

Διάλεξη 21 - Λίστες

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινήσεις

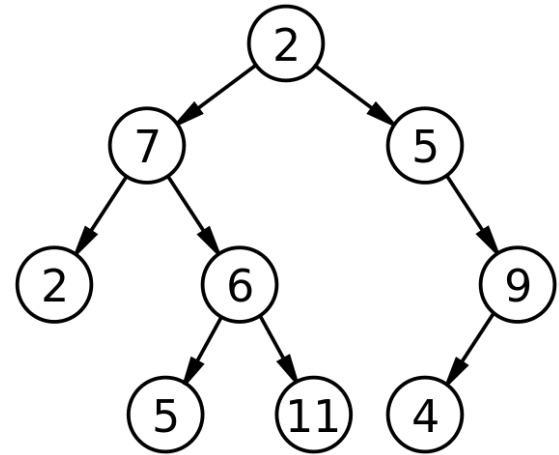
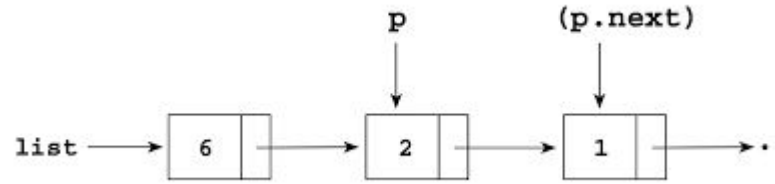
- BMP format: ο έλεγχος ορθότητας μετράει στην βαθμολογία
- DNA: χρειάζεται ελαχιστοποίηση της μνήμης για τις "δύσκολες" αλυσίδες

Την προηγούμενη φορά

- Προχωρημένες Δομές
 - Πεδία Δυφίων (Bit Fields)
 - Ενώσεις (Unions)
 - Απαριθμήσεις (Enumerations)
 - Αυτοαναφορικές (Self-Referential)

Σήμερα

- Αυτοαναφορικές Δομές
 - Λίστες
 - ~~Δέντρα~~
 - Αλγόριθμοι χρήσης και διάσχισης
 - Hands-on χειρισμός δυαδικών αρχείων



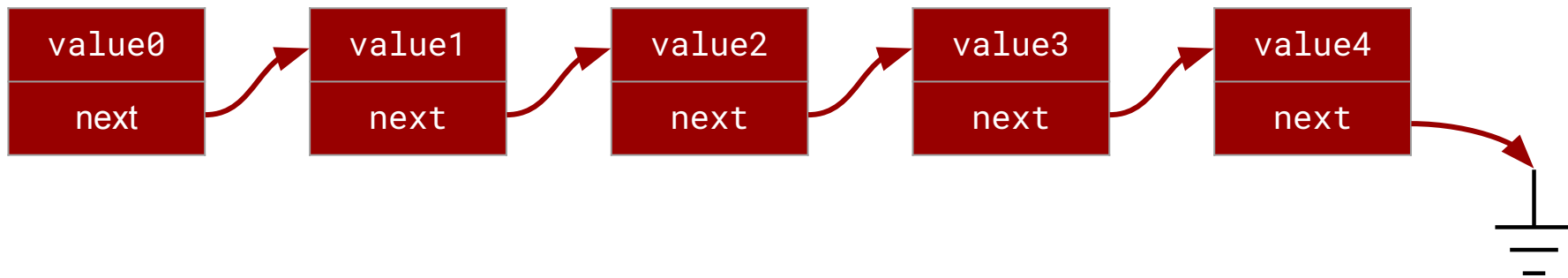
Απλά Συνδεδεμένη Λίστα (Single Linked List)

Στην γενική μορφή δηλώνεται ως:

```
struct listnode {  
    int value;  
    struct listnode * next;  
};
```

Απλά Συνδεδεμένη Λίστα (Single Linked List)

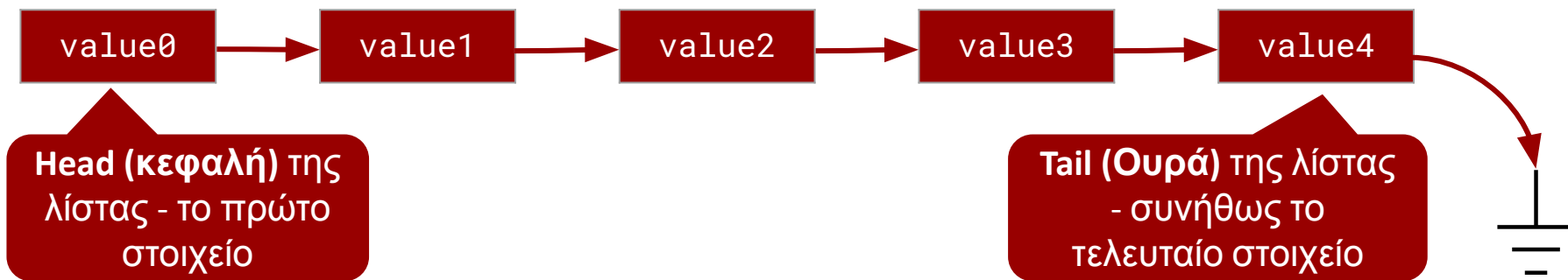
Η απλά συνδεδεμένη λίστα (single linked list) είναι ένας τύπος δεδομένων που το κάθε στοιχείο δείχνει (links) στο επόμενο και το τελευταίο δείχνει στο NULL.



Μήκος λίστας (list length): ο αριθμός των στοιχείων που περιέχει (5 παραπάνω)

Απλά Συνδεδεμένη Λίστα (Single Linked List)

Η απλά συνδεδεμένη λίστα (single linked list) είναι ένας τύπος δεδομένων που το κάθε στοιχείο δείχνει (links) στο επόμενο και το τελευταίο δείχνει στο NULL.



Μήκος λίστας (list length): ο αριθμός των στοιχείων που περιέχει (5 παραπάνω)

Βασικές Λειτουργίες με Λίστες

1. `is_empty`: Έλεγχος αν η λίστα είναι άδεια
2. `insert`: Προσθήκη στοιχείου στην λίστα
3. `print`: Τύπωμα στοιχείων λίστας
4. `length`: Εύρεση μήκους λίστας
5. `find`: Εύρεση στοιχείου σε λίστα
6. `delete`: Αφαίρεση στοιχείου από λίστα

is_empty: Έλεγχος αν η λίστα είναι άδεια

```
#include <stdio.h>

typedef struct listnode {int value; struct listnode * next;} * List;

int is_empty(List list) {

    return list == NULL;

}

int main() {

    struct listnode node = {42, NULL};

    List list1 = &node;

    List list2 = NULL;

    printf("Is empty: %d\n", is_empty(list1));

    printf("Is empty: %d\n", is_empty(list2));

    return 0;

}
```

is_empty: Έλεγχος αν η λίστα είναι άδεια

```
#include <stdio.h>

typedef struct listnode {int value; struct listnode * next;} * List;

int is_empty(List list) {

    return list == NULL;

}

int main() {

    struct listnode node = {42, NULL};

    List list1 = &node;

    List list2 = NULL;

    printf("Is empty: %d\n", is_empty(list1));

    printf("Is empty: %d\n", is_empty(list2));

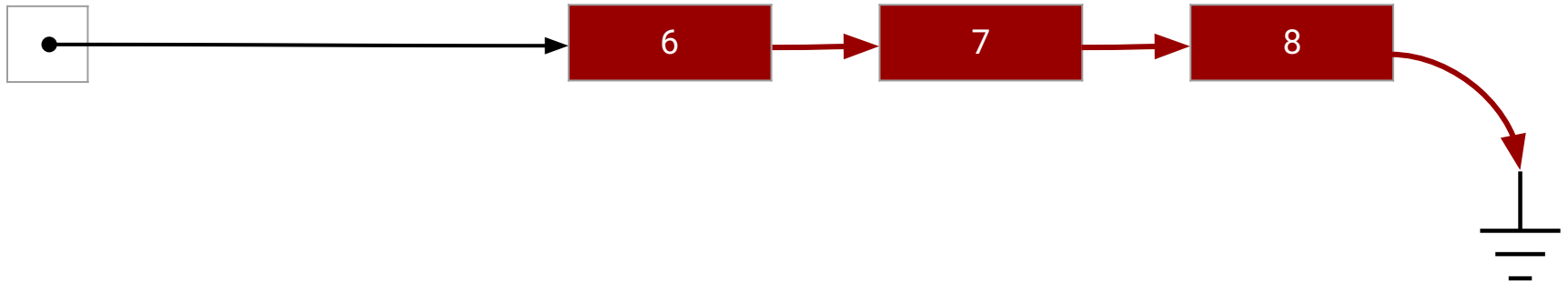
    return 0;

}
```

```
$ ./list
Is empty: 0
Is empty: 1
```

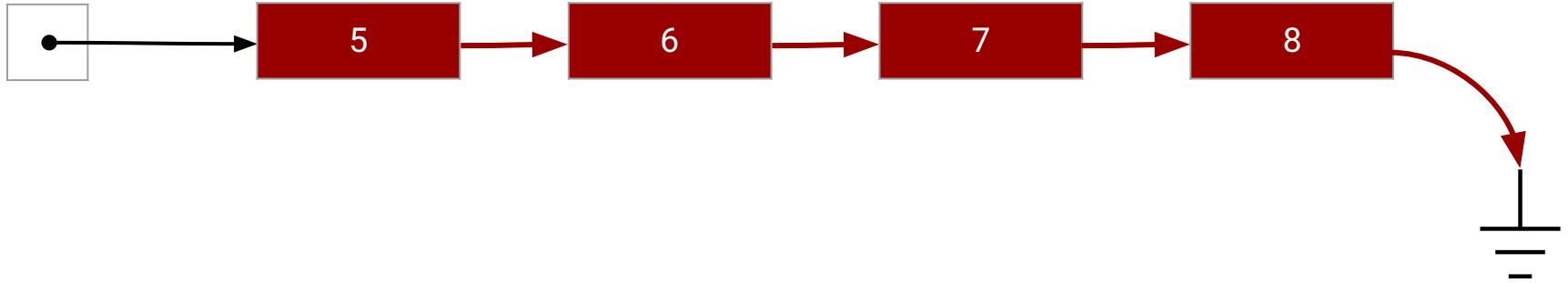
Θέλω να προσθέσω ένα στοιχείο (π.χ., το 5) σε λίστα. Πως;

`list`



Θέλω να προσθέσω ένα στοιχείο (π.χ., το 5) σε λίστα. Πως;

`list`



insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

    List current_head = *list;

    List new_head = malloc(sizeof(struct listnode));

    new_head->value = value;

    new_head->next = current_head;

    *list = new_head;

}

int main() {

    List list = NULL;

    insert(&list, 42); insert(&list, 43); insert(&list, 44);

    print(list);

    return 0;

}
```

Δημιουργία νέου
κόμβου λίστας
στον σωρό (heap)

Αρχικοποίηση
κόμβου

Ο νέος κόμβος γίνεται η νέα
κεφαλή της λίστας

insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

    List current_head = *list;

    List new_head = malloc(sizeof(struct listnode));

    new_head->value = value;

    new_head->next = current_head;

    *list = new_head;

}

int main() {

    List list = NULL;

    insert(&list, 42); insert(&list, 43); insert(&list, 44);

    print(list); // commented out for size

    return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

    List current_head = *list;

    List new_head = malloc(sizeof(struct listnode));

    new_head->value = value;

    new_head->next = current_head;

    *list = new_head;

}

int main() {

    List list = NULL;

    insert(&list, 42); insert(&list, 43); insert(&list, 44);

    print(list); // commented out for size

    return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

```
$ ./insert
list:  -> 44 -> 43 -> 42 -> NULL
```

print: Τύπωμα στοιχείων λίστας

```
void print(List list) {  
    printf("list: ");  
    while(list) {  
        printf(" -> %d", list->value);  
        list = list->next;  
    }  
    printf(" -> NULL\n");  
}
```


length: Εύρεση μήκους λίστας

length: Εύρεση μήκους λίστας

```
int length(List list) {  
    int counter = 0;  
    while(list) {  
        counter++;  
        list = list->next;  
    }  
    return counter;  
}
```

Πως θα το κάναμε αναδρομικά;

length: Εύρεση μήκους λίστας

```
int length(List list) {  
    int counter = 0;  
    while(list) {  
        counter++;  
        list = list->next;  
    }  
    return counter;  
}
```

```
int length(List list) {  
    if (!list) return 0;  
    return 1 + length(list->next);  
}
```

Ποια η πολυπλοκότητα των παραπάνω
ως προς χρόνο και χώρο;

length: Εύρεση μήκους λίστας

```
int length(List list) {  
    int counter = 0;  
    while(list) {  
        counter++;  
        list = list->next;  
    }  
    return counter;  
}
```

Χρόνος: $O(n)$
Χώρος: $O(1)$

```
int length(List list) {  
    if (!list) return 0;  
    return 1 + length(list->next);  
}
```

Ποια η πολυπλοκότητα των παραπάνω ως προς χρόνο και χώρο;

Χρόνος: $O(n)$
Χώρος: $O(n)$

find: Εύρεση στοιχείου σε λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

List find(List list, int value) {

    while(list && list->value != value) {

        list = list->next;

    }

    return list;

}

int main() {

    List list = NULL;

    insert(&list, 42); insert(&list, 43); insert(&list, 44);

    printf("Found 43: %x\n", find(list, 43));

    printf("Found 34: %x\n", find(list, 34));

    return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

find: Εύρεση στοιχείου σε λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

List find(List list, int value) {

    while(list && list->value != value) {

        list = list->next;

    }

    return list;

}

int main() {

    List list = NULL;

    insert(&list, 42); insert(&list, 43); insert(&list, 44);

    printf("Found 43: %x\n", find(list, 43));

    printf("Found 34: %x\n", find(list, 34));

    return 0;

}
```

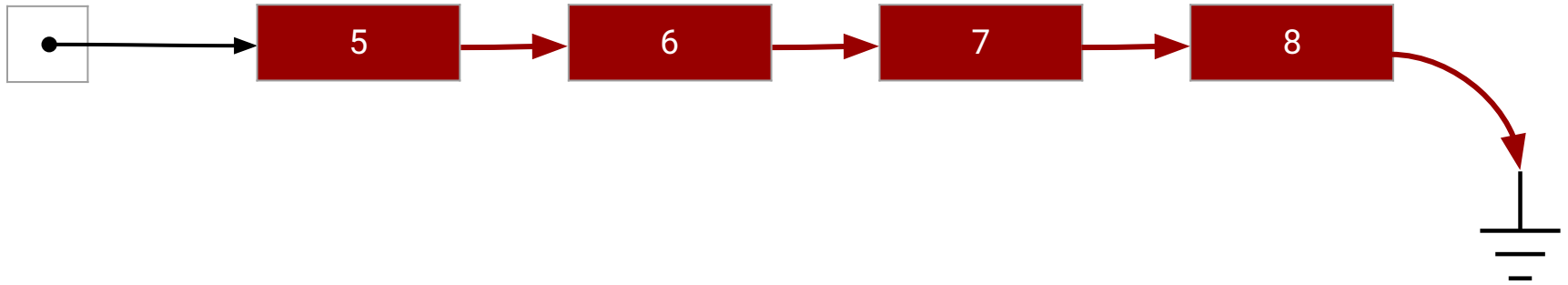
Τι θα τυπώσει το πρόγραμμα;

```
$ ./find
Found 43: 161862c0
Found 34: 0
```

Χρόνος: $O(n)$
Χώρος: $O(1)$

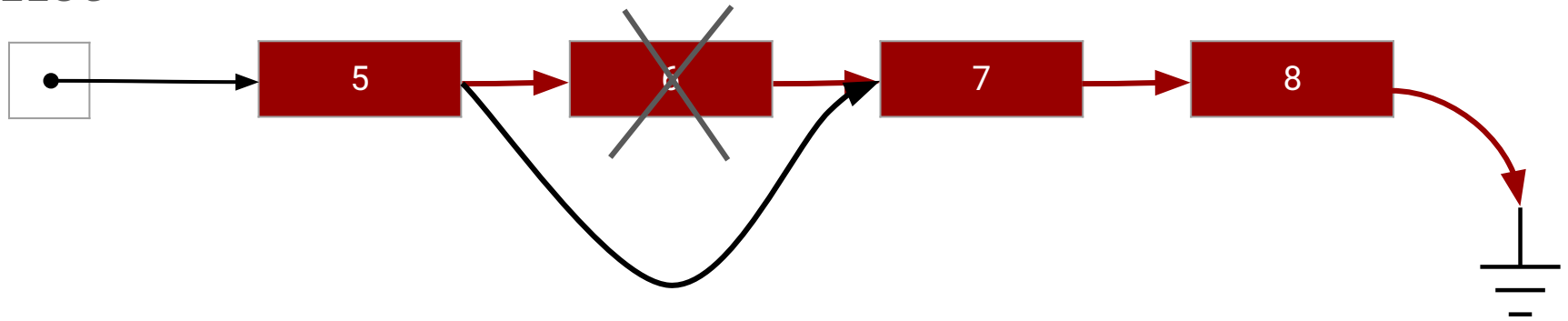
Θέλω να αφαιρέσω ένα στοιχείο (π.χ., το 6). Πως;

`list`



Θέλω να αφαιρέσω ένα στοιχείο (π.χ., το 6). Πως;

list



delete: Αφαίρεση στοιχείου από λίστα

```
void delete(List * list, int value) {  
    List temp;  
    while(*list && (*list)->value != value) {  
        list = &((*list)->next);  
    }  
    if (*list) {  
        temp = *list;  
        *list = temp->next;  
        free(temp);  
    }  
}
```

Πίνακες vs Λίστες

Πίνακες

1. Τα στοιχεία αποθηκεύονται σε **συνεχόμενες θέσεις** στην μνήμη
2. Ο χώρος μνήμης είναι **όσος χρειάζεται** για την αποθήκευση των στοιχείων
3. **Πρόσβαση** σε κάθε στοιχείο **σε σταθερό χρόνο** ($O(1)$) χρησιμοποιώντας `array[i]`
4. **Αναδιάταξη** στοιχείων συνήθως παίρνει **$O(n)$ χρόνο** (π.χ., εισαγωγή στο `array[0]`)
5. Στην **δήλωσή** τους χρειάζεται να **ξέρουμε πόσα στοιχεία** θα εισάγουμε (πιο στατικό ως δομή δεδομένων)

Λίστες

1. Τα στοιχεία μπορούν να αποθηκευτούν **σε οποιαδήποτε θέση** στην μνήμη
2. Ο χώρος μνήμης είναι **επαυξημένος κατά ένα `sizeof(pointer)`** για κάθε στοιχείο
3. **Πρόσβαση** σε κάθε στοιχείο **σε γραμμικό χρόνο** ($O(n)$)
4. Εύκολη / γρήγορη **αναδιάταξη** στοιχείων με **χρήση δεικτών**
5. Στην **δήλωσή** τους **δεν χρειάζεται να ξέρουμε πόσα στοιχεία** θα εισάγουμε (πιο δυναμικό ως δομή δεδομένων)

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου προτείνω να διαβάσετε τις σελίδες

120-135

- [Linked List](#)
- [Αφηρημένοι Τύποι Δεδομένων \(ΑΤΔ\)](#) - Abstract Data Types

Ευχαριστώ και καλή μέρα εύχομαι!
Keep Coding ;)