# Ενώσεις δεδομένων

## Απαριθμητές

## Ψηφιακοί τελεστές

## Αναδρομικές συναρτήσεις

*Διδάσκων*: Β. Ράπτης

## Ενώσεις δεδομένων (union) – τι και γιατί

Συσκευές με μικρή μνήμη => ανάγκη εξοικονόμησης πόρων

Παρατήρηση: αχρησιμοποίητη μνήμη.
Έστω τύπος δεδομένων που απαιτεί 8 bytes, π.χ. double


Όσο/όταν δε χρειάζεται μπορεί να αποθηκεύσει οτιδήποτε απαιτεί <= 8 bytes (char, int, float, double, struct μέχρι 8 bytes)

Λύση: μία περιοχή μνήμης από κοινού για περισσότερες μεταβλητές

Πρακτικός ορισμός: ένωση δεδομένων = μεταβλητή που μπορεί να αλλάζει τύπους.

# Ενώσεις δεδομένων (union) –πώς

```c
#include <stdio.h>
#include <string.h>

typedef union Student {
    int AM;
    char *name;
    char *fname;
    int semester;
    double grade;
} STUDENT;

int main(int argc, char **argv) {
    STUDENT someone;

    printf("size of STUDENT is %d\n", sizeof(STUDENT));
    someone.AM = 124;
    printf("someone's AM = %d\n", someone.AM);
    someone.fname="Tadopoulos";
    printf("someone's family name is %s\n", someone.fname);
    someone.grade = 8.5;
    printf("someone's grade = %f\n", someone.grade);
    return 0;
}
```

Θα τυπώσει:

```
size of STUDENT is 8
someone's AM = 124
someone's family name is Tadopoulos
someone's grade = 8.500000
```

*Αν ήταν struct θα είχε μέγεθος* 4 + 4 + 4 + 4 + 8 = 24

# Ενώσεις δεδομένων
# Απαριθμητές
# Ψηφιακοί τελεστές
# Αναδρομικές συναρτήσεις

*Διδάσκων*: Β. Ράπτης

## Τύποι απαρίθμησης (enum)

Επιτρέπουν αντιστοίχιση ακεραίων σταθερών σε συμβολικά ονόματα.

Παράδειγμα 1

```
enum day
{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
typedef enum day DAY;
...
DAY today, tomorrow;
...
today = Monday;
tomorrow = today + 1;
printf("Αύριο θα είναι %d\n", tomorrow); /* θα τυπώσει 2 */
```

Παράδειγμα 2

```
typedef enum spring_months {March=3, April=4, May=5} SPRING;
SPRING this_montht, next_month;
```

Παράδειγμα 3

```
enum dekades {ena = 1, deka = 10, ekato = 100};
enum dekades x, y, z;
```

# Ενώσεις δεδομένων
# Απαριθμητές
# Ψηφιακοί τελεστές
# Αναδρομικές συναρτήσεις

*Διδάσκων*: Β. Ράπτης

## Χειρισμός δυαδικών δεδομένων: ψηφιακοί τελεστές

**Τι είναι:** οι ψ. τ. (bitwise operators) "βλέπουν" τα bits μιας μεταβλητής και επιδρούν σε αυτά. Επιτρέπουν χειρισμούς *χαμηλού επιπέδου*.

**Για τι δεδομένα**: ακέραια (int και παραλλαγές τους, αλλά και char)

**Πώς λειτουργούν:**

α) παίρνουν ένα-ένα τα bits μίας ή δύο μεταβλητών και τα αλλάζουν
β) μετατοπίζουν τα ψηφία κατά ορισμένες θέσεις

**Ποιοι είναι και τι κάνουν:**

| Σύμβολο | Όνομα | Λειτουργία |
|---|---|---|
| & | ψηφιακό AND | 1 & 1 → 1, αλλιώς → 0 |
| \| | ψηφιακό OR | 0 \| 0 → 0, αλλιώς → 1 |
| ^ | ψηφιακό XOR | p ^ q → 1 για p == q, αλλιώς → 0 |
| ~ | συμπλήρωμα ως προς 1 | ~1 → 0, ~0 → 1 |
| >> | τελεστής ολίσθησης | μετατόπιση ψηφίων δεξιά |
| << | τελεστής ολίσθησης | μετατόπιση ψηφίων αριστερά |

