

ΜΥΥ502

Προγραμματισμός Συστημάτων



Β. Δημακόπουλος

dimako@cse.uoi.gr

<http://www.cse.uoi.gr/~dimako>

❖ Αντικείμενο μαθήματος:

- Εκμάθηση βασικών εργαλείων, τεχνικών και μεθόδων για προχωρημένο προγραμματισμό που στοχεύει περισσότερο στο «σύστημα» παρά στην «εφαρμογή»
- Η γλώσσα C είναι μονόδρομος στον προγραμματισμό συστημάτων
 - ✧ Με μεγάλη προγραμματιστική βάση και στο χώρο των εφαρμογών
 - ✧ Προγραμματισμός συστήματος:
 - «κάτω» από το επίπεδο εφαρμογών (υποστηρίζει τις εφαρμογές)
 - Ανάμεσα στις εφαρμογές και το hardware
 - π.χ. μεταφραστές, λειτουργικά συστήματα, συστήματα υποστήριξης εκτέλεσης, ενσωματωμένα συστήματα κλπ.
- Το UNIX/POSIX και τα «POSIX-οειδή» περιβάλλοντα είναι η σημαντικότερη και πιο ολοκληρωμένη πλατφόρμα για εργασία σε επίπεδο συστήματος
 - ✧ Βασικές γνώσεις χρήσιμες και σε επόμενα μαθήματα (π.χ. λειτουργικά συστήματα, παράλληλα συστήματα, μεταφραστές κλπ)

❖ 'Υλη μαθήματος (από τον οδηγό σπουδών):

- “ Η γλώσσα προγραμματισμού C: στοιχειώδης C (βασικοί τύποι δεδομένων, εκφράσεις, τελεστές, δομές ελέγχου ροής, συναρτήσεις), προχωρημένα στοιχεία (πίνακες, δείκτες, δομές), δυναμική διαχείριση μνήμης, είσοδος/έξοδος, προεπεξεργαστής. ”
- “ Βασικές κλήσεις UNIX (διεργασίες, I/O, σήματα). Διαδιεργασιακή επικοινωνία (κοινόχρηστη μνήμη, sockets). Εισαγωγή στον παράλληλο προγραμματισμό (νήματα, mapReduce). Προχωρημένα θέματα (ασφάλεια, γλώσσα μηχανής, εργαλεία ανάπτυξης μεγάλων προγραμμάτων). ”



❖ Δύο μέρη:

➤ Γλώσσα προγραμματισμού C

- ✧ Υποθέτει γνώση προγραμματιστικών τεχνικών
- ✧ Υποθέτει γνώση γλωσσών προγραμματισμού «συγγενών» με την C (π.χ. Java)
- ✧ Καλύπτονται από τις «Τεχνικές Αντικειμενοστραφούς Προγραμματισμού» και «Ανάπτυξη Λογισμικού»

Π1: Περίπου 50% της ύλης

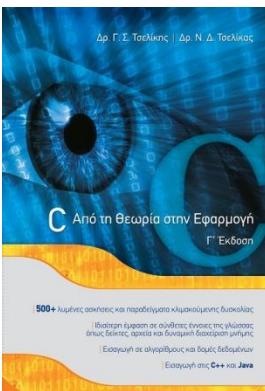
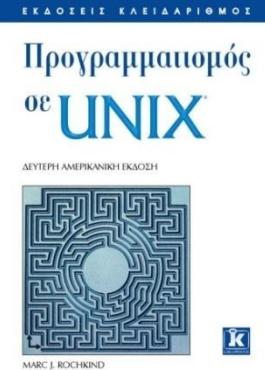
➤ Προγραμματισμός συστημάτων POSIX και προχωρημένα θέματα

- ✧ Εμβάθυνση σε προχωρημένες δυνατότητες της C
- ✧ Γνωριμία με διαδικασίες και εργαλεία ανάπτυξης εφαρμογών συστήματος
- ✧ Βασικές κλήσεις POSIX (διεργασίες, σήματα, επικοινωνίες, νήματα κλπ)
- ✧ Άλλα προχωρημένα θέματα και τεχνικές

Π2: Περίπου 50% της ύλης

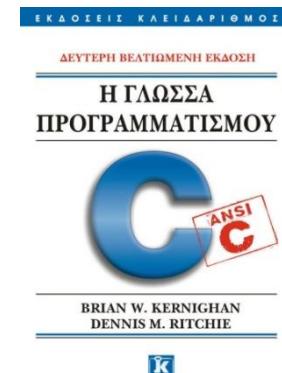
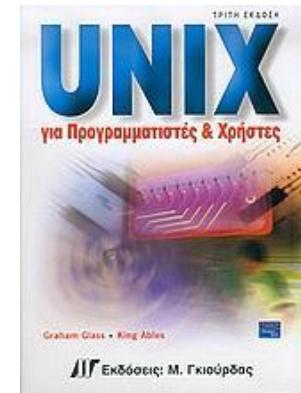
Συγγράμματα

- ❖ Υπάρχουν πολλά βιβλία για C
 - Και πάρα πολύ υλικό στο διαδίκτυο
- ❖ Για προγραμματισμό συστημάτων (POSIX) όχι τόσα πολλά μεταφρασμένα
 - Συνήθως θεωρούν δεδομένη τη γνώση της C
 - Πολλά που είναι για χρήση / διαχείριση του UNIX, όχι προγραμματισμό
 - ✧ Δεν μας αφορούν



Εύδοξος

- ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ UNIX,
M.J. Rochkind (2007)
 - ❖ Εξαιρετικό βιβλίο για προγραμματισμό συστημάτων UNIX
 - ❖ Υποθέτει γνώση της C
- UNIX ΓΙΑ ΠΡΟΓΡΑΜΜΑΤΙΣΤΕΣ ΚΑΙ ΧΡΗΣΤΕΣ,
G.Glass, K. Ables (2005)
 - ❖ Χρήση, διαχείριση, εσωτερικά του UNIX
 - ❖ Δύο κεφάλαια αφιερώνονται στα εργαλεία προγραμματισμού και στις κλήσεις συστήματος του UNIX και υποθέτει γνώση της C
- Σ ΑΠΟ ΤΗ ΘΕΩΡΙΑ ΣΤΗΝ ΕΦΑΡΜΟΓΗ,
Γ. Τσελίκης, Ν. Τσελίκης (2016)
 - ❖ Πάρα πολύ καλό βιβλίο C (ελληνικό!)
- Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C,
B.W. Kernighan, D.M. Ritchie (2008)
 - ❖ «Ευαγγέλιο» της C (C90)
 - ❖ Ίσως όχι το καλύτερο για εκμάθηση.



- ❖ Πολλές ιστοσελίδες για C
- ❖ Πολλές ιστοσελίδες για προγραμματισμό σε UNIX/Linux/POSIX κλπ.
- ❖ Πολύ καλό βιόήθημα στο διάβασμά σας:
 - “Programming in C; Unix System Calls and Subroutines using C.”, A. D. Marshall
 - <http://www.cs.cf.ac.uk/Dave/C/CE.html>
 - Καλύπτει και εκμάθηση της C αλλά και αρκετό προγραμματισμό συστημάτων POSIX

Ώρες μαθήματος

❖ Διαλέξεις:

- Δευτέρα: 12:00 – 14:00
- Τετάρτη: 12:00 – 14:00 (*)
- Αίθουσα: 15

❖ Εργαστήρια (το πιθανότερο):

- Τρίτη: 14:00 – 20:00
- ΠΕΠ I, ΠΕΠ II, ΠΕΛΣ

❖ Ήρες γραφείου διδάσκοντα:

- Τρίτη: 09:00 – 11:00
- Teams (& ίσως B33)

	Δε	Τρ	Τε	Πε	Πα
08:00					
09:00					
10:00		Γρ			
11:00					
12:00	Δ			Δ	
13:00					
14:00		Εργ			
15:00		Εργ			
16:00		Εργ			
17:00		Εργ			
18:00		Εργ			
19:00		Εργ			

Ιστοσελίδα μαθήματος:

<http://www.cse.uoi.gr/~dimako/teaching/>

Εργαστήρια (αν εγγράφεστε ΠΡΩΤΗ ΦΟΡΑ ΣΤΟ ΕΡΓΑΣΤΗΡΙΟ)

❖ Σκοπός/λειτουργία εργαστηρίων:

- «Φροντιστηριακού» τύπου για ενίσχυση διδασκαλίας
- Σχεδιασμός προγράμματος και υλοποίηση «επί τόπου»
- (Πολύ) έμπειροι μεταπτυχιακοί και προσωπικό για να σας βοηθήσουν

❖ Οργανωτικά:

- Ξεκινούν (λογικά) σε 2 εβδομάδες – Θα σας ενημερώσω εγκαίρως
- Εφόσον γίνουν πλήρως δια ζώσης:
 - ✧ Θα εργάζεστε ένας σε κάθε θέση
 - ✧ 14:00 – 18:00 (3 σειρές της 1½ ώρας με ενδιάμεση απολύμανση)
- Θα υπάρξει πρόβλεψη και για εξ αποστάσεως
(Θα υπάρξει ενημέρωση)
- **Παρουσίες**

Όσοι ΠΕΡΑΣΟΥΝ το εργαστήριο,
το κατοχυρώνουν για πάντα

Και έρχονται μόνο στις τελικές
εξετάσεις.

❖ Βαθμός εργαστηρίου:

- Θα βγει από τη συμμετοχή σας και από 2 προόδους
- **20% οι παρουσίες + 40% πρόοδος1 + 40% πρόοδος2**
- Κάθε απουσία σας αφαιρεί 10%. Από 2 και μετά δεν έχει διαφορά...



Τι σημαίνει «ΠΕΡΝΑΩ» το εργαστήριο

- ❖ Για να **ΠΕΡΑΣΕΙ** κάποιος το εργαστήριο, θα πρέπει ο βαθμός του (20% οι παρουσίες + 40% πρόοδος 1 + 40% πρόοδος 2) να είναι $\geq 4,5$.
 - Όποιος το περάσει, το κατοχυρώνει για ΠΑΝΤΑ.

FAQ

- ❖ ΕΡΩΤΗΣΗ: Αν κάνω όλες τις παρουσίες περνάω;
- ❖ ΑΠΑΝΤΗΣΗ: ΟΧΙ, πρέπει να περάσετε και τις προόδους
- ❖ ΕΡΩΤΗΣΗ: Αν ΔΕΝ περάσω για κάποιο λόγο, μπορώ να ξαναπαρακολουθήσω το εργαστήριο του χρόνου;
- ❖ ΑΠΑΝΤΗΣΗ: ΟΧΙ, δείτε την επόμενη διαφάνεια
- ❖ ΕΡΩΤΗΣΗ: Δεν με βολεύει η ημ/νια της προόδου / έχω κανονίσει να λείπω / έχω πληρώσει εισιτήρια / θα είμαι άρρωστος εκείνη την ημέρα. Μπορώ να δώσω κάποια άλλη μέρα;
- ❖ ΑΠΑΝΤΗΣΗ: ΟΧΙ

Εργαστήρια (για όσους είχαν ΕΓΓΡΑΦΕΙ ΠΑΛΑΙΟΤΕΡΑ)

❖ Αφορά μόνο όσους:

1. Έχουν ξαναεγγραφεί στο εργαστήριο κατά το παρελθόν KAI
2. Δεν έχουν περάσει το εργαστήριο ποτέ

❖ Εγγραφή στα εργαστήρια:

- Υποχρεωτική, πλήρης (ηλεκτρονική) εγγραφή όπως όλοι
- Αλλιώς ΔΕΝ ΘΑ ΜΠΟΡΕΣΟΥΝ ΝΑ ΕΞΕΤΑΣΤΟΥΝ

❖ Εξέταση και βαθμός εργαστηρίων:

- Εξέταση στις 2 προόδους του εργαστηρίου, όπως όλοι.
- Απαραίτητη επικοινωνία με διδάσκοντα πριν από κάθε πρόοδο
- Ο βαθμός θα βγει ως εξής:
50% πρόοδος1 + 50% πρόοδος2

Εργαστήρια (για όσους είχαν ΕΓΓΡΑΦΕΙ ΠΑΛΑΙΟΤΕΡΑ)

- ❖ Για να **ΠΕΡΑΣΕΙ** κάποιος παλιός το εργαστήριο, θα πρέπει ο βαθμός του (50% πρόοδος1 + 50% πρόοδος2) να είναι $\geq 4,5$.

FAQ

- ❖ ΕΡΩΤΗΣΗ: *Είχα κάνει παρουσίες παλιά. Μπορώ να πάρω το 20% από αυτές;*
- ❖ ΑΠΑΝΤΗΣΗ: **ΟΧΙ**, οι παρουσίες μετρούν μόνο την 1^η φορά που εγγράφεστε στο εργαστήριο
- ❖ ΕΡΩΤΗΣΗ: *Μπορώ να ξαναπαρακολουθήσω το εργαστήριο να πάρω το 20% από τις παρουσίες;*
- ❖ ΑΠΑΝΤΗΣΗ: **ΟΧΙ**
- ❖ ΕΡΩΤΗΣΗ: *Μπορώ να ξαναπαρακολουθήσω το εργαστήριο ΧΩΡΙΣ να πάρω το 20% από τις παρουσίες;*
- ❖ ΑΠΑΝΤΗΣΗ: Πολύ δύσκολο λόγω χώρου και κορωνοϊού. Αν όμως θέλετε να ασχοληθείτε μόνοι σας θα υπάρχουν και οι ασκήσεις και οι απαντήσεις και οι βοηθοί.

Βαθμολόγηση

❖ Επιτυχία στο μάθημα προϋποθέτει:

1. Επιτυχία στο εργαστήριο (Βαθμός εργαστηρίων $\geq 4,5$)
 - ✧ Όσοι δεν επιτύχουν στο εργαστήριο, δεν έχουν δικαίωμα εξετάσεων
2. Τουλάχιστον βαθμό 4,5 στις εξετάσεις
 - ✧ Όσοι έχουν βαθμό < 4 στις εξετάσεις, δεν περνούν ακόμα και άριστα να πήγαν στο εργαστήριο.

❖ Τελικός βαθμός (το πιθανότερο):

- **50% εργαστήριο + 50% τελικές εξετάσεις**

Το σημερινό μάθημα

Εισαγωγικά στοιχεία για τη C



MYY502

❖ D. Ritchie, Bell Labs, 1972

- Με βάση προηγούμενη γλώσσα (B)
- Χρησιμοποιήθηκε για την υλοποίηση του λειτουργικού συστήματος UNIX
- Ευρεία διάδοση από τότε.

