ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

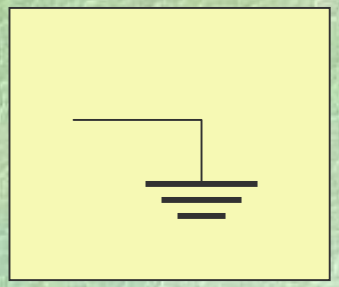
http://www.softlab.ntua.gr/~nickie/Courses/progtech/

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

Διαφάνειες παρουσίασης #5 (β)

- ✓ Δείκτες (συνέχεια)
- ✓ Μετατροπές τύπων
- ✓ Δυναμική παραχώρηση μνήμης
- ✓ Δυναμικοί ΣΤΔ
- ✓ Απλά συνδεδεμένες λίστες
- ✓ Ουρές

Κενός δείκτης και δείκτης σε κενό



- Ο κενός δείκτης: **`NULL`**
  - Απαγορεύεται να αποδεικτοδοτηθεί!

```
int *p = NULL;
*p = 42;            /* wrong! */
```

  - Ο μόνος δείκτης που αντιστοιχεί στην ψευδή λογική τιμή, όταν χρησιμοποιείται σε συνθήκη
- Ο δείκτης σε κενό: **`void *`**
  - Γενική μορφή δείκτη
  - Απαγορεύεται να αποδεικτοδοτηθεί!
  - Απαγορεύεται η αριθμητική δεικτών!

Δείκτες αντί περάσματος με αναφορά

- Κώδικας Pascal

```
procedure inc (var x : integer);
begin
   x := x+1
end;

... inc(a) ...
```

- Ισοδύναμος κώδικας C

```
void inc (int * px)
{
   (*px)++;
}

... inc(&a) ...
```

Δείκτες σε εγγραφές

- Συντομογραφία

  `p->x` είναι ισοδύναμο με `(*p).x`

- Παράδειγμα

```
struct {
   int x, y;
} coordinates, *p;

coordinates.x = 1;
coordinates.y = 3;

p = &coordinates;

printf("%d\n", p->x);
printf("%d\n", p->y);
```

Μετατροπές τύπων (i)

- Έμμεσες (coercions)

```
double d = 3;
int x = 3.14;
```

(με ανάθεση)

```
int f (int x);
f(3.14);
```

(με πέρασμα παραμέτρου)

- Άμεσες (type casting)

(*τύπος*) *έκφραση*

```
(double) 3
(int) 3.14
(int *) NULL
```

Μετατροπές τύπων (ii)

- Πρόβλημα: πώς μπορώ να τυπώσω το αποτέλεσμα της πραγματικής διαίρεσης δυο ακεραίων αριθμών, χωρίς νέες μεταβλητές;

```
int x = 5, y = 3;
```

- Λάθος λύση #1: `printf("%d", x/y);`
- Λάθος λύση #2: `printf("%lf", x/y);`
- Σωστές λύσεις:

```
printf("%lf", 1.0 * x / y);
printf("%lf", (double) x / (double) y);
printf("%lf", (double) x / y);
```

Δυναμική παραχώρηση μνήμης (i)

- Συναρτήσεις βιβλιοθήκης <stdlib.h>
  - **`void * malloc (size_t n);`**

    Δυναμική παραχώρηση μνήμης μήκους **`n`** bytes.
    Το αποτέλεσμα πρέπει να μετατραπεί στο σωστό τύπο.
    Επιστρέφεται **`NULL`** αν εξαντληθεί η μνήμη.
  - **`void free (void * p);`**

    Αποδέσμευση της μνήμης στην οποία δείχνει το **`p`**.
    Το **`p`** πρέπει να έχει δημιουργηθεί με προηγούμενη κλήση στη **`malloc`**.
- Πόσα bytes χρειάζονται;
  - **`sizeof(`***type***`)`** π.χ. **`sizeof(int)`**
  - **`sizeof(`***variable***`)`** π.χ. **`sizeof(x)`**

Δυναμική παραχώρηση μνήμης (ii)

- Παράδειγμα

```
int *p;
int i;

p = (int *) malloc(sizeof(int));
*p = 42;
free(p);

p = (int *) malloc(10 * sizeof(int));
for (i = 0; i < 10; i++)
   p[i] = 42;
free(p);
```

Το αποτέλεσμα της **malloc** πρέπει να ελεγχθεί ότι δεν είναι **NULL**!

Δυναμικοί ΣΤΔ (i)

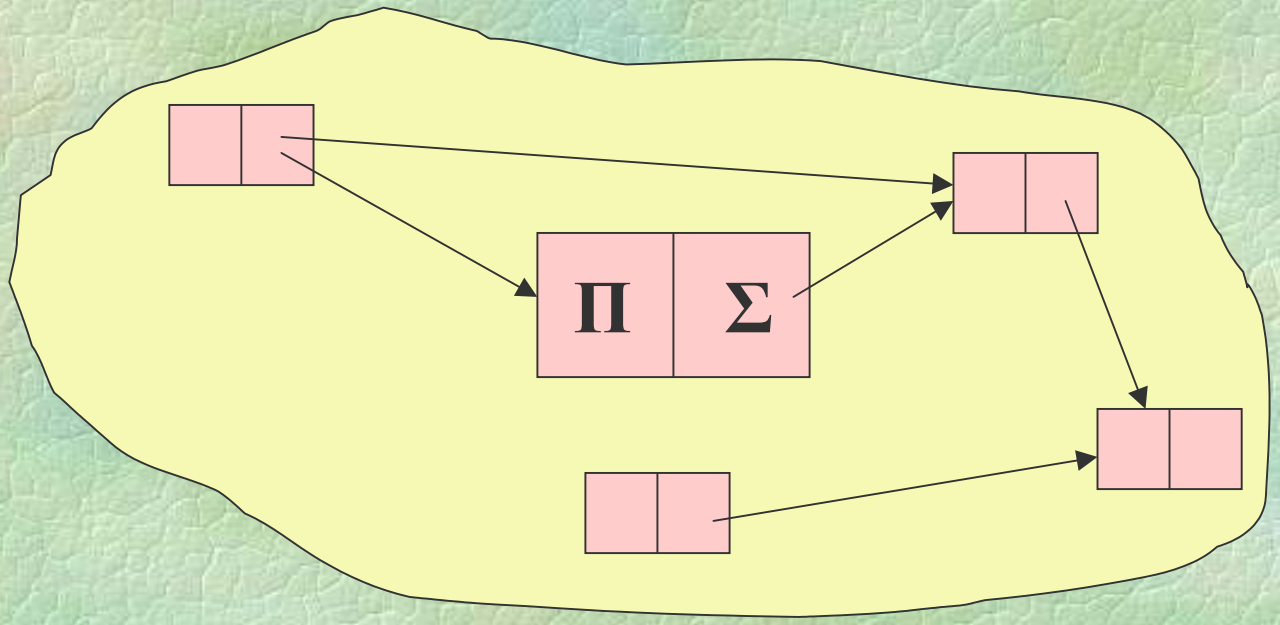
- Χρησιμοποιούνται για την υλοποίηση ΑΤΔ
- Παραδείγματα
  - συνδεδεμένες λίστες
  - δέντρα
  - γράφοι
- Πώς υλοποιούνται στη C;
  - με κατάλληλο συνδυασμό δομών (`struct`) και δεικτών (pointers), και
  - με δυναμική παραχώρηση μνήμης

Δυναμικοί ΣΤΔ (ii)

- Πλεονεκτήματα έναντι στατικών ΣΤΔ
  - Δεν επιβάλλουν περιορισμούς στο μέγιστο πλήθος των δεδομένων
  - Η μνήμη που χρησιμοποιείται είναι ανάλογη του πραγματικού πλήθους των δεδομένων
  - Κάποιες πράξεις υλοποιούνται αποδοτικότερα
- Μειονεκτήματα έναντι στατικών ΣΤΔ
  - Για σταθερό και γνωστό πλήθος δεδομένων, χρησιμοποιούν συνήθως περισσότερη μνήμη
  - Κάποιες πράξεις υλοποιούνται λιγότερο αποδοτικά

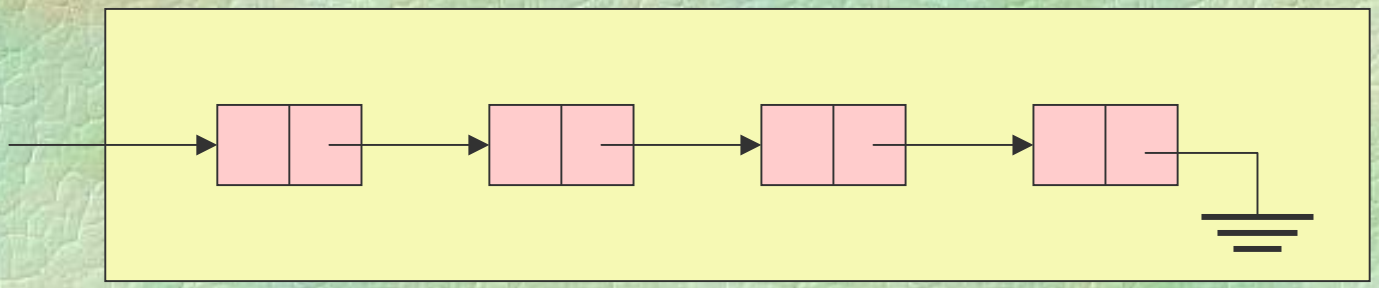
Δυναμικοί ΣΤΔ (iii)

- Ιδέα
  - Κατασκευάζεται ένα σύνολο κόμβων
  - Κάθε κόμβος περιέχει πληροφορίες και συνδέσμους προς άλλους κόμβους



Απλά συνδεδεμένες λίστες (i)

- Είναι γραμμικές διατάξεις
- Κάθε κόμβος περιέχει ένα σύνδεσμο στον επόμενο κόμβο
- Ο τελευταίος κόμβος έχει κενό σύνδεσμο

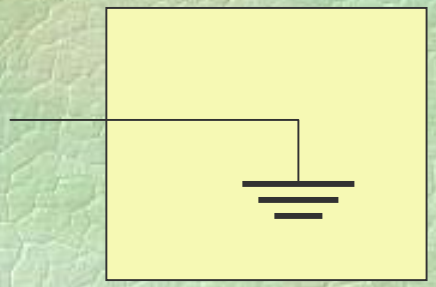


Απλά συνδεδεμένες λίστες (ii)

- Παράδειγμα: λίστα ακεραίων

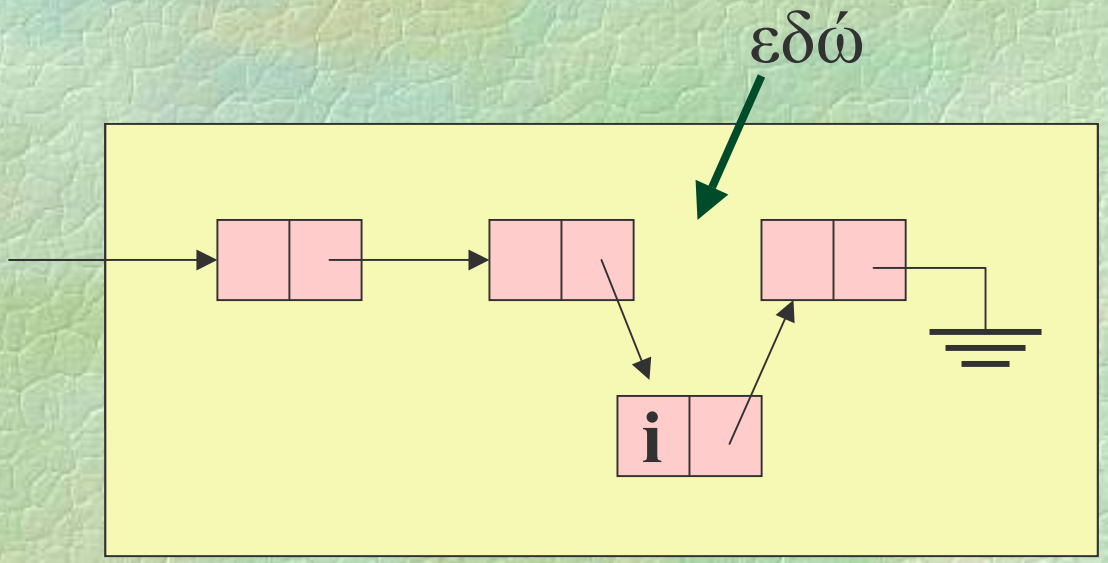
```
struct node_tag {
    int data;
    struct node_tag * next;
};

typedef struct node_tag
            Node, * LinkedList;
```

- Κενή λίστα



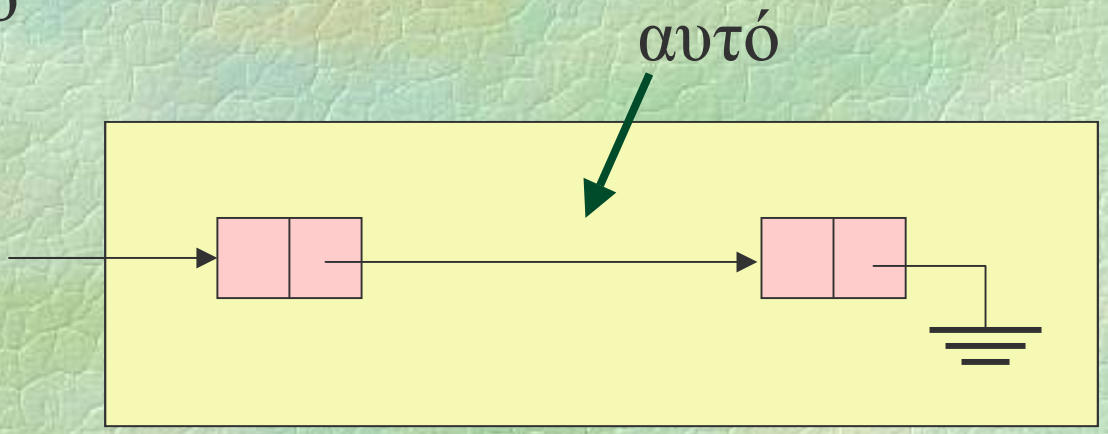
Απλά συνδεδεμένες λίστες (iii)

- Προσθήκη στοιχείων
  - απόφαση πού θα προστεθεί
  - δημιουργία νέου κόμβου
  - αντιγραφή πληροφορίας
  - σύνδεση νέου κόμβου



Απλά συνδεδεμένες λίστες (iv)

- Αφαίρεση στοιχείων
  - απόφαση ποιο στοιχείο θα αφαιρεθεί
  - καταστροφή κόμβου
  - σύνδεση υπόλοιπων κόμβων

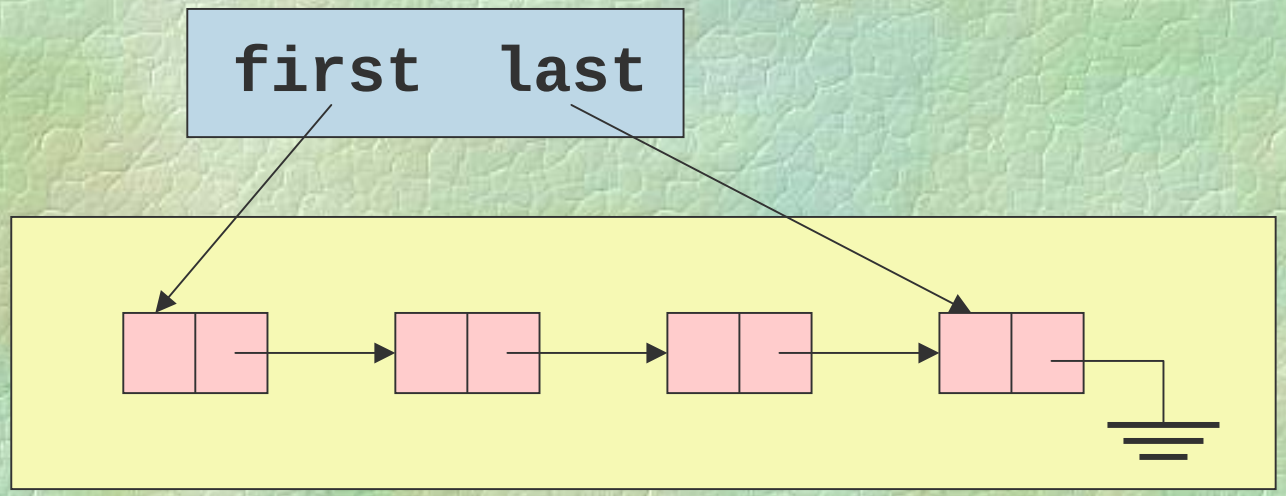


Ουρές (i)

- First In First Out (FIFO)
  ό,τι μπαίνει πρώτο, βγαίνει πρώτο
- Ουρά ακεραίων
  - ΑΤΔ: **queue**
  - `const queue queueEmpty;`
  - `void queueInsert (queue * qp, int t);`
  - `int queueRemove (queue * qp);`
  - `int queueHead (queue q);`

Ουρές (ii)

- Υλοποίηση με απλά συνδεδεμένη λίστα



Υλοποίηση ουράς σε C (i)

- Υλοποίηση με απλά συνδεδεμένη λίστα

```
typedef struct list_tag {
    int data;
    struct list_tag * next;
} ListNode;
```

- Τύπος queue

```
typedef struct {
    ListNode * first;
    ListNode * last;
} queue;
```

Υλοποίηση ουράς σε C (ii)

- Άδεια ουρά

```
const queue queueEmpty = { NULL, NULL };
```

- Εισαγωγή στοιχείου

```
void queueInsert (queue * qp, int t)
{
    ListNode * n = (ListNode *)
       malloc(sizeof(ListNode));

    if (n == NULL) {
       printf("Out of memory\n");
       exit(1);
    }
```

Υλοποίηση ουράς σε C (iii)

- Εισαγωγή στοιχείου (συνέχεια)

```
    n->data = t;
    n->next = NULL;

    if (qp->last == NULL)
        qp->first = qp->last = n;
    else {
        qp->last->next = n;
        qp->last = n;
    }
}
```