

# Διάλεξη 19 - Δομές

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

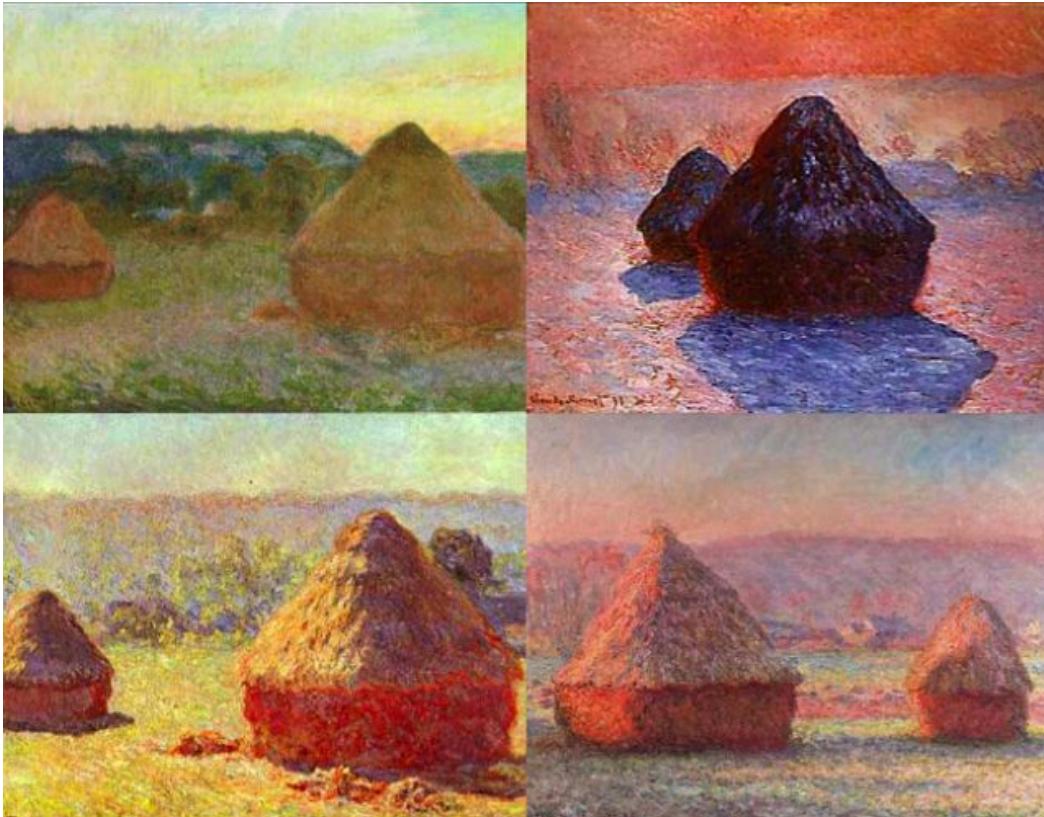
Θανάσης Αυγερινός / Τάκης Σταματόπουλος

## Ανακοινώσεις / Διευκρινίσεις

- Από μετεγγραφή / νιώθεις πως έχεις μείνει πίσω στα εργαστήρια;  
Φροντιστηριακό μάθημα στο Linux Lab 9-11πμ την Τρίτη 9 Δεκεμβρίου!
- Δεν θα κρατήσω ώρες γραφείου σήμερα

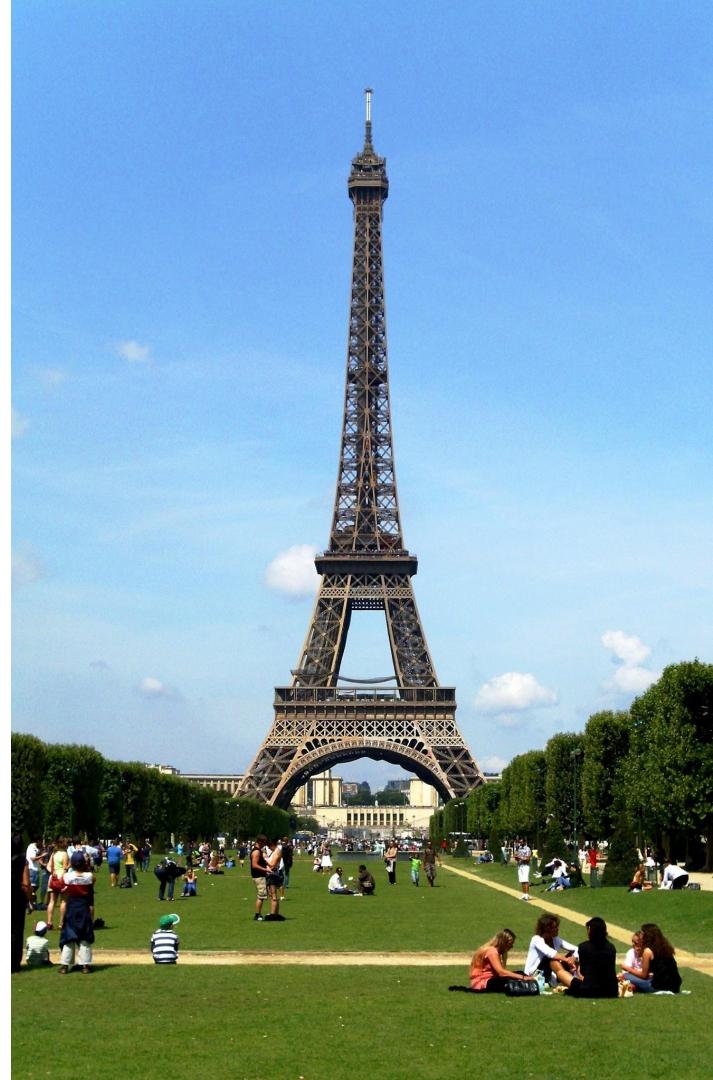
# Την Προηγούμενη Φορά

- Αλγόριθμοι Ταξινόμησης  
(Sorting Algorithms)
- Δεδομένα Εισόδου #2
  - Λίγο παραπάνω για την `scanf`
  - Αρχεία (Files) και ρεύματα δεδομένων (data streams)
  - Συναρτήσεις χειρισμού αρχείων



