και Κατηγορίες Πινάκων

- *Στατική (Static):* Τα εύρη των δεικτών (subscript ranges) είναι στατικά προσδεδεμένα και η κατανομή μνήμης γίνεται στατικά (πριν από το χρόνο εκτέλεσης).
  - Πλεονέκτημα: Αποδοτικότητα, καθώς αποφεύγεται η δυναμική κατανομή μνήμης
- *Σταθερή δυναμική-στοίβας (Fixed stack-dynamic):* Τα εύρη των δεικτών είναι στατικά προσδεδεμένα, αλλά η κατανομή μνήμης γίνεται κατά το χρόνο δήλωσης.
  - Πλεονέκτημα: Αποδοτικότητα χώρου

Πρόσδεση Δεικτών και Κατηγορίες Πινάκων (συνέχεια)

- *Σταθερή δυναμική-σωρού (Fixed heap-dynamic)*: Είναι παρόμοια με τη σταθερή δυναμική-στοίβας, καθώς η δέσμευση μνήμης είναι δυναμική, αλλά μένει σταθερή μετά την κατανομή, δηλαδή η πρόσδεση γίνεται όταν ζητηθεί και ο χώρος αποθήκευσης διατίθεται από το σωρό (heap) και όχι από τη στοίβα (stack)

Πρόσδεση Δεικτών και Κατηγορίες Πινάκων (συνέχεια)

- *Δυναμική–Σωρού (Heap–dynamic):* Η πρόσδεση του εύρους των δεικτών και η κατανομή μνήμης είναι δυναμικές και μπορούν να αλλάξουν πολλές φορές
  - Πλεονέκτημα: Ευελιξία (οι πίνακες μπορούν να μεγαλώσουν ή να συρρικνωθούν κατά τη διάρκεια της εκτέλεσης του προγράμματος)

Πρόσδεση Δεικτών και Κατηγορίες Πινάκων (συνέχεια)

- Στη C και στη C++ οι πίνακες που περιέχουν τον προσδιοριστή `static` είναι στατικές
- Στη C και στη C++ οι πίνακες χωρίς τον προσδιοριστή `static` είναι σταθερές δυναμικές-στοίβας
- Η C και η C++ διαθέτει διατάξεις σταθερές δυναμικές-σωρού
- Η C# περιέχει μια δεύτερη κλάση διατάξεων την `ArrayList` που είναι σταθερή δυναμική-σωρού
- Η Perl, η JavaScript, η Python, και η Ruby υποστηρίζουν διατάξεις δυναμικές-σωρού

Αρχικοποίηση Πινάκων

- Ορισμένες γλώσσες επιτρέπουν την αρχικοποίηση τη χρονική στιγμή που γίνεται η δέσμευση χώρου μνήμης
  - C, C++, Java, C#

```
int list[] = {4, 5, 7, 83}
```

  - Συμβολοσειρές χαρακτήρων στη C και στη C++

```
char name[] = "freddie";
```

  - Πίνακες συμβολοσειρών στην C και στην C++

```
char *names[] = {"Bob", "Jake", "Joe"];
```

  - Αρχικοποίηση αντικειμένων συμβολοσειρών στην Java

```
String[] names = {"Bob", "Jake", "Joe"};
```

Ετερογενείς Πίνακες

- Ένας ετερογενής πίνακας είναι ένας πίνακας που τα στοιχεία του δεν είναι απαραίτητο να είναι όλα του ίδιου τύπου
- Οι ετερογενείς πίνακες υπάρχουν στις Perl, Python, JavaScript, Ruby κ.α.

Αρχικοποίηση Πινάκων

- Σε C-based γλώσσες
  - `int list [] = {1, 3, 5, 7}`
  - `char *names [] = {"Mike", "Fred", "Mary Lou"};`
- Python
  - List comprehensions

```
list = [x ** 2 for x in range(12) if x % 3 == 0]
```

puts `[0, 9, 36, 81]` in `list`

Λειτουργίες Πινάκων

- Η γλώσσα APL παρέχει τις πιο ισχυρές λειτουργίες επεξεργασίας για διανύσματα και πίνακες καθώς και μονομελείς τελεστές (για παράδειγμα, για αντιστροφή των στοιχείων μιας στήλης)
- Η Python υποστηρίζει αναθέσεις πινάκων, αλλά πρόκειται μόνο για αλλαγές αναφορών. Επίσης, υποστηρίζει συνένωση πινάκων και λειτουργίες ελέγχου ύπαρξης τιμών σε πίνακες
- Η Ruby διαθέτει επίσης δυνατότητες συνένωσης πινάκων

Ορθογώνιοι και Οδοντωτοί Πίνακες

- Ένας ορθογώνιος πίνακας είναι ένας πίνακας πολλών διαστάσεων στον οποίο όλες οι γραμμές έχουν το ίδιο πλήθος στοιχείων και όλες οι στήλες επίσης έχουν το ίδιο πλήθος στοιχείων
- Ένας οδοντωτός πίνακας (jagged matrix) έχει γραμμές με μεταβλητό πλήθος στοιχείων
  - Είναι πιθανό να συμβεί όταν πίνακες πολλών διαστάσεων, είναι πίνακες με περιεχόμενα άλλους πίνακες
- Οι C, C++, και Java υποστηρίζουν οδοντωτούς πίνακες
- Η F# και η C# υποστηρίζουν ορθογώνιους και οδοντωτούς πίνακες

Κομμάτια (Slices)

- Ένα slice είναι ένα τμήμα ενός πίνακα. Στην ουσία πρόκειται για έναν μηχανισμό αναφοράς στα ίδια στοιχεία του αρχικού πίνακα
- Τα slices είναι χρήσιμα μόνο στις γλώσσες προγραμματισμού που διαθέτουν λειτουργίες που μπορούν να εφαρμοστούν σε πίνακες

Παραδείγματα Slicing Πινάκων

- Python

```
vector = [2, 4, 6, 8, 10, 12, 14, 16]
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

`vector[3:6]` είναι ένας πίνακας 3 στοιχείων

`mat[0][0:2]` είναι το πρώτο και το δεύτερο στοιχείο της πρώτης σειράς του `mat`

- Η Ruby υποστηρίζει slices με τη μέθοδο `slice`

`list.slice(2, 2)` επιστρέφει το τρίτο και το τέταρτο στοιχείο του `list`

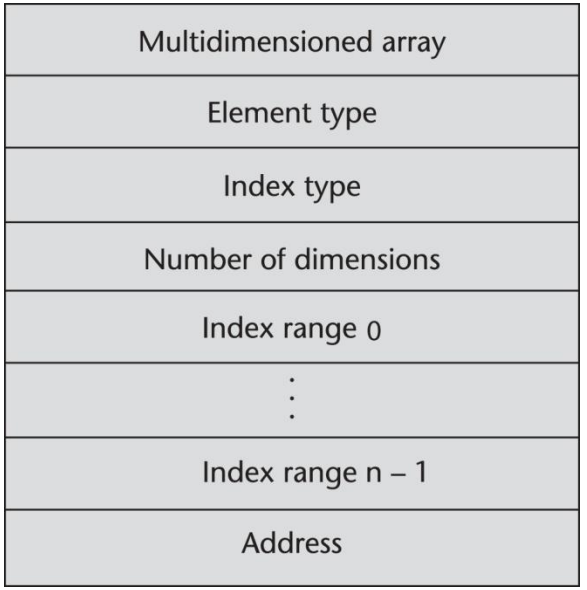
Υλοποίηση Πινάκων

- Η συνάρτηση πρόσβασης (access function) αντιστοιχεί τις εκφράσεις των δεικτών (subscript expressions) σε μια διεύθυνση στον πίνακα
- Η συνάρτηση πρόσβασης για πίνακες μιας διάστασης είναι:

διεύθυνση(list[k]) = διεύθυνση(list[0]) + k * μέγεθος_στοιχείου_πίνακα

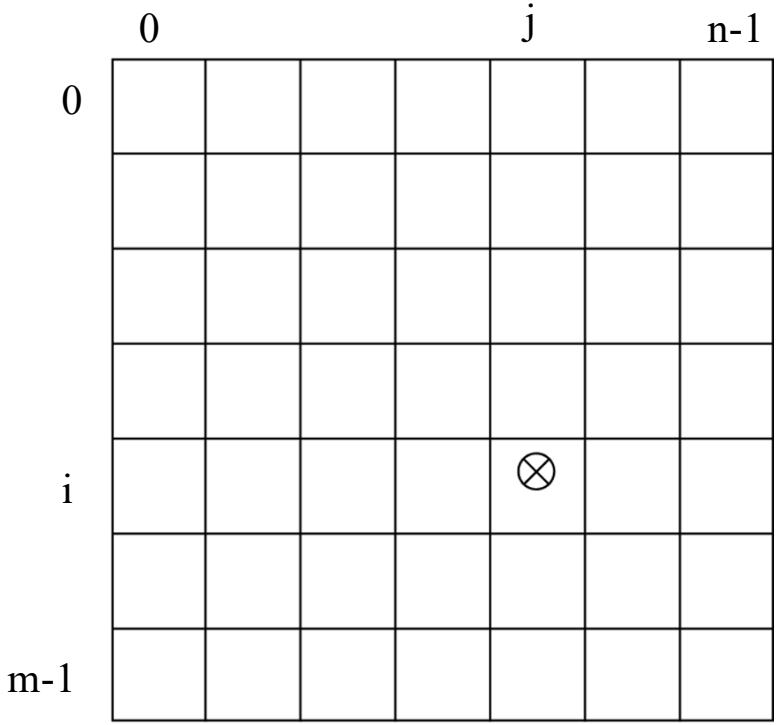
Προσπέλαση Πινάκων Πολλών Διαστάσεων

- Δύο συνηθισμένοι τρόποι:
  - **Row major διάταξη** (κατά σειρές) – χρησιμοποιείται στις περισσότερες γλώσσες προγραμματισμού
  - **Column major διάταξη** (κατά στήλες) – χρησιμοποιείται στην Fortran και σε άλλες γλώσσες
  - Ένας compile-time descriptor για μια διάταξη πολλών διαστάσεων

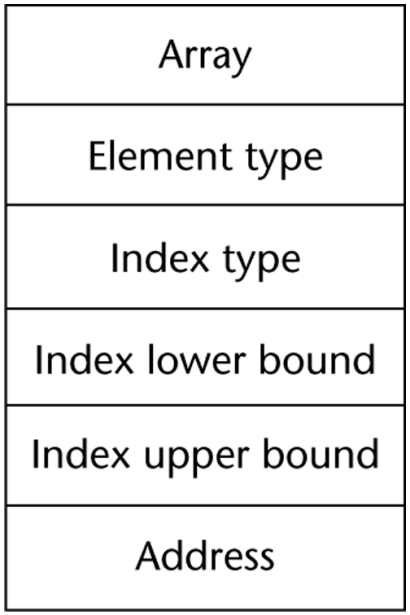


Εντοπισμός Ενός Στοιχείου σε Έναν Πίνακα Πολλών Διαστάσεων

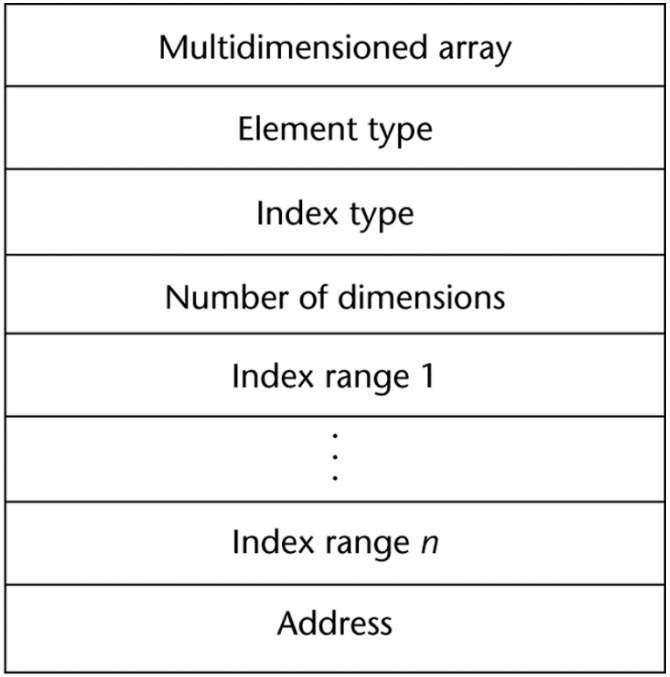
διεύθυνση(a[i,j]) =
διεύθυνση(a[0,0]) + (i * n + j) * μέγεθος_στοιχείου_πίνακα



Compile-Time Descriptors



Πίνακας μιας διάστασης



Πίνακας πολλών διαστάσεων

Συσχετιστικοί Πίνακες (associative arrays)

- Ένας *συσχετιστικός* πίνακας είναι μια μη διατεταγμένη συλλογή από στοιχεία δεδομένων που δεικτοδοτούνται από ένα ίσο σε πλήθος σύνολο τιμών που ονομάζονται κλειδιά
  - Τα ορισμένα από το χρήστη κλειδιά πρέπει να αποθηκεύονται
- Θέματα σχεδίασης:
  - Ποια είναι η μορφή με την οποία θα γίνεται αναφορά στα στοιχεία;
  - Είναι το μέγεθος στατικό ή δυναμικό;
- Στις Perl, Python, Ruby, Lua και σε άλλες γλώσσες ο συσχετιστικός πίνακας είναι built-in τύπος
  - Στην Lua, οι συσχετιστικοί πίνακες υποστηρίζονται από τα tables που διαθέτει η γλώσσα

Συσχετιστικοί Πίνακες στην Perl

- Το όνομα ενός συσχετιστικού πίνακα στην Perl ξεκινά με % και τα ζεύγη κλειδιού, τιμής που περιέχει περικλείονται σε παρενθέσεις όπως στο παράδειγμα:

```
%hi_temps = ("Mon" => 77, "Tue" => 79, "Wed" => 65, …);
```

- Η δεικτοδότηση γίνεται με αγκύλες και κλειδιά

```
$hi_temps{"Wed"} = 83;
```

  - Τα στοιχεία μπορούν να αφαιρεθούν με το **delete**

```
delete $hi_temps{"Tue"};
```

Τύποι Εγγραφών (Record Types)

- Μια εγγραφή (*record)* είναι μια πιθανά ετερογενής συνάθροιση στοιχείων δεδομένων όπου τα επιμέρους στοιχεία αναγνωρίζονται με ονόματα
- Θέματα σχεδίασης:
  - Ποιος τρόπος σύνταξης θα χρησιμοποιείται για αναφορά στα επιμέρους πεδία των εγγραφών;
  - Θα επιτρέπονται αναφορές σε εγγραφές με ελλείψεις;

Ορισμός Εγγραφών στην COBOL

- Η COBOL χρησιμοποιεί επίπεδα αριθμών για να υποδηλώσει εμφωλευμένες εγγραφές, ενώ άλλες γλώσσες χρησιμοποιούν αναδρομικούς ορισμούς

```
01 EMP-REC.
   02 EMP-NAME.
      05 FIRST PIC X(20).
      05 MID   PIC X(10).
      05 LAST  PIC X(20).
   02 HOURLY-RATE PIC 99V99.
```

Αναφορές σε Εγγραφές

- Αναφορές σε πεδία εγγραφών
  1. COBOL
  field_name `OF` record_name_1 `OF` ... `OF` record_name_n
  2. Άλλες γλώσσες (χρησιμοποιούν τον dot συμβολισμό)
  record_name_1.record_name_2. ... record_name_n.field_name

- Οι πλήρως προσδιορισμένες αναφορές πρέπει να περιλαμβάνουν όλα τα ονόματα των εγγραφών.

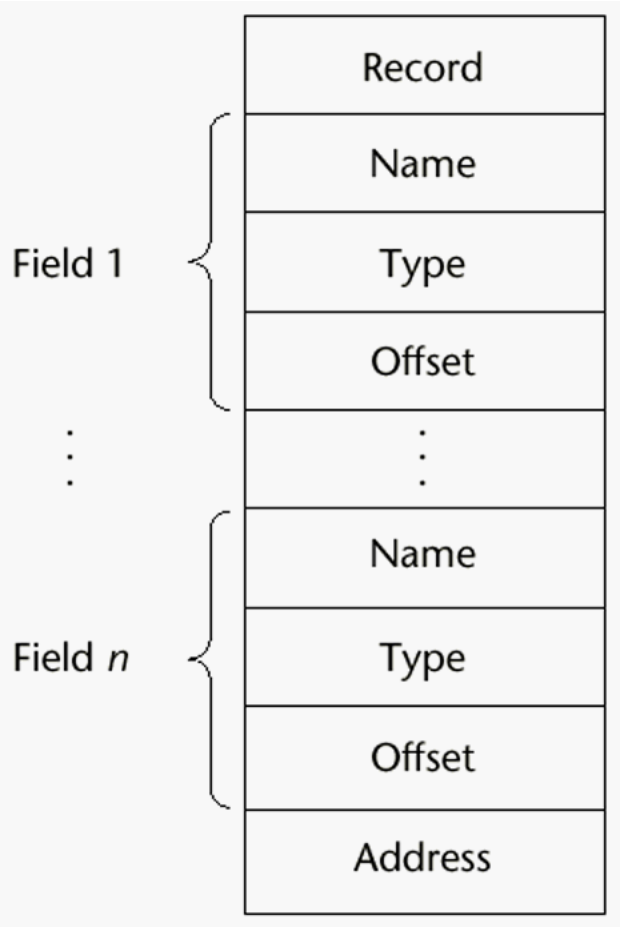
- Οι ελλιπείς αναφορές επιτρέπουν την παράληψη ονομάτων εγγραφών, εφόσον αυτό δεν καθιστά την αναφορά ασαφή, για παράδειγμα στην COBOL
  `FIRST`, `FIRST OF EMP-NAME`, και `FIRST OF EMP-REC` είναι ελλιπείς, αλλά έγκυρες αναφορές στο όνομα ενός υπαλλήλου

Αποτίμηση Χρησιμότητας Εγγραφών και Σύγκριση με Πίνακες

- Οι εγγραφές χρησιμοποιούνται όταν μια συλλογή δεδομένων είναι ετερογενής
- Η πρόσβαση στα στοιχεία των πινάκων είναι πολύ βραδύτερη από την πρόσβαση στα πεδία εγγραφών, λόγω του ότι οι δείκτες (subscripts) είναι δυναμικοί (ενώ τα ονόματα των πεδίων είναι στατικά)
- Δυναμικοί δείκτες (subscripts) θα μπορούσαν να χρησιμοποιηθούν για την πρόσβαση στα πεδία εγγραφών, αλλά αυτό θα ακύρωνε τον έλεγχο τύπων και θα ήταν πολύ βραδύτερο

Υλοποίηση του Τύπου Εγγραφής

Μια διεύθυνση μετατόπισης (offset address) σε σχέση με την αρχή των εγγραφών συσχετίζεται με κάθε πεδίο



Τύποι Πλειάδας

- Μια πλειάδα (tuple) είναι ένας τύπος δεδομένων που είναι παρόμοιος με τις εγγραφές, με τη διαφορά ότι τα στοιχεία δεν έχουν ονόματα
- Χρησιμοποιείται στην Python, στην ML, και στην F# για να επιτρέψει σε συναρτήσεις να επιστρέφουν πολλές τιμές
  - Python
    - Οι πλειάδες είναι παρόμοιες με τις λίστες, αλλά τα στοιχεία μιας πλειάδας δεν μπορούν να αλλάξουν, είναι δηλαδή immutable
    - Δημιουργείται με το κυριολεκτικό tuple ή με απλή παράθεση των τιμών χωρισμένων με κόμματα (οι παρενθέσεις είναι προαιρετικές):

```
myTuple = tuple((3, 5.8, 'apple'))
myTuple2 = (3, 5.8, 'apple')
myTuple3 = 3, 5.8, 'apple'
```

      Η αναφορά γίνεται με δείκτες (που ξεκινούν από το 0)
      Συνένωση πλειάδων γίνεται με το + και διαγραφή τους με το del

Τύποι Πλειάδας (συνέχεια)

- ML

```
val myTuple = (3, 5.8, 'apple');
```

  - Η πρόσβαση γίνεται ως εξής:

    `#1(myTuple)` είναι το πρώτο στοιχείο

  - Ένας νέος τύπος πλειάδας μπορεί να οριστεί ως εξής:

    `type intReal = int * real;` Ο νέος τύπος πλειάδας αποτελείται από έναν ακέραιο και έναν πραγματικό

- F#

```
let tup = (3, 5, 7)
let a, b, c = tup
```

αναθέτει την πλειάδα `tup` στο πρότυπο πλειάδας `(a, b, c)`

Τύποι Λίστας

- Οι λίστες στην Lisp και στην Scheme αποτελούνται από στοιχεία που περικλείονται σε παρενθέσεις και δεν χωρίζονται μεταξύ τους με κόμματα

  `(A B C D)` και `(A (B C) D)`

- Τα δεδομένα και ο κώδικας έχουν την ίδια μορφή

  Ως δεδομένα, το `(A B C)` είναι τρεις τιμές

  Ως κώδικας, το `(A B C)` είναι η συνάρτηση `A` που εφαρμόζεται στις παραμέτρους `B` και `C`

- Ο διερμηνευτής πρέπει να γνωρίζει τι είδους λίστα είναι οπότε αν πρόκειται για δεδομένα, θα πρέπει αυτό να σηματοδοτηθεί με μια απόστροφο ως εξής:

  `'(A B C)` είναι δεδομένα

Τύποι Λίστας (συνέχεια)

- Λειτουργίες λιστών στην Scheme
  - `CAR` επιστρέφει το πρώτο στοιχείο της λίστας που δέχεται ως παράμετρο

    `(CAR '(A B C))` επιστρέφει `A`
  - `CDR` επιστρέφει το υπόλοιπο της λίστας που δέχεται ως παράμετρο εφόσον το πρώτο στοιχείο έχει αφαιρεθεί

    `(CDR '(A B C))` επιστρέφει `(B C)`
  - `CONS` τοποθετεί την πρώτη παράμετρο στη δεύτερη παράμετρο που είναι λίστα, έτσι ώστε να δημιουργηθεί μια νέα λίστα

    `(CONS 'A (B C))` επιστρέφει `(A B C)`
  - `LIST` επιστρέφει μια νέα λίστα με στοιχεία τις παραμέτρους της

    `(LIST 'A 'B '(C D))` επιστρέφει `(A B (C D))`

Τύποι Λίστας (συνέχεια)

- Λειτουργίες λιστών στην ML
  - Οι λίστες γράφονται σε αγκύλες και τα στοιχεία χωρίζονται με κόμματα
  - Τα στοιχεία της λίστας πρέπει να είναι του ίδιου τύπου
  - Η συνάρτηση `CONS` της Scheme είναι στην ML ένας δυαδικός τελεστής, ο τελεστής `::`

    `3 :: [5, 7, 9]` αποτιμάται σε `[3, 5, 7, 9]`
  - Οι συναρτήσεις της Scheme `CAR` και `CDR` ονομάζονται στην ML, `hd` και `tl`, αντίστοιχα

Τύποι Λίστας (συνέχεια)

- Λίστες στην F#
  - Παρόμοια με την ML, με τη διαφορά ότι τα στοιχεία διαχωρίζονται μεταξύ τους με ελληνικά ερωτηματικά (;) και οι συναρτήσεις `hd` και `tl` είναι μέθοδοι της κλάσης `List`
- Λίστες στην Python
  - Ο τύπος δεδομένων λίστας χρησιμοποιείται και για τους πίνακες της Python
  - Σε αντίθεση με τις Scheme, Common Lisp, ML, και F#, οι λίστες στην Python μπορούν να τροποποιούνται, είναι δηλαδή mutable
  - Τα στοιχεία μιας λίστας μπορεί να είναι οποιουδήποτε τύπου
  - Μια λίστα μπορεί να δημιουργηθεί με μια ανάθεση όπως η ακόλουθη:

```
myList = [3, 5.8, "grape"]
```

Τύποι Λίστας (συνέχεια)

- Λίστες στην Python (συνέχεια)
  - Τα στοιχεία των λιστών αναφέρονται με δείκτες που ξεκινούν από το μηδέν

    `x = myList[1]` Θέτει το `x` σε `5.8`
  - Τα στοιχεία των λιστών μπορούν να διαγραφούν με το `del`

    ```
    del myList[1]
    ```
  - Περιφραστικές λίστες (List Comprehensions) – έχουν προκύψει από τον συμβολισμό των μαθηματικών για τα σύνολα

    ```
    [x * x for x in range(7) if x % 3 == 0]
    ```

    Η **`range`**`(7)` δημιουργεί το `[0, 1, 2, 3, 4, 5, 6]`

    Η λίστα που δημιουργείται από την περιφραστική είναι: `[0, 9, 36]`

Τύποι Λίστας (συνέχεια)

- Περιφραστικές λίστες στην Haskell

```
[n * n | n <- [1..10]]
```

- Περιφραστικές λίστες στην F#

```
let myArray = [|for i in 1 .. 5 -> [i * i) |]
```

- Τόσο η C# όσο και η Java υποστηρίζουν λίστες μέσω των κλάσεων, `List` και `ArrayList`, αντίστοιχα

Τύποι Ενώσεων

- Μια ένωση *union* είναι ένας τύπος δεδομένων που οι μεταβλητές του επιτρέπεται να αποθηκεύουν τιμές διαφορετικών τύπων σε διαφορετικές χρονικές στιγμές κατά την εκτέλεση
- Θέμα σχεδίασης
  - Θα πρέπει ο έλεγχος τύπων να είναι υποχρεωτικός;

Ενώσεις με Διάκριση Τύπου έναντι Ελεύθερων Ενώσεων

- Οι ενώσεις στη C και C++ δεν πραγματοποιούν έλεγχο τύπων και ονομάζονται ελεύθερες ενώσεις (*free unions)*
- Για να μπορεί να υπάρχει έλεγχος τύπων στις ενώσεις θα πρέπει κάθε ένωση να περιέχει έναν προσδιοριστή τύπου που ονομάζεται *discriminant*
  - Αυτό υποστηρίζεται από γλώσσες όπως οι ML, Haskell, και F#

Ενώσεις στην F#

- Οι ενώσεις στην F# ορίζονται με μια εντολή type που χρησιμοποιεί το σύμβολο | που σημαίνει ή

```
type intReal =
    | IntValue of int
    | RealValue of float;;
```

`intReal` είναι ο νέος τύπος

`IntValue` και `RealValue` είναι κατασκευαστές

Για να δημιουργηθεί μια τιμή τύπου `intReal`:

```
let ir1 = IntValue 17;;
let ir2 = RealValue 3.4;;
```

Ενώσεις στην F# (συνέχεια)

- Η προσπέλαση στην τιμή μιας ένωσης γίνεται με pattern matching

```
match pattern with
    | expression_list1 -> expression1
    | ...
    | expression_listn -> expressionn
```

  - Ένα pattern μπορεί να είναι οποιοσδήποτε τύπος δεδομένων
  - Οι λίστες εκφράσεων μπορεί να έχουν wild cards (_)

Ενώσεις στην F# (συνέχεια)

Παράδειγμα:

```
let a = 7;;
let b = "grape";;
let x = match (a, b) with
      | 4, "apple" -> apple
      | _, "grape" -> grape
      | _ -> fruit;;
```

Ενώσεις στην F# (συνέχεια)

Για να εμφανιστεί ο τύπος της ένωσης `intReal` μπορεί να οριστεί η συνάρτηση:

```
let printType value =
   match value with
        | IntVale value -> printfn "int"
        | RealValue value -> printfn "float";;
```

Αν το `ir1` και `ir2` έχουν οριστεί ως:

```
let ir1 = IntValue 17;;
let ir2 = RealValue 3.4;;
```

τότε,

`printType ir1` επιστρέφει `int`

`printType ir2` επιστρέφει `float`

Αποτίμηση Χρησιμότητας Ενώσεων

- Οι free unions δεν είναι ασφαλείς
  - Δεν επιτρέπουν έλεγχο τύπων
- Η Java και η C# δεν υποστηρίζουν unions
  - Είναι χαρακτηριστική περίπτωση απόφασης που λήφθηκε λόγω των αυξανόμενων ανησυχιών για την ασφάλεια στις γλώσσες προγραμματισμού

Τύποι Δεικτών και Τύποι Αναφορών

- Μια μεταβλητή με τύπο δείκτη (pointer type variable) έχει ένα εύρος τιμών που αποτελείται από διευθύνσεις μνήμης και μια ιδιαίτερη τιμή, το *nil*
- Οι δείκτες προσφέρουν στον προγραμματιστή τη δύναμη της έμμεσης διευθυνσιοδότησης
- Οι δείκτες παρέχουν έναν τρόπο δυναμικής διαχείρισης μνήμης
- Ένας δείκτης μπορεί να χρησιμοποιηθεί για να προσπελαστεί η θέση σε μια περιοχή μνήμης όπου ο χώρος αποθήκευσης έχει δημιουργηθεί δυναμικά (συνήθως ο χώρος αυτός ονομάζεται σωρός–*heap*)

Θέματα Σχεδίασης για τους Δείκτες

- Ποια είναι η εμβέλεια και η διάρκεια ζωής μιας μεταβλητής δείκτη;
- Ποια είναι η διάρκεια ζωής μιας heap-dynamic μεταβλητής;
- Είναι οι δείκτες περιορισμένοι ως προς τον τύπο της τιμής στον οποίο μπορούν να δείχνουν;
- Μπορούν οι δείκτες να χρησιμοποιηθούν μόνο για δυναμική διαχείριση του χώρου αποθήκευσης ή μόνο για έμμεση διευθυνσιοδότηση ή και για τα δύο;
- Θα πρέπει μια γλώσσα να υποστηρίζει μόνο τύπους δεικτών ή μόνο τύπους αναφορών ή και τα δύο;

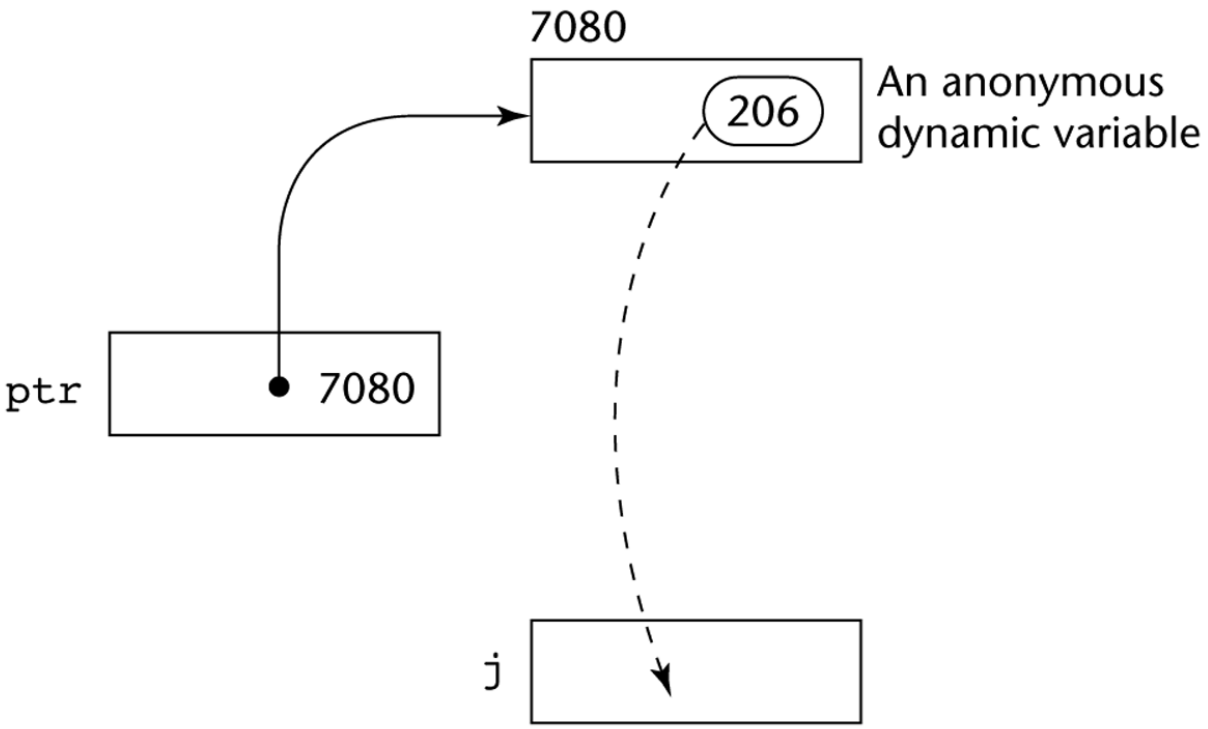
Λειτουργίες Δεικτών

- Δύο θεμελιώδεις λειτουργίες: ανάθεση (assignment) και αποαναφορά (dereference)
- Η ανάθεση χρησιμοποιείται για να θέσει την τιμή μιας μεταβλητής δείκτη σε μια διεύθυνση μνήμης που θα είναι χρήσιμη για τη συνέχεια του προγράμματος
- Η αποαναφορά επιστρέφει την τιμή που είναι αποθηκευμένη στη θέση μνήμης που αναπαρίσταται από την τιμή του δείκτη
  - Η αποαναφορά μπορεί να είναι ρητή ή υπονοούμενη
  - Η C++ πραγματοποιεί ρητή αποαναφορά με το *

    `j = *ptr`

    θέτει το j στην τιμή που είναι αποθηκευμένη εκεί που δείχνει ο δείκτης `ptr`

Παράδειγμα Αποαναφοράς Δείκτη



Η λειτουργία ανάθεσης j = *ptr

Προβλήματα που προκύπτουν κατά τη χρήση Δεικτών

- Αιωρούμενοι (dangling) pointers (επικίνδυνο)
  - Ένας δείκτης που δείχνει σε μια heap-dynamic μεταβλητή που έχει αποδεσμευτεί
- Χαμένη heap-dynamic μεταβλητή
  - Μια δεσμευμένη heap-dynamic μεταβλητή που δεν είναι πλέον προσπελάσιμη από το πρόγραμμα (συχνά ονομάζεται σκουπίδι-*garbage*)
    - Ο δείκτης `p1` τίθεται να δείχνει σε μια νέα heap-dynamic μεταβλητή
    - Ο δείκτης `p1` αργότερα τίθεται να δείχνει σε μια άλλη νέα heap-dynamic μεταβλητή
    - Η διαδικασία της απώλειας heap-dynamic μεταβλητών ονομάζεται διαρροή μνήμης *(memory leakage)*

Δείκτες στη C και στην C++

- Εξαιρετικά ευέλικτοι, αλλά πρέπει να χρησιμοποιούνται με προσοχή
- Οι δείκτες μπορούν να δείχνουν σε οποιαδήποτε μεταβλητή, ανεξάρτητα από το πότε ή που έχει δεσμευτεί χώρος μνήμης για αυτή
- Χρησιμοποιούνται για δυναμική διαχείριση μνήμης και διευθυνσιοδότηση
- Μπορεί να χρησιμοποιηθεί αριθμητική δεικτών
- Χρησιμοποιούνται ρητές αποναφορές δεικτών και ο τελεστής διεύθυνσης &
- Ο τύπος του δείκτη δεν απαιτείται να είναι σταθερός (`void *`)

  ένας δείκτης `void *` μπορεί να δείξει σε οποιοδήποτε τύπο, υπόκειται σε έλεγχο τύπου (ωστόσο, δεν μπορεί να γίνει αποαναφορά void * δεικτών)

Αριθμητική Δεικτών στη C και στη C++

```
float stuff[100];
float *p;
p = stuff;
```

`*(p+5)` είναι ισοδύναμο με `stuff[5]` και `p[5]`

`*(p+i)` είναι ισοδύναμο με `stuff[i]` και `p[i]`

Τύποι Αναφορών

- Η C++ περιέχει έναν ειδικό τύπο δεικτών που ονομάζεται τύπος αναφοράς (*reference type)* που χρησιμοποιείται κυρίως για τυπικές παραμέτρους (formal parameters)
  - Έχει τα πλεονεκτήματα τόσο του pass-by-reference όσο και του pass-by-value
- Η Java χρησιμοποιεί αναφορές και αντικαθιστά με αυτές τους δείκτες της C++
  - Οι αναφορές είναι αναφορές σε αντικείμενα, αντί να είναι διευθύνσεις
- Η C# περιέχει τόσο τις αναφορές της Java όσο και τους δείκτες της C++

Αποτίμηση Χρησιμότητας Δεικτών

- Οι dangling δείκτες και τα dangling αντικείμενα αποτελούν προβλήματα όπως και η διαχείριση του σωρού
- Οι δείκτες είναι σαν `goto`'s—διευρύνουν τα κελιά μνήμης που μπορούν να προσπελαστούν από μια μεταβλητή
- Οι δείκτες ή οι αναφορές είναι απαραίτητοι/ες σε δυναμικές δομές δεδομένων – συνεπώς, δεν μπορεί να σχεδιαστεί μια γλώσσα χωρίς να συμπεριλάβει τουλάχιστον ένα από αυτά

Αναπαραστάσεις Δεικτών

- Οι μεγάλοι υπολογιστές χρησιμοποιούν απλές τιμές
- Οι μικροεπεξεργαστές της Intel χρησιμοποιούν segment και offset

Το Πρόβλημα του Αιωρούμενου (dangling) Δείκτη

- *Tombstone*: Πρόκειται για ένα επιπλέον κελί μνήμης στον σωρό που είναι δείκτης σε μια heap-dynamic μεταβλητή
  - Οι πραγματικές μεταβλητές δείκτη δείχνουν μόνο σε tombstones
  - Όταν μια heap-dynamic μεταβλητή γίνεται de-allocate, το tombstone εξακολουθεί να υπάρχει αλλά τίθεται στην τιμή nil
  - Έχουν υψηλό κόστος χρόνου και χώρου

. *Locks-and-keys*: Οι τιμές των δεικτών αναπαρίστανται ως ζεύγη (κλειδί, διεύθυνση)
  - Οι heap-dynamic μεταβλητές αναπαρίστανται ως μεταβλητές συν ένα κελί μνήμης που διατηρεί μια ακέραια τιμή κλειδώματος (lock value)
  - Όταν γίνεται δέσμευση μνήμης μιας heap-dynamic μεταβλητής, δημιουργείται η lock τιμή και τοποθετείται σε ένα lock cell και key cell του δείκτη

Διαχείριση Σωρού (Heap Management)

- Πρόκειται για μια σύνθετη διεργασία που γίνεται κατά το χρόνο εκτέλεσης
- Δύο επιλογές: κελιά ενός μεγέθους έναντι κελιών μεταβλητού μεγέθους
- Δύο κύριες προσεγγίσεις για την ανάκτηση χώρου που καταλαμβάνεται από σκουπίδια (garbage)
  - Μετρητές αναφορών reference counters (*ανυπόμονη προσέγγιση – eager approach*): η ανάκτηση χώρου μνήμης είναι σταδιακή
  - Mark-sweep (οκνηρή προσέγγιση – *lazy approach*): η ανάκτηση χώρου συμβαίνει όταν δεν υπάρχει πλέον χώρος

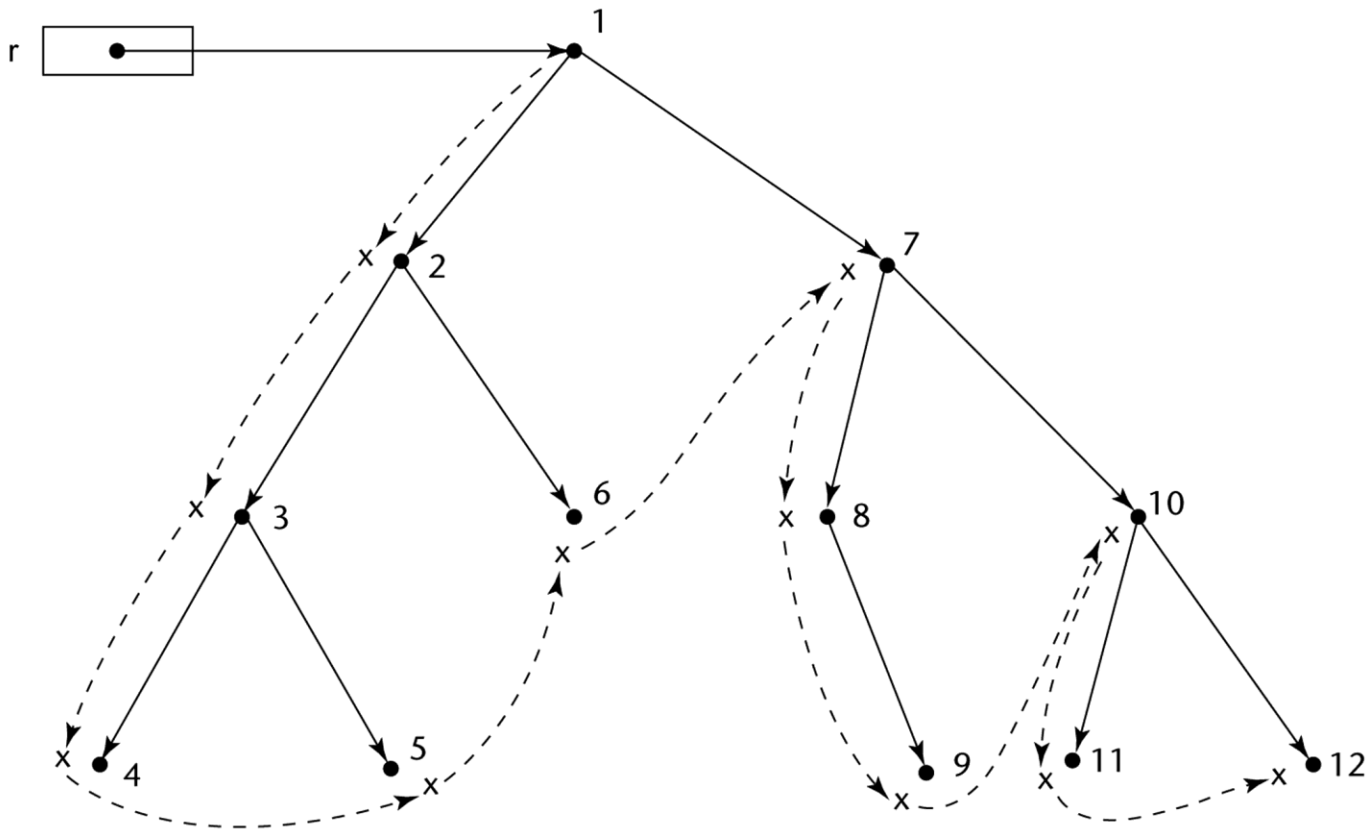
Καταμέτρηση Αναφορών (Reference Counter)

- Μετρητές αναφορών (reference counters): διατηρείται ένας μετρητής σε κάθε κελί που αποθηκεύει το πλήθος των δεικτών που την τρέχουσα στιγμή δείχνουν στο κελί
  - *Μειονεκτήματα*: απαιτείται επιπλέον χώρος, απαιτείται επιπλέον χρόνος εκτέλεσης, προκύπτουν προβλήματα σε κελιά που συνδέονται κυκλικά
  - *Πλεονέκτημα*: είναι εγγενώς αυξητικό, συνεπώς αποφεύγονται σημαντικές καθυστερήσεις στην εκτέλεση των εφαρμογών

Mark-Sweep (σήμανση και εκκαθάριση)

- Το σύστημα χρόνου εκτέλεσης δεσμεύει κελιά μνήμης καθώς ζητούνται και αποσυνδέει τους δείκτες από τα κελιά όταν αυτό ζητείται, στη συνέχεια ξεκινά ο αλγόριθμος mark-sweep:
  - Κάθε κελί του σωρού έχει ένα επιπλέον bit που χρησιμοποιείται από τον αλγόριθμο συλλογής
  - Όλα τα κελιά αρχικά θεωρούνται garbage
  - Όλοι οι δείκτες ιχνηλατούνται στο σωρό, και τα κελιά που είναι προσβάσιμα από την ιχνηλάτηση σηματοδοτούνται ως μη-garbage
  - Όλα τα κελιά garbage επιστρέφονται στη λίστα ων διαθέσιμων κελιών
  - Μειονεκτήματα: στην αρχική του μορφή γινόταν πολύ σπάνια. Όταν συνέβαινε, προκαλούσε σημαντικές καθυστερήσεις στην εκτέλεση της εφαρμογής. Οι σύγχρονοι mark-sweep αλγόριθμοι αποφεύγουν αυτή τη συμπεριφορά με συχνότερες ενεργοποιήσεις τους — ονομάζονται incremental mark-sweep αλγόριθμοι

Αλγόριθμος Marking



Dashed lines show the order of node_marking

Κελιά Μεταβλητού-Μεγέθους

- Παρουσιάζουν όλες τις δυσκολίες των κελιών ενός μεγέθους και μερικά ακόμα
- Απαιτείται από τις περισσότερες γλώσσες προγραμματισμού
- Αν χρησιμοποιείται mark-sweep, εμφανίζονται επιπλέον προβλήματα:
  - Η αρχική ρύθμιση των indicators για όλα τα κελιά στο σωρό είναι δύσκολη
  - Η διαδικασία marking δεν είναι απλοϊκή
  - Η διατήρηση της λίστας του διαθέσιμου χώρου προκαλεί μια ακόμα επιβάρυνση

Έλεγχος Τύπων (Type Checking)

- Πρόκειται για γενίκευση της έννοιας των τελεστών και των τελεστέων έτσι ώστε να περιλαμβάνει υποπρογράμματα υποπρογράμματα και αναθέσεις
- Ο *έλεγχος τύπων* είναι η ενέργεια διασφάλισης ότι οι τελεστές ενός τελεστέου διαθέτουν συμβατούς τύπους
- Ένας συμβατός τύπος είναι ένας τύπος που είτε είναι έγκυρος για τον τελεστή, είτε επιτρέπεται με βάση τους κανόνες της γλώσσας, υπονοούμενα να μετατραπεί από τον μεταγλωττιστή, σε έγκυρο τύπο
  - Η αυτόματη μετατροπή ονομάζεται *coercion (εξαναγκασμός)*.
- Ένα *σφάλμα τύπου* παρουσιάζεται όταν εφαρμόζεται ένας τελεστής σε έναν τελεστέο μη αποδεκτού τύπου

Έλεγχος Τύπων (συνέχεια)

- Αν όλες οι προσδέσεις τύπων είναι στατικές, σχεδόν όλοι οι έλεγχοι τύπων μπορεί να είναι και αυτές στατικές
- Αν η πρόσδεση τύπων είναι δυναμική, ο έλεγχος τύπων πρέπει να είναι δυναμικός
- Μια γλώσσα προγραμματισμού είναι *strongly typed* αν τα σφάλματα τύπων ανιχνεύονται πάντα
- Πλεονέκτημα του strong typing: επιτρέπει την ανίχνευση λανθασμένων χρήσεων των μεταβλητών που οδηγούν σε σφάλματα τύπων

Strong Typing (Ισχυροί Τύποι)

Παραδείγματα γλωσσών:

- Η C και η C++ μπορούν να θεωρηθούν ότι δεν είναι strong typing γλώσσες: ο έλεγχος τύπου παραμέτρων μπορεί να παραληφθεί, για παράδειγμα οι ενώσεις (unions) δεν ελέγχονται σε σχέση με τον τύπο τους
- Η Java και η C# είναι, σχεδόν strong typing γλώσσες (αλλά μπορούν οι τύποι να μετατραπούν μέσω ρητών type casting)
- Η ML και η F# θεωρούνται strong typing γλώσσες

Strong Typing (συνέχεια)

- Οι κανόνες εξαναγκασμού τύπων (coercion) επηρεάζουν ισχυρά το strong typing – μπορούν να το αδυνατίσουν σημαντικά (C++ έναντι της ML και F#)

- Αν και η Java έχει περίπου τους μισούς εξαναγκασμούς τύπων σε σχέση με τη C++, το strong typing της είναι πολύ λιγότερο αποτελεσματικό από της Ada

Ισοδυναμία Ονομάτων Τύπων

- *Ισοδυναμία ονόματος τύπου (name type equivalence)* σημαίνει ότι δύο μεταβλητές έχουν ισοδύναμους τύπους αν είτε και οι δύο μεταβλητές βρίσκονται στην ίδια δήλωση είτε αν οι δηλώσεις τους χρησιμοποιούν το ίδιο όνομα τύπου
- Εύκολο να υλοποιηθεί αλλά πολύ περιοριστικό:
  - Οι υποπεριοχές ακεραίων τύπων δεν είναι ισοδύναμοι με τους ακέραιους τύπους
  - Οι τυπικές παράμετροι θα πρέπει να έχουν τους ίδιους τύπους με τους τύπους των αντίστοιχων πραγματικών παραμέτρων

Ισοδυναμία Δομής Τύπου

- *Ισοδυναμία δομής τύπου (structure type equivalence)* σημαίνει ότι δύο μεταβλητές έχουν ισοδύναμους τύπους αν οι τύποι τους έχουν πανομοιότυπες δομές
- Περισσότερο ευέλικτο, αλλά δυσκολότερο να υλοποιηθεί

Ισοδυναμία Τύπων (συνέχεια)

- Έστω δύο δομημένοι τύποι:
  - Είναι δύο τύποι εγγραφής ισοδύναμοι, αν είναι δομικά ίδιοι, αλλά χρησιμοποιούν διαφορετικά ονόματα πεδίων;
  - Είναι δύο τύποι πινάκων ισοδύναμοι, αν είναι ίδιοι με εξαίρεση τους δείκτες τους που είναι διαφορετικοί; (π.χ. `[1..10]` και `[0..9]`)
  - Είναι δύο τύποι απαρίθμησης ισοδύναμοι αν τα συστατικά τους απλά γράφονται διαφορετικά;
  - Με την ισοδυναμία δομής τύπου, δεν μπορούν να διαφοροποιηθούν τύποι με την ίδια δομή (π.χ. διαφορετικές μονάδες ταχύτητας, που είναι και οι δύο float)

Θεωρία Τύπων και Τύποι Δεδομένων

- Η θεωρία τύπω είναι ένα ευρύ πεδίο μελέτης των μαθηματικών της λογικής, της επιστήμης υπολογιστών και της φιλοσοφίας
- Υπάρχουν δύο κλάδοι της θεωρίας τύπων στην επιστήμη υπολογιστών:
  - Πρακτικός κλάδος – τύποι δεδομένων σε εμπορικές γλώσσες
  - Αφηρημένος κλάδος – λογισμός λάμδα με τύπους (typed lambda calculus)
- Ένα σύστημα τύπων είναι ένα σύνολο από τύπους μαζί με τους κανόνες που καθορίζουν τη χρήση τους σε προγράμματα

Θεωρία Τύπων και Τύποι Δεδομένων (συνέχεια)

- Το τυπικό μοντέλο (formal model) ενός συστήματος τύπων είναι ένα σύνολο τύπων και μια συλλογή από συναρτήσεις που ορίζουν τους κανόνες των τύπων
  - Για τις συναρτήσεις μπορεί να χρησιμοποιηθεί είτε μια γραμματική χαρακτηριστικών (attribute grammar) ή μια αντιστοίχιση τύπων (type map)
  - Πεπερασμένες αντιστοιχίσεις (finite mappings) – μοντελοποιούν πίνακες και συναρτήσεις
  - Καρτεσιανα γινόμενα – μοντελοποιούν πλειάδες και εγγραφές
  - Ενώσεις συνόλων – μοντελοποιούν τύπους ενώσεων
  - Υποσύνολα – μοντελοποιούν υποτύπους

Σύνοψη

- Οι τύποι δεδομένων μιας γλώσσας καθορίζουν σε μεγάλο βαθμό το στυλ της γλώσσας και τη χρησιμότητά της
- Οι πρωτογενείς τύποι δεδομένων των περισσότερων προστακτικών γλωσσών περιέχουν αριθμητικούς τύπους, τον τύπο χαρακτήρα και τύπους Boolean
- Οι απαριθμήσεις που ορίζονται από τον χρήστη και οι subrange types είναι βολικοί μηχανισμοί και βοηθούν την αναγνωσιμότητα και την αξιοπιστία των προγραμμάτων
- Οι πίνακες και οι εγγραφές περιέχονται στις περισσότερες γλώσσες
- Οι δείκτες χρησιμοποιούνται για αυξημένης ευελιξία κατά τον προγραμματισμό και για δυναμική διαχείριση μνήμης