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 #6 (α)

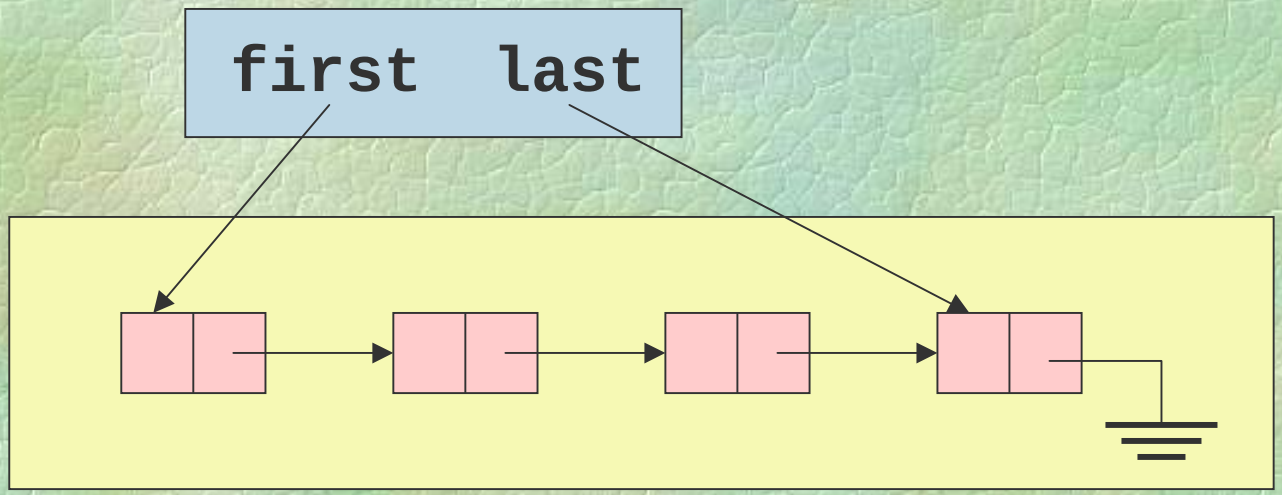
- ✓Ουρές (επανάληψη και συνέχεια)
- ✓Στοίβες
- ✓Διπλά συνδεδεμένες λίστες
- ✓Κυκλικές λίστες

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;
    }
}
```

Υλοποίηση ουράς (iv)

- Αφαίρεση στοιχείου

```
int queueRemove (queue * qp)
{
    ListNode * n;
    int result;

    if (qp->first == NULL) {
        printf("Nothing to remove"
               " from an empty queue\n");
        exit(1);
    }
```

Υλοποίηση ουράς (v)

- Αφαίρεση στοιχείου (συνέχεια)

```
    n = qp->first;
    result = qp->first->data;
    qp->first = qp->first->next;
    free(n);

    if (qp->first == NULL)
        qp->last = NULL;

    return result;
}
```

Υλοποίηση ουράς (vi)

- Εξέταση στοιχείου

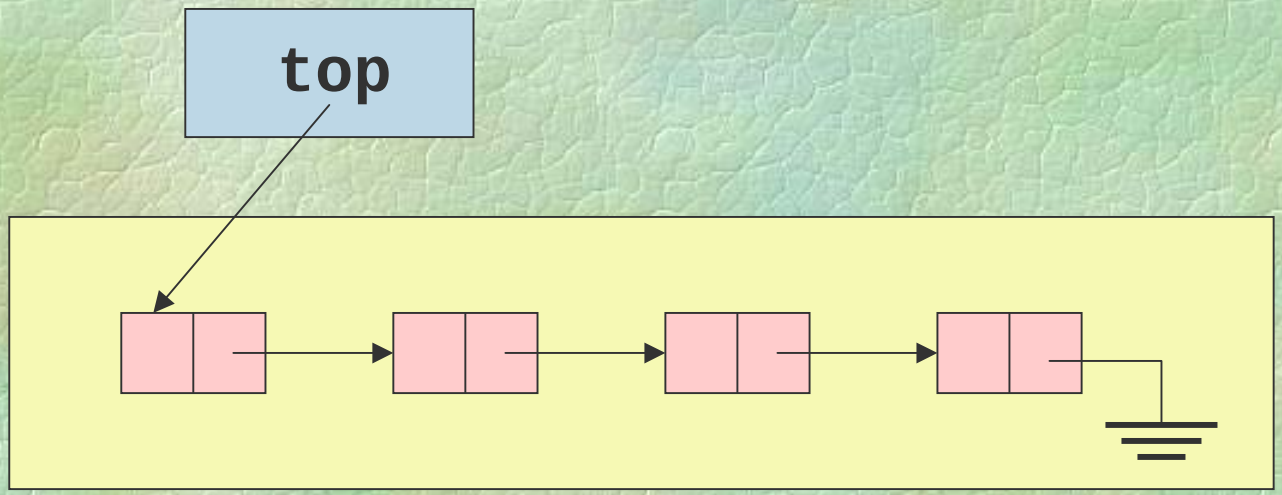
```
int queueHead (queue q)
{
    if (q.first == NULL) {
        fprintf(stderr, "Nothing to see"
          " in an empty queue\n");
        exit(1);
    }

    return q.first->data;
}
```

Στοίβες (i)

- Last In First Out (LIFO)
  ό,τι μπαίνει τελευταίο, βγαίνει πρώτο
- Στοίβα ακεραίων
  - ΑΤΔ: **stack**
  - `const stack stackEmpty;`
  - `void stackPush (stack * sp, int t);`
  - `int stackPop (stack * sp);`
  - `int stackTop (stack s);`

Στοίβες (ii)

- Υλοποίηση με απλά συνδεδεμένη λίστα



Υλοποίηση στοίβας σε C (i)

- Τύπος stack

```
typedef ListNode * stack;
```

- Άδεια στοίβα

```
const stack stackEmpty = NULL;
```

- Εισαγωγή στοιχείου

```
void stackPush (stack * sp, int t)
{
    ListNode * n = (ListNode *)
       malloc(sizeof(ListNode));

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

Υλοποίηση στοίβας σε C (ii)

- Εισαγωγή στοιχείου (συνέχεια)

```
    n->data = t;
    n->next = *sp;
    *sp = n;
}
```

- Αφαίρεση στοιχείου

```
int stackPop (stack * sp)
{
    ListNode * n;
    int result;

    if (*sp == NULL) {
        printf("Nothing to remove"
               " from an empty stack\n");
        exit(1);
    }
```

Υλοποίηση στοίβας σε C (iii)

- Αφαίρεση στοιχείου (συνέχεια)

```
    n = *sp;
    result = (*sp)->data;
    *sp = (*sp)->next;
    free(n);
    return result;
}
```

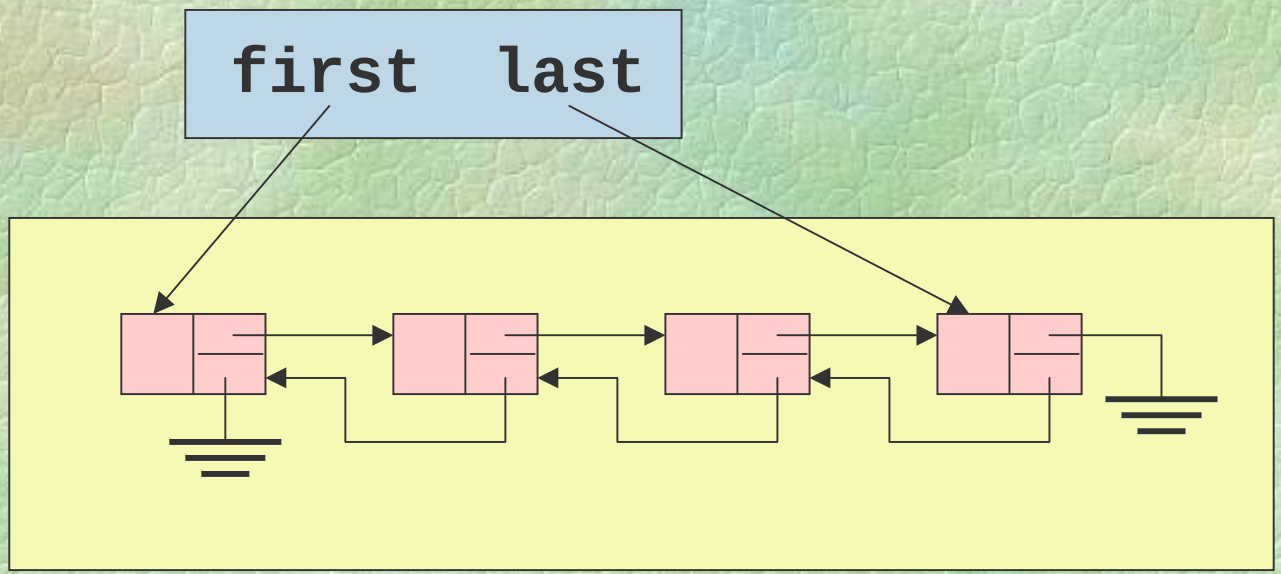
Υλοποίηση στοίβας σε C (iv)

- Εξέταση στοιχείου

```
int stackTop (stack s)
{
    if (s == NULL) {
        printf("Nothing to see"
               " in an empty stack\n");
        exit(1);
    }

    return s->data;
}
```

Διπλά συνδεδεμένες λίστες (i)

- Επίσης γραμμικές διατάξεις
- Δυο σύνδεσμοι σε κάθε κόμβο, προς τον επόμενο και προς τον προηγούμενο
- Γενική μορφή, π.χ. για υλοποίηση ουράς:



Διπλά συνδεδεμένες λίστες (ii)

- Τύπος κόμβου DListNode

```
typedef struct DListNode_tag {
    int data;
    struct DListNode_tag * next;
    struct DListNode_tag * prev;
} DListNode;
```

- Τύπος dlist

```
typedef struct {
    DListNode * first;
    DListNode * last;
} dlist;
```

- Άδεια λίστα

```
const dlist dlistEmpty = { NULL, NULL };
```

Διπλά συνδεδεμένες λίστες (iii)

- Εισαγωγή στοιχείου στην αρχή

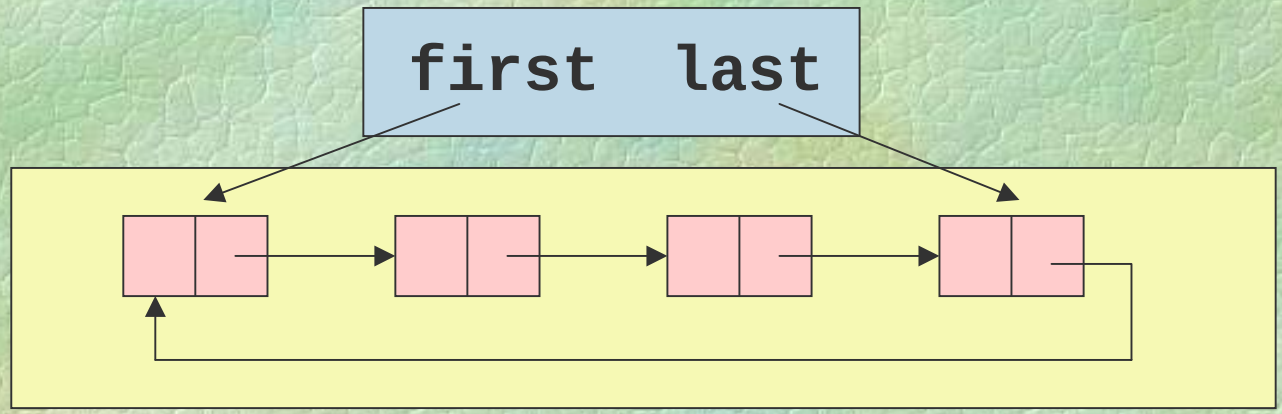
```
void dlistInsert (dlist * lp, int t)
{
    DListNode * n = (DListNode *)
       malloc(sizeof(DListNode));

    if (n == NULL) {
       fprintf(stderr, "Out of memory\n");
       exit(1);
    }

    n->data = t;
```

Διπλά συνδεδεμένες λίστες (iv)

- Εισαγωγή στοιχείου στην αρχή (συνέχεια)

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

Κυκλικές λίστες (i)



- Τύπος clist

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

- Άδεια λίστα

```
const clist clistEmpty = { NULL, NULL };
```

Κυκλικές λίστες (ii)

- Εισαγωγή στοιχείου

```
void clistInsert (clist * lp, int t)
{
    ListNode * n = (ListNode *)
       malloc(sizeof(ListNode));

    if (n == NULL) {
       fprintf(stderr, "Out of memory\n");
       exit(1);
    }

    n->data = t;
```

Κυκλικές λίστες (iii)

- Εισαγωγή στοιχείου (συνέχεια)

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

Κυκλικές λίστες (iv)

- Αφαίρεση στοιχείου

```
int clistRemove (clist * lp)
{
    int result;

    if (lp->first == NULL) {
        fprintf(stderr, "Nothing to remove"
          " from empty list\n");
        exit(1);
    }

    result = lp->first->data;
```

Κυκλικές λίστες (v)

- Αφαίρεση στοιχείου (συνέχεια)

```
    if (lp->first == lp->last) {
        free(lp->first);
        lp->first = lp->last = NULL;
    }
    else {
        lp->first = lp->first->next;
        free(lp->last->next);
        lp->last->next = lp->first;
    }

    return result;
}
```

Κυκλικές λίστες (vi)

- Εκτύπωση στοιχείων

```
void clistPrint (clist l)
{
    ListNode * n;

    for (n = l.first; n != NULL;
            n = n->next) {
        printf("%d\n", n->data);
        if (n->next == l.first)
            break;
    }
}
```