Διάλεξη 20 - Προχωρημένες Δομές

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινήσεις

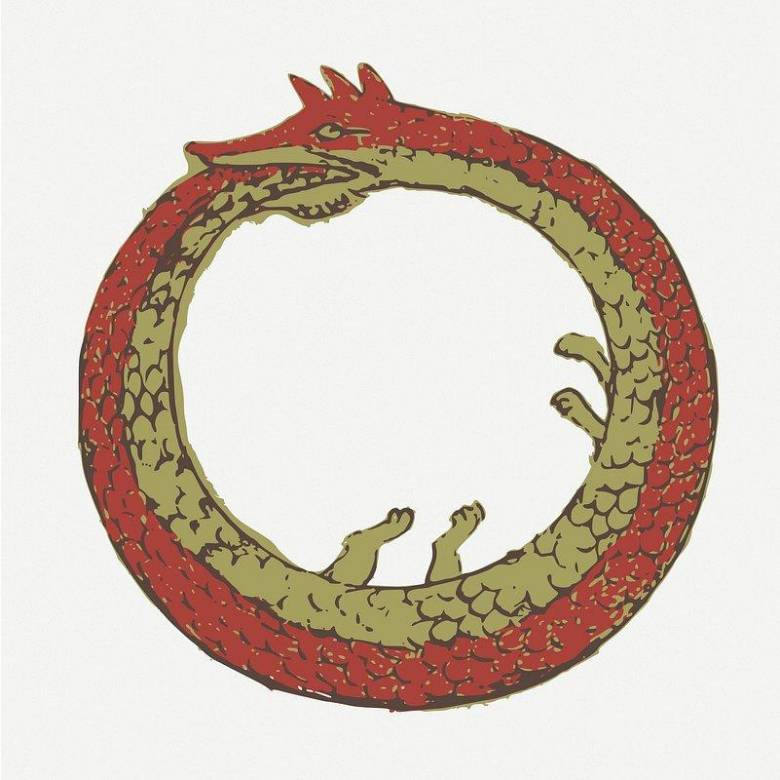
- Την Τρίτη είναι η προθεσμία για την υποβολή της εργασίας #1
    - Μην το ξεχάσουμε!
- Επόμενη βδομάδα: επαναλήψεις, προτείνετε θέματα

Την Προηγούμενη Φορά

- Δομές
- Επίλυση Προβλημάτων

Σήμερα

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



Πεδία Δυφίων σε Δομές (Struct Bit Fields)

Ένα πεδίο δομής μπορεί να οριστεί έτσι ώστε να έχει ως μέγεθος στην μνήμη ένα συγκεκριμένο αριθμό από bits. Συγκεκριμένα, ακολουθούμε την σύνταξη:

Γιατί; Για να σώσουμε μνήμη

```
struct όνομα {
    τύπος1 πεδίο1 : αριθμός_bits1;
    τύπος2 πεδίο2 : αριθμός_bits2;
    τύπος3 πεδίο3 : αριθμός_bits3;
    ...
};
```

Τυπικά οι μεταγλωττιστές υποστηρίζουν τύπους int, long, char

Πεδία Δυφίων σε Δομές (Struct Bit Fields)

Ένα πεδίο δομής μπορεί να οριστεί έτσι ώστε να έχει ως μέγεθος στην μνήμη ένα συγκεκριμένο αριθμό από bits. Συγκεκριμένα, ακολουθούμε την σύνταξη:

```
struct student_status {
  int registered : 1;
  int year : 3;
  int grade : 4;
};
```

Δύο καταστάσεις (boolean): registered ή όχι - 1 bit αρκεί

Ο βαθμός είναι από το 0 μέχρι το 10, οπότε 4 bits φτάνουν για να αναπαραστήσουμε όλες τις δυνατές τιμές

Πεδία Δυφίων σε Δομές (Struct Bit Fields)

Ένα πεδίο δομής μπορεί να οριστεί έτσι ώστε να έχει ως μέγεθος στην μνήμη ένα συγκεκριμένο αριθμό από bits. Συγκεκριμένα, ακολουθούμε την σύνταξη:

```
struct student_status {
  int registered : 1;
  int year : 3;
  int grade : 4;
};
printf("%zu\n", sizeof(struct student_status));
```

Πεδία Δυφίων σε Δομές (Struct Bit Fields)

Ένα πεδίο δομής μπορεί να οριστεί έτσι ώστε να έχει ως μέγεθος στην μνήμη ένα συγκεκριμένο αριθμό από bits. Συγκεκριμένα, ακολουθούμε την σύνταξη:

```
struct student_status {
  int registered : 1;
  int year : 3;
  int grade : 4;
};
printf("%zu\n", sizeof(struct student_status));
```

```
$ ./bitfields
4
```

Πεδία Δυφίων σε Δομές (Struct Bit Fields)

Ένα πεδίο δομής μπορεί να οριστεί έτσι ώστε να έχει ως μέγεθος στην μνήμη ένα συγκεκριμένο αριθμό από bits. Συγκεκριμένα, ακολουθούμε την σύνταξη:

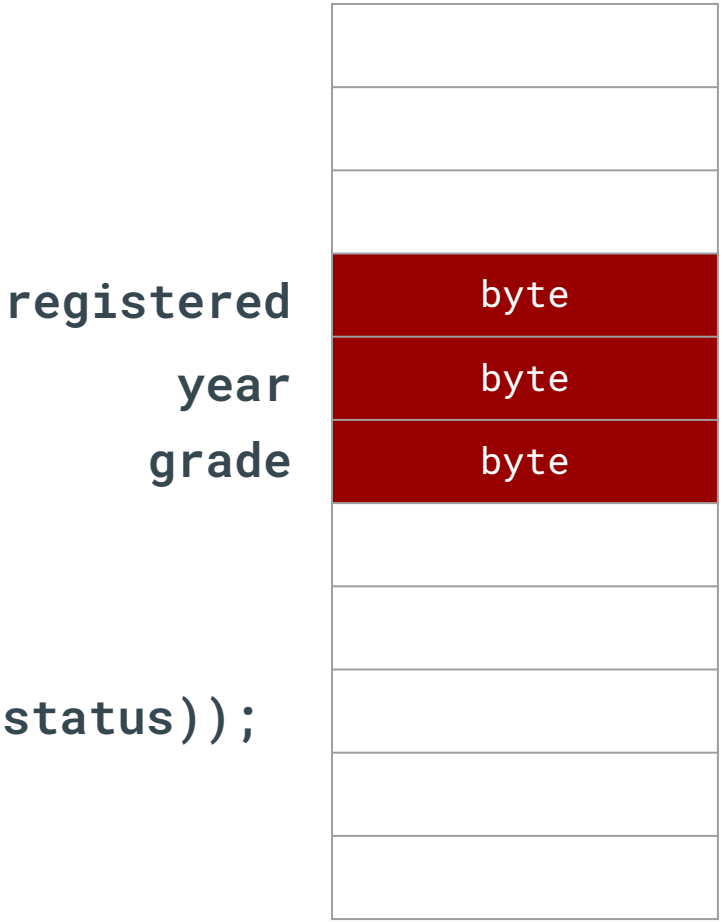
```
struct student_status {
  char registered : 1;
  char year : 3;
  char grade : 4;
};
printf("%zu\n", sizeof(struct student_status));
```

```
$ ./bitfields2
1
```

Bit Fields - Αναπαράσταση

```
struct student_status {
  char registered;
  char year;
  char grade;
};
printf("%zu\n", sizeof(struct student_status));
```

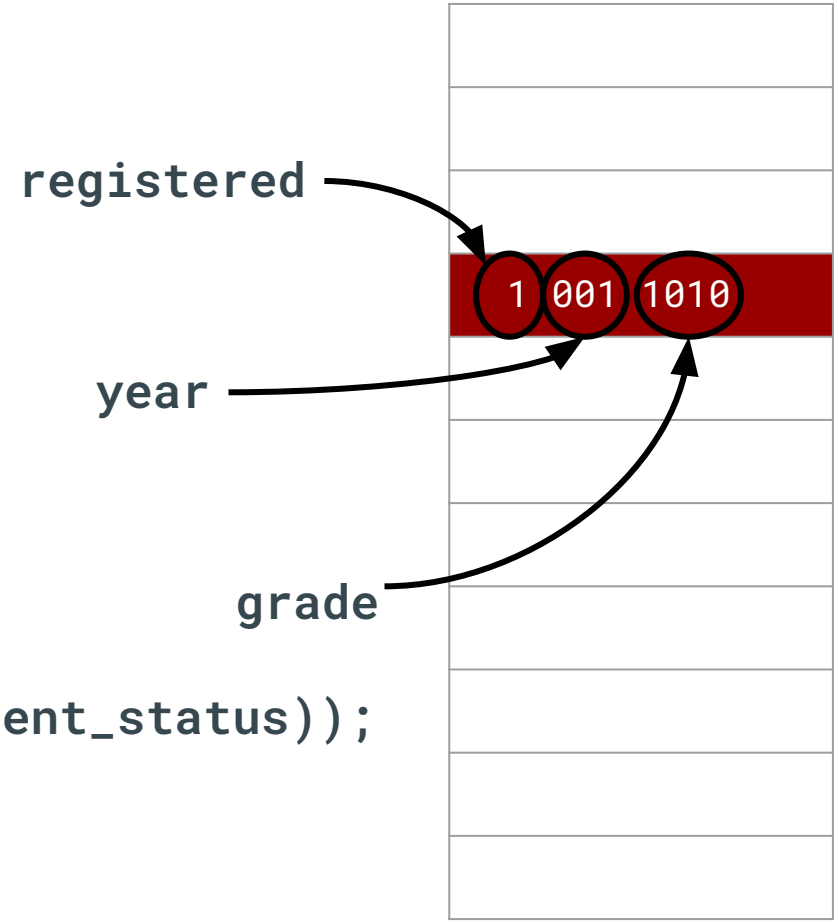
```
$ ./bitfields2
3
```



Bit Fields - Αναπαράσταση

```
struct student_status {
  char registered : 1;
  char year : 3;
  char grade : 4;
};
printf("%zu\n", sizeof(struct student_status));
```

```
$ ./bitfields2
1
```



Bit Fields - Ανάθεση

```c
#include <stdio.h>

typedef struct {
  unsigned char registered : 1;
  unsigned char year : 3;
  unsigned char grade : 4;
} status;

int main() {
  status st = {1, 1, 10};
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year = 2;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year += 7;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

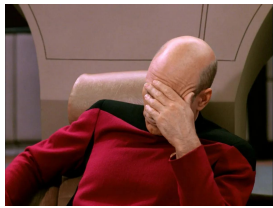
Bit Fields - Ανάθεση

```c
#include <stdio.h>

typedef struct {
  unsigned char registered : 1;
  unsigned char year : 3;
  unsigned char grade : 4;
} status;

int main() {
  status st = {1, 1, 10};
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year = 2;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year += 7;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

```
$ ./bitfields3
Status: 1 1 10
Status: 1 2 10
Status: 1 1 10
```

2 + 7 = 1??



Bit Fields - Ανάθεση

```c
#include <stdio.h>

typedef struct {
  unsigned char registered : 1;
  unsigned char year : 3;
  unsigned char grade : 4;
} status;

int main() {
  status st = {1, 1, 10};
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year = 2;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year += 7;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

```
$ ./bitfields3
Status: 1 1 10
Status: 1 2 10
Status: 1 1 10
```

2 + 7 % 8 = 1 % 8
Το **year** έχει μόλις **3 bits** - αν προσπαθήσουμε να αποθηκεύσουμε μεγαλύτερες τιμές του 7 (2^3 - 1) θα υποστούμε τις συνέπειες (aka Προσοχή)

Περιορισμοί Bit Fields

1. Δεν μπορούμε να ζητήσουμε το μέγεθος (sizeof αποτυγχάνει)

```
bitfields.c:14:25: error: ‘sizeof’ applied to a bit-field
   14 |   printf("%zu\n", sizeof(st.year));
```

2. Δεν μπορούμε να πάρουμε την διεύθυνση ενός bit-field

```
bitfields.c:14:23: error: cannot take address of bit-field ‘year’
   14 |   unsigned char * c = &st.year;
```

3. Μπορεί να οδηγήσει σε προβλήματα απόδοσης, δυσκολία ανάγνωσης, ή προβλήματα συμβατότητας. Αποφεύγουμε ή κάνουμε χρήση με φειδώ.

Ενώσεις (Unions)

Η ένωση (union) στην C μοιάζει με μία δομή (struct) με μία διαφορά: τα πεδία της ένωσης μοιράζονται την ίδια μνήμη. Η κοινή μνήμη επιτρέπει εξοικονόμηση χώρου.

Για μία μεταβλητή τύπου union επομένως, συνήθως έχει νόημα να χρησιμοποιούμε μόνο ένα από τα πεδία της.

Δήλωση Ένωσης (Union Declaration)

Η ένωση (union) στην C μοιάζει με μία δομή (struct) με μία διαφορά: τα πεδία της ένωσης μοιράζονται την ίδια μνήμη. Γενική σύνταξη:

```
union όνομα {
 τύπος1 πεδίο1;
 τύπος2 πεδίο2;
 τύπος3 πεδίο3;
 ...
};
```

Όμοια με την δήλωση struct απλά χρησιμοποιεί το keyword union

Δήλωση Ένωσης (Union Declaration)

Η ένωση (union) στην C μοιάζει με μία δομή (struct) με μία διαφορά: τα πεδία της ένωσης μοιράζονται την ίδια μνήμη. Γενική σύνταξη:

```
union anything {
  char c;
  int i;
  float f;
  double d;
};
```

Χρήση Ένωσης και Μέγεθος

```c
#include <stdio.h>
union anything {
  char c;  int i;  float f;  double d;
};
int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
  printf("%zu\n", sizeof(a1));
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

Χρήση Ένωσης και Μέγεθος

```c
#include <stdio.h>
union anything {
  char c;  int i;  float f;  double d;
};
int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
  printf("%zu\n", sizeof(a1));
  return 0;
}
```

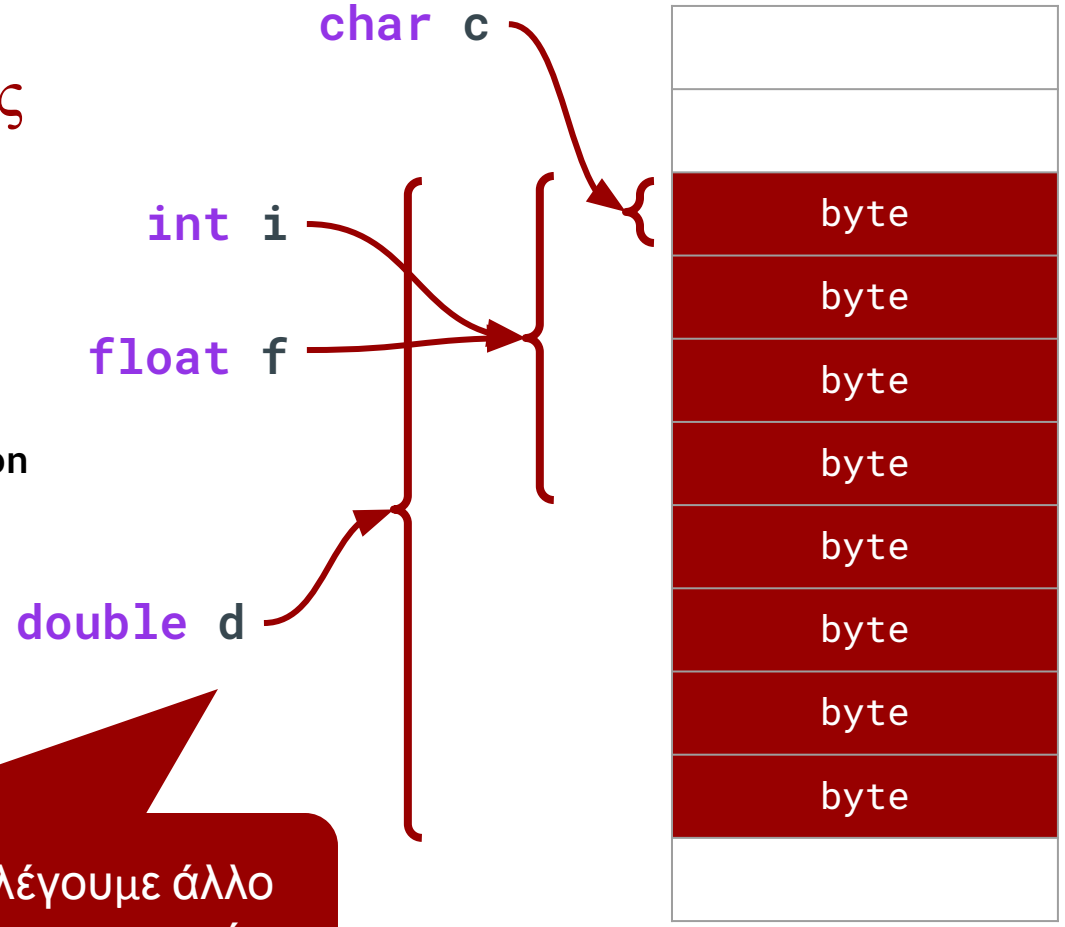
Τι θα τυπώσει το πρόγραμμα;

```
$ ./union
66 B
67 C
8
```

Χρήση Ένωσης και Μέγεθος

```c
#include <stdio.h>
union anything {
  char c;  int i;  float f;  double d;
};
int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
```

```
$ ./union
66 B
67 C
8
```

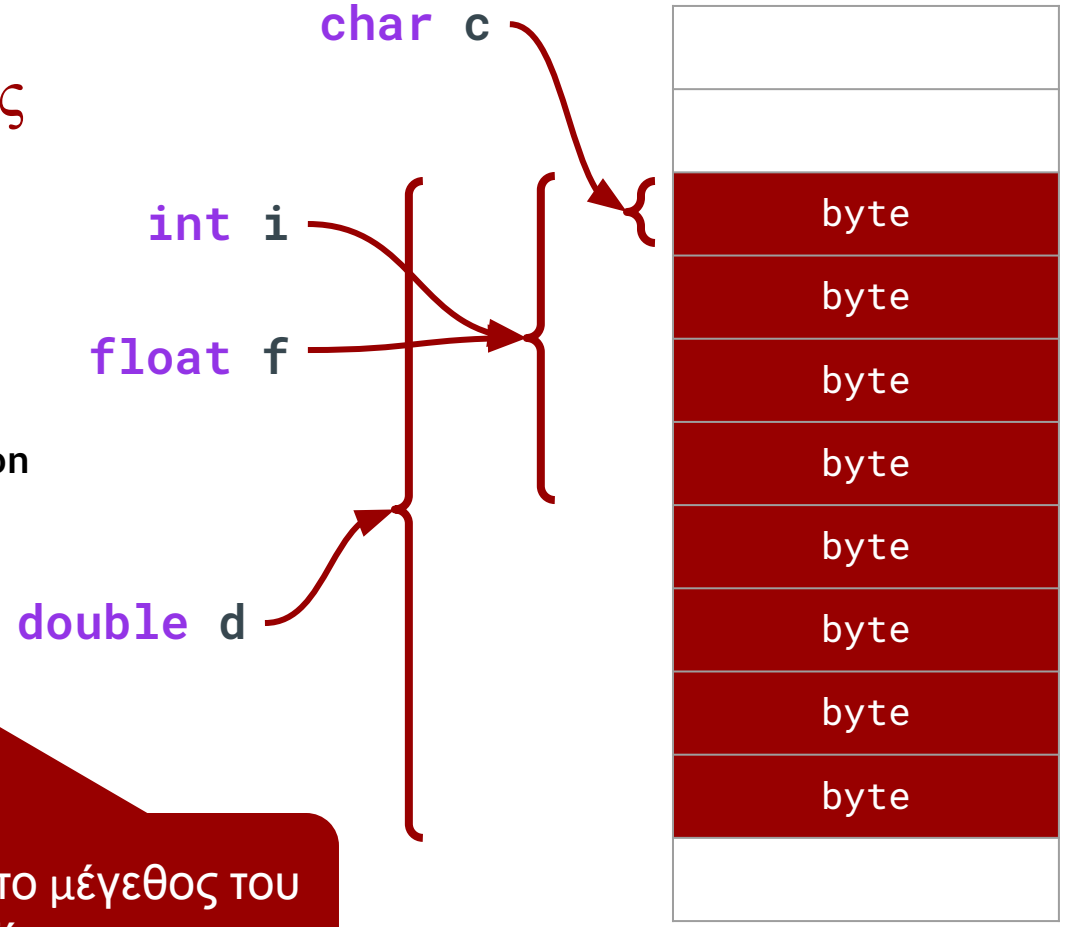


Διαλέγοντας διαφορετικό πεδίο, διαλέγουμε άλλο τρόπο αναπαράστασης των ίδιων bytes στην μνήμη

Χρήση Ένωσης και Μέγεθος

```c
#include <stdio.h>
union anything {
  char c;  int i;  float f;  double d;
};
int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
```

```
$ ./union
66 B
67 C
8
```



Το μέγεθος της ένωσης ταυτίζεται με το μέγεθος του τύπου του "μεγαλύτερου" πεδίου της

Απαριθμήσεις (Enumerations)

Ο τύπος απαρίθμησης `enum` (enumeration type) μας επιτρέπει να ορίζουμε ένα σύνολο ακεραίων με συγκεκριμένα ονόματα και σταθερές τιμές.

Η πρώτη σταθερά αρχικοποιείται στο 0 (εκτός αν της δοθεί συγκεκριμένη τιμή).

Κάθε σταθερά στην οποία δεν δίνουμε τιμή παίρνει την τιμή της προηγούμενης σταθεράς αυξημένη κατά 1.

Δήλωση Απαρίθμησης (Enum Declaration)

Ο τύπος απαρίθμησης enum (enumeration type) μας επιτρέπει να ορίζουμε ένα σύνολο ακεραίων με συγκεκριμένα ονόματα και σταθερές τιμές. Σύνταξη:

```
enum όνομα { επιλογή1, επιλογή2, ... };
```

Όνομα της απαρίθμησης

Σταθερές που αποτελούν την απαρίθμηση

Παράδειγμα:

```
enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

Χρήση Απαρίθμησης (Enum Usage)

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>
enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
int main() {
  enum weekday day1 = Mon, day2;
  day2 = Fri;
  printf("%d %d\n", day1, day2);
  return 0;
}
```

Χρήση Απαρίθμησης (Enum Usage)

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>
enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
int main() {
  enum weekday day1 = Mon, day2;
  day2 = Fri;
  printf("%d %d\n", day1, day2);
  return 0;
}
```

```
$ ./enum1
0 4
```

Χρήση Απαρίθμησης (Enum Usage) #2

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>
enum weekday {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
int main() {
  for(enum weekday day = Mon; day <= Sun; day++) {
    if(day == Mon || day == Fri)
      printf("We have class on day # %d of the week\n", day);
  }
  return 0;
}
```

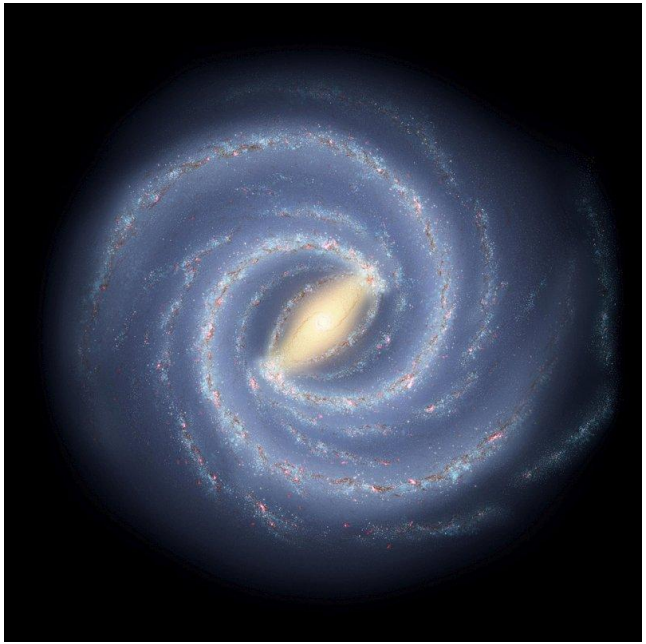
Χρήση Απαρίθμησης (Enum Usage) #2

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>
enum weekday {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
int main() {
  for(enum weekday day = Mon; day <= Sun; day++) {
    if(day == Mon || day == Fri)
      printf("We have class on day # %d of the week\n", day);
  }
  return 0;
}
```

```
$ ./enum2
We have class on day # 1 of the week
We have class on day # 5 of the week
```

Αυτοαναφορά (Self-Reference)

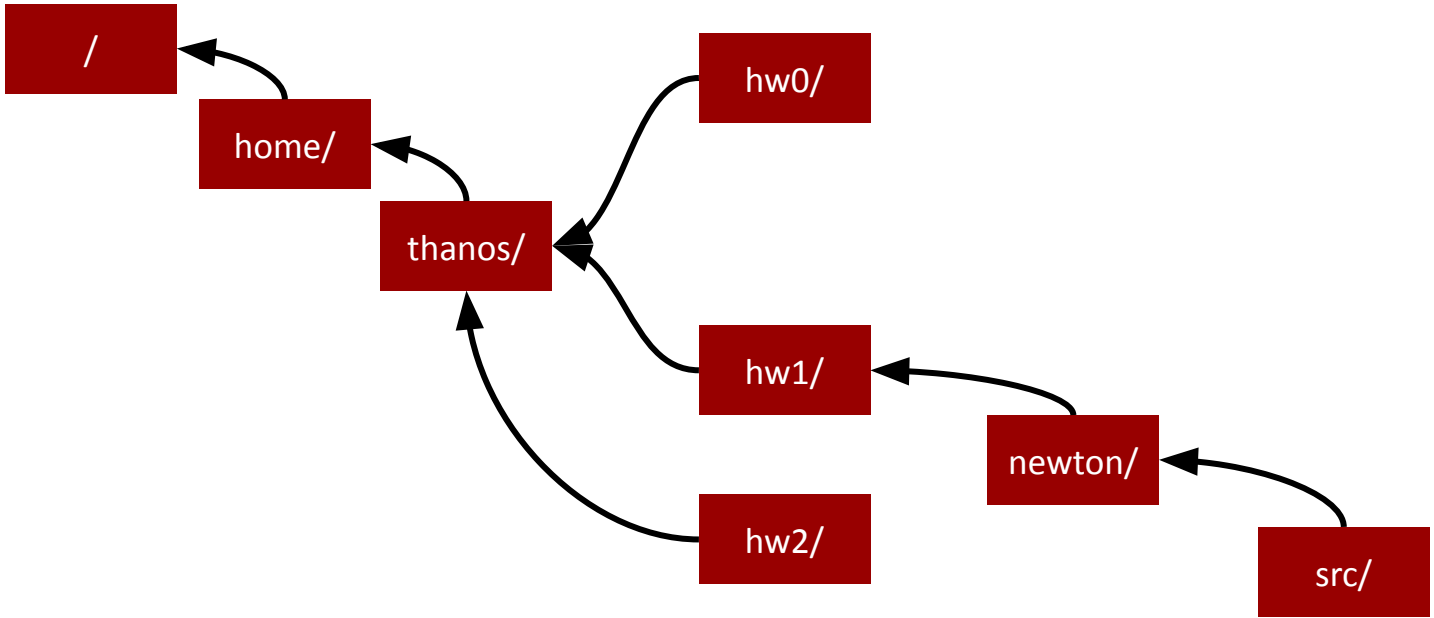


Αυτή η πρόταση είναι ψευδής

(Επιμενίδης, 6ος αιώνας π.Χ.)

Θέλω να φτιάξω μια δομή (struct) που να αναπαριστά έναν φάκελο σε ένα σύστημα αρχείων. Πως θα το κάνω;

Κάθε φάκελος έχει ένα όνομα και βρίσκεται μέσα σε κάποιον άλλο (parent) φάκελο. Ο αρχικός φάκελος (root /) δεν βρίσκεται μέσα σε κάποιο φάκελο. #1 εκδοχή:



Αυτοαναφορικές Δομές (Self-Referential Structs)

Τα μέλη μιας δομής μπορεί να είναι οποιουδήποτε τύπου, ακόμα και δείκτες του ίδιου τύπου. Για παράδειγμα:

```
struct folder {
  char name[128];
  struct folder * parent;
};
```

Κάθε φάκελος έχει ένα όνομα

Κάθε φάκελος έχει ένα δείκτη στον `parent` φάκελο

Χρήση Αυτοαναφορικών Δομών

```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder * parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  printf("%s -> %s -> %s\n", newton.name, newton.parent->name, newton.parent->parent->name);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

Χρήση Αυτοαναφορικών Δομών

```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder * parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  printf("%s -> %s -> %s\n", newton.name, newton.parent->name, newton.parent->parent->name);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

```
$ ./self
newton/ -> hw0/ -> thanos/
```

Χρήση Αυτοαναφορικών Δομών #2

```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder * parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  for(Folder * iterator = &newton; iterator; iterator = iterator->parent)
    printf("folder: %s\n", iterator->name);
  return 0;
}
```

Τι θα τυπώσει το πρόγραμμα;

Χρήση Αυτοαναφορικών Δομών #2

```c
#include <stdio.h>

typedef struct folder {

  char name[128];

  struct folder * parent;

} Folder;

int main() {

  Folder root = {"/", NULL};

  Folder home = {"home/", &root};

  Folder user = {"thanos/", &home};

  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};

  Folder newton = {"newton/", &hw0};

  for(Folder * iterator = &newton; iterator; iterator = iterator->parent)

    printf("folder: %s\n", iterator->name);

  return 0;

}
```

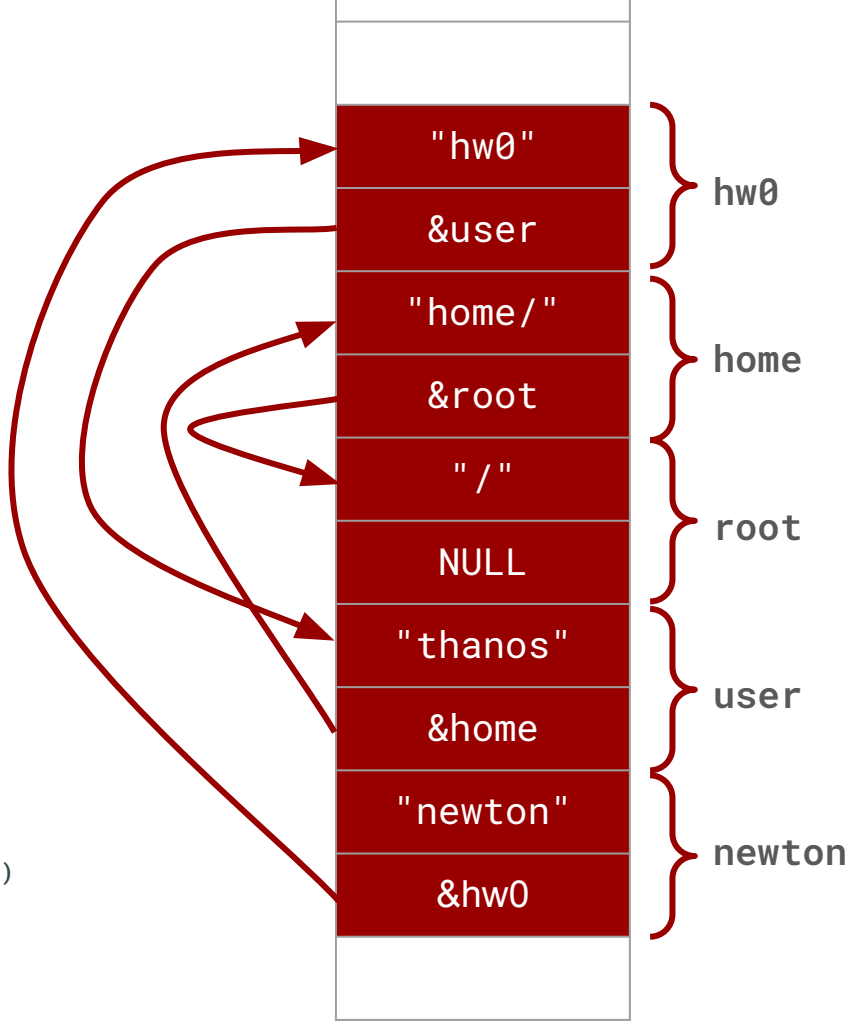
Τι θα τυπώσει το πρόγραμμα;

```
$ ./self2
folder: newton/
folder: hw0/
folder: thanos/
folder: home/
folder: /
```

Αυτοαναφορικές Δομές - Μνήμη

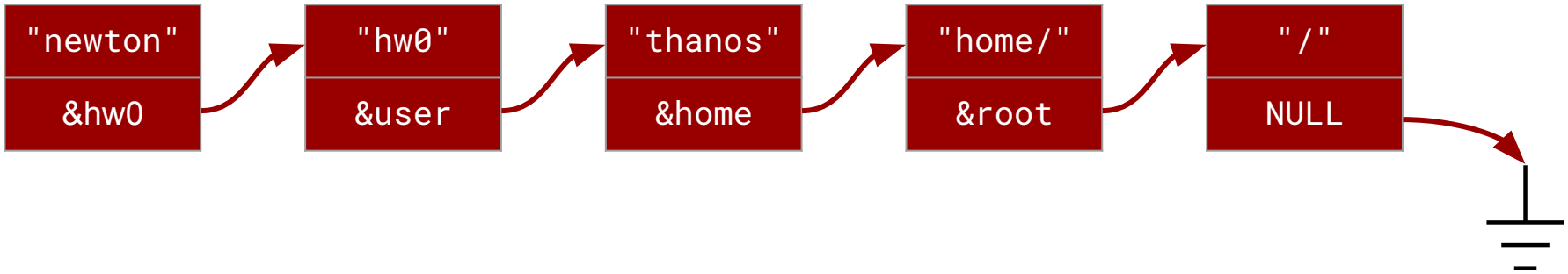
```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder * parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  for(Folder * iterator = &newton; iterator; iterator = iterator->parent)
    printf("folder: %s\n", iterator->name);
  return 0;
}
```