❖ Από αυτήν προέκυψαν / επηρεάστηκαν οι περισσότερες από τις πιο δημοφιλείς γλώσσες:

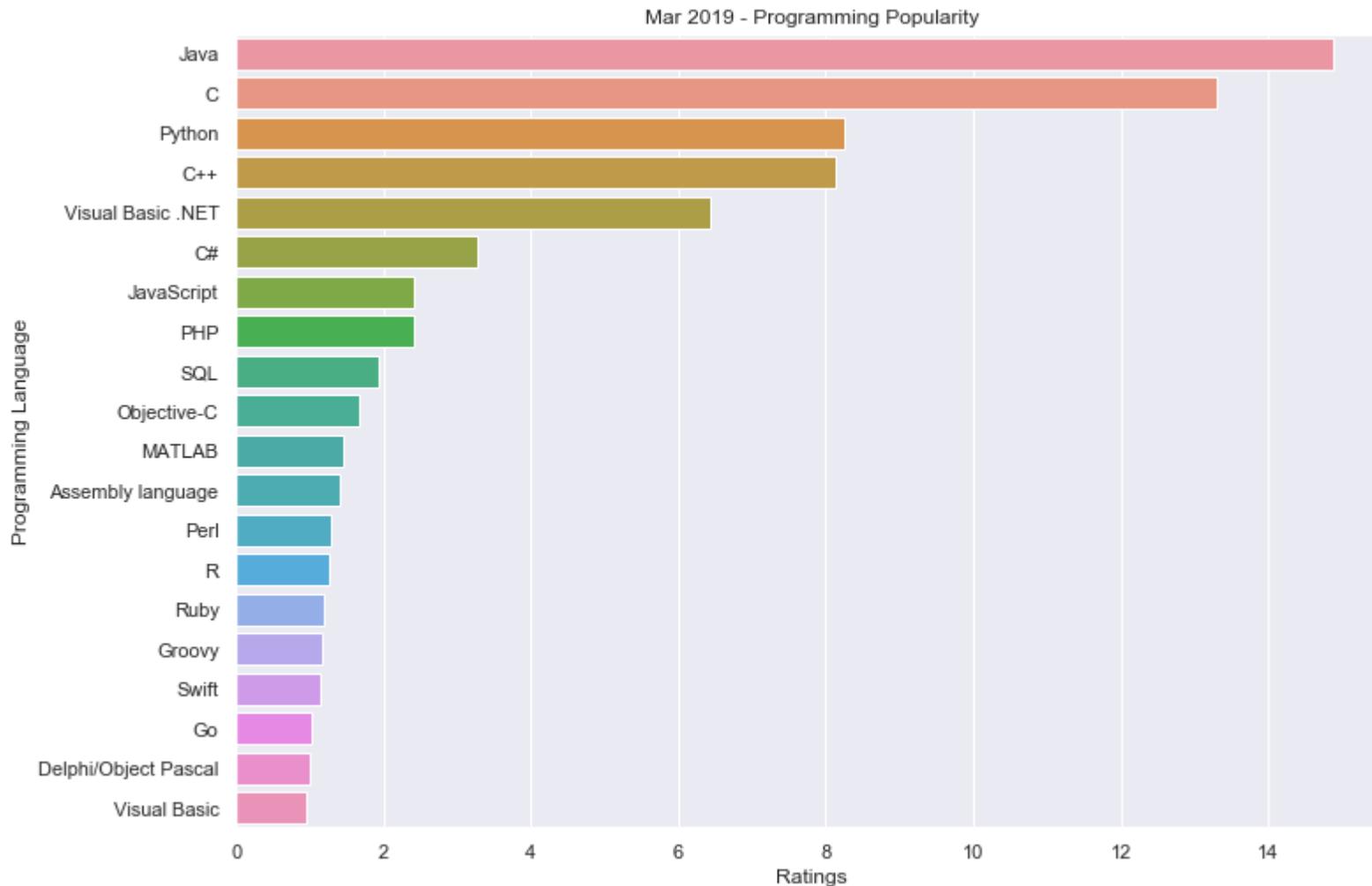
- Π.χ. C++, Java, PHP κλπ.

❖ Σε κάποιους τομείς (π.χ. ενσωματωμένα συστήματα) η C είναι ουσιαστικά η μοναδική επιλογή

- Δυνατή, μικρή, εύκολα μεταφράσιμη γλώσσα

Δημοτικότητα της C (TIOBE index, 2019)

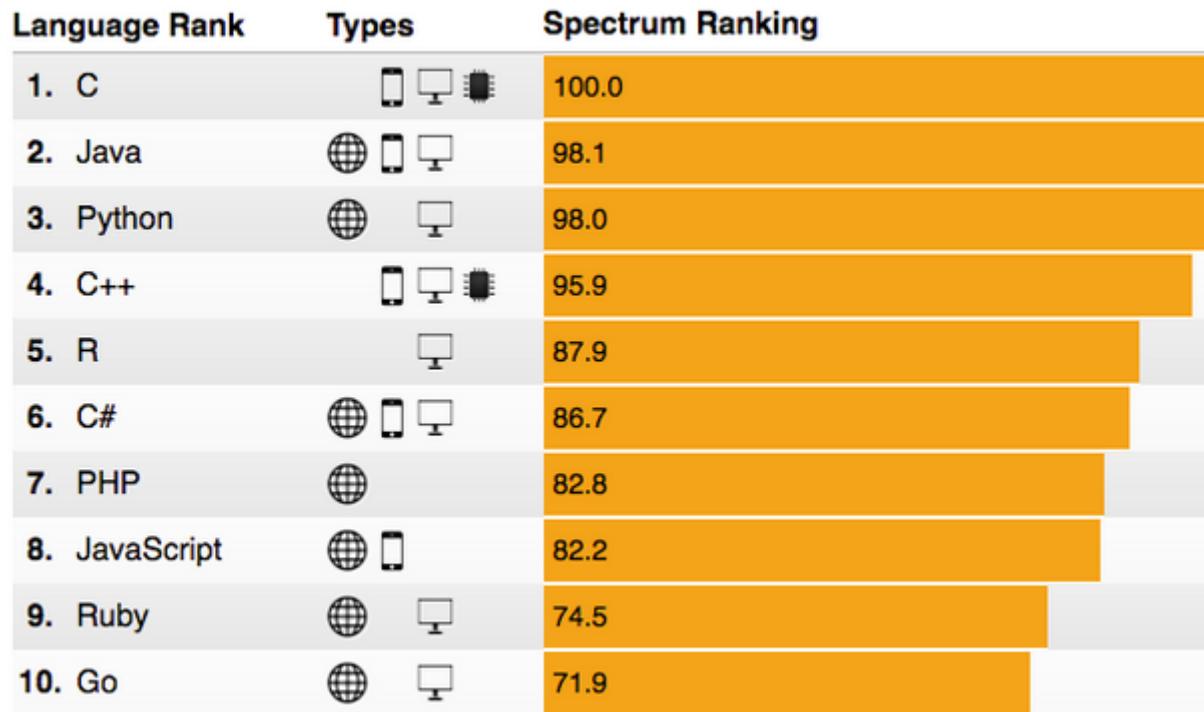
❖ Όλες οι κορυφαίες με βάση την C!



Κατάταξη γλωσσών προγραμματισμού (IEEE Spectrum)

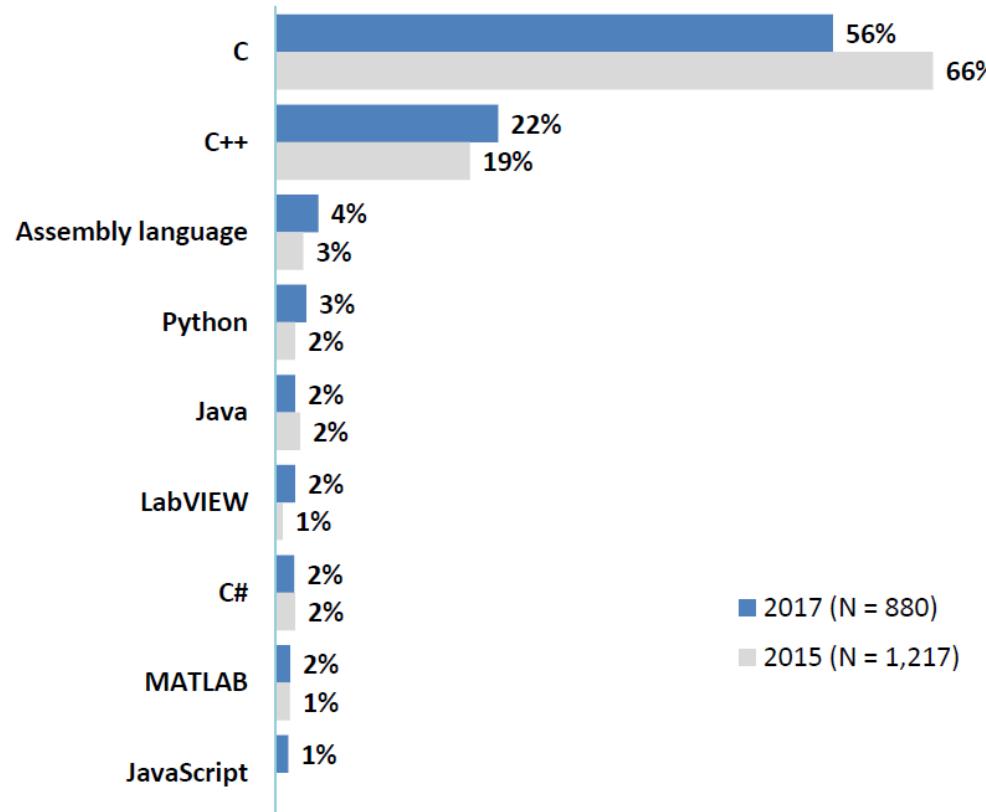
❖ IEEE, Αυγ. 2016

<http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>



Στα ενσωματωμένα συστήματα

My current embedded project is programmed mostly in:



EE Times embedded

2017 Embedded Markets Study

© 2017 Copyright by AspenCore. All rights reserved.

- ❖ Το UNIX γράφτηκε σε C
 - ❖ Ο πυρήνας του Linux είναι γραμμένος σε C
 - ❖ Σχεδόν όλες οι εφαρμογές συστήματος είναι σε C
 - ❖ Η πλειονότητα εφαρμογών ανοιχτού κώδικα είναι σε C
 - ❖ ...
-
- ❖ **Προσοχή:** Η C ΔΕΝ είναι πανάκεια...
 - «Επικίνδυνη» αν κάποιος δεν την ξέρει καλά
 - «Χαμηλότερου» επιπέδου από άλλες γλώσσες (π.χ. Java)
 - Δεν βολεύει πάντα για εφαρμογές χρήστη, ειδικά όταν υπάρχει γραφική αλληλεπίδραση

Εισαγωγή στη C

C για προγραμματιστές Java



MYY502

Hello world

```
public class hello          #include <stdio.h>
{
    public static         int main() {
        void main (String args []) {     puts("Hello world");
            System.out.println      return 0;
            ("Hello world");           }
        }
}
```

❖ Κλάσεις

- Μόνο δεδομένα (μεταβλητές) και συναρτήσεις
- Η συνάρτηση `main()` είναι αυτή που εκτελείται αρχικά

❖ Boolean

- Με ακεραίους «προσομοιώνουμε» τα boolean
- Το **0** θεωρείται FALSE
- Οτιδήποτε μη-μηδενικό θεωρείται TRUE

❖ Strings (τουλάχιστον όπως τα χειρίζεται η Java)

- Χειρισμός μέσω πινάκων και δεικτών

❖ try ... catch μπλοκ (exceptions)

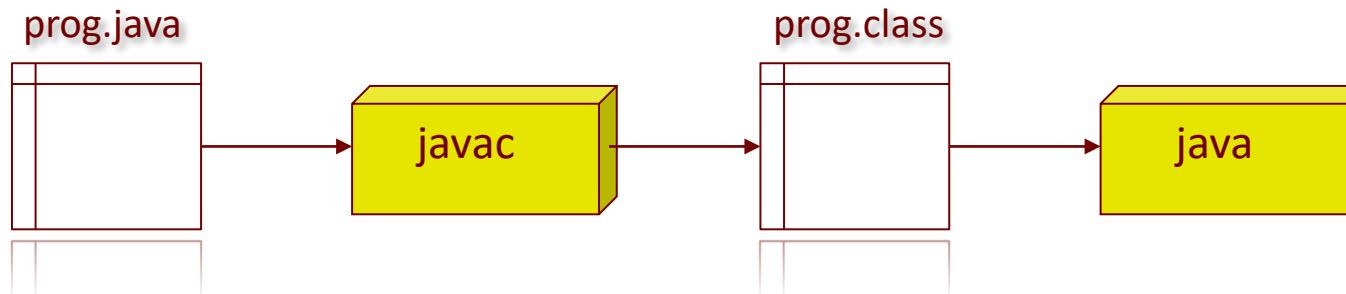
- Δεν υπάρχει ανάλογο, μόνο μέσω συναρτήσεων συστήματος

- ❖ Pointers (δείκτες)!
 - Όχι μόνο απλό πέρασμα με αναφορά
- ❖ «Ελευθερία» στους τύπους των δεδομένων (π.χ. int/short/char είναι πάνω-κάτω ίδιοι) και δεν γίνεται πλήρης έλεγχος κατά τη χρήση τους.
- ❖ «Ελευθερία» στη διαχείριση της μνήμης
 - Επαφίεται πλήρως στον προγραμματιστή
 - Η java έχει garbage collector που αυτόματα αποδεσμεύει άχρηστη μνήμη

Java vs C

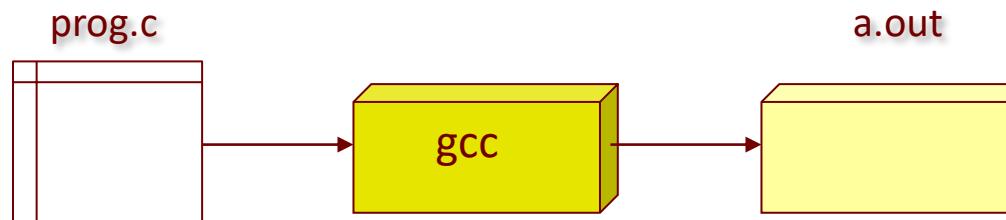
❖ Η Java είναι (βασικά) ερμηνευόμενη (*interpreted*)

- Συνήθως μετατρέπεται (javac) σε bytecode,
 - ✧ ο οποίος ερμηνεύεται από μία εικονική μηχανή (JVM),
 - η οποία εκτελείται (java) στην πραγματική μηχανή



❖ Η C είναι (βασικά) μεταφραζόμενη (*compiled*)

- Μετατρέπεται απευθείας σε εντολές assembly της πραγματικής μηχανής που θα εκτελέσει το πρόγραμμα
 - ✧ Το πρόγραμμα εκτελείται αυτόνομα

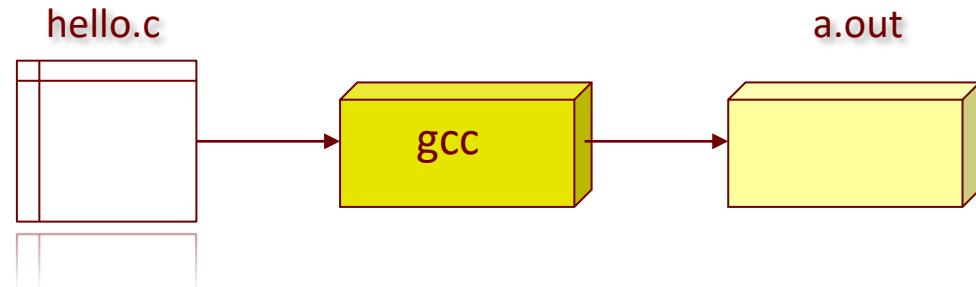


- ❖ Σελίδα στο ecourse:
 - Password: sys20prog20
- ❖ Προκαταρκτικό πλάνο εργαστηρίων
 - [Ιστοσελίδα...](#)
- ❖ Διαφάνειες
 - Αναρτώνται μετά το τέλος της κάθε ενότητας – υπάρχουν οι περσινές ούτως ή άλλως.
- ❖ Επικοινωνία με email
 - [Ιστοσελίδα...](#)

Το πρώτο πρόγραμμα σε C (hello.c)

```
#include <stdio.h>           ← HEADER (αρχείο επικεφαλίδων)  
                                Θυμίζει το import της java  
  
int main()                   ← Συνάρτηση εκκίνησης  
{  
    /* Just show a simple message */   ← Σχόλιο  
    printf("Hello, World\n");          ← Οθόνη  
}  
                                ← Τερματισμός προγράμματος  
                                ← Αλλαγή γραμμής
```

Μετάφραση του προγράμματος



- ❖ Στο τερματικό:

```
% ls
```

```
hello.c
```

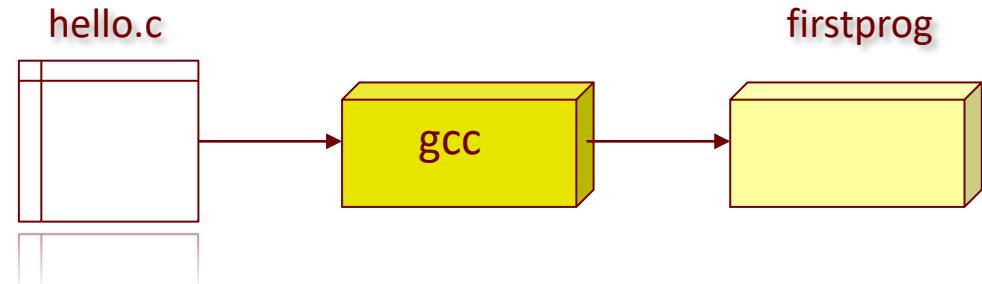
```
% gcc hello.c
```

```
% ls
```

```
a.out hello.c
```

- ❖ Ο μεταφραστής (gcc) ονομάζει το εκτελέσιμο “a.out”

Μετάφραση του προγράμματος με δικό μας όνομα

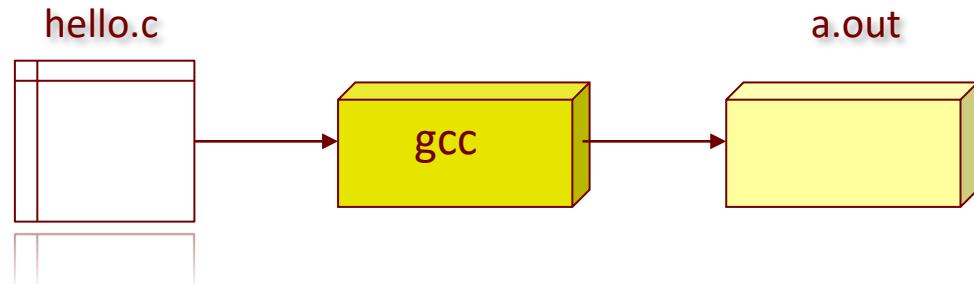


- ❖ Στο τερματικό:

```
% ls  
hello.c  
% gcc -o firstprog hello.c  
% ls  
firstprog hello.c
```

- ❖ Με το “–o” ο μεταφραστής αντί για “a.out” ονομάζει το εκτελέσιμο με ότι όνομα μας αρέσει.

Εκτέλεση του προγράμματος



- ❖ Στο τερματικό:

```
% ls
```

```
hello.c
```

```
% gcc hello.c
```

```
% ls
```

```
a.out hello.c
```

```
% ./a.out
```

```
Hello, World
```

```
%
```

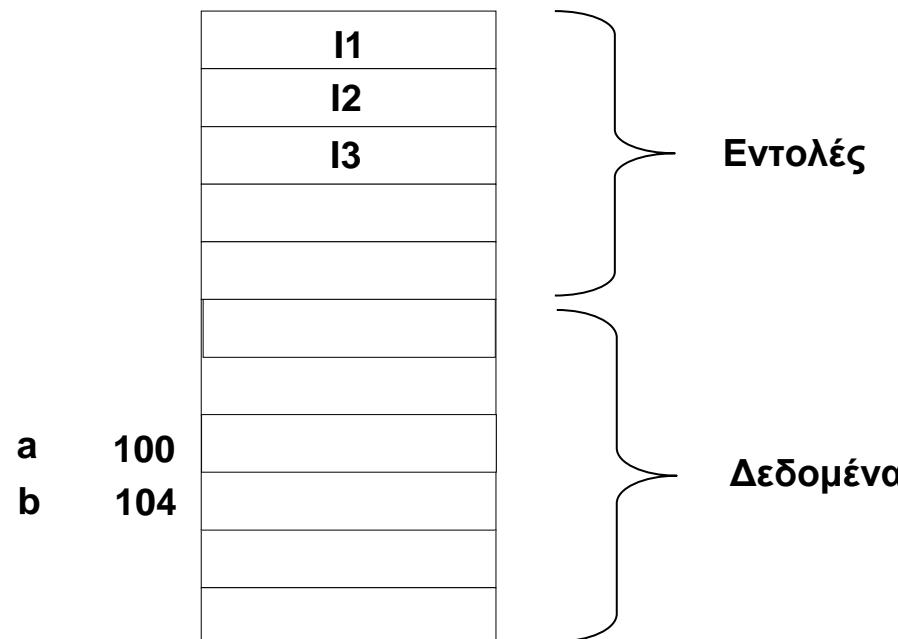
- ❖ Το “.” εννοεί το τρέχον directory – ίσως και να μην χρειάζεται.

- ❖ Πρόγραμμα = {δεδομένα} + {κώδικας (συναρτήσεις)}
- ❖ Συναρτήσεις = {main, ...}
- ❖ Δεδομένα = {μεταβλητές, σταθερές}
- ❖ Σταθερές = ποσότητες που δεν μεταβάλλονται κατά την εκτέλεση του προγράμματος
 - Π.χ. $\pi = 3.14$
- ❖ Μεταβλητές = ποσότητες που μεταβάλλονται
 - I/O, υπολογισμοί

- ❖ Τα δεδομένα αποθηκεύονται στη μνήμη του υπολογιστή
- ❖ Δηλώνοντας μια μεταβλητή δεσμεύω θέσεις στη μνήμη και καθορίζω ένα όνομα που χρησιμοποιώ για να αναφερθώ σε αυτή τη θέση μνήμης
- ❖ π.χ.
 - int const n = 10;
 - int a;

Εισαγωγικά

- ❖ Και οι εντολές αποθηκεύονται στη μνήμη του υπολογιστή
- ❖ Π.χ. εντολή I1: πρόσθεσε την τιμή της μεταβλητής a με αυτή της b → πρόσθεσε τα περιεχόμενα της θέσης 100 στα περιεχόμενα της θέσης 104



- ❖ Κάθε μεταβλητή, σταθερά έχει ένα τύπο
- ❖ Ο τύπος καθορίζει το μέγεθος του «κελιού» που θα δεσμευτεί στη μνήμη
 - char: 1 byte
 - int: 4 bytes (συνήθως)
 - float: 4 bytes
 - double: 8 bytes
- ❖ Προσδιοριστές
 - short int: 2 bytes
 - long int: 8 bytes
 - long double: ? bytes (≥ 10)

❖ Προσδιοριστές: `unsigned`

- Χωρίς πρόσημο (μόνο θετικοί) – όλα τα bits για την τιμή του αριθμού

❖ `unsigned int`, `unsigned short int`, `unsigned char`

- 32 bits, 16 bits, 8 bits

- ✧ 0, 1, ..., 4294967295
- ✧ 0, 1, ..., 65535
- ✧ 0, 1, ..., 255

❖ `int`, `short int`, `char`

- 31 bits, 15 bits, 7 bits (συν 1 για το πρόσημο)

- ✧ -2147483648, ..., 2147483647
- ✧ -32768, ..., 32767
- ✧ -128, ..., 127

Εκτυπώνοντας τις τιμές των μεταβλητών

- ❖ Η βασικότερη και γενικότερη συνάρτηση εκτύπωσης είναι η “printf”.
 - Αρχικά πρέπει να δώσω ως πρώτο όρισμα μια συμβολοσειρά με το ΤΙ ΤΥΠΟ θα εκτυπώσει
 - Στη συνέχεια, τα επόμενα ορίσματα είναι οι αντίστοιχες μεταβλητές ή εκφράσεις
- ❖ Παράδειγμα:

```
int main() {  
    int x = 5;  
    float y = 1.2;  
  
    printf("%d", x); /* %d ή %i για ακεραίους */  
    printf("%f", y); /* %f για πραγματικούς */  
    printf("x = %d and y = %f", x, y);  
    return 0;  
}
```

- ❖ Όσες μεταβλητές βρίσκονται εντός ενός μπλοκ εντολών (π.χ. μέσα σε μία συνάρτηση) είναι **τοπικές (local)** και μπορούν να χρησιμοποιηθούν μόνο εντός του μπλοκ.
- ❖ Όσες είναι εκτός των συναρτήσεων είναι **καθολικές (global)** και μπορούν να χρησιμοποιηθούν παντού.
- ❖ Περισσότερα αργότερα...

- ❖ Όλα τα δεδομένα σε ένα υπολογιστή κωδικοποιούνται ως ακολουθίες 0, 1
- ❖ Ένας χαρακτήρας κωδικοποιείται ως ακολουθία 0, 1
- ❖ Άρα στην ουσία ο υπολογιστής τον αντιλαμβάνεται σαν ένα αριθμό
 - Ο χαρακτήρας '0' αντιστοιχεί στον αριθμό 48
 - `char ch = 'x';`
 - Λέγοντας `ch = 'x'` είναι σαν να λέμε: βάλε στη μεταβλητή `ch` την τιμή (αριθμό) που αντιστοιχεί στο χαρακτήρα 'x'
 - `ch++;`
- ❖ Αριθμητική τιμή = κωδικός ASCII

Παράδειγμα με printf

```
#include <stdio.h>

int main()
{
    char ch = 'x';      /* Τοπική μεταβλητή τύπου χαρακτήρα */

    printf("ch = %d, ch = %c\n", ch, ch);

    return 0;
}
```

Ακέραιοι σε διάφορες μορφές...

```
int main() {
    int x = 95; /* 000...0 0101 1111 */

    printf("%d", x); /* 95 */
    printf("%x", x); /* 5f */
    printf("%o", x); /* 137 */
    printf("%c", x); /* _ */

    x = 95; /* Καταχωρώντας το 95 σε διάφορες μορφές */
    x = 0x5F;
    x = 0137;

    x = 'd'; /* Ποιος αριθμός είναι αυτός; */
    printf("%d", x); /* 100 */
    printf("%x", x); /* 64 */
    printf("%o", x); /* 144 */
    printf("%c", x); /* d */
    return 0;
}
```

- ❖ Αριθμητικοί τελεστές
 - +, -, *, /, % (το τελευταίο μόνο για ακεραίους)
- ❖ Συγκριτικοί τελεστές
 - >, <, <=, >=, ==, !=
- ❖ Λογικοί τελεστές
 - &&, ||, !
- ❖ Υπάρχουν και κάποιοι άλλοι τελεστές που θα μας απασχολήσουν αργότερα
 - Bitwise operators: ~, &, |, ^, >>, <<

Πράξεις και μετατροπές

- ❖ Αριθμητικοί τελεστές για δεδομένα ίδιου τύπου κυρίως (π.χ. πρόσθεση δύο ακεραίων)
 - ❖ Όμως, μπορούμε να κάνουμε και πράξεις με μεταβλητές διαφορετικού τύπου, π.χ.

```
int x; float f;  
f = f+x;
```

- Γίνεται εσωτερική μετατροπή των «κατώτερων» τύπων σε «ανώτερους»
 - Και το αποτέλεσμα ανώτερου τύπου

- ❖ Τέτοιες μετατροπές γίνονται αυτόματα αλλά μπορούμε να τις ζητήσουμε και εμείς σε ένα πρόγραμμα με το μηχανισμό των “type casts”

- `f + x` (το `x` μετατρέπεται αυτόματα σε `float`)
 - `f + ((float) x)` (cast του προγραμματιστή)

❖ Τελεστές σύντμησης:

`++, --, +=, -=, *=, /=`

❖ Παράδειγμα

- `++i` και `i++` (pre-increment, post-increment)
- `i=i+1` και `i+=1`

❖ Παράδειγμα

```
i = 3;  
x = ++i; /* πρώτα γίνεται η αύξηση και μετά η αποτίμηση της έκφρασης */  
x = i++; /* πρώτα γίνεται η αποτίμηση της έκφρασης και μετά η αύξηση */  
i = i++ + ++i; /* εδώ τι τιμή θα πάρει τελικά το i? */
```

Ο «τριαδικός» τελεστής

μεταβλητή = συνθήκη ? τιμή1 : τιμή2;

❖ Η εκτέλεση ισοδυναμεί με:

```
if (συνθήκη)
    μεταβλητή = τιμή1;
else
    μεταβλητή = τιμή2;
```

❖ Παράδειγμα:

```
x = (y > 0) ? 1 : 0;
```

2 «στυλ» σταθερών

❖ Στη java το “final” μπορεί να χρησιμοποιηθεί για να ορίσει «σταθερές»

❖ Στη C υπάρχουν 2 τρόποι να οριστούν σταθερές:

1. Με προσθήκη του «const» στον τύπο της δήλωσης, π.χ.

```
const int x = 5; /* Δεν μπορεί να αλλάξει τιμή */
```

2. Με ορισμό σταθεράς προεπεξεργαστή (#define)

```
#define M 10           /* Το σύνηθες ... */
```

```
#define PI 3.14
```

```
#define NEWLINE '\n'
```

❖ Διαφορά:

- Οι const καταλαμβάνουν μνήμη για αποθήκευση
- Οι #define ΑΝΤΙΚΑΘΙΣΤΑΝΤΑΙ ΠΡΙΝ ΓΙΝΕΙ Η ΜΕΤΑΦΡΑΣΗ του προγράμματος (και άρα δεν υπάρχουν στο εκτελέσιμο)

Διάβασμα (scanf) / εκτύπωση (printf)

```
#include <stdio.h> /* Απαραίτητο */

char c;                  /* Καθολική μεταβλητή */

int main() {             /* Συνάρτηση main */
    int i;                /* Τοπικές δηλώσεις ΠΑΝΤΑ στην αρχή της συνάρτησης */
    float f;

    printf("Dwse 1 xaraktira, 1 akeraio kai enan pragmatiko\n");
    scanf("%c", &c);
    scanf("%d%f", &i, &f);           /* Η scanf ΘΕΛΕΙ & στις μεταβλητές */
    printf("c = %c, i = %d, f = %f\n", c, i, f);
    return 0;
}
```

Εισαγωγή στη Σ

Εντολές



ΜΥΥ502

❖ Όπως και στην java:

- if
- if-else
- switch
- for
- while
- do-while
- break

❖ Επιπλέον:

- goto

if & if-else

```
if (condition) {  
    statements;  
}  
  
if (condition) {  
    statements;  
} else {  
    statements;  
}
```

Αν υπάρχει ΜΟΝΟ ΈΝΑ statement,
όπως και στην Java, δε χρειάζονται οι
αγκύλες. Π.χ. ο παρακάτω κώδικας

```
if (x == 3)  
    x++;  
    y++;  
z = x+y;
```

είναι ισοδύναμος με:

```
if (x == 3) {  
    x++;  
}  
y++;  
z = x+y;
```

switch

```
switch (variable) {  
    case const1:  
        statements;  
        break;  
    case const2:  
        statements;  
        break;  
    ...  
    default:  
        statements;  
        break;  
}
```

Εντολές ελέγχου – switch

```
switch (variable) {  
    case const1:  
        statements;  
        break;  
    case const2:  
        statements;  
        break;  
    ...  
    default:  
        statements;  
        break;  
}
```

```
switch ( choice )  
{  
    case 'a':  
    case 'A':  
        do_thing_1();  
        break;  
  
    case 'b':  
    case 'B':  
        do_thing_2();  
        break;  
    ...  
    default:  
        printf("Wrong choice.");  
}
```

❖ Προσοχή:

- Αν δεν υπάρχει break, η εκτέλεση ενός case συνεχίζεται με τον κώδικα του επόμενου case...

while & do-while

```
while (condition) {  
    statements;  
}
```

```
x = 0;  
while (x < 10)  
    x++;
```

```
x = 0;  
while (x < 10);  
    x++;
```

```
do {  
    statements;  
} while (condition);
```

Αν υπάρχει ΜΟΝΟ ΈΝΑ statement,
όπως και στη Java, δε χρειάζονται οι
αγκύλες.

for (I)

```
for (initialization; condition; iteration) {  
    statements;  
}
```

```
/* Ισοδύναμος κώδικας: */  
initialization;  
while (condition) {  
    statements;  
    iteration;  
}
```

Αν υπάρχει ΜΟΝΟ ΕΝΑ statement,
όπως και στην Java, δε χρειάζονται οι
αγκύλες.

- ❖ Τα initialization / iteration μπορούν να περιέχουν πολλές εκφράσεις, χωρισμένες με κόμμα.

for (II)

- ❖ Τα initialization / iteration μπορούν να περιέχουν πολλές εκφράσεις, χωρισμένες με κόμμα.

Π.χ.

```
int i, sum;  
sum = 0;  
for (i = 1; i <= 10; i++) {  
    sum += i;  
}
```

```
for (i = 1, sum = 0; i <= 10; sum += (i++))  
    ;
```

- ❖ Δεν υπάρχει το for each

~~for (δήλωση : συλλογή)~~

της Java.

break

- ❖ Έξοδος από switch ή από βρόχους for/while/do, π.χ.

```
while (1) {  
    if (w == 3) {  
        break; /* Βγαίνει αμέσως μετά το while */  
    }  
    . . .  
}
```

- ❖ Δεν υπάρχει ονοματισμένο break (τύπου break <label>)

goto

- ❖ Η εκτέλεση μεταπηδά σε συγκεκριμένη ετικέτα, π.χ.

```
if (x == 1) {  
    goto before;  
}  
y = 2;  
goto after;  
before: y = 1;  
after: ...
```

- ❖ Επικίνδυνη / μη-προβλέψιμη εντολή, π.χ.

```
while (1) {  
    if (w == 3) {  
        Strange: x=1; ...  
    }  
    ...  
    if (condition) goto Strange;
```

- ❖ «Ακατάλληλη» δια ανηλίκους. Κακή προγραμματιστική τεχνική. Δεν πρέπει να χρησιμοποιείται σχεδόν ποτέ.

Αγκύλες και μπλοκ κώδικα

- ❖ Γράψτε τον παρακάτω κώδικα ΧΩΡΙΣ αγκύλες, όπου γίνεται:

```
if (i >= 81) {  
    x = 3;  
}  
if (row >= 1 && col<= 9) {  
    q = 5;  
    y = x;  
}  
else {  
    for (S[row][col] = 1; S[row][col] <= 9; S[row][col]++) {  
        if ( sudoku_solve(i) ) {  
            return (1);  
        }  
    }  
}
```

Αγκύλες και μπλοκ κώδικα

❖ Πρώτη προσπάθεια:

```
if (i >= 81) {  
    x = 3;  
}  
if (row >= 1 && col<= 9) {  
    q = 5;  
    y = x;  
}  
else {  
    for (S[row][col] = 1; S[row][col] <= 9; S[row][col]++) {  
        if ( sudoku_solve(i) ) {  
            return (1);  
        }  
    }  
}
```

Αγκύλες και μπλοκ κώδικα

❖ Δεύτερη προσπάθεια:

```
if (i >= 81
    x = 3;

if (row >= 1 && col<= 9) {
    q = 5;
    y = x;
}
else {
    for (S[row][col] = 1; S[row][col] <= 9; S[row][col]++) {
        if ( sudoku_solve(i) )
            return (1);

    }
}
```

Αγκύλες και μπλοκ κώδικα

❖ Τρίτη προσπάθεια:

```
if (i >= 81
    x = 3;

if (row >= 1 && col<= 9) {
    q = 5;
    y = x;
}
else {
    for (S[row][col] = 1; S[row][col] <= 9; S[row][col]++)
        if ( sudoku_solve(i) )
            return (1);

}
```

Αγκύλες και μπλοκ κώδικα

❖ Τελικός κώδικας:

```
if (i >= 81
    x = 3;

if (row >= 1 && col<= 9) {
    q = 5;
    y = x;
}
else
    for (S[row][col] = 1; S[row][col] <= 9; S[row][col]++)
        if ( sudoku_solve(i) )
            return (1);
```

❖ ΠΑΝΤΑ ΝΑ ΒΑΖΕΤΕ ΑΓΚΥΛΕΣ, ΑΚΟΜΑ ΚΑΙ ΌΤΑΝ ΔΕΝ ΧΡΕΙΑΖΕΤΑΙ

ΜΥΥ502

Προγραμματισμός Συστημάτων



Β. Δημακόπουλος

dimako@cse.uoi.gr

<http://www.cse.uoi.gr/~dimako>

Εισαγωγή στη Σ

Πίνακες



ΜΥΥ502

❖ Διαφορές με java:

- Όταν τους ορίζεις, πρέπει να δηλώσεις και το μέγεθος
- Οι αγκύλες πάνε μετά το όνομα της μεταβλητής

❖ Δηλ. δεν παίζει το:

```
int [] myarray;  
myarray = new int[30];
```

❖ Σωστή (και μοναδική) δήλωση:

```
int myarray[30];
```

❖ Η αρίθμηση των στοιχείων ξεκινάει από τη θέση μηδέν (0)

Πίνακες

- ❖ ΔΕΝ μπορεί να αλλάξει το μέγεθος του πίνακα.
- ❖ ΔΕΝ θυμάται / ελέγχει η C τα όρια του πίνακα.
- ❖ ΔΕΝ μπορεί να γίνει αρχικοποίηση του πίνακα παρά μόνο τη στιγμή της δήλωσής του. Μετά καταχώρηση στοιχείο-στοιχείο. Παράδειγμα:

```
#include <stdio.h>
#define N 10

int main() {
    int myarray[N] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

    myarray[3] = 400;
    myarray[10] = 900;      /* Εκτός ορίων - ΔΕΝ μας προειδοποιεί
                           με κάποιο σφάλμα η C */
    ...
}
```

Πίνακες χωρίς μέγεθος

- ❖ Επιτρέπεται να μην δοθεί το μέγεθος του πίνακα κατά τη δήλωση *μόνο αν προκύπτει από την αρχικοποίησή του:*

```
int main() {  
    int myarray[] = {10,20,30,40};  
    ...  
}
```

- Προκύπτει (και δεν μπορεί να αλλάξει) ότι το μέγεθος είναι 4 στοιχεία.
- ❖ Το παρακάτω δεν επιτρέπεται!

```
int main() {  
    int myarray[];  
    ...  
}
```

- ❖ Αν ο πίνακας δεν αρχικοποιηθεί κατά τη δήλωσή του
 - τα στοιχεία του θα έχουν τυχαίες τιμές («σκουπίδια»).
- ❖ Αν κατά τη δήλωση του πίνακα αρχικοποιηθούν λιγότερα στοιχεία από αυτά που έχει,
 - τα υπόλοιπα αρχικοποιούνται αυτόματα στο 0 (μηδέν)
- ❖ Π.χ. εύκολος τρόπος να αρχικοποιήσω όλα τα στοιχεία ενός πίνακα στο 0:
`int myarray[100] = { 0 };`

Συμβολοσειρές (strings): βασικά πίνακες χαρακτήρων

- ❖ Παράδειγμα:

```
int main() {  
    char str1[5];  
    char str2[6] = {'b', 'i', 'l', 'l', '\0'};  
  
    str2[1] = 'a'; /* b a l l */  
    ...  
}
```

- ❖ Απλά πρέπει να υπάρχει στο τέλος και ο ειδικός χαρακτήρας '\0' (λεπτομέρειες αργότερα).
- ❖ Πολύ πιο εύκολη αρχικοποίηση, μόνο για συμβολοσειρές:

```
char str3[6] = "maria"; /* Υπονοείται το \0 στο τέλος */  
char str4[] = "demi"; /* Μέγεθος 5 */
```

Πίνακες 2D

- ❖ Για διδιάστατους (τριδιάστατους κλπ), κατά τη δήλωση χρησιμοποιούνται πολλαπλές αγκύλες για να δηλώσουν το μέγεθος κάθε διάστασης.

```
int array1[10][20];           /* 10 γραμμών, 20 στηλών */  
  
int main() {  
    int array2[30][20];         /* 30 γραμμών, 20 στηλών */  
  
    array1[0][0] = array2[10][10];  
    array1[5] = array2[3]; /* Δεν επιτρέπεται */  
}
```

- ❖ Οι πίνακες αυτοί είναι ΠΡΑΓΜΑΤΙΚΑ διδιάστατοι και ΣΤΑΘΕΡΟΙ
 - Όχι σαν τη java όπου βασικά πρόκειται για διάνυσμα όπου κάθε στοιχείο του είναι άλλο διάνυσμα (γραμμή).
 - Όλες οι γραμμές ίδιου (αμετάβλητου) μεγέθους (σε αντίθεση με τη java που κάθε γραμμή μπορούσε να έχει διαφορετικό μέγεθος).

Η γλώσσα C

Συναρτήσεις



ΜΥΥ502

- ❖ Πρόκειται απλά για μια ακολουθία εντολών την οποία χρησιμοποιούμε σε αρκετά σημεία του προγράμματός μας και την οποία την ξεχωρίζουμε ως «υποπρόγραμμα», ώστε να μπορεί να κληθεί και να εκτελεστεί όσες φορές θέλουμε.
- ❖ «Μέθοδοι» στη Java.
- ❖ Στη C δεν αποτελούν μέρος καμίας κλάσης (δεν υπάρχουν κλάσεις), καλούνται από παντού.



Τα 3 βασικά σημεία των συναρτήσεων: (α) πρωτότυπο

❖ Πρωτότυπο (prototype) – **function declaration**

- **δήλωση** (declaration) που περιγράφει τι χρειάζεται η συνάρτηση για να λειτουργήσει και τι είδους αποτέλεσμα θα επιστρέψει:

<τύπος_επιστροφής> όνομα_συνάρτησης (<τύπος_παραμέτρου> παράμετρος, ...) ;

Π.χ.

```
int power(int base, int exponent);
```

❖ Είναι απαραίτητο το πρωτότυπο;

- Απάντηση: όχι πάντα, αλλά αν δεν το γράψετε μπορεί να δημιουργηθούν προβλήματα.
- Θεωρείστε το ισοδύναμο με τη **δήλωση μίας μεταβλητής** (η οποία πρέπει να οριστεί / δηλωθεί πριν την χρησιμοποιήσετε)

Τα 3 βασικά σημεία των συναρτήσεων: (β) ορισμός

❖ Ορισμός / υλοποίηση της συνάρτησης – **function definition**

```
<πρωτότυπο>          /* Προσοχή: χωρίς το ';' */
{
    ...
    /* οι εντολές της συνάρτησης */
    return (<τιμή>);
}
```

❖ Η **return**: τερματίζει τη συνάρτηση και επιστρέφει το αποτέλεσμα.

- Τι σημαίνει επιστρέφει το αποτέλεσμα?
- Αν έχω στη **main()** : “εκτέλεσε τη συνάρτηση **power** για τα 2, 4” τότε ό,τι τιμή γίνεται **return** αποθηκεύεται σε μια προσωρινή θέση μνήμης **tmp**.
 - ✧ **res = power(2, 4)** → **res = tmp** όπου στο **tmp** έχει μπει το 2^4 , δηλ. το 16

❖ Ειδικός τύπος συνάρτησης: **void**

- Αν η συνάρτηση είναι τύπου **void** τότε ΔΕΝ επιστρέφει τίποτε
- Έχουμε σκέτο:
return;

Τα 3 βασικά σημεία των συναρτήσεων: (γ) κλήση

❖ Κλήση μίας συνάρτησης – **function call**

- Μέσα στο πρόγραμμα
 - ✧ “εκτέλεσε τη συνάρτηση power για τα 2, 4 και υπολόγισε το αποτέλεσμα”
- Τρόπος κλήσης:
`<όνομα_συνάρτησης>(τιμές για πραγματικές παραμέτρους)`

Παράδειγμα

```
#include <stdio.h>

int main() {
    int x, y, k, n, res;
    int power(int base, int n); /* πρωτότυπο */

    x = 2; y=3; k=4; n=2;
    res = power(x, k); /* κλήση */
    printf("%d \n", res);
    res = power(y, n); /* κλήση */
    printf("%d \n", res);

    return 0; /* Η main() πρέπει να επιστρέψει 0 αν όλα OK */
}

/* Ορισμός (υπολογίζει το base υψωμένο στη δύναμη n, basen) */
int power(int base, int n) {
    int i, p;
    for (i = p = 1; i <= n; i++)
        p = p*base;
    return p; /* τιμή επιστροφής */
}
```

Παράμετροι (parameters)
ή
τυπικές παράμετροι
(formal parameters)

Ορίσματα (arguments) ή
πραγματικές παράμετροι

Παράδειγμα – αλλού το πρωτότυπο

```
#include <stdio.h>
int power(int base, int n);           /* πρωτότυπο */

int main() {
    int x, y, k, n, res;

    x = 2; y=3; k=4; n=2;
    res = power(x, k);             /* κλήση */
    printf("%d \n", res);
    res = power(y, n);             /* κλήση */
    printf("%d \n", res);

    return 0;                      /* Η main() πρέπει να επιστρέψει 0 αν όλα OK */
}

/* Ορισμός (υπολογίζει το  $base^n$ ) */
int power(int base, int n) {
    int i, p;
    for (i = p = 1; i <= n; i++)
        p = p*base;
    return p;                      /* τιμή επιστροφής */
}
```

ΔΙΑΦΟΡΑ ΜΕ ΠΡΙΝ:

Όχι μόνο η main(), αλλά ΟΛΕΣ οι επόμενες συναρτήσεις θα «γνωρίζουν» την power().

Πριν, ΜΟΝΟ στη main() είχε γίνει γνωστή!

Παράδειγμα χωρίς πρωτότυπο. Τι θα γίνει;

```
#include <stdio.h>

int main() {
    int x, y, k, n, res;

    x = 2; y=3; k=4; n=2;
    res = power(x, k);           /* κλήση */
    printf("%d \n", res);
    res = power(y, n);           /* κλήση */
    printf("%d \n", res);

    return 0;                    /* Η main() πρέπει να επιστρέψει 0 αν όλα OK */
}

/* Ορισμός (υπολογίζει το basen) */
int power(int base, int n) {
    int i, p;
    for (i = p = 1; i <= n; i++)
        p = p*base;
    return p;                   /* τιμή επιστροφής */
}
```

ΑΠΟΤΕΛΕΣΜΑ:

Compiler **warnings**: δεν «γνωρίζει» την `power()` στα σημεία που γίνονται οι δύο κλήσεις. Πολλές φορές υπάρχουν και καταστροφικά αποτελέσματα.



Παράδειγμα χωρίς πρωτότυπο. Τι θα γίνει;

```
#include <stdio.h>

/* Ορισμός (υπολογίζει το  $base^n$ ) */
int power(int base, int n) {
    int i, p;
    for (i = p = 1; i <= n; i++)
        p = p*base;
    return p;          /* τιμή επιστροφής */
}

int main() {
    int x, y, k, n, res;

    x = 2; y=3; k=4; n=2;
    res = power(x, k);           /* κλήση */
    printf("%d \n", res);
    res = power(y, n);           /* κλήση */
    printf("%d \n", res);

    return 0;                    /* Η main() πρέπει να επιστρέψει 0 αν όλα OK */
}
```

ΑΠΟΤΕΛΕΣΜΑ:

Μιας και η συνάρτηση ορίστηκε ΠΡΙΝ τα σημεία των δύο κλήσεων, ΘΕΩΡΕΙΤΑΙ ΓΝΩΣΤΗ.

Επομένως δεν δημιουργείται πρόβλημα και άρα δεν είναι απαραίτητο το πρωτότυπο.

ΜΗΝ ΤΟ ΚΑΝΕΤΕ!!

ΠΑΝΤΑ ΝΑ ΓΡΑΦΕΤΕ ΤΑ ΠΡΩΤΟΤΥΠΑ!!

```
int power(int base, int n) {  
    int i, p;  
    for (i = p = 1; i <= n; i++)  
        p = p*base;  
    return p;  
}
```

- ❖ Κανένα πρόβλημα;
 - Τι γίνεται αν $n < 0$?
- ❖ Πώς θα το áλλαζα για να ελέγχει αν το n είναι θετικός;

```
if (n < 0) {  
    return (-1);  
}
```

Έτοιμες μαθηματικές συναρτήσεις στη C

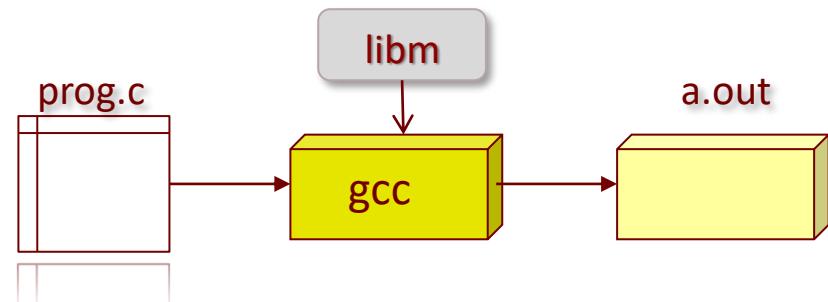
❖ Πρέπει:

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

int main()
{
    printf("cos(0) = %lf\n", cos(0.0));
    return 0;
}
```

❖ Επίσης, κατά τη μεταγλώττιση, πρέπει να συμπεριληφθεί και η βιβλιοθήκη των μαθηματικών συναρτήσεων με το **-lm**:

```
% gcc prog.c -lm
% ls
a.out prog.c
```



Συνήθεις μαθηματικές συναρτήσεις

```
double acos(double);           double sqrt(double);
double asin(double);           double exp(double);
double atan(double);          double pow(double, double);
double cos(double);            double log(double);
double cosh(double);          double log10(double);
double sin(double);            double fabs(double);
double sinh(double);          double tan(double);
double tanh(double);
```

Πέρασμα Παραμέτρων

- ❖ Οι μεταβλητές βασικού τύπου (int, char, float, ...) περνιούνται ως παράμετροι σε μια συνάρτηση **με τιμή (by value)** .
 - δηλαδή αντιγράφονται οι τιμές τους σε τοπικές μεταβλητές της συνάρτησης.
 - όποιες αλλαγές γίνουν σε αυτές τις τιμές των μεταβλητών στο εσωτερικό της συνάρτησης δεν είναι εμφανείς στο κομμάτι του προγράμματος στο οποίο έγινε η κλήση της συνάρτησης.
- ❖ Αν θέλουμε να ξεπεράσουμε τον παραπάνω περιορισμό, δηλ. ότι αλλαγές γίνουν στις τιμές των μεταβλητών στο εσωτερικό της συνάρτησης να είναι εμφανείς στο κομμάτι του προγράμματος στο οποίο έγινε η κλήση της συνάρτησης, μπορούμε να περάσουμε τις διευθύνσεις των θέσεων μνήμης που έχουν δεσμευθεί για τις μεταβλητές, να κάνουμε όπως λέμε **πέρασμα με αναφορά (by reference, με χρήση παραμέτρων τύπου pointer)**.
- ❖ Οι παράμετροι τύπου **πίνακα** πάντα περνιούνται **με αναφορά**.
 - όποιες αλλαγές γίνουν στα στοιχεία του πίνακα είναι εμφανείς στο κομμάτι του προγράμματος στο οποίο έγινε η κλήση της συνάρτησης.

swap

```
#include <stdio.h>
void swap (int x, int y);
```

```
#include <stdio.h>
void swap (int *x1, int *x2);
```

```
int main() {
    int a,b;
    a = 6;
    b = 7;
    swap(a, b); —————>
    printf("%d %d\n", a, b);
    return 0;
}
```

```
int main() {
    int a,b;
    a = 6;
    b = 7;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
```

```
void swap (int x, int y) { —————>
    int temp;
    temp = x;
    x = y;
    y = temp;
    return;
}
```

```
void swap (int *x1, int *x2) { —————>
    int temp;
    temp = *x1;
    *x1 = *x2;
    *x2 = temp;
    return;
}
```

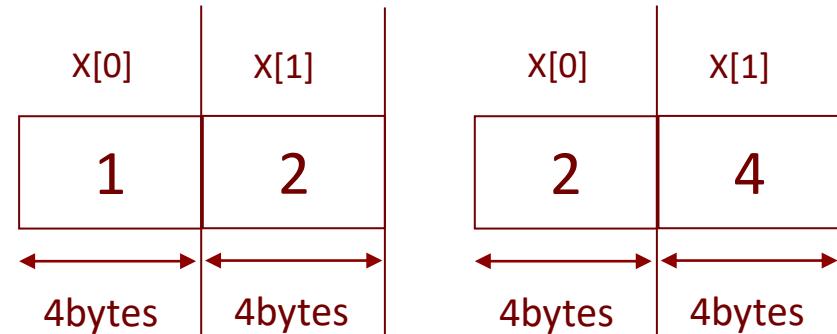
```
Parallel Processing Group  
UNIVERSITY OF IOANNINA
```

Παράδειγμα πέρασμα by reference με πίνακα

```
#include <stdio.h>
void f(int a[2]);

int main() {
    int i;
    int x[] = {1, 2};
    f(x);
    for (i = 0; i < 2; i++)
        printf("%d\n", x[i]);
    return 0;
}

void f(int k[2]) {
    int i;
    for (i = 0; i < 2; i++)
        k[i] = k[i]*2;
}
```



Στοίβα (stack)

- ❖ Για κάθε κλήση συνάρτησης έχω ένα κομμάτι μνήμης αφιερωμένο σε αυτή
- ❖ Δεσμεύεται αυτόματα κατά την είσοδο στη συνάρτηση και αποδεσμεύεται επίσης αυτόματα κατά την έξοδο από αυτή
- ❖ Ο χώρος μνήμης που χρησιμοποιείται για το σκοπό αυτό καλείται **stack** (στοίβα)
 - Το συγκεκριμένο κομμάτι μνήμης για μια συνάρτηση καλείται **stack frame**
- ❖ Στη μνήμη αυτή (**stack frame**) τοποθετούνται οι παράμετροι και οι τοπικές μεταβλητές μιας συνάρτησης

Η γλώσσα C

*Εμβέλεια και διάρκεια ζωής
μεταβλητών
(scope & lifetime)*



MYY502

Εμβέλεια μεταβλητών (scope / visibility)

- ❖ Τι είναι: Η **εμβέλεια** του ονόματος μιας μεταβλητής είναι το **τμήμα του προγράμματος στο οποίο η μεταβλητή μπορεί να χρησιμοποιηθεί**
 - Global (καθολική)
 - Local (τοπική, σε συνάρτηση)
 - Block (σε δομημένο μπλοκ εντολών { .. })
- ❖ Εμβέλεια τοπικών μεταβλητών → δηλώνονται στην αρχή κάθε συνάρτησης και μπορούν να χρησιμοποιηθούν μόνο μέσα στη συνάρτηση
- ❖ Εμβέλεια block → δηλώνεται στην αρχή ενός δομημένου block { ... } και είναι ορατό μόνο μέσα στο *block* αυτό!
- ❖ Εμβέλεια καθολικών μεταβλητών → από εκεί που δηλώθηκαν μέχρι τέλος αρχείου

Εμβέλεια

```
int i=2, j=7;

int f(int i) {
    return i+3;
}

int g() {
    int k;
    if (j > 5) {
        int j;
        j = f(i);
        k = 2*j;
    }
    return (k);
}

main() {
    i = g();
}
```

Η εμβέλεια προκύπτει ΜΟΝΟ από το κείμενο του κώδικα – απλά παρατηρώντας που είναι τοποθετημένες οι δηλώσεις σε σχέση με τις συναρτήσεις και τα blocks

KAI 'OXI

από το πως εκτελείται ο κώδικας.

Τι θα τυπωθεί;

```
#include <stdio.h>
char color = 'B';

void paintItGreen() {
    char color = 'G';
    printf("color@pIG: %c\n", color);
}

void paintItRed() {
    char color = 'R';
    printf("\n\t-----start pIR----\n\t");
    printf("color@pIR: %c\n\t", color);
    paintItGreen();
    printf("\tcolor@pIR: %c\n\t", color);
    printf("-----end pIR----\n\n");
}

main() {
    printf("\tcolor@main: %c\n", color);
    paintItGreen();
    printf("\tcolor@main: %c\n", color);
    paintItRed();
    printf("\tcolor@main: %c\n", color);
}
```

- ❖ Το **χρονικό διάστημα για το οποίο δεσμεύεται μνήμη για τις μεταβλητές**.
- ❖ **Τοπικές** μεταβλητές: ως την ολοκλήρωση της συνάρτησης στην οποία είναι ορισμένες
- ❖ **Καθολικές**: ως την ολοκλήρωση της εκτέλεσης του προγράμματος
- ❖ **Τοπικές** μεταβλητές:
 - **auto** (το default, υπάρχουν όσο διαρκεί το block / συνάρτηση, μιας και αποθηκεύονται συνήθως στη στοίβα)
 - **static** (διατηρείται ο χώρος ακόμα και αν τελειώσει μία συνάρτηση)

Διάρκεια – μνήμη

```
int i=2, j=7;
```

```
int f(int i) {  
    return i+3;  
}
```

```
int g() {  
    int k;  
    if (j > 5) {  
        int j;  
        j = f(i);  
        k = 2*j;  
    }  
    return (k);  
}
```

```
main() {  
    i = g();  
}
```

Η διάρκεια προκύπτει από την
εκτέλεση του κώδικα.

Μεταβλητές με διάρκεια static

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

```
void f(void) {  
    static int y = 0;  
    y++;  
    printf("%d\n", y);  
}
```

```
int main() {  
    int i;  
    for (i=0; i<5; i++)  
        f();  
    return 0;  
}
```

1. Η αρχικοποίηση γίνεται κατά την εκκίνηση του προγράμματος – δεν γίνεται στην κάθε κλήση της συνάρτησης.
2. Η μεταβλητή συνεχίζει να υπάρχει στη μνήμη ακόμα και μετά τη λήξη της συνάρτησης

Αρχικοποίηση μεταβλητών

❖ Με τη δήλωση:

- Οι καθολικές (global) και οι τοπικές static
 - ✧ Αρχικοποιούνται ΑΥΤΟΜΑΤΑ στο μηδέν (0)
- Οι τοπικές (εκτός των static)
 - ✧ Τυχαία αρχική τιμή (συνήθως σκουπίδια)

❖ Αρχικοποίηση πινάκων:

- `int a[] = {1, 2, ...};`
 - ✧ Από το πλήθος των στοιχείων προκύπτει το μέγεθος του πίνακα
- `int a[5] = {1, 2, ...};`
 - ✧ Αν παραπάνω από 5 στοιχεία στις αγκύλες → Error!
 - ✧ Αν λιγότερα από 5 → 0 στις υπόλοιπες θέσεις του a.

❖ Αρχικοποίηση πινάκων χαρακτήρων:

- `char txt[] = {'A', 'l', 'a', 'l', 'a', '\0'};`
- ή πιο εύκολα: `char txt[] = "Alala";`

Η γλώσσα C

*Εμβέλεια extern &
πολλαπλά αρχεία κώδικα*



MYY502

Πολλαπλά αρχεία κώδικα - scope.c

```
#include <stdio.h>
#define SCALE 2
int number;      /* Δήλωση/ορισμός καθολικής μεταβλητής */
int f(int param); /* Πρωτότυπο */

int main() {
    int y;
    number = 5;
    y= f(4*SCALE);
    printf("%d, %d\n", number, y);
    return 0;
}

int f(int x) {
    int y;
    y = x*SCALE + number;
    return (y);
}
```

Σε δύο αρχεία

scope.c

```
#include <stdio.h>
#define SCALE 2

int number, /* Δήλωση/ορισμός */
    f(int param); /* Πρωτότυπο */

int main() {
    int y;
    number = 5;
    y = f(4*SCALE);
    printf("%d,%d\n",number,y);
    return 0;
}
```

func.c

```
#define SCALE 2

/* Μεταβλητή ορισμένη αλλού */
extern int number;

int f(int x) {
    int y;
    y = x*SCALE + number;
    return (y);
}
```

Με αρχείο επικεφαλίδας

scope.c

```
#include <stdio.h>
#include "myscope.h"

int number;

int main() {
    int y;
    number = 5;
    y = f(4*SCALE);
    printf("%d,%d\n",number,y);
    return 0;
}
```

func.c

```
#include "myscope.h"

int f(int x) {
    int y;
    y = x*SCALE + number;
    return (y);
}
```

myscope.h

```
#define SCALE 2
int f(int param);
extern int number;
```

❖ Χρήση συναρτήσεων από το ένα στο άλλο

- Δήλωση prototype
- Εναλλακτικά, δημιουργία include file με prototypes
- Γενικός ρόλος που παίζουν τα include files?

❖ Πώς τα κάνω compile?

- Είτε όλα μαζί:
 - ✧ gcc *.c
- Είτε με χρήση **Makefile** γιατί πάντα ελέγχει τι έχει αλλάξει

Με Makefile

scope.c

```
#include <stdio.h>
#include "myscope.h"

int number;

int main() {
    int y;
    number = 5;
    y = f(4*SCALE);
    printf("%d,%d\n",number,y);
    return 0;
}
```

func.c

```
#include "myscope.h"

int f(int x) {
    int y;
    y = x*SCALE + number;
    return (y);
}
```

myscope.h

```
#define SCALE 2
int f(int param);
extern int number;
```

Makefile

```
scope: scope.o func.o myscope.h
        gcc -o scope scope.o func.o

func.o: func.c myscope.h
        gcc -c func.c

scope.o: scope.c myscope.h
        gcc -c scope.c
```

scope.c

Makefile

```
scope: scope.o func.o myscope.h  
        gcc -o scope scope.o func.o  
  
func.o: func.c myscope.h  
        gcc -c func.c  
  
scope.o: scope.c myscope.h  
        gcc -c scope.c
```

Κανόνας

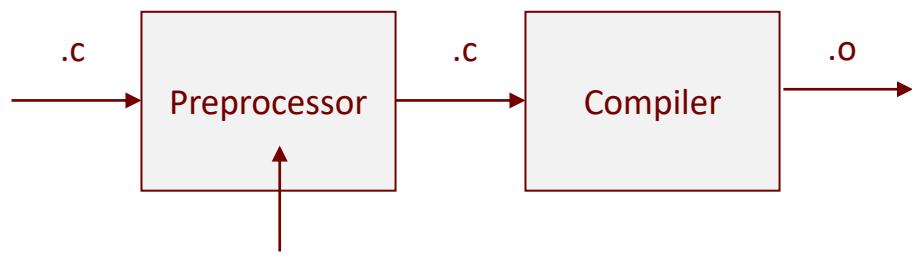
στόχος (target)

αρχεία απ' τα οποία εξαρτάται (προαπαιτούμενα)

εντολή / ενέργεια για να γίνει (η γραμμή αρχίζει με TAB!!!)

ελεογμ \ ελεβλερα λαν λα λανερ (μ λαστικι μ αδχρεσ τε TAB!!!)

Για κάθε αρχείο : `gcc -c x.c`



Αρχεία .o

άλλα αρχεία .o

`gcc -o scope.exe scope.o f.o`



Η γλώσσα C

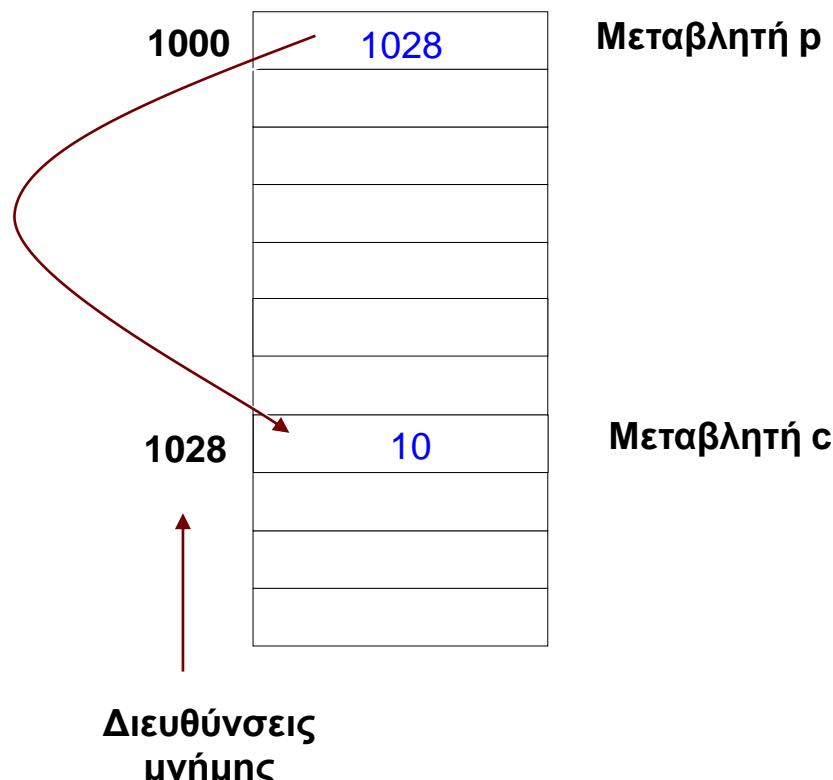
Δείκτες (*pointers*)



ΜΥΥ502

Δείκτες - Pointers

- ❖ Δείκτης: τι είναι;
 - Μια μεταβλητή που περιέχει τη διεύθυνση μιας άλλης μεταβλητής



```
int c = 10;
```

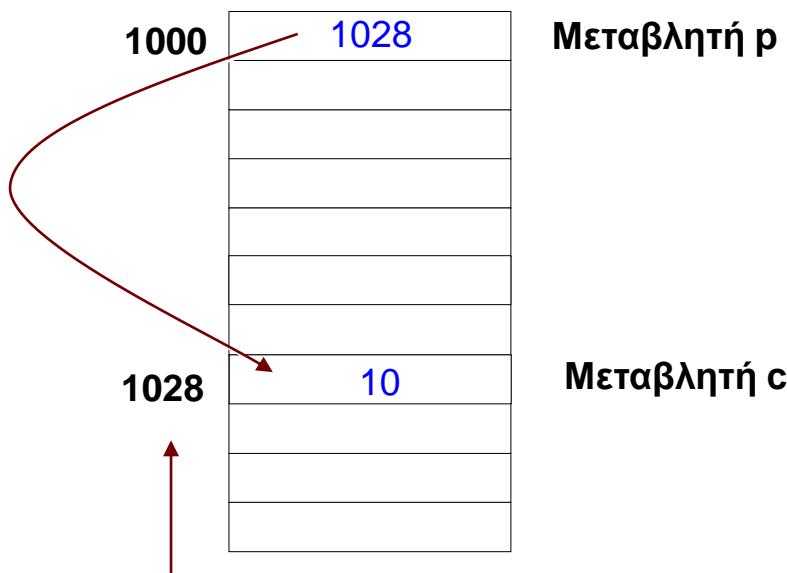
```
int *p;
```

```
p = &c;
```

*p : προσπέλαση του αντικειμένου που δείχνει ο δείκτης p.

Δείκτες - Pointers

- ❖ Τα περιεχόμενα του κελιού στο οποίο δείχνει ο δείκτης p δίνονται από το *p
- ❖ Τα περιεχόμενα του p = η διεύθυνση του κουτιού
- ❖ Η διεύθυνση στην οποία βρίσκεται ο c: &c



```
int c = 10;  
int *p;  
  
p = &c;
```

*p : προσπέλαση του αντικειμένου που δείχνει ο δείκτης p.

Μεταβλητές & μνήμη

```
#include <stdio.h>

int main() {
    int var1;
    char var2[10];

    printf("Address of var1 variable: %p\n", &var1 );
    printf("Address of var2 variable: %p\n", &var2 );
    return 0;
}
```

Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6



Μεταβλητές & μνήμη

```
#include <stdio.h>

int main() {
    int var = 20; /* actual variable declaration */
    int * ip;      /* pointer variable declaration */

    ip = &var; /* store address of var in pointer variable*/
    printf("Address of var variable: %p\n", &var );
    printf("Address stored in ip variable: %p\n", ip );
    printf("Value of *ip variable: %d\n", *ip );
    return 0;
}
```

Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

Αναλογία με αποθήκη προϊόντων

- ❖ Γνωστό κατάστημα διαθέτει αποθήκη με πολλά ράφια όπου σε κάθε ράφι υπάρχει ένα προϊόν (έπιπλο). Τα προϊόντα είναι συσκευασμένα έτσι ώστε ΔΕΝ ΜΠΟΡΕΙΣ να καταλάβεις τι είναι αν πας στα ράφια της αποθήκης. Για να πάρεις το έπιπλο πρέπει να επισκεφτείς την έκθεση του καταστήματος και να σημειώσεις σε ειδικά χαρτάκια το ΡΑΦΙ της αποθήκης που έχει το προϊόν που σε ενδιαφέρει. Με το συμπληρωμένο χαρτάκι πας στο σωστό ράφι και το παίρνεις – αλλιώς δεν μπορείς να βρεις το προϊόν.
- ❖ Αναλογία:
 - **Μνήμη** = αποθήκη
 - **Μεταβλητή** (κελί στη μνήμη) = το ράφι (ο αριθμός του)
 - **Τιμή μεταβλητής** = προϊόν στο ράφι
 - **Δείκτης** = το χαρτάκι



Έννοιες & αναλογίες

Λειτουργία / έννοια	Αναλογία
$p = \&x;$	Γράφω σε χαρτάκι το ράφι.
$*p$	Το προϊόν που υπάρχει στο ράφι («ταξιδεύω» εκεί και το βρίσκω).
$q = \&y;$	Άλλο χαρτάκι που γράφει άλλο ράφι.
$p = q;$	Στο πρώτο χαρτάκι αλλάζω τι έγραψα και γράφω το ίδιο ράφι που γράφει το δεύτερο χαρτάκι.
$temp = *a;$ $*a = *b;$ $*b = *temp;$	«Ταξιδεύω» στα ράφια που γράφουν τα χαρτάκια a και b και εναλλάσσω τα προϊόντα.



Δηλώσεις

- ❖ Δήλωση μεταβλητής: <type> *<name>;
- ❖ Παράδειγμα

```
int x, y;
```

```
int * p;
```

```
y = 3;
```

```
p = &y;
```

```
x = *p + 1;
```

- ❖ Προσοχή: η έκφραση **a*b** δεν περιλαμβάνει δείκτη!

❖ Σωστή αρχικοποίηση

```
int c;  
int * p;  
p = NULL; /* Χρειάζεται το stdio.h */  
p = 0; /* Ισοδύναμο με NULL */  
p = &c;
```

❖ Λάθος

```
p = 100;  
p = c;
```

Δήλωση μαζί με αρχικοποίηση δεικτών

- ❖ Μπορείτε να δηλώσετε έναν δείκτη και ταυτόχρονα να τον αρχικοποιήσετε:

```
int c, * p = &c; /* μην το κάνετε, μπερδεύει!! */
```

- Είναι ισοδύναμο με:

```
int c, * p;  
p = &c;
```

- ❖ **ΠΡΟΣΟΧΗ:**

- **Στις δηλώσεις το «*p» σημαίνει ότι το p είναι δείκτης. ΔΕΝ σημαίνει ότι πηγαίνει εκεί που δείχνει και παίρνει το περιεχόμενο!**
- **Μόνο όταν το «*p» εμφανίζεται μέσα σε πράξεις έχει την έννοια του «πηγαίνω και παίρνω το περιεχόμενο»**

- ❖ Για να μην υπάρχει σύγχυση, καλύτερα **να μην αρχικοποιείτε με αυτόν τον τρόπο τους δείκτες**. Το σωστότερο είναι να τους αρχικοποιείτε πάντα στην τιμή NULL και μετά να κάνετε ότι τροποποιήσεις θέλετε:

```
int c, * p = NULL;  
p = &c;
```

Δήλωση δείκτη

- ❖ Παράδειγμα:

```
char * ptr; /* Ισοδύναμα: char* ptr, char *ptr */
```

- ❖ «Διαβάζοντας» τη δήλωση:

```
char * ptr;
```



Πρόκειται για μία **μεταβλητή** που ονομάζεται `ptr`.

Η μεταβλητή είναι **δείκτης**.

Η θέση στην κύρια μνήμη στην οποία θα δείχνει, αποθηκεύει έναν **char**.

ελαλ υπακ

ουορα θα ρετχει' αυτοθικερει

Δείκτες και πέρασμα παραμέτρων

❖ Παραδείγματα:

```
void foo(char *s);
```

```
void bar(int *result);
```

❖ Γιατί να περάσω δείκτη;

- Κλήση δια αναφοράς (call by reference)
- Αν θέλω, δηλαδή, η συνάρτηση να κάνει αλλαγές που να επηρεάζουν τα πραγματικά δεδομένα-ορίσματά της (δηλ. τις μεταβλητές της καλούσας συνάρτησης)
- Επίσης, αν θέλω να περάσω ως όρισμα μία μεγάλη δομή αποφεύγοντας τη χρονοβόρα αντιγραφή στη στοίβα (αργότερα αυτά)

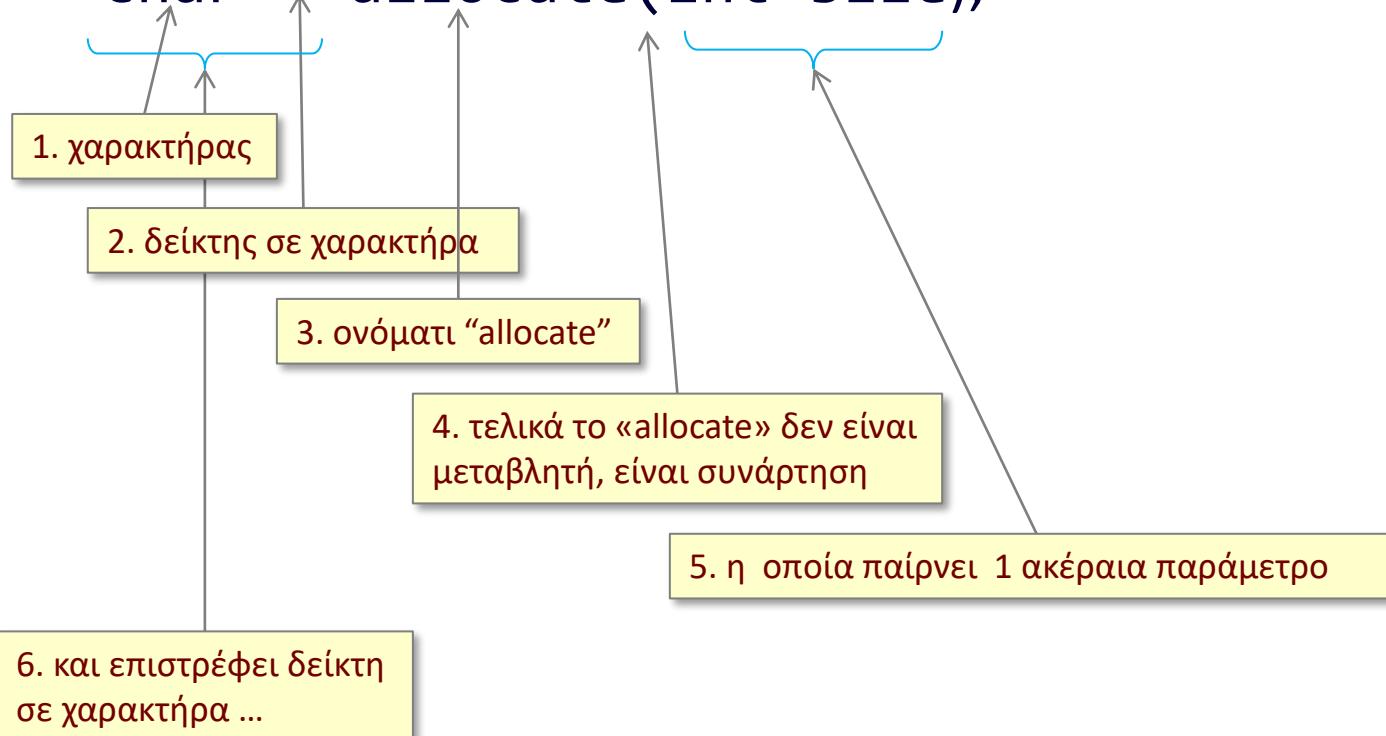
Συναρτήσεις που επιστρέφουν δείκτη

❖ Παράδειγμα:

```
char * allocate(int size);
```

❖ «Διαβάζοντας» τη δήλωση:

```
char * allocate(int size);
```



Ερωτήματα

❖ Υποθέστε τη δήλωση «`int x;`»

Τι ακριβώς κάνουν τα παρακάτω αν βρεθούν σε μια έκφραση;

- `&x;`
- `*(&x);`
- `&(*x);`

❖ Τι μεταβλητή χρειάζομαι για να βάλω μέσα τη διεύθυνση ενός δείκτη;

Κι άλλα ερωτήματα...

```
int i = 3, j = 5, k, * p=NULL, * q=NULL, * r=NULL;  
p = &i; q = &j;
```

* * & p

ισοδύναμο: $\ast(\ast(\&p))$

αποτέλεσμα: 3 (αφού είναι ισοδύναμο με $\ast(p)$)

3 * - * p / * q + 7

ισοδύναμο: $((3 \ast (- (*p))) / (*q)) + 7$

αποτέλεσμα: 6

3 * - * p /* q + 7

ισοδύναμο: λάθος! Το /* ξεκινά σχόλιο!!

* (r = & k) = *p * * q

ισοδύναμο: $(\ast(r = \&k)) = ((\ast p) * (\ast q))$

αποτέλεσμα: 15

Προτεραιότητες τελεστών C

Καλύτερα να μην θυμάστε τη σειρά!

Να καθορίζετε μόνοι σας τη σειρά των πράξεων βάζοντας στα σωστά σημεία

ΠΑΡΕΝΘΕΣΕΙΣ.

Operator	Description	Associativity
()	Function call	Left to right
[]	Array element	
->	Structure member pointer reference	
.	Class, structure or union member reference	
sizeof	Storage size in bytes of object / type	
++	Postfix Increment	
--	Postfix Decrement	
++	Prefix Increment	Right to left
--	Prefix Decrement	
-	Unary minus	
+	Unary plus	
!	Logical negation	
~	One's complement	
&	Address of	
*	Indirection	
(type)	Type conversion (cast)	Right to left
*	Multiplication	Left to right
/	Division	
%	Modulus	
+	Addition	Left to right
-	Subtraction	
<<	Bitwise left shift	Left to right
>>	Bitwise right shift	
<	Scalar less than	Left to right
<=	Scalar less than or equal to	
>	Scalar greater than	
>=	Scalar greater than or equal to	
==	Scalar equal to	Left to right
!=	Scalar not equal to	
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
	Bitwise inclusive OR	Left to right
&&	Logical AND	Left to right
	Logical inclusive OR	Left to right
? :	Conditional expression	Right to left
=	Assignment	Right to left
+= -= *=	Assignment	
/= %= &=	Assignment	
^= =	Assignment	
<<= >>=	Assignment	
,	Comma	Left to right

Πέρασμα παραμέτρων με αναφορά – swap

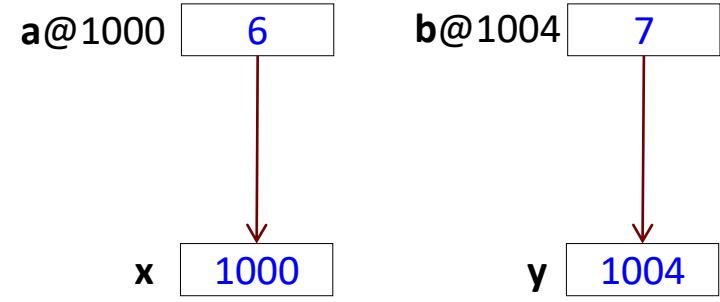
```
#include <stdio.h>
void swap (int x, int y);
int main() {
    int a, b;
    a = 6;
    b = 7;
    swap(a, b);
    printf("%d %d\n", a, b);
    return 0;
}
void swap (int x, int y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
    return;
}
```

Πέρασμα παραμέτρων με αναφορά – swap

```
#include <stdio.h>
void swap (int x, int y);
int main() {
    int a, b;
    a = 6;
    b = 7;
    swap(a, b);           → swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
void swap (int x, int y) {           void swap (int *x1, int *x2) {
    int temp;
    temp = x;             }
    x = y;               →   { temp = *x1;
    y = temp;             *x1 = *x2;
    return;               *x2 = temp;
}
```

Swap με δείκτες

```
#include <stdio.h>
void swap (int *x, int *y);
int main() {
    int a, b;
    a = 6;
    b = 7;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
void swap (int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```



Τα **x**, **y** περιέχουν διευθύνσεις

Swap με δείκτες – εναλλακτικός κώδικας

```
#include <stdio.h>
void swap (int *x, int *y);
int main() {
    int a, b;
    a = 6;
    b = 7;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
void swap (int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

```
#include <stdio.h>
void swap (int *x, int *y);
int main() {
    int a = 6, b = 7, *pa, *pb;
    pa = &a;
    pb = &b;
    swap(pa, pb);
    printf("%d %d\n", a, b);
    return 0;
}
void swap (int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

Παράδειγμα

```
#include <stdio.h>
int max(int x, int y);

int main() {
    int a, b, res;
    a = 5;
    b = 3;
    res = max(a, b);
    return 0;
}

int max(int x, int y) {
    int res;
    if (x > y) res = x;
    else res = y;
    return res;
}
```

```
#include <stdio.h>
void max(int x, int y, int * res);

int main() {
    int a, b, res;
    a = 5;
    b = 3;
    max(a, b, &res);
    return 0;
}

void max(int x, int y, int *res) {
    if (x > y) *res = x;
    else *res = y;
}
```

Παράδειγμα

```
#include <stdio.h>
void init(int * x, int * y);

int main() {
    int a, b;
    init(&a, &b);
    return 0;
}

void init(int *x, int *y) {
    *x = 12;
    *y = 15;
}
```

Πέρασμα παραμέτρων με αναφορά - πίνακες

- ❖ Αν θυμάστε, το πέρασμα των πινάκων σε μία συνάρτηση γίνεται πάντα με αναφορά.
 - Αυτό που γίνεται είναι ότι περνιέται **ΜΟΝΟ ΕΝΑΣ ΔΕΙΚΤΗΣ ΣΤΟ ΠΡΩΤΟ ΣΤΟΙΧΕΙΟ ΤΟΥ ΠΙΝΑΚΑ** (μπορείτε να φανταστείτε γιατί;;)
 - Τα παρακάτω είναι ισοδύναμα:

```
void initzero(int x[100]) {    // Αγνοείται το μέγεθος!
    x[0] = 10;
}
void initzero(int x[]) {        // Άρα δεν χρειάζεται
    x[0] = 10;
}
void initzero(int *x) {        // Είναι απλός δείκτης
    *x = 10;
}
```

- ❖ Μόνο σε παράμετρο συνάρτησης επιτρέπεται να μην δοθεί το πλήθος των γραμμών ενός πίνακα διότι ΑΓΝΟΕΙΤΑΙ – ο πίνακας περνιέται ως ένας απλός δείκτης, χωρίς πληροφορία για το πλήθος των στοιχείων του πίνακα.

Παράδειγμα

```
#include <stdio.h>
void initarr(int x[10]);

int main() {
    int a[10];
    initarr(a);
    return 0;
}

void initarr(int x[10]) {
    int i;
    for (i = 0; i < 10; i++)
        x[i] = i;
}
```

```
#include <stdio.h>
void initarr(int x[], int n);

int main() {
    int a[10];
    initarr(a, 10);
    return 0;
}

void initarr(int x[], int n) {
    int i;
    for (i = 0; i < n; i++)
        x[i] = i;
}
```

Παρατήρηση

- ❖ Δεν έχει νόημα μια συνάρτηση να επιστρέφει δείκτη που δείχνει τοπικά, δηλαδή τη διεύθυνση μιας τοπικής μεταβλητής:

```
int * test1() {  
    int i;  
    return &i;  
}
```

- ❖ Έχει διαφορά η παρακάτω περίπτωση;

```
int * test2() {  
    static int i;  
    return &i;  
}
```

Παρένθεση - υπενθύμιση

- ❖ Το α και β παρακάτω είναι ίδια; Αν όχι υπάρχει κάποιο πρόβλημα;

(a)	(b)
<code>int x, * y = &x;</code>	<code>int x, * y; *y = &x;</code>

- ❖ Θυμηθείτε ότι άλλο σημαίνουν οι τελεστές της C μέσα σε μία ΔΗΛΩΣΗ και άλλο μέσα σε μία εντολή/πράξη. Το (b) λοιπόν είναι διαφορετικό (και λάθος). Το (a) θα ήταν ισοδύναμο με το (c):

(a)	(c)
<code>int x, * y = &x;</code>	<code>int x, * y; y = &x;</code>

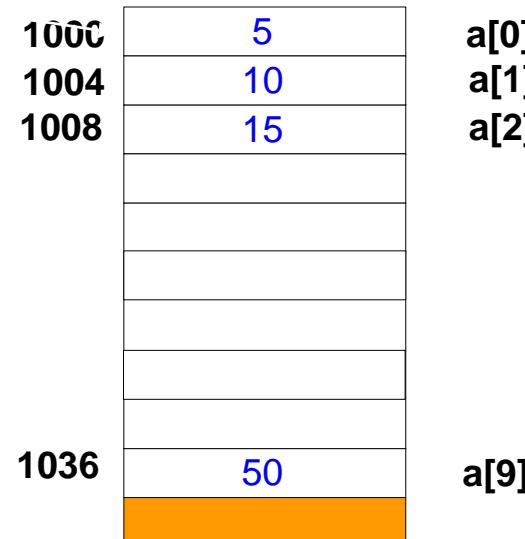
- ❖ Καλό είναι να μην κάνουμε αρχικοποίηση ενός δείκτη κατά τη δήλωσή του (παρά μόνο με *NULL*).

Δείκτες και Πίνακες

❖ Παράδειγμα:

```
int * p=NULL;  
int a[10] = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};  
p = &a[0]; /* διεύθυνση 1ου στοιχείου του πίνακα */  
p = a; /* ισοδύναμο με το παραπάνω */  
p[i] <=> a[i] ⇔ *(p+i)
```

❖ Τι είναι η μεταβλητή a? Είναι ένας **σταθερός** δείκτης που αντιστοιχεί σε μια διεύθυνση. Ο σταθερός δείκτης δεν μπορεί να δείξει κάπου αλλού.



Ο τελεστής []

- ❖ Μέχρι τώρα γνωρίζαμε ότι ο τελεστής [] χρησιμοποιείται για επιλογή κάποιου στοιχείου ενός πίνακα, π.χ. το a[3] υποδηλώνει το 3^o στοιχείο του πίνακα.
- ❖ Όπως αυτή είναι η «μισή» αλήθεια:
 - Βασικά το a[3] σημαίνει το στοιχείο που είναι 3 θέσεις μετά από εκεί που δείχνει (ξεκινά) ο δείκτης a.
 - Επομένως, μπορεί να χρησιμοποιηθεί με γενικούς δείκτες και όχι μόνο με πίνακες!
 - Π.χ. αν $p = &a[0]$, μπορώ να πω $p[1] = 4$;
 - Π.χ. αν $p = &a[2]$, μπορώ να πω $p[1] = 4$;
- ❖ Η έκφραση **A[B]** είναι ισοδύναμη με ***(A+B)**
 - ✧ “Μυστικό”: επομένως το a[2] είναι ισοδύναμο με το 2[a] !!!

Παράδειγμα

```
#include <stdio.h>
main() {
    int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int * p=NULL, * q=NULL;
    int n = 3;

    p = & src[0];
    q = & src[2];
    printf("%d %d\n", *p, p[0]);
    printf("%d %d\n", *q, q[0]);

    p[n] += 11;
    printf("%d\n", p[n]);

    q[n] += 11;
    printf("%d\n", q[n]);
}
```

Εκτυπώσεις:

1 1

3 3

15

17

❖ Μπορούμε να έχουμε εκφράσεις με δείκτες

- $p < q$ Ποιος δείχνει πιο μπροστά και ποιο πιο πίσω?
- $p + n$ Το κελί που βρίσκεται η κελιά πιο μετά
- $p - n$ Το κελί που βρίσκεται η κελιά πιο πριν
- $p - q$ Πόσο μακριά βρίσκονται (σε κελιά)?

❖ Άλλες δυνατές εκφράσεις

- $p = p + n$
- $p += n$

❖ Δεν δουλεύουν:

- $p + q$
- p^*4

Αριθμητική δεικτών

```
char c, * pc=NULL;  
int i, * pi=NULL;  
  
pc = &c; pi = &i;  
printf("pc = %p, pc+1 = %p", pc, pc+1);  
printf("pi = %p, pi+1 = %p", pi, pi+1);
```

Δίνει:

```
pc = 251af3c8, pc+1 = 251af3c9  
pi = 251af3c4, pi+1 = 251af3c8
```

Γιατί:

Διότι το +1 δεν είναι +1 byte αλλά +1 τύπος (όσα bytes πιάνει ο τύπος του δείκτη)

```
printf("char: %d bytes", sizeof(char));  
printf("int: %d bytes", sizeof(int));
```

Δίνει:

```
char: 1 bytes  
int: 4 bytes
```

Παράδειγμα

```
#include <stdio.h>
main() {
    int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int * p=NULL, * q=NULL;
    int n = 3;

    p = & src[0];
    q = & src[2];
    printf("%d %d\n", *p, p[0]);
    printf("%d %d\n", *q, q[0]);

    *(p+n) += 11;
    printf("%d\n", p[n]);
    .....  

}
```

Εκτυπώσεις:

1 1	*(p+n) += 11;
3 3	printf("%d\n", p[n]);
15
17	*(q+n) += 11;

Συνεχίζεται...

Παράδειγμα

```
#include <stdio.h>
main() {
    int src[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int * p=NULL, * q=NULL;
    int n = 3;           ... από πριν
.....
if (q > p)
    printf("%p\n", q - p);           2
else
    printf("%p\n", p - q);

p++;
q+=2;

printf("%d %d\n", *p, p[0]);           2 2
printf("%d %d\n", *q, q[0]);           5 5
}
```

Εκτυπώσεις:

Δείκτες και πίνακες, πάλι

- ❖ Υποθέτουμε ότι έχουν δηλωθεί: `int a[10], *p;`
- ❖ **Ισχύει:**

$$\&a[i] \Leftrightarrow a+i$$

$$a[i] \Leftrightarrow *(a+i)$$

- ❖ Αν έχω εκτελέσει την εντολή: `p = a`, τότε **Ισχύει:**

$$\&a[i] \Leftrightarrow a+i \Leftrightarrow \&p[i] \Leftrightarrow p+i$$

$$a[i] \Leftrightarrow *(a+i) \Leftrightarrow p[i] \Leftrightarrow *(p+i)$$

- ❖ Ενώ ισχύει η έκφραση `p++`, **δεν ισχύουν** οι εκφράσεις: **a++** και **a = p**,
 - παρά μόνο αν το `a[]` είναι παράμετρος σε μια συνάρτηση

Παράδειγμα, πάλι

```
#include <stdio.h>
void initarr(int x[], int n);

int main() {
    int a[10];
    initarr(a, 10);
    return 0;
}
```

```
void initarr(int x[], int n) {
    int i;
    for (i = 0; i < n; i++)
        x[i] = i;
}
```

```
#include <stdio.h>
void initarr(int *x, int n);
```

```
int main() {
    int a[10];
    initarr(a, 10);
    return 0;
}
```

```
void initarr(int *x, int n) {
    int i;
    for (i = 0; i < n; i++)
        x[i] = i;
}
```

Παρένθεση: ο τελεστής sizeof

- ❖ Το sizeof() (το οποίο δεν είναι συνάρτηση αλλά **ΤΕΛΕΣΤΗΣ** της C), επιστρέφει το μέγεθος σε BYTES είτε ενός τύπου είτε μίας μεταβλητής:
- ❖ Τι θα τυπωθεί παρακάτω (υποθέστε 32-μπιτο σύστημα);

```
char x, s[5], * p = NULL;
int y, a[10], * q = NULL;
p = &x; q = a;
printf("char: %d, int: %d", sizeof(char), sizeof(int));
    char: 1, int: 4
printf("x: %d, y: %d", sizeof(x), sizeof(y));
    x: 1, y: 4
printf("s: %d, a: %d", sizeof(s), sizeof(a));
    s: 5, a: 40
printf("p: %d, q: %d", sizeof(p), sizeof(q));
    p: 4, q: 4
printf("*p: %d, *q: %d", sizeof(*p), sizeof(*q));
    p: 1, q: 4
```

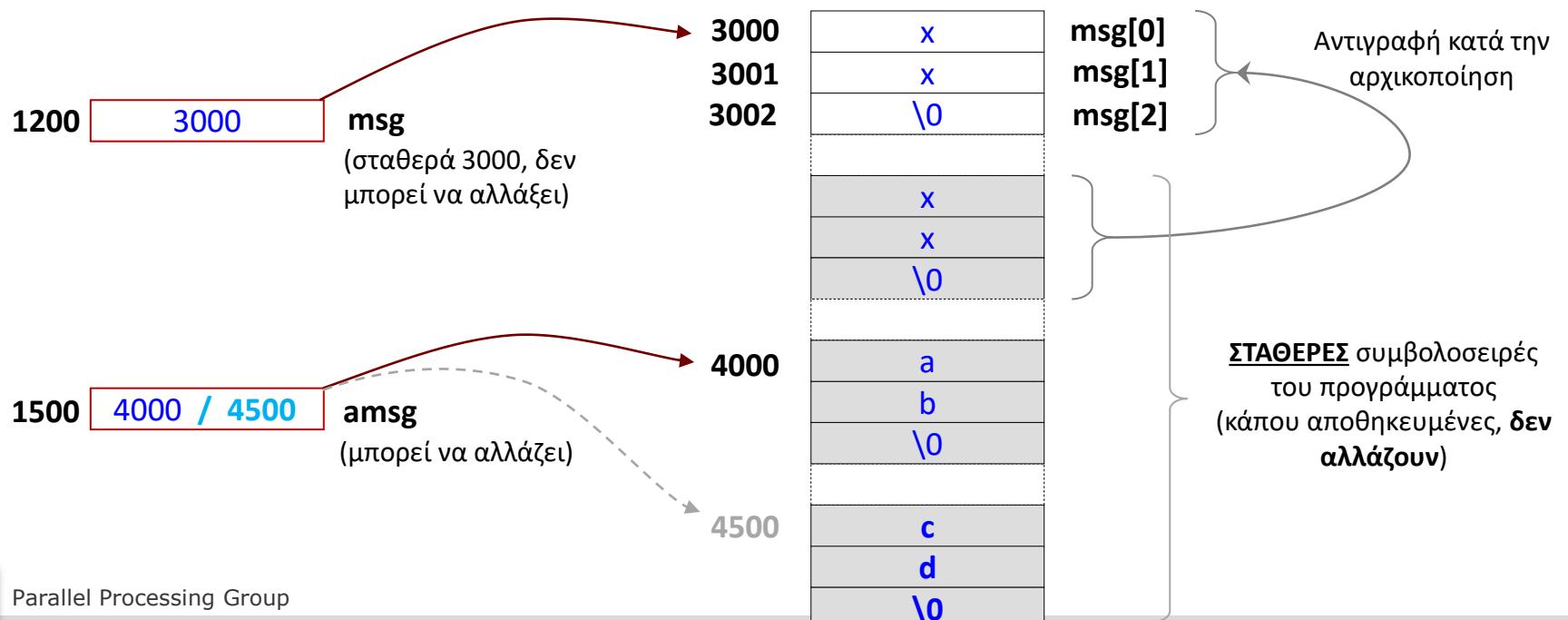
Τρικ με pointers

- ❖ Πώς μπορώ να δω αν το μηχάνημά μου αποθηκεύει με σειρά big endian ή little endian?
- ❖ *Homework ;-)*

Δείκτες και συμβολοσειρές

(ΥΠΕΝΘΥΜΙΣΗ: Κάθε string περιέχει και έναν παραπάνω χαρακτήρα, το '\0')

```
char msg[] = "xx"; /* Πίνακας 3 στοιχείων με αρχικοποίηση */
msg = "yyyy";        /* Δεν επιτρέπεται (σταθερός δείκτης)! */
char * amsg = "ab"; /* Δείκτης σε σταθερή συμβολοσειρά */
amsg = "cd";         /* OK - δείχνει σε άλλο χώρο! */
```



str.c

```
#include <stdio.h>
int main() {
    char buf[] = "hello";
    char * ptr = "geia sou";
    char * ptr1 = "file mou";
    buf[2] = '$';
    printf("%s\n", buf);

    ptr[2] = '%'; /* crash (why??) */

    ptr = ptr1;
    printf("%s\n", ptr);

    ptr[2] = '#'; /* crash (why??) */
    buf = ptr; /* compile error (why??) */

    ptr = buf;
    ptr[2] = 'l';
    printf("%s\n", ptr);

    printf("buf occupies %d bytes in memory.\n", sizeof(buf)); /* ??? */
    printf("ptr occupies %d bytes in memory.\n", sizeof(ptr)); /* ??? */
    return 0;
}
```

Παραδείγματα

```
#include <stdio.h>
void strcpy1(char s[], char d[]);

int main() {
    char x[30] = "abcdef";
    char y[10];
    strcpy1(x, y);
    printf("%s", y);
    return 0;
}

void strcpy1(char s[], char d[]) { /* Copy s to d */
    int i;
    i = 0;
    while (s[i] != '\0') {
        d[i] = s[i];
        i++;
    }
    d[i] = '\0';
}
```

Αντιγραφή – Εναλλακτικά

```
#include <stdio.h>
void strcpy2(char * s, char * d);

int main() {
    char x[30] = "abcdef";
    char y[10];
    strcpy2(x, y);
    printf("%s", y);
    return 0;
}

void strcpy2(char * s, char * d) { /* Copy s to d */
    while (*s != '\0') {
        *d = *s;
        s++; d++;
    }
    *d = '\0';
}
```

Αντιγραφή – Εναλλακτικά

```
void strcpy3 (char s[], char d[]) {  
    int i = 0;  
    while ((d[i] = s[i]) != '\0') {  
        i++;  
    }  
}
```

```
void strcpy4 (char * s, char * d) {  
    while ((*d = *s) != '\0') {  
        s++; d++;  
    }  
}
```

Σύγκριση

```
int strcmp1(char s1[], char s2[]) {  
    int i;  
    for (i = 0; s1[i] == s2[i]; i++) {  
        if (s1[i] == '\0') /* και τα δύο ίσα με \0 */  
            return 0;  
    }  
    return (s1[i] - s2[i]);  
}
```

- ❖ Η συνάρτηση επιστρέφει: 0 αν ΙΣΑ, < