ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

# Προγραμματισμός Η/Υ

## Ενότητα 9: Ειδικά θέματα Δομών Δεδομένων

Νίκος Καρακαπιλίδης, Καθηγητής
Δημήτρης Σαραβάνος, Καθηγητής
Πολυτεχνική Σχολή
Τμήμα Μηχανολόγων & Αεροναυπηγών Μηχανικών

# Σκοποί ενότητας

- Κατανόηση της έννοιας της Δομής (struct)

# Περιεχόμενα ενότητας

- Η έννοια της Δομής
- Δομές και συναρτήσεις
- Δείκτες σε Δομές
- Ενώσεις
- Εισαγωγή στις Δυναμικές Δομές Δεδομένων

Μέρος 1ο

# Η έννοια της Δομής

# Η έννοια της Δομής

Ορισμός, δήλωση, αρχικοποίηση, αναφορά, ένθεση

# Η έννοια της Δομής

Αφαιρετικότητα

- Διεργασίες
- Δεδομένα

Παράδειγμα: «Ποια είναι η διεύθυνσή σου;»

- Οδός, αριθμός, Τ.Κ., πόλη

# Η έννοια της Δομής

Διαχείριση ομάδων δεδομένων διαφορετικού τύπου τα οποία έχουν σχέση μεταξύ τους

- Κάθε δεδομένο έχει το δικό του όνομα
- Παράβαλε: τύπος πίνακα

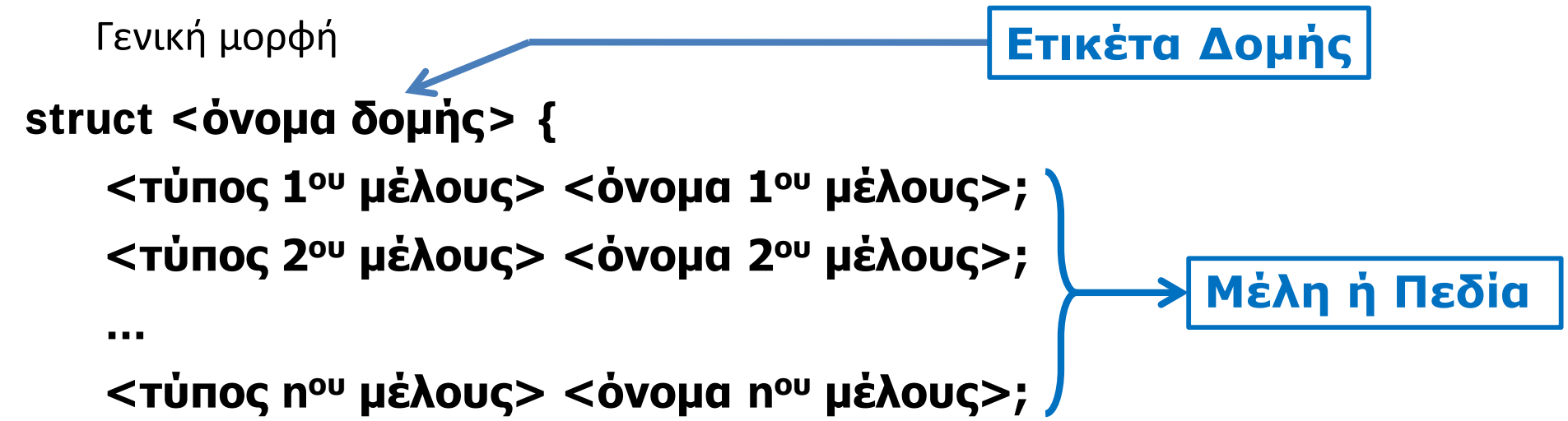
Γλώσσες Προγραμματισμού

- Εγγραφή (record)
  - Πεδίο (field)
- Δομή (structure)