```
$ ./self2
folder: newton/
folder: hw0/
folder: thanos/
folder: home/
folder: /
```



Σε πιο αφηρημένη (abstract) μορφή

Η πραγματική διάταξη (layout) στην μνήμη μπορεί να είναι περίπλοκη. Σε αφηρημένη μορφή είναι κάπως έτσι:



Μας θυμίζει κάτι αυτή η διάταξη;

Απλά Συνδεδεμένη Λίστα (Single Linked List)

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

Μοιάζει με πίνακα αλλά έχει βασικές διαφορές (next time).

Συνήθως αποθηκεύεται εξ'ολοκλήρου δυναμικά (στον σωρό).

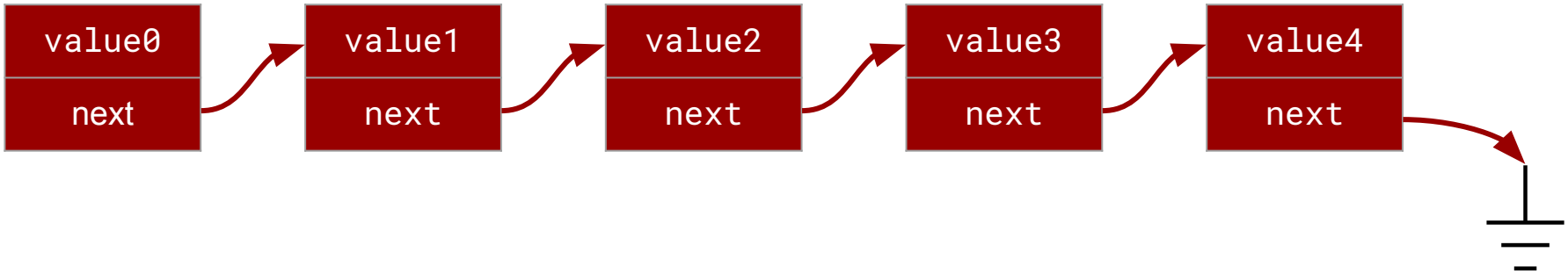
Απλά Συνδεδεμένη Λίστα (Single Linked List)

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.



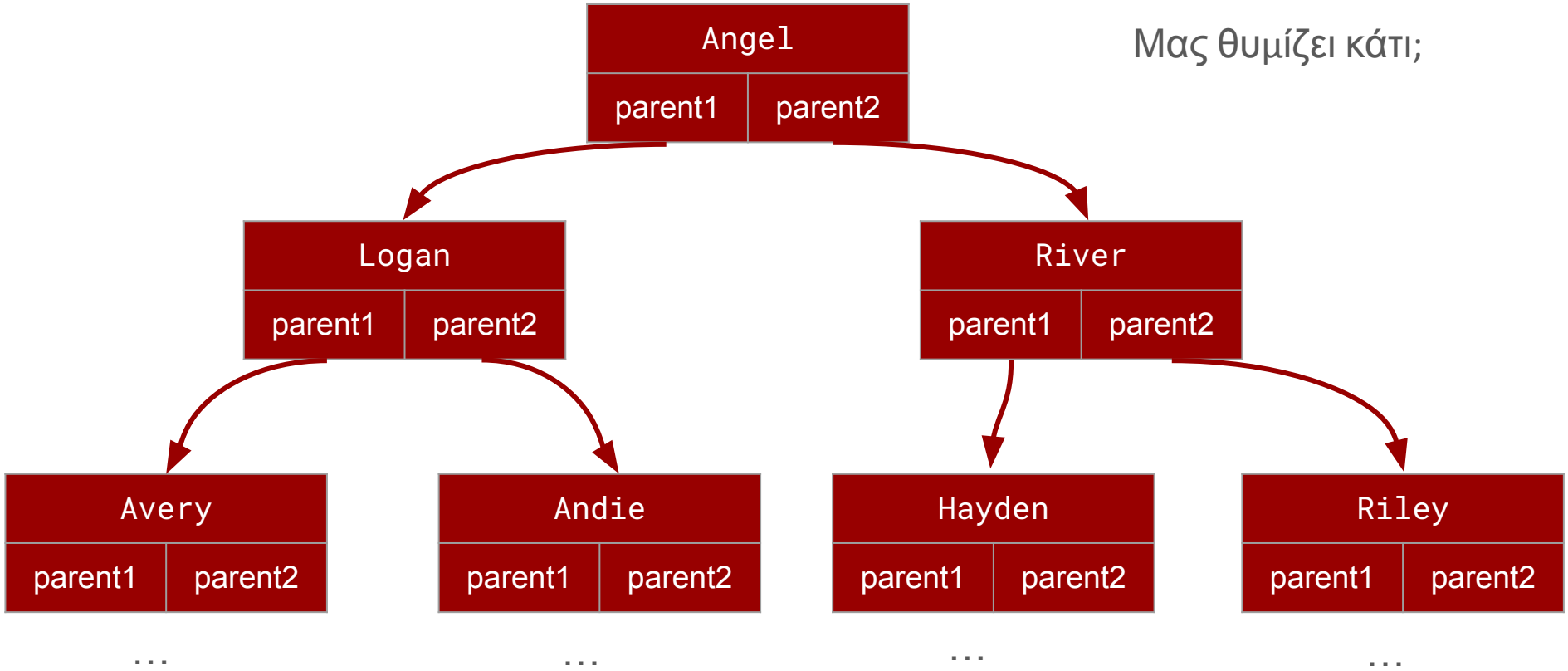
Μήκος λίστας (list length): ο αριθμός των στοιχείων που περιέχει (5 παραπάνω)

Αυτοαναφορικές Δομές (Self-Referential Structs)

Τα μέλη μιας δομής μπορεί να είναι οποιουδήποτε τύπου, ακόμα και **πολλοί** δείκτες του ίδιου τύπου. Για παράδειγμα:

```
struct person {
  char name[128];
  struct person * parent1;
  struct person * parent2;
};
```

Αυτοαναφορικές Δομές (Self-Referential Structs)



Μας θυμίζει κάτι;

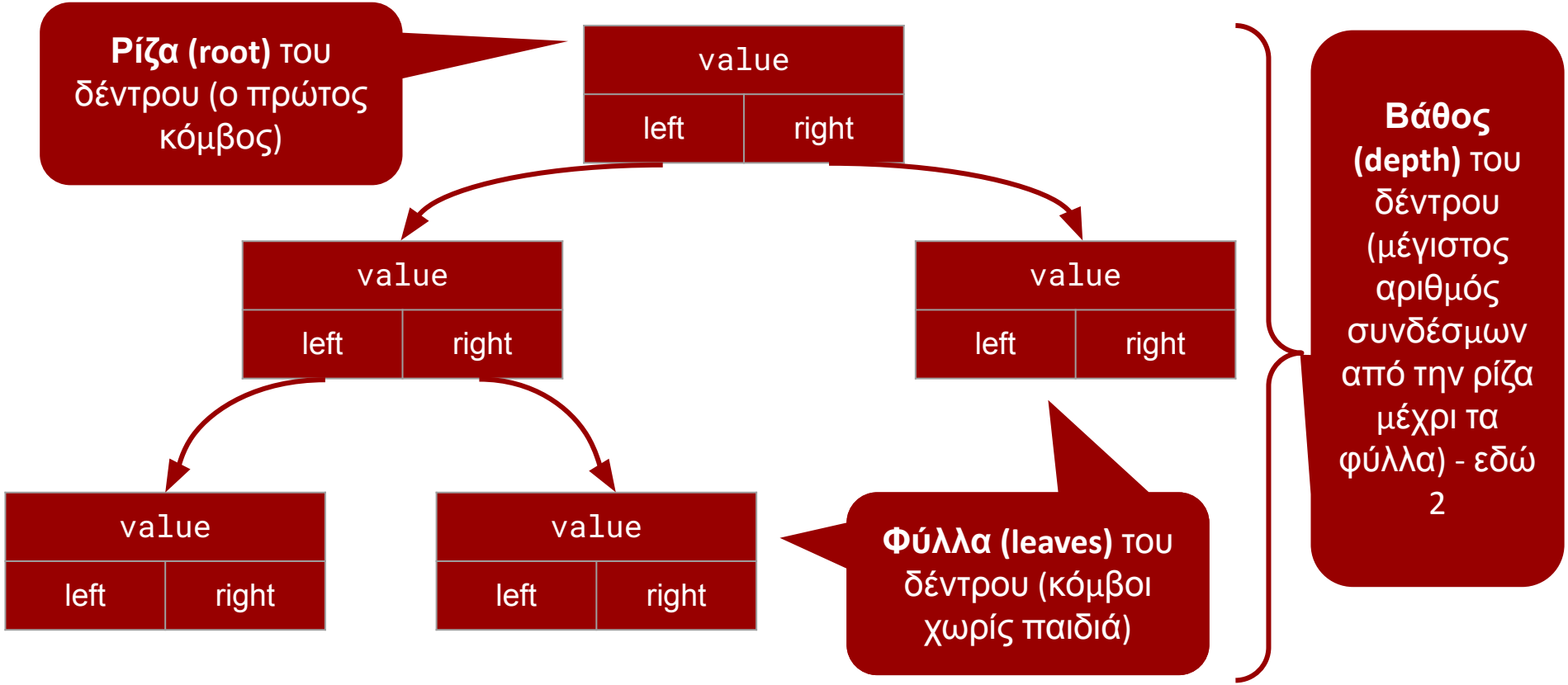
Δυαδικό Δέντρο (Binary Tree)

Το δυαδικό δέντρο (binary tree) είναι ένας τύπος δεδομένων που μας επιτρέπει να οργανώνουμε τα δεδομένα μας σε δενδρική διάταξη, καθένας από τους κόμβους του δέντρου μπορεί να έχει από 0 μέχρι 2 κόμβους-παιδιά.

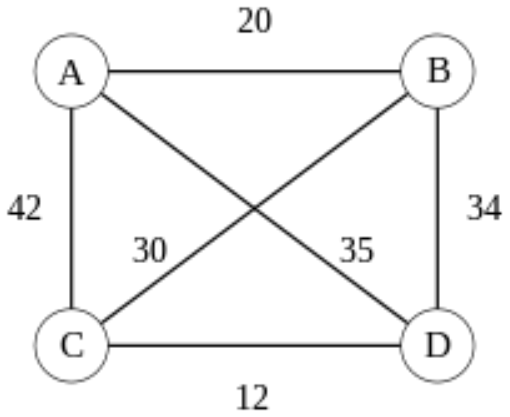
```
struct treenode {
  int value;
  struct treenode * left;
  struct treenode * right;
};
```

Εφαρμογές: από βάσεις δεδομένων/αναζήτηση μέχρι μεταγλωττιστές και από συμπίεση δεδομένων μέχρι κρυπτογραφία (όπου απαιτείται αναπαράσταση γνώσης)

Δυαδικά Δέντρα (Binary Trees)



Θέλω να αναπαραστήσω ένα χάρτη σε μορφή γράφου με αυτοαναφορικές δομές. Πως;



Θέλω να αναπαραστήσω ένα σύστημα αρχείων (filesystem) ώστε να βρίσκω άμεσα τους υποφακέλους και τον parent φάκελο. Πως;

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου προτείνω να διαβάσετε τις σελίδες 108, 120-135

- Bit fields και reference
- Unions
- Enums
- Linked List
- Binary Tree

Ευχαριστώ και καλό σαβκο εύχομαι!
Keep Coding ;)