# Σήμερα

- Δομές
- Επίλυση Προβλημάτων



# Εισαγωγή στις Δομές

Μέχρι στιγμής είδαμε βασικούς τύπους

`int`, `double`, `char`, δείκτες, πίνακες

όμως τα δεδομένα μπορεί να είναι πιο δομημένα/περίπλοκα από έναν πίνακα ακεραίων. Τι κάνουμε για αυτά;

# Δομή (Struct)

Δομή (struct) (ή αλλιώς εγγραφή / record) στην C λέγεται μια συλλογή πεδίων που συνήθως χρησιμοποιούνται για την ομαδοποίηση πληροφορίας που περιγράφει μια λογική οντότητα.

Για παράδειγμα, έστω ότι θες να γράψεις ένα πρόγραμμα διαχείρισης μιας λίστας φοιτητών. Θα χρειαστείς μεταβλητές για τα ακόλουθα:

```
char first_name[128];  
char last_name[128];  
unsigned int year;  
double grade;
```

Έστω ότι έχουμε 100 φοιτητές, τι θα κάνουμε για να τους αναπαραστήσουμε στο πρόγραμμά μας;

# Δήλωση Δομής (Struct Declaration)

Η δήλωση δομής επιτρέπει να δημιουργούμε τους **δικούς μας** (user-defined) τύπους μεταβλητών που μπορούν να έχουν μια συλλογή από τα πεδία που επιθυμούμε.

**struct** όνομα {

τύπος1 πεδίο1;

τύπος2 πεδίο2;

τύπος3 πεδίο3;

...

} ;

To keyword  
struct  
υποδηλώνει  
ότι ορίζουμε  
μια δομή

To όνομα (ή αλλιώς ετικέτα / tag) της δομής μας επιτρέπει να αναφερόμαστε σε αυτήν



Τα πεδία / fields της δομής περιέχουν ορισμούς τύπων που αποτελούν την δομή

# Δήλωση Τύπου Δομής (Struct Type Declaration)

Η δήλωση δομής επιτρέπει να δημιουργούμε τους **δικούς μας** (user-defined) τύπους μεταβλητών που μπορούν να έχουν μια συλλογή από τα πεδία που επιθυμούμε.

```
struct student {  
    char first_name[128];  
    char last_name[128];  
    unsigned int year;  
    double grade;  
};
```

To keyword  
struct  
υποδηλώνει  
ότι ορίζουμε  
μια δομή

Το όνομα (ή αλλιώς ετικέτα / tag) της δομής μας επιτρέπει να αναφερόμαστε σε αυτήν

Τα πεδία / fields της δομής περιέχουν ορισμούς τύπων που αποτελούν την δομή

# Δήλωση Μεταβλητής Τύπου Δομής

Δηλώνουμε μια μεταβλητή με τύπο δομής ως εξής:

```
struct όνομα_δομής όνομα_μεταβλητής;
```

Για παράδειγμα:

```
struct student st1, st2;
```

Δύο μεταβλητές st1, st2  
ΤΥΠΟΥ struct student

```
struct student *student_ptr;
```

Δείκτης σε δομή τύπου struct  
student

```
struct student student_array[100];
```

Πίνακας με 100 δομές τύπου  
struct student

# Προσπέλαση Πεδίων Δομής (Struct Field Access)

Για να προσπελάσουμε το πεδίο μιας δομής χρησιμοποιούμε την σύνταξη:

**όνομα\_μεταβλητής\_τύπου\_δομής.όνομα\_πεδίου**

Για παράδειγμα:

**st1.year**

Αναφέρεται στο ακέραιο  
πεδίο **year** της δομής  
student που αντιστοιχεί στην  
μεταβλητή st1

Το πεδίο μπορεί να χρησιμοποιηθεί όπως μια μεταβλητή στην C

# Ανάθεση και Χρήση Πεδίων (Field Assignment and Usage)

```
int main() {
    struct student st1;
    st1.year = 1;
    st1.grade = 7.54;
    strncpy(st1.first_name, "Thanos", sizeof(st1.first_name) - 1);
    st1.first_name[sizeof(st1.first_name) - 1] = '\0';
    strncpy(st1.last_name, "Barbounis", sizeof(st1.last_name) - 1);
    st1.last_name[sizeof(st1.last_name) - 1] = '\0';
    printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade, st1.year);
    return 0;
}
```

Τι τυπώνει αυτό το πρόγραμμα;

# Ανάθεση και Χρήση Πεδίων (Field Assignment and Usage)

```
int main() {  
    struct student st1;  
  
    st1.year = 1;  
    st1.grade = 7.54;  
  
    strncpy(st1.first_name, "Thanos", sizeof(st1.first_name) - 1);  
    st1.first_name[sizeof(st1.first_name) - 1] = '\0';  
  
    strncpy(st1.last_name, "Barbounis", sizeof(st1.last_name) - 1);  
    st1.last_name[sizeof(st1.last_name) - 1] = '\0';  
  
    printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade, st1.year);  
  
    return 0;  
}
```

Τι τυπώνει αυτό το πρόγραμμα;

Ανάθεση τιμών στα πεδία  
της δομής (integer, double,  
char[128])

Τύπωμα των τιμών των  
πεδίων της δομής

\$ ./struct  
Thanos Barbounis: 7.540000 [1 year]

# Αρχικοποίηση Δομής (Struct Initialization)

Χρησιμοποιώντας σύνταξη παρόμοια με την αρχικοποίηση πινάκων μπορούμε να αρχικοποιήσουμε δομές:

```
struct student st1;

struct student st1 = {
    "Thanos",
    "Barbounis",
    1,
    7.54
};

printf("%s %s: %f  [%u year]\n", st1.first_name, st1.last_name, st1.grade,
st1.year);
```

# Αρχικοποίηση Δομής (Struct Initialization)

Χρησιμοποιώντας σύνταξη παρόμοια με την αρχικοποίηση πινάκων μπορούμε να αρχικοποιήσουμε δομές:

```
struct student st1;

struct student st1 = {    Ιδια σειρά με την
    "Thanos",           δήλωση της δομής      struct student {
    "Barbounis",         ←→                         char first_name[128];
    1,                  ←→                         char last_name[128];
    7.54                ←→                         unsigned int year;
};

printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade,
st1.year);
```

```
$ ./struct
Thanos Barbounis: 7.540000 [1 year]
```

# Αρχικοποίηση Δομής (Struct Initialization)

Χρησιμοποιώντας σύνταξη παρόμοια με την αρχικοποίηση πινάκων μπορούμε να αρχικοποιήσουμε δομές:

```
struct student st1;

struct student st1 = {
    "Thanos",
    "Barbounis",
    1
};

printf("%s %s: %f  [%u year]\n", st1.first_name, st1.last_name, st1.grade,
st1.year);

$ ./struct
Thanos Barbounis: 0.000000 [1 year]
```

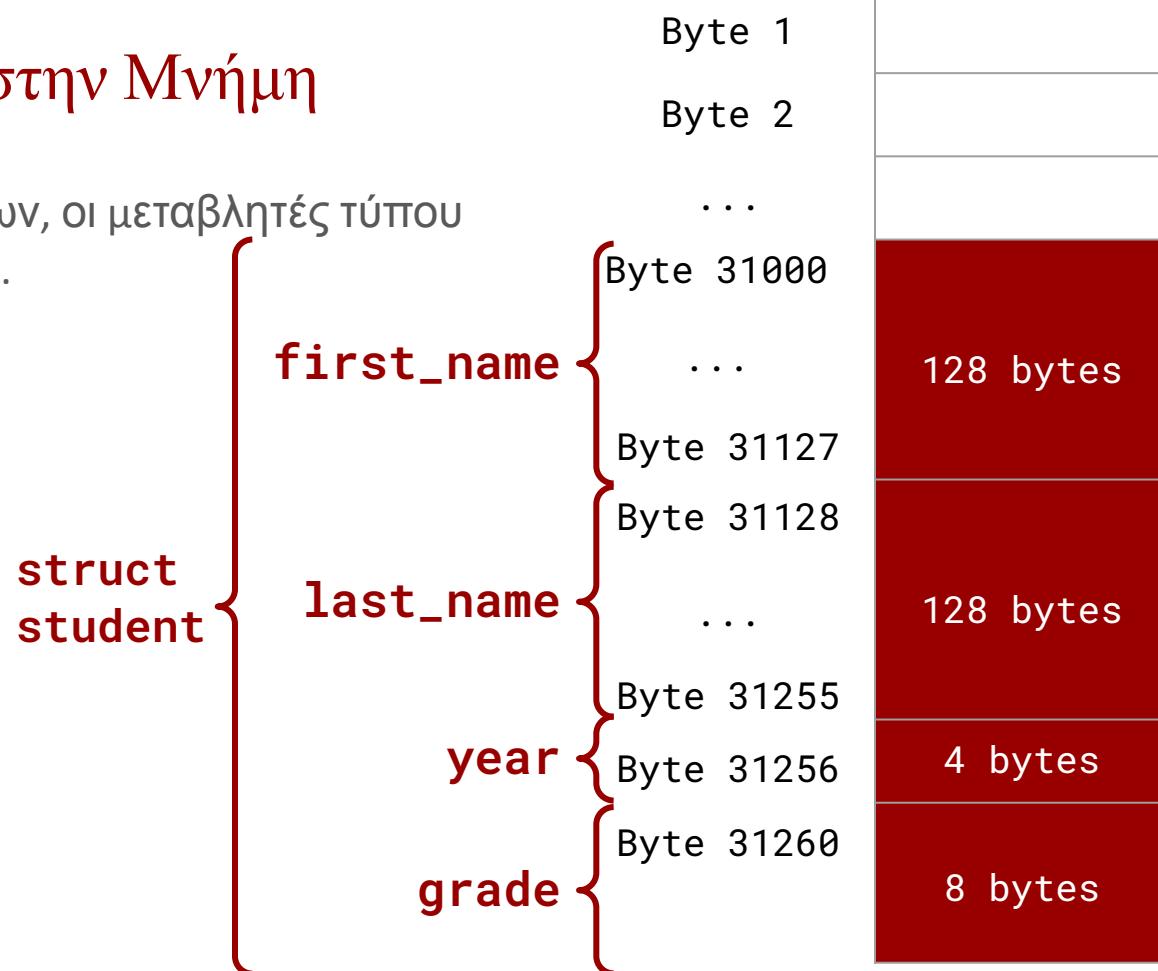


Αν παραλείψουμε μια τιμή αρχικοποιείται στο 0

# Αναπαράσταση Δομής στην Μνήμη

Όπως οι μεταβλητές άλλων τύπων, οι μεταβλητές τύπου δομής πιάνουν bytes στην μνήμη.

```
struct student {  
    char first_name[128];  
    char last_name[128];  
    unsigned int year;  
    double grade;  
} st1;
```

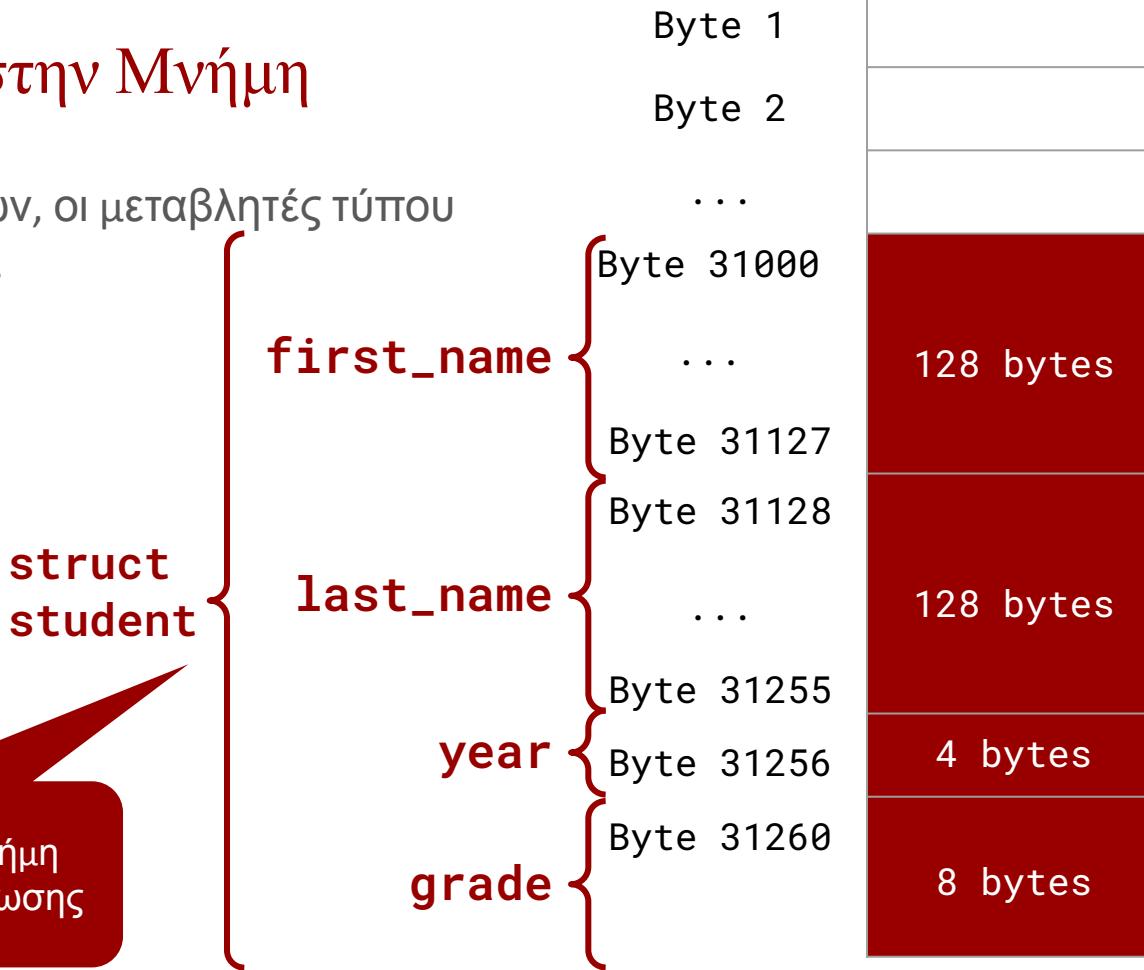


# Αναπαράσταση Δομής στην Μνήμη

Όπως οι μεταβλητές άλλων τύπων, οι μεταβλητές τύπου δομής πιάνουν bytes στην μνήμη.

```
struct student {  
    char first_name[128];  
    char last_name[128];  
    unsigned int year;  
    double grade;  
} st1;
```

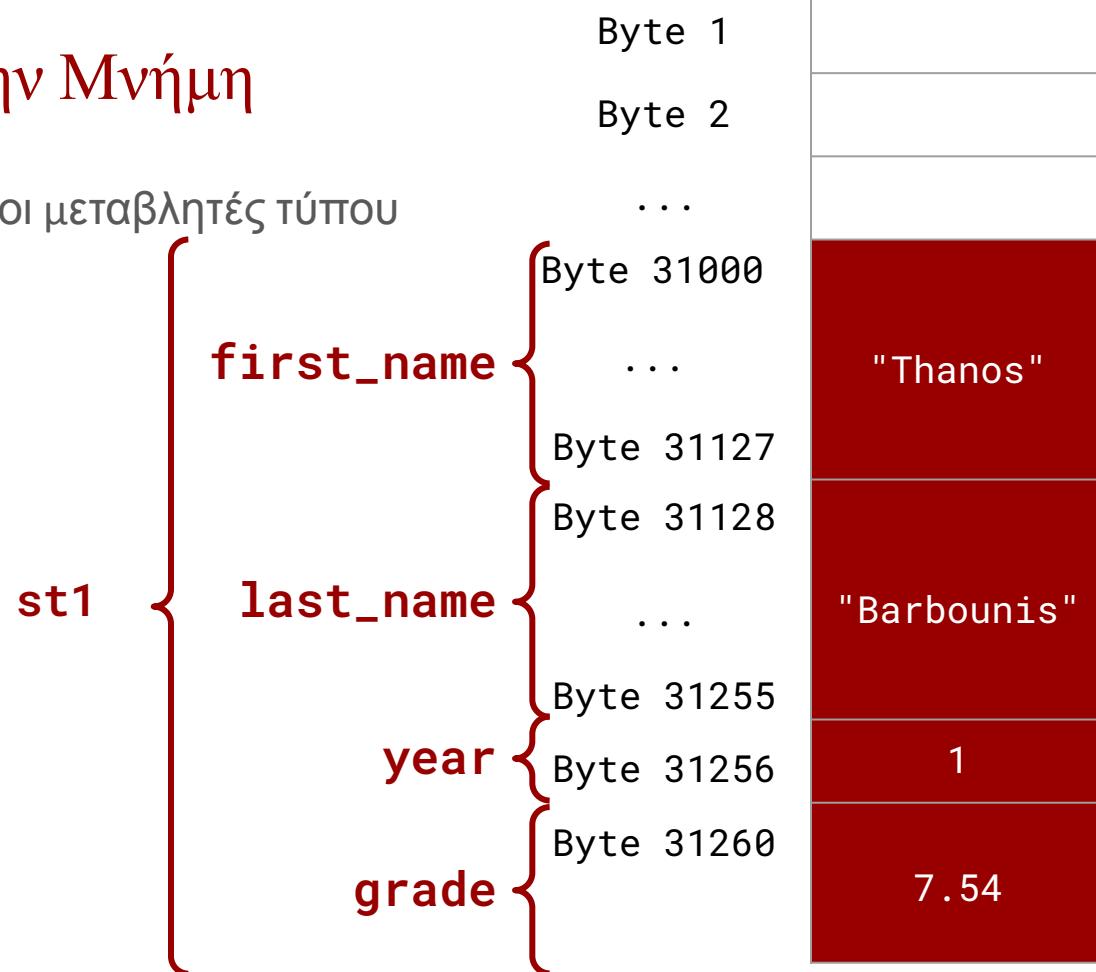
Σειρά πεδίων στην μνήμη  
όμοια με αυτήν της δήλωσης



# Αναπαράσταση Δομής στην Μνήμη

Όπως οι μεταβλητές άλλων τύπων, οι μεταβλητές τύπου δομής πιάνουν bytes στην μνήμη.

```
struct student st1 = {  
    "Thanos",  
    "Barbounis",  
    1,  
    7.54  
};
```

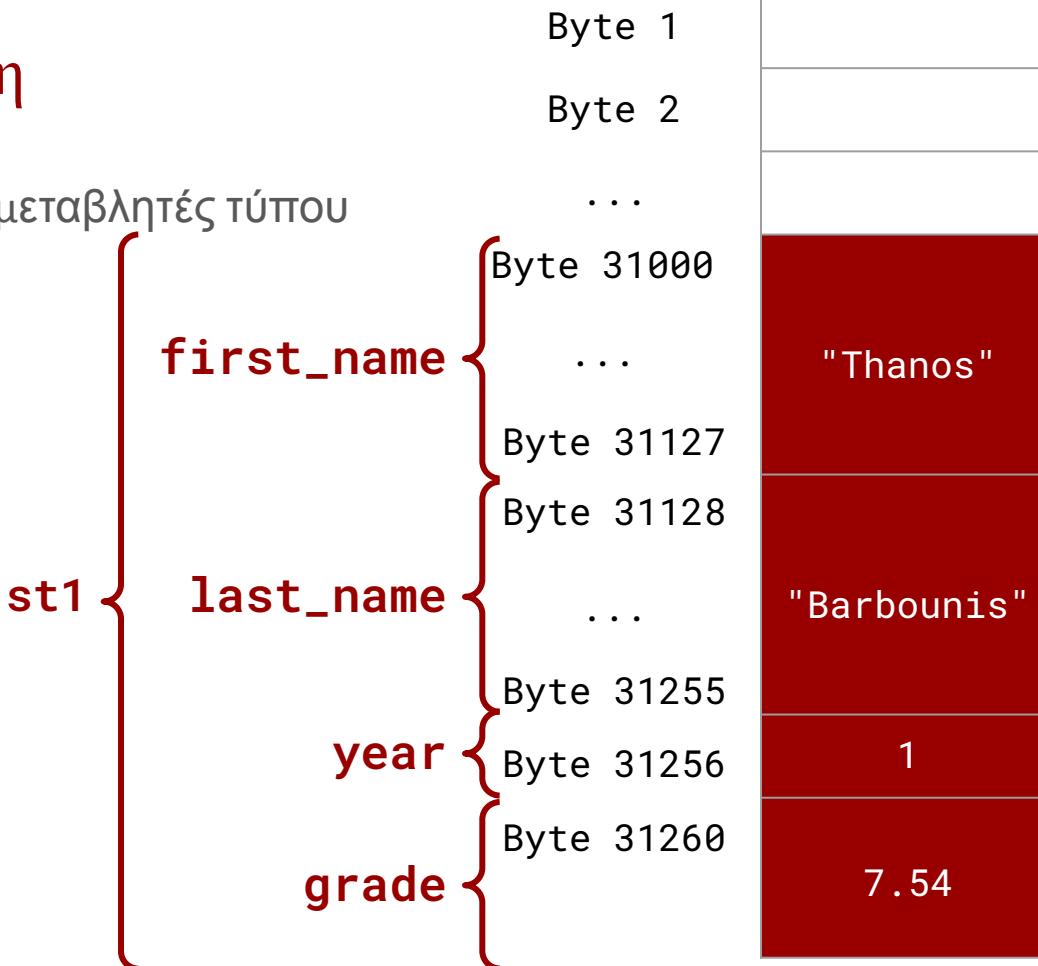


# Μέγεθος Δομής στην Μνήμη

Όπως οι μεταβλητές άλλων τύπων, οι μεταβλητές τύπου δομής πιάνουν bytes στην μνήμη.

Τι θα τυπώσει το ακόλουθο;

```
printf("%zu\n",
      sizeof(struct student));
```



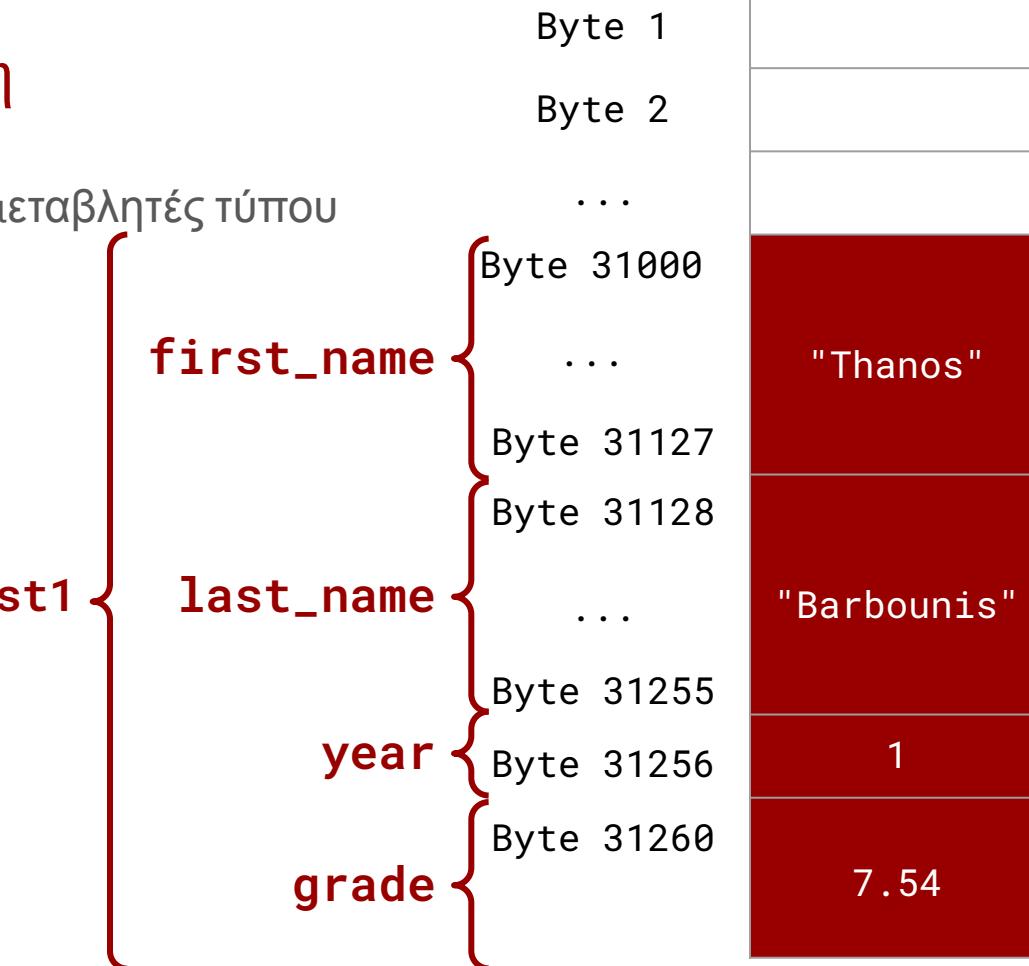
# Μέγεθος Δομής στην Μνήμη

Όπως οι μεταβλητές άλλων τύπων, οι μεταβλητές τύπου δομής πιάνουν bytes στην μνήμη.

Τι θα τυπώσει το ακόλουθο;

```
printf("%zu\n",
      sizeof(struct student));
```

```
$ gcc -m32 -o struct struct.c
$ ./struct
268
```



# Μέγεθος Δομής στην Μνήμη - Padding

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red;
        char green;
        char blue;
        int alpha;
    } pixel = {0xFF, 0xFF, 0xFF, 42};
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

# Μέγεθος Δομής στην Μνήμη - Padding

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red;
        char green;
        char blue;
        int alpha;
    } pixel = {0xFF, 0xFF, 0xFF, 42};
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

\$ gcc -m32 -o struct2 struct2.c  
\$ ./struct2  
8  
???

# Μέγεθος Δομής στην Μνήμη - Padding

Μόνο 7 από τα 8 bytes  
χρησιμοποιούνται

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red;
        char green;
        char blue;
        int alpha;
    } pixel = {0xFF, 0xFF, 0xFF, 42};
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

```
$ gcc -m32 -o struct2 struct2.c
$ ./struct2
8
```

red	0xFF
green	0xFF
blue	0xFF
padding	(κενό)
alpha	42

Ο μεταγλωτιστής μπορεί να αποφασίσει να προσθέσει padding ("κενά") που δεν χρησιμοποιείται προκειμένου οι διευθύνσεις των πεδίων να είναι πολλαπλάσιο του 4 (ή του 8 / sizeof(void\*)) για λόγους απόδοσης ([memory alignment](#))

# Μέγεθος Δομής στην Μνήμη - Padding

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red; int alpha;
        char green; int beta;
        char blue; int gamma;
    } pixel;
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

# Μέγεθος Δομής στην Μνήμη - Padding

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red; int alpha;
        char green; int beta;
        char blue; int gamma;
    } pixel;
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

```
$ gcc -m32 -o struct3 struct3.c
$ ./struct3
24
```

red

padding  
(κενό)

alpha

green

padding  
(κενό)

# Μέγεθος Δομής στην Μνήμη - Padding

```
#include <stdio.h>
int main() {
    struct pixel_tag {
        char red; int alpha;
        char green; int beta;
        char blue; int gamma;
    } pixel;
    printf("%zu\n", sizeof(pixel));
    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

```
$ gcc -m32 -o struct3 struct3.c
$ ./struct3
24
```

Μόνο 15 από τα 24 bytes  
χρησιμοποιούνται

...

# Συνοψίζοντας

Ποιο είναι το μέγεθος του **struct student**;

Ότι επιστρέψει το `sizeof(struct student)`

# Ανάθεση με Δομές

Για να αντιγραφούν τα περιεχόμενα μιας δομής σε μια άλλη, χρησιμοποιούμε τον τελεστή ανάθεσης:

```
#include <stdio.h>

struct point { int x; int y; };

int main() {
    struct point pt1 = { 3, 4 };

    struct point pt2;

    printf("%d %d\n", pt2.x, pt2.y);

    pt2 = pt1;

    printf("%d %d\n", pt2.x, pt2.y);

    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

# Ανάθεση με Δομές

Για να αντιγραφούν τα περιεχόμενα μιας δομής σε μια άλλη, χρησιμοποιούμε τον τελεστή ανάθεσης:

```
#include <stdio.h>

struct point { int x; int y; };

int main() {

    struct point pt1 = { 3, 4 };

    struct point pt2;

    printf("%d %d\n", pt2.x, pt2.y);

    pt2 = pt1;

    printf("%d %d\n", pt2.x, pt2.y);

    return 0;
}
```

Τι θα τυπώσει αυτό το πρόγραμμα;

```
$ ./copy
-29387249 0
3 4
```

Δεν έχει αρχικοποιηθεί οπότε θα τυπώσει  
ότι υπήρχε στην μνήμη

Όλα τα περιεχόμενα του pt1  
αντιγράφηκαν στο pt2 με την ανάθεση

# Ανάθεση με Δομές

Για να αντιγραφούν τα περιεχόμενα μιας δομής σε μια άλλη, πρέπει να ακριβώσ το **ΐδιο όνομα** τύπου:

```
#include <stdio.h>

struct point1 { int x; int y; };

struct point2 { int x; int y; };

int main() {
    struct point1 pt1 = { 3, 4 };
    struct point2 pt2;

    pt2 = pt1;

    printf("%d %d\n", pt2.x, pt2.y);

    return 0;
}
```

```
$ gcc -o badcopy badcopy.c
badcopy.c: In function 'main':
badcopy.c:7:9: error: incompatible types when assigning
to type 'struct point2' from type 'struct point1'
      7 |     pt2 = pt1;
           |           ^~~
```

# Σύγκριση με δομές

Μπορώ να συγκρίνω δομές με τελεστές σύγκρισης;

```
struct point pt1 = { 3, 4 };

struct point pt2;

pt2 = pt1;

if (pt1 == pt2) printf("impossible\n");

$ gcc -o copy copy.c
copy.c: In function 'main':
copy.c:13:11: error: invalid operands to binary == (have 'struct point' and
'struct point')
13 |     if (pt1 == pt2) printf("impossible\n");
      ^~
```

ΟΧΙ, πρέπει να συγκρίνω τα πεδία της δομής ένα-ένα

# Το προσδιοριστικό **typedef**

Το προσδιοριστικό **typedef** (type definition) χρησιμοποιείται για τον ορισμό συνωνύμων για τύπους. Χρησιμοποιούμε την σύνταξη:

```
typedef υπάρχων_τύπος νέος_τύπος;
```

Μετά από αυτόν τον ορισμό οι δύο τύποι είναι συνώνυμοι. Για παράδειγμα:

```
typedef unsigned int myuint;
```

Με αυτόν τον ορισμό οι ακόλουθες δηλώσεις μεταβλητών είναι ισοδύναμες:

```
unsigned int x;  myuint x;
```

# Πίνακες και `typedef`

Έστω ότι θέλουμε να ορίσουμε έναν δικό μας τύπο `page` που αντιστοιχεί σε έναν πίνακα 1024 ακεραίων, μπορούμε να ορίσουμε:

```
typedef int page[1024];
```

Με αυτόν τον ορισμό μπορούμε να δηλώσουμε έναν πίνακα ως εξής:

```
page mypage;
```

```
...
printf("%zu\n", sizeof(mypage));
...
$ ./page
4096
```

# Structs και typedef

Μπορούμε να δημιουργήσουμε μια συντομογραφία Student για τον τύπο struct student ως εξής:

```
#include <stdio.h>

typedef struct student {
    char first_name[128]; char last_name[128];
    int year; double grade;
} Student;

int main() {
    Student st1 = {"Thanos", "Barbounis", 1, 7.54};
    printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade, st1.year);
    return 0;
}
```

Ορισμός τύπου Student που αντιστοιχεί  
στο struct student

# Structs και typedef

Μπορούμε να δημιουργήσουμε μια συντομογραφία student για τον τύπο struct student ως εξής:

```
#include <stdio.h>

typedef struct student {
    char first_name[128]; char last_name[128];
    int year; double grade;
} student;

int main() {
    student st1 = {"Thanos", "Barbounis", 1, 7.54};
    printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade, st1.year);
    return 0;
}
```

Ορισμός τύπου student που αντιστοιχεί  
στο struct student

# Structs και typedef

Μπορούμε να δημιουργήσουμε μια συντομογραφία student για τον ανώνυμο τύπο struct ως εξής:

```
#include <stdio.h>

typedef struct {

    char first_name[128]; char last_name[128];

    int year; double grade;

} student;

int main() {

    student st1 = {"Thanos", "Barbounis", 1, 7.54};

    printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade, st1.year);

    return 0;
}
```

Η ετικέτα του struct μπορεί να παραλειφθεί

# Εμφωλευμένες/Ένθετες Δομές (Nested Structs)

Μια δομή μπορεί να περιέχει **μία ή περισσότερες δομές**, οι οποίες ονομάζονται εμφωλευμένες/ένθετες δομές (nested structs). Γενική μορφή:

```
struct όνομα1 {  
    ...  
};  
  
struct όνομα2 {  
    ...  
    struct όνομα1 πεδίο3;  
    ...  
};
```

Η δομή όνομα1 είναι εμφωλευμένη στην δομή όνομα2

# Εμφωλευμένες Δομές (Nested Structs)

```
#include <stdio.h>
int main() {
    struct date {
        int day;
        int month;
        int year;
    };
    struct product {
        char * name;
        double price;
        struct date created;
        struct date updated;
    };
    struct product prod = {"eclasse", 3.14, {1, 1, 2021}, {11, 12, 2022}};
    printf("%zu\n", sizeof(prod));
    return 0;
}
```

\$ ./nested  
40

Η δομή date είναι εμφωλευμένη μέσα στην δομή product και χρησιμοποιείται από δύο πεδία

# Εμφωλενμένα Πεδία Δομών (Nested Struct Fields)

```
#include <stdio.h>

struct date {
    int day;
    int month;
    int year;
};

struct product {
    char * name;
    double price;
    struct date created;
    struct date updated;
};

int main() {
    struct product prod = {"eclasse", 3.14, {1, 1, 2021}, {11, 12, 2022}};
    prod.updated.year = 2023;
    printf("%s [eu: %.2f] [created: %d/%d/%d] [updated: %d/%d/%d]\n",
           prod.name, prod.price,
           prod.created.day, prod.created.month, prod.created.year,
           prod.updated.day, prod.updated.month, prod.updated.year);
    return 0;
}
```

Χρησιμοποιούμε  
`.` όσες φορές  
χρειαστεί για να  
αναφερθούμε στο  
πεδίο που θέλουμε

\$ ./nested2  
eclasse [eu: 3.14] [created: 1/1/2021] [updated: 11/12/2023]

# Εμφωλευμένα Πεδία Δομών (Nested Struct Fields)

```
#include <stdio.h>

typedef struct {
    int day;
    int month;
    int year;
} Date;

typedef struct {
    char * name;
    double price;
    Date created;
    Date updated;
} Product;

int main() {
    Product prod = {"eclasse", 3.14, {1, 1, 2021}, {11, 12, 2022}};
    prod.updated.year = 2023;
    printf("%s [eu: %.2f] [created: %d/%d/%d] [updated: %d/%d/%d]\n",
           prod.name, prod.price,
           prod.created.day, prod.created.month, prod.created.year,
           prod.updated.day, prod.updated.month, prod.updated.year);
    return 0;
}

$ ./nested3
eclasse [eu: 3.14] [created: 1/1/2021] [updated: 11/12/2023]
```

# Δείκτες σε Δομές

```
#include <stdio.h>
#include <stdlib.h>

typedef struct { int day; int month; int year; } Date;

int main() {
    Date d1 = {1, 10, 2023};

    Date * d2 = &d1;
    (*d2).day = 2;

    Date * d3 = malloc(sizeof(Date));
    *d3 = *d2;

    (*d3).month = 12; (*d3).day = 11;

    printf("Diff: %d/%d/%d\n", (*d3).day - d1.day, (*d3).month - d1.month, (*d3).year - d1.year);

    return 0;
}
```

Οι δείκτες μπορούν να συνδυαστούν με δομές όπως όλοι οι άλλοι τύποι. Τι θα τυπώσει το διπλανό πρόγραμμα;

# Δείκτες σε Δομές

```
#include <stdio.h>
#include <stdlib.h>

typedef struct { int day; int month; int year; } Date;

int main() {
    Date d1 = {1, 10, 2023};

    Date * d2 = &d1;
(*d2).day = 2;

    Date * d3 = malloc(sizeof(Date));
*d3 = *d2;

    (*d3).month = 12; (*d3).day = 11;

    printf("Diff: %d/%d/%d\n", (*d3).day - d1.day, (*d3).month - d1.month, (*d3).year - d1.year);

    return 0;
}
```

Οι δείκτες μπορούν να συνδυαστούν με δομές όπως όλοι οι άλλοι τύποι. Τι θα τυπώσει το διπλανό πρόγραμμα;

```
$ ./date
Diff: 9/2/0
```

# Συντόμευση: Προσπέλαση Πεδίων Δομής μέσω Δείκτη (Arrow Operator)

Για να προσπελάσουμε το πεδίο μιας δομής, όταν έχουμε έναν δείκτη στην δομή χρησιμοποιούμε:

όνομα\_μεταβλητής\_τύπου\_δείκτη\_σε\_δομή->όνομα\_πεδίου

Για παράδειγμα:

```
student * st1;
```

```
...
```

```
st1->year
```

Αντί να γράφουμε (\*st1).year μπορούμε να χρησιμοποιήσουμε την σύνταξη st1->year. Οι δύο εκφράσεις (\*ptr).field και ptr->field είναι **ισοδύναμες**

# Δείκτες σε Δομές με χρήση ->

```
#include <stdio.h>
#include <stdlib.h>

typedef struct { int day; int month; int year; } Date;

int main() {
    Date d1 = {1, 10, 2023};

    Date * d2 = &d1;
    d2->day = 2;

    Date * d3 = malloc(sizeof(Date));
    *d3 = *d2;

    d3->month = 12; d3->day = 11;
    printf("Diff: %d/%d/%d\n", d3->day - d1.day, d3->month - d1.month, d3->year - d1.year);
    return 0;
}
```

Χρησιμοποιώντας -> γράφουμε έναν χαρακτήρα λιγότερο :) και υποδεικνύουμε στον αναγνώστη ότι η μεταβλητή είναι δείκτης

```
$ ./dateptr
Diff: 9/2/0
```

# Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 109-119, 126

- struct και C structures και άλλα initializations
- typedef
- Padding και alignment
- Η δομή FILE (λέγαμε είναι ~216 bytes, μπορείτε να δείτε τι περιέχει)
- Arrow operator

Ευχαριστώ και καλή μέρα εύχομαι!  
Keep Coding ;)