**Ευκολότερος χειρισμός τέτοιων δεδομένων**

# Ορισμός Δομής

Γενική μορφή




```
struct <όνομα δομής> {
    <τύπος 1ου μέλους> <όνομα 1ου μέλους>;
    <τύπος 2ου μέλους> <όνομα 2ου μέλους>;
    ...
    <τύπος nου μέλους> <όνομα nου μέλους>;
    };

struct <όνομα δομής> {
    <τύπος μέλους> <όνομα 1ου μέλους>, <όνομα 2ου μέλους>;
        ...
    };
```

# Ορισμός Δομής

Παραδείγματα

```
struct rectangle {
        int x1, y1;  /* συντεταγμένες κάτω αριστερής γωνίας */
        int x2, y2;  /* συντεταγμένες πάνω δεξιάς γωνίας */
        int line_color; /* χρώμα εξωτερικής γραμμής */
        int fill_color; /* χρώμα σχήματος */
        };
```

# Ορισμός Δομής

```
struct Person {
        char firstName[15];
        char lastName[15];
        char gender;  /* M ή F */
        int age;
        };
```

# Δήλωση μεταβλητών

Γενική μορφή

**struct <όνομα δομής> <όνομα μεταβλητής>;**

**ή**

**struct <όνομα δομής> <λίστα ονομάτων μεταβλητών>;**

# Δήλωση μεταβλητών

Εναλλακτική μορφή

- Ταυτόχρονα με τον ορισμό της δομής

**struct <όνομα δομής> {**

**<τύπος 1ου μέλους> <όνομα 1ου μέλους>;**

**<τύπος 2ου μέλους> <όνομα 2ου μέλους>;**

**…**

**<τύπος nου μέλους> <όνομα nου μέλους>;**

**} <λίστα ονομάτων μεταβλητών>;**

# Δήλωση μεταβλητών

Παραδείγματα

```
struct rectangle {
        int x1, y1;
        int x2, y2;
        int line_color;
        int fill_color;
        } rect1, rect2;

        ή ...
```

**Συνίσταται να γίνεται διαχωρισμός του ορισμού από τις δηλώσεις**

# Δήλωση μεταβλητών

**… ή**

(με την προϋπόθεση ότι έχει προηγηθεί ο ορισμός της δομής **rectangle**)

**struct rectangle rect1, rect2;**

Δήλωση πίνακα Δομών

**struct Person people[10];**

# Απόδοση αρχικών τιμών

Απόδοση τιμών στα μέλη της Δομής

```
struct Person x = {"Νίκος", "Ιωάννου", 'Μ', 43};

struct Person people[10] = {
                        {"Νίκος", "Ανδρέου", 'Μ', 43},
                        {"Μαρία", "Γεωργίου", 'F', 38},
                        {"Βασίλης", "Λάκης", 'Μ', 28}
                        };
```

**Αρχικοποιούνται τα τρία πρώτα στοιχεία του πίνακα**

# Αναφορά στα μέλη μιας Δομής

Μέσω της κατασκευής

**<όνομα μεταβλητής>.<όνομα μέλους>**

Παραδείγματα

**rect1.x1**

**rect2.line_color**

```
if (x.age > 40)
        printf("%s age: %d \n", x.lastName, x.age);
```

# Ένθεση Δομών

Δομές μπορεί να φωλιάζουν η μία μέσα στην άλλη, δημιουργώντας έτσι πιο πολύπλοκες δομές

```
struct family {
        struct Person father;
        struct Person mother;
        int num_of_child;
        struct Person children[5];
};
```

# Ένθεση Δομών

```
int main(void){
        struct Person x, y, z;
        struct family fml;
        fml.num_of_child = 2;
        strcpy(fml.father.firstName, “Joe”);
        strcpy(fml.children[0].firstName, “Mary”);
        …
}
```

Μέρος 2ο

# Δομές και συναρτήσεις

# Δομές και συναρτήσεις

# Δομές και συναρτήσεις

Επιτρεπτές πράξεις σε μια δομή είναι:

- η αντιγραφή της
- η απόδοση τιμής σε αυτή σαν σύνολο
- η εξαγωγή της διεύθυνσής της
- η προσπέλαση των μελών της

# Δομές και συναρτήσεις

Μεταβλητές τύπου δομής μπορούν να περαστούν ως ορίσματα σε συναρτήσεις όπως επίσης και να επιστραφούν ως αποτελέσματα συναρτήσεων

```
struct Person inc_age (struct Person x){
        x.age += 1;
        return x;
}
...
struct Person x1, x2;
x2 = inc_age(x1);
```

# Δομές και συναρτήσεις

Μη επιτρεπτή πράξη

- Σύγκριση δύο δομών (**x1 == x2**)

Προβλήματα

- Να γράψετε συνάρτηση η οποία παίρνει δύο παραμέτρους τύπου **struct Person** και τις συγκρίνει
- Να γράψετε συνάρτηση η οποία παίρνει ως δεδομένο εισόδου μια οικογένεια (τύπου **struct family**) και ένα άτομο (τύπου **struct Person**) και αν το άτομο δεν ανήκει ήδη στα παιδιά της οικογένειας τότε το προσθέτει
- Αλγόριθμοι;

Μέρος 3ο

# Δείκτες σε Δομές

# Δείκτες σε Δομές

# Δείκτες σε Δομές

Η μεταβλητή **addr1** που ορίζεται από τη δήλωση

**struct address addr1**

έχει κάποια διεύθυνση

- Η έκφραση **&addr1** δίνει τη διεύθυνση της δομής **addr1**

# Δείκτες σε Δομές

Η πρόταση

```
struct address *addr_ptr = &addr1;
```

έχει ως επακόλουθο ότι ο δείκτης **addr_ptr** δείχνει στη δομή **addr1** παρέχοντας έτσι ένα εναλλακτικό τρόπο πρόσβασης στα μέλη της.

Πολύ συχνή χρήση δεικτών σε δομές

# Δείκτες σε Δομές

Παρέχεται ο εναλλακτικός συμβολισμός:

**(*p).μέλος_δομής = p->μέλος_δομής**

Παραδείγματα

```
struct Person p;
. . .
initPerson(&p);
. . .
void initPerson(struct Person *p){
        strcpy( p->firstName, “Ανδρέας”);
        strcpy( p->lastName, “Ανδρέου”);
        p->gender = ’M’;
        p->age = 43;
}
```

```
struct address addr[10];
struct address *addr_ptr = &addr[0];
addr_ptr++;
```

Μέρος 4ο

# Ενώσεις

# Ενώσεις

Η χρήση των Ενώσεων

# Ενώσεις

Γενικά

- Η ένωση (union) είναι κατασκευή της C που μοιάζει με αυτή της δομής (structure)
- Η διαφορά έγκειται στο ότι τα μέλη των ενώσεων επικαλύπτουν το ένα το άλλο, μοιράζονται δηλαδή ένα κοινό κομμάτι μνήμης
- Ο συνολικός χώρος που καταλαμβάνει η ένωση είναι αυτός που απαιτεί το μέλος με το μεγαλύτερο μέγεθος
- Ο τύπος union χρησιμοποιείται στη C για τη δήλωση μεταβλητών που μπορούν να αποθηκεύσουν σε διαφορετικές στιγμές δεδομένα διαφορετικών τύπων και μεγεθών στην ίδια περιοχή μνήμης

# Ενώσεις

- **Παράδειγμα:** Έστω ότι θέλουμε μια σταθερά σε ένα πρόγραμμα να δέχεται είτε μια ακέραια τιμή (int) είτε μια πραγματική τιμή (float). Θα μπορούσαμε να δηλώσουμε:

```
struct {
    int ival;
    float fval;
} constant;
```

και να τοποθετούμε τη σταθερά μας σε ένα από τα δύο πεδία της δομής ανάλογα με την τιμή της.

**Μειονέκτημα:** αυτής της λύσης είναι ότι σπαταλά άσκοπα μνήμη (δεσμεύει μνήμη για δύο πεδία ενώ θα χρησιμοποιηθεί κάθε φορά μόνο το ένα)

# Ενώσεις

Χρήση ένωσης (union)

- Η προηγούμενη δήλωση γίνεται ως εξής:

```
union {
  int ival;
  float fval;
} constant;
```

- Η μνήμη που δεσμεύεται σε αυτή την περίπτωση είναι η μέγιστη που απαιτείται για αποθήκευση ακριβώς μιας μεταβλητής από οποιονδήποτε από τους τύπους των πεδίων
- Η τιμή που ανακτάται κάθε φορά είναι η αυτή που αποθηκεύτηκε πιο πρόσφατα κατά την εκτέλεση. Είναι ευθύνη του προγραμματιστή να παρακολουθεί και να γνωρίζει ποιος τύπος είναι αποθηκευμένος σε μια ένωση

# Παράδειγμα

```
#define INT_TYPE 1;
#define REAL_TYPE 2;

struct item {
        int type;
        union {
                int ival;
                float fval;
                } constant;
        }
...
struct item x;
...
if (x.type == INT_TYPE)
        printf("value of const = %d\n", x.constant.ival);
if (x.type == REAL_TYPE)
        printf("value of const = %f\n", x.constant.fval);
```

Μέρος 5ο

# Εισαγωγή στις Δυναμικές Δομές Δεδομένων

# Εισαγωγή στις Δυναμικές Δομές Δεδομένων

Δυναμική διαχείριση μνήμης

# Δυναμικές Δομές Δεδομένων

Ίσως η σημαντικότερη καινοτομία που συναντάμε μέχρι τώρα κατά τη γνωριμία μας με τη γλώσσα C

Ένα πρόγραμμα κατά την εκτέλεσή του μπορεί να ζητά δυναμικά μνήμη για την αποθήκευση δεδομένων καθώς και να ελευθερώνει δυναμικά τη μνήμη που δεν χρειάζεται

# Δυναμικές Δομές Δεδομένων

Ως σήμερα, δηλώναμε εκ των προτέρων μέσω απλών μεταβλητών ή πινάκων το μέγιστο αριθμό θέσεων μνήμης που ένα πρόγραμμα μπορεί να χειριστεί κατά την εκτέλεσή του

- Έτσι μηνύματα όπως “Array is full” ήταν πολύ συνηθισμένα όταν ο πίνακας γέμιζε

Η υλοποίηση τύπων δεδομένων όπως η λίστα, η στοίβα και η ουρά μπορεί να μη βασιστεί σε πίνακες οι οποίοι μας αναγκάζουν να ορίσουμε τις διαστάσεις τους και συνεπώς το μέγιστο πλήθος στοιχείων που μπορούμε να αποθηκεύσουμε

# Δυναμική Δέσμευση Μνήμης

**void *malloc(size)**

- Η **malloc()** επιστρέφει ένα δείκτη στο χώρο μνήμης που δέσμευσε δυναμικά και στον οποίο μπορούμε να αποθηκεύσουμε ένα αντικείμενο μεγέθους **size**. Η **malloc()** επιστρέφει NULL όταν η αίτηση δεν μπορεί να ικανοποιηθεί (δεν υπάρχει άλλη διαθέσιμη μνήμη για να δοθεί)

# Δυναμική Δέσμευση Μνήμης

Παράδειγμα:

- Η **malloc(sizeof(int))** δεσμεύει μνήμη για την αποθήκευση ενός ακεραίου και επιστρέφει ένα δείκτη (τη διεύθυνση δηλαδή) του χώρου μνήμης που δέσμευσε.

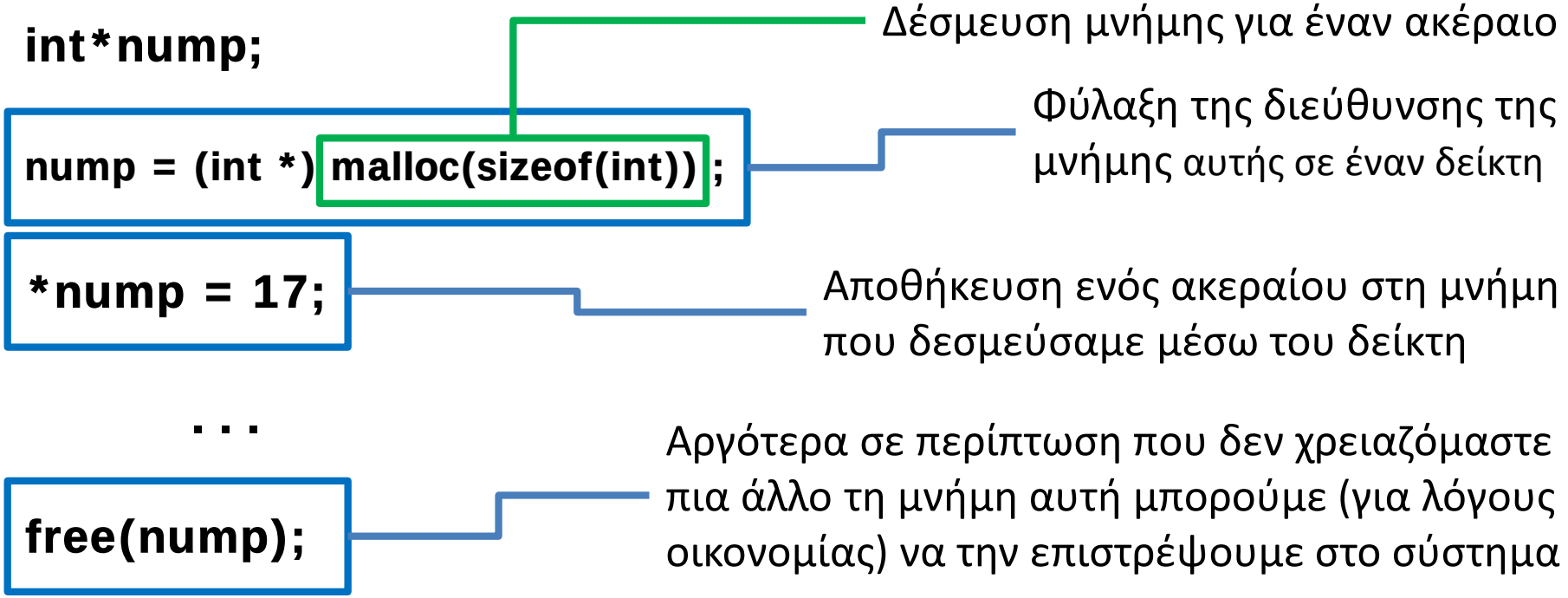
**Προσοχή:**

- Επειδή στη C μιλάμε πάντα για δείκτες ενός συγκεκριμένου τύπου πρέπει πάντα να κάνουμε **cast** τον δείκτη που επιστρέφει η malloc() στον αντίστοιχο τύπο. Δηλαδή:

```
(int *) malloc(sizeof(int))
```

# Δυναμική Δέσμευση Μνήμης

Παράδειγμα δηλώσεων και κλήσεων




Σημαντική είναι η χρήση της **sizeof(type)** η οποία μας επιστρέφει το μέγεθος αντικειμένου τύπου **type** και μας διευκολύνει στην εύρεση του ποσού μνήμης που απαιτείται για την αποθήκευση ενός αντικειμένου τύπου **type**

# Αυτοαναφορικές δομές

Χρήση δυναμικής δέσμευσης μνήμης

- Για την αποθήκευση όχι τόσο ακεραίων ή άλλων απλών τύπων δεδομένων αλλά αντικειμένων τύπου structure. Αυτό γιατί μπορούμε μέσω αντικειμένων τύπου structure να φτιάξουμε κόμβους, να τους συνδέσουμε μεταξύ τους και να δημιουργήσουμε έτσι μία συνδεδεμένη λίστα ή άλλες εξελιγμένες δομές όπως στοίβες, ουρές, λίστες αναμονής, δέντρα κλπ.

# Αυτοαναφορικές δομές

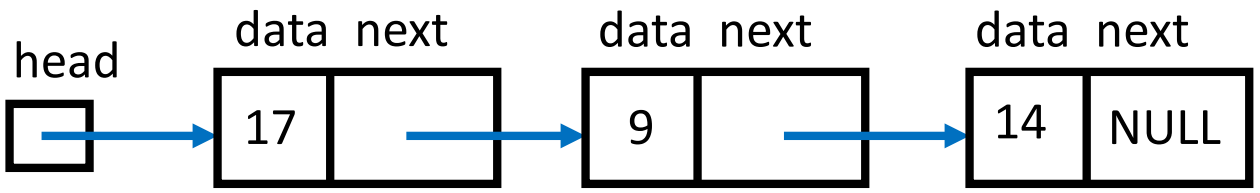
Απλός ορισμός κόμβου μιας συνδεδεμένης λίστας

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

- Προσέξτε ότι ένα πεδίο της δομής που υλοποιεί τον κόμβο είναι δείκτης στην ίδια δομή που ορίζεται. Αυτό το φαινόμενο στις δομές τύπου structure το ονομάζουμε αυτοαναφορικές δομές (self-referential structures)

# Αυτοαναφορικές δομές

Συνδεδεμένη λίστα με κόμβους τύπου **struct node**



# Αυτοαναφορικές δομές

```
struct node *head, *p1, *p2;
head  = (struct node *) malloc (sizeof(struct node));
p1    = (struct node *) malloc (sizeof(struct node));
p2    = (struct node *) malloc (sizeof(struct node));

head->data    = 17;
p1->data    = 9;
p2->data    = 14;

head->next   = p1;
p1->next     = p2;
p2->next     = NULL;
```

**Σημ.: οι δείκτες p1 και p2 είναι βοηθητικοί για την υλοποίηση της λίστας**

# Η δήλωση typedef στη C

Η C παρέχει μέσω της δήλωσης typedef τη δημιουργία νέων ονομάτων σε τύπους δεδομένων που ήδη υπάρχουν

Προσοχή: δεν δημιουργούμε νέους τύπους δεδομένων

- Απλά “βαφτίζουμε” με νέα ονόματα τύπους που ήδη υπάρχουν

# Η δήλωση typedef στη C

Παραδείγματα

```
typedef int             Akeraios;
typedef char            *String;
typedef struct node     NODE;     /* το struct node
                                     έχει οριστεί νωρίτερα */
...

Akeraios        en, I, arr[20];
String          p, lineptr[MAXLINES];
NODE            n1, n2, *pn;
```

# Η δήλωση typedef στη C

Πολλές φορές «βαφτίζουμε» μία δομή ταυτόχρονα με τον ορισμό της

```
typedef struct node {
        int data;
        struct node *next;
} NODE;
```

# Η δήλωση typedef στη C

Είναι καλή συνήθεια να χρησιμοποιούμε ως όνομα στο typedef το ίδιο το όνομα της δομής αλλά με κεφαλαία γράμματα

Χρήση της δήλωσης typedef

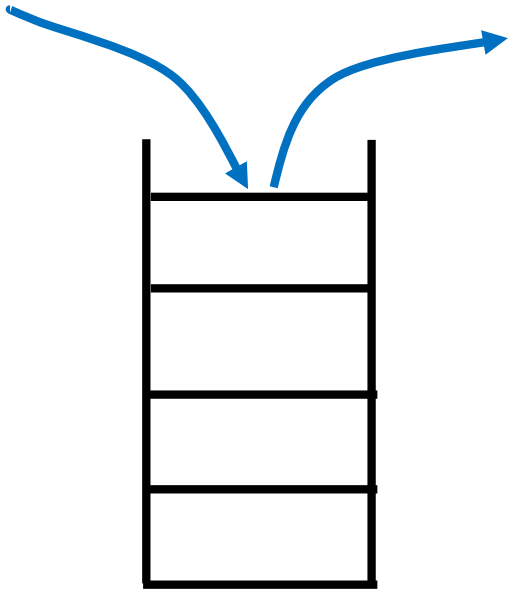
- Δημιουργεί πιο αναγνώσιμο και πιο κατανοητό κώδικα
    - Το να δηλώσεις NODE *ptr; είναι πιο κατανοητό από το να δηλώσεις ένα δείκτη σε μία πολύπλοκη δομή (struct node *ptr;)
- Ο κώδικας γίνεται πιο λιτός. Παράδειγμα:

```
ptr = (NODE *)malloc(sizeof(NODE));
```

# Στοίβες

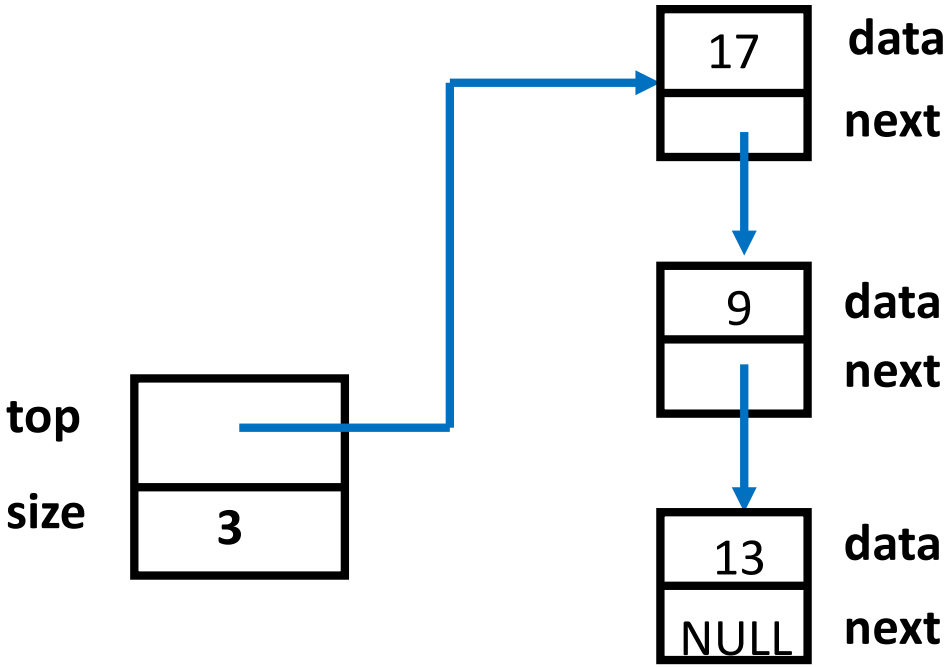
Μια στοίβα είναι μια λίστα στοιχείων που συνοδεύεται από τις διαδικασίες

- **push,** για εισαγωγή στοιχείου στη λίστα
- **pop,** για εξαγωγή του τελευταία εισαγομένου στοιχείου της λίστας



# Στοίβες

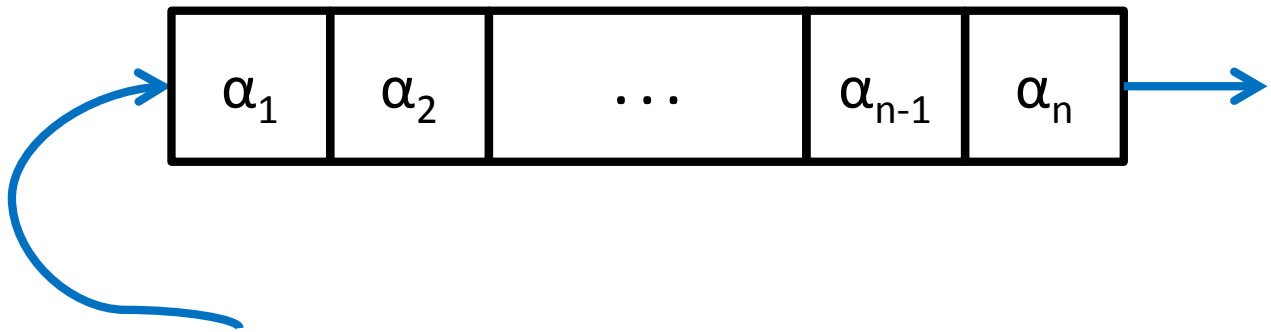
Για ευκολία στην διεκπεραίωση των δύο διαδικασιών θα ήταν χρήσιμο να γνωρίζουμε ανά πάσα στιγμή που βρίσκεται το τέλος της λίστας (και το μέγεθός της)



# Ουρές

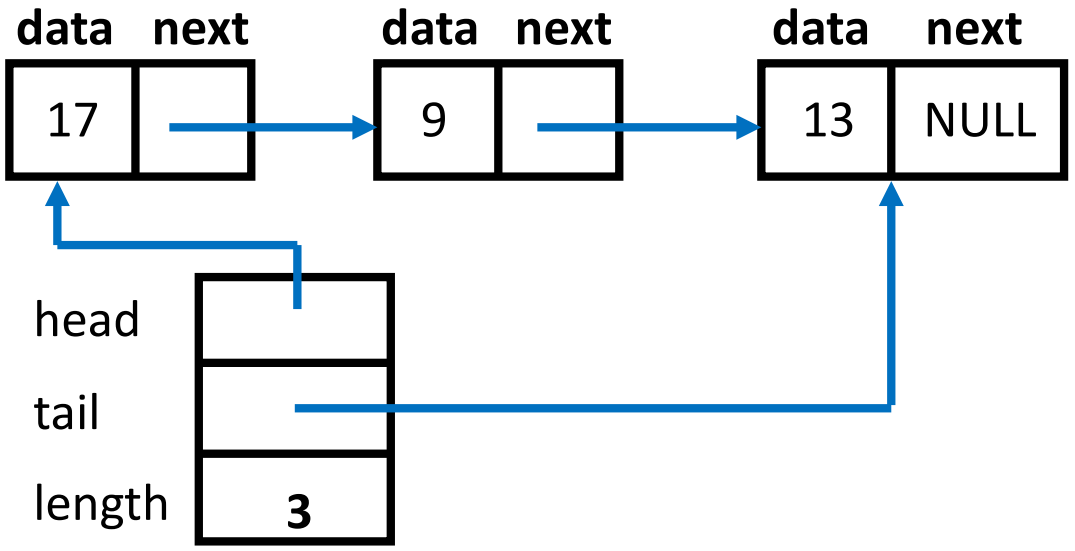
Μια ουρά είναι μια λίστα στοιχείων που συνοδεύεται από τις διαδικασίες

- **queque,** για εισαγωγή στοιχείου στη λίστα
- **unqueue,** για εξαγωγή του στοιχείου που εισήχθηκε πρώτο στη λίστα



# Ουρές

Για ευκολία στην διεκπεραίωση των δύο διαδικασιών θα ήταν χρήσιμο να γνωρίζουμε ανά πάσα στιγμή που βρίσκεται το τέλος της λίστας (και το μέγεθός της)



# Τέλος Ενότητας

# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου των διδασκόντων.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

# Σημείωμα Αναφοράς

# Σημείωμα Αδειοδότησης

# Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

# Σημείωμα Χρήσης Έργων Τρίτων