Επιτρέπονται και σύνθετοι τελεστές: &=, |=, ^=

**Χαρακτηριστικά-εφαρμογές:** πολύ γρήγοροι, κατάλληλοι για βιβλιοθήκες γραφικών και άλλες εφαρμογές βασισμένες σε ακέραια δεδομένα.

## ψ. τ. - παραδείγματα

**AND**: εφαρμογή “μάσκας” για το μηδενισμό συγκεκριμένων ψηφίων.
Π.χ. $n = (65)_{10} = (01000001)_2$ (κωδικός ascii για 'A')

Εφαρμογή της μάσκας $(11011111)_2 = (223)_{10}$ αντιστρέφει το έκτο από δεξιά ψηφίο:
n1 = n & 223 δίνει $n1 = (01100001)_2 = (97)_{10}$ (κωδικός ascii για 'a').
Δηλ. η μάσκα $(223)_{10}$ μετατοπίζει κατά $(32)_{10}$ και κάνει τα κεφαλαία μικρά.

Ανάλογες εφαρμογές και για OR, XOR και συμπλήρωμα ως προς 1.

**Τελεστές ολίσθησης**: στο δυαδικό, μετατόπιση αριστερά = διπλασιασμός,
μετατόπιση δεξιά = υποδιπλασιασμός.

Χρήση: X >> n (διαίρεση διά $2^n$), X << n (πολλαπλασιασμός επί $2^n$)

Εξήγηση: έστω χ = 41.
X >> 3 <=> 00101001 → 00000101 = 41 / 8 = 5
X << 2 <=> 00101001 → 10100100 = 41 * 4 = 164

Ενώσεις δεδομένων
Απαριθμητές
Ψηφιακοί τελεστές

# Αναδρομικές συναρτήσεις

*Διδάσκων*: Β. Ράπτης

## Αναδρομικές συναρτήσεις

**Τι είναι**: συναρτήσεις που καλούν τον εαυτό τους.

**Παρατήρηση 1**: η αναδρομική κλήση ΔΕΝ συμμορφώνεται με το στυλ του Δομημένου Προγραμματισμού (διαδοχή, επιλογή, επανάληψη).

**Παρατήρηση 2**: κάθε αναδρομικός αλγόριθμος μπορεί να μετασχηματιστεί σε μη αναδρομικό (θεώρημα!).

**Χρειάζεται;;;**
Πλεονέκτημα: Επιτρέπει πιο συμπαγή, εύκολο στη γραφή και κατανοητό κώδικα.
Παράδειγμα: βλ. υλοποίηση της ταχείας ταξινόμησης (ασκήσεις 10ης εβδομάδας).
Μειονέκτημα: κάθε κλήση => νέα αντίγραφα τοπικών μεταβλητών => περισσότερη μνήμη (κίνδυνος εξάντλησης στοίβας).

# Παραδείγματα αναδρομής

```
int fact(int n) {
    int p = 1;
    if(n > 0) p = n * fact(n-1);
    return p;
}
```

Υπολογισμός παραγοντικού:

$$N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$$

άρα,

$$N! = (N-1)! \cdot N$$

```
double dpow(double x, int n) {
    if( x == 0)
        y = 0.0;
    else if( n == 0 )
        y = 1.0;
    if (n > 0)
      y = dpow(x, n-1) * x;
    else if (n < 0)
      y = dpow(x, n+1) / x;
    return y;
}
```

Υπολογισμός δύναμης:

$$x^n = x \; x \; x \; \ldots \; x = x^{n-1} \cdot x$$

Παρόμοια,

$$x^{-n} = x^{-(n-1)} / x$$