ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

http://www.softlab.ntua.gr/~nickie/Courses/progtech/

Διδάσκοντες: Γιάννης Μαΐστρος (maistros@cs.ntua.gr)
Στάθης Ζάχος (zachos@cs.ntua.gr)
Νίκος Παπασπύρου (nickie@softlab.ntua.gr)

Διαφάνειες παρουσίασης #3

- ✓Πίνακες
- ✓Δομές δεδομένων, αλγόριθμοι, πολυπλοκότητα
- ✓Συγκεκριμένοι και αφηρημένοι τύποι δεδομένων

Πίνακες (arrays) (i)

- Ακολουθία μεταβλητών
  - με το ίδιο όνομα,
  - με τον ίδιο τύπο δεδομένων,
  - σε συνεχόμενες θέσεις μνήμης.
- Παράδειγμα

```
int a[50];
double d1[5], d2[10];
```

- Η αρίθμηση αρχίζει από το 0 !

```
for (i = 0; i < 50; i++)
    a[i] = i;
```

Πίνακες (arrays) (ii)

- Αρχικοποίηση πινάκων

```
int a[3] = {6, 7, 42};
char c[] = {'a', 'b', 'c', 'd', 'e'};
```

- Συμβολοσειρές
  - Είναι πίνακες χαρακτήρων

```
char s[6] = "abcde";
```

  - Τερματίζονται με τον κενό χαρακτήρα `'\0'`

```
char s[6] = {'a', 'b', 'c',
             'd', 'e', '\0'};
```

Πίνακες (arrays) (iii)

- Παράδειγμα

```
int a[3] = {5, 6, 7};
int b[3] = {2, 3, 1};
int i;

for (i = 0; i < 3; i++)
   printf("%d times %d is %d\n",
      a[i], b[i], a[i]*b[i]);
```

```
5 times 2 is 10
6 times 3 is 18
7 times 1 is 7
```

Πολυδιάστατοι πίνακες (i)

- Παράδειγμα

```
int a[3][5];
char c[5][10][4];
```

- Αρχικοποίηση

```
int i[3][3] = { {1, 0, 0},
                {0, 1, 0},
                {0, 0, 1} };
char s[][10] = {
    "my", "name", "is", "joe"
};
```

Πολυδιάστατοι πίνακες (ii)

- Παράδειγμα: πολλαπλασιασμός πινάκων

```
double a[3][4] = ... ,
       b[4][5] = ... ,
       c[3][5];
int i, j, k;

for (i=0; i<3; i++)
   for (j=0; j<5; j++) {
      c[i][j] = 0.0;
      for (k=0; k<4; k++)
         c[i][j] += a[i][k] * b[k][j];
   }
```

Εισαγωγή στις δομές δεδομένων (i)

- Αλγόριθμος

  Πεπερασμένο σύνολο εντολών που,
  όταν εκτελεστούν,
  επιτυγχάνουν κάποιο επιθυμητό αποτέλεσμα

  - Δεδομένα εισόδου και εξόδου
  - Κάθε εντολή πρέπει να είναι:
    - καλά ορισμένη
    - απλή
  - Η εκτέλεση πρέπει να σταματά

Εισαγωγή στις δομές δεδομένων (ii)

- Πολυπλοκότητα

  Μέτρο εκτίμησης της απόδοσης αλγορίθμων ως συνάρτηση του μεγέθους του προβλήματος που επιλύουν

  - Χρόνος εκτέλεσης
  - Χωρητικότητα μνήμης που απαιτείται
- Ακριβής μέτρηση πολυπλοκότητας

  $t = f(n)$
- Τάξη μεγέθους, συμβολισμοί Ο, Ω, Θ

  $t = O(f(n))$

Εισαγωγή στις δομές δεδομένων (iii)

- Ορισμός των κλάσεων O, Ω, Θ
  - Άνω όριο

    $g = O(f) \Leftrightarrow \exists c.\ \exists n_0.\ \forall n > n_0.\ g(n) < c\ f(n)$
  - Κάτω όριο

    $g = \Omega(f) \Leftrightarrow \exists c.\ \exists n_0.\ \forall n > n_0.\ g(n) > c\ f(n)$
  - Τάξη μεγέθους

    $g = \Theta(f) \Leftrightarrow \exists c_1, c_2.\ \exists n_0.\ \forall n > n_0.$
    $c_1\ f(n) < g(n) < c_2\ f(n)$
- Διάταξη μερικών κλάσεων πολυπλοκότητας

  $O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n)$
  $< O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$

Εισαγωγή στις δομές δεδομένων (iv)

- Αφαίρεση δεδομένων (data abstraction)
  - Διαχωρισμός ιδιοτήτων και υλοποίησης
  - Ιδιότητες ενός τύπου δεδομένων:
    ο τρόπος με τον οποίο δημιουργεί κανείς και χρησιμοποιεί δεδομένα αυτού του τύπου
  - Υλοποίηση ενός τύπου δεδομένων:
    ο τρόπος με τον οποίο αναπαρίστανται τα δεδομένα στη μνήμη του υπολογιστή και προγραμματίζονται οι διαθέσιμες πράξεις

Εισαγωγή στις δομές δεδομένων (v)

- Οι λεπτομέρειες υλοποίησης επηρεάζουν την πολυπλοκότητα των αλγορίθμων
- Είναι συχνή επομένως η αλλαγή τρόπου υλοποίησης κάποιου τύπου δεδομένων
- Η αφαίρεση δεδομένων ελαχιστοποιεί τις αλλαγές που απαιτούνται στο πρόγραμμα που χρησιμοποιεί έναν τύπο δεδομένων, όταν αλλάξει ο τρόπος υλοποίησης

Αφηρημένοι τύποι δεδομένων

- Αφηρημένος τύπος δεδομένων – ΑΤΔ (abstract data type)
  - καθορίζει τις ιδιότητες του τύπου δεδομένων
  - δεν καθορίζει την υλοποίησή του
- Αλγεβρικός ορισμός ΑΤΔ
  - σύνταξη: όνομα του τύπου και επικεφαλίδες των πράξεων
  - σημασιολογία: κανόνες που περιγράφουν τη λειτουργία των πράξεων

Παράδειγμα ΑΤΔ: Σύνολα (i)

- Σύνταξη
  - Όνομα τύπου: **`set`** (σύνολα ακεραίων)
  - Επικεφαλίδες πράξεων:

```
const set empty;
set add (int x, set s);
boolean member (int x, set s);
set union (set s1, set s2);
boolean subset (set s1, set s2);
boolean equal (set s1, set s2);
```

Σημείωση: θεωρούμε ότι έχει οριστεί κατάλληλα ο τύπος **`boolean`**.

Παράδειγμα ΑΤΔ: Σύνολα (ii)

- Σημασιολογία
  - Αξιωματικός ορισμός **member**:

```
member(x, empty) = false
member(x, add(x, s)) = true
member(x, add(y, s)) = member(x, s)
    αν x != y
```

  - Αξιωματικός ορισμός **union**:

```
union(empty, s) = s
union(add(x, s1), s2) =
    add(x, union(s1, s2))
```

Παράδειγμα ΑΤΔ: Σύνολα (iii)

- Σημασιολογία
  - Αξιωματικός ορισμός **subset**:

```
subset(empty, s) = true
subset(add(x, s1), s2) =
    band(member(x, s2), subset(s1, s2))
```

  - Αξιωματικός ορισμός **equal**:

```
equal(s1, s2) =
    band(subset(s1, s2), subset(s2, s1))
```

Συγκεκριμένοι τύποι δεδομένων

- Συγκεκριμένος τύπος δεδομένων – ΣΤΔ (concrete data type)
  - καθορίζει τις ιδιότητες του τύπου δεδομένων
  - καθορίζει επακριβώς την υλοποίησή του
- Κάθε γλώσσα προγραμματισμού υποστηρίζει ορισμένους ΣΤΔ, π.χ.
  - απλοί τύποι: ακέραιοι αριθμοί, πραγματικοί αριθμοί, χαρακτήρες, λογικές τιμές
  - σύνθετοι τύποι: πίνακες (arrays), εγγραφές (records), δείκτες (pointers), σύνολα (sets)

Σχέση ΑΤΔ και ΣΤΔ (i)

- ΑΤΔ set: σύνολα ακεραίων $s = \{ 1, 3, 7, 9 \}$
- *Υλοποίηση 1:* πίνακας από boolean

false	true	false	true	false	false	false	true	false	true	false
s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]

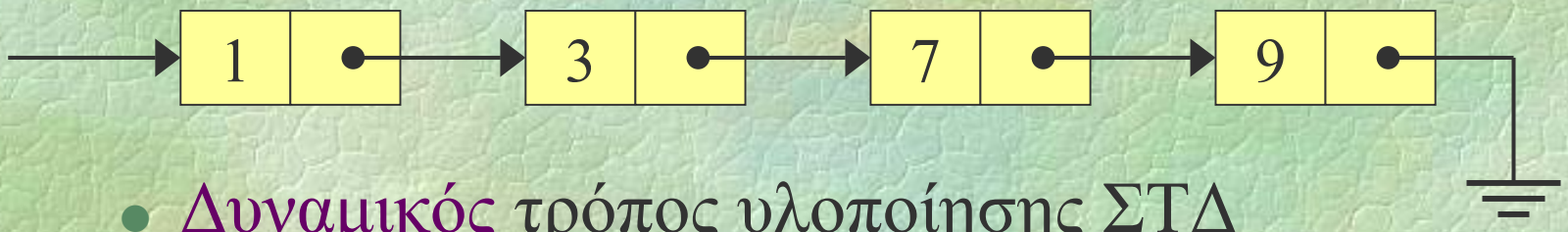
- *Υλοποίηση 2:* bits ενός πίνακα από int

0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0
	s[1]		s[3]				s[7]		s[9]						

  - Στατικός τρόπος υλοποίησης ΣΤΔ

Σχέση ΑΤΔ και ΣΤΔ (ii)

- *Υλοποίηση 3:* απλά συνδεδεμένη λίστα



  - Δυναμικός τρόπος υλοποίησης ΣΤΔ
- *Υλοποίηση 4:* συμβολοσειρά που περιέχει το μαθηματικό ορισμό του συνόλου

```
"{x | x=1 \/ x=3 \/ x=7 \/ x=9}"
"{x | 1<=x<=10 /\ x%2=1 /\ x<>5}"
```

  - Δύσκολη η υλοποίηση των πράξεων
  - Απειροσύνολα: `"{x | x > 100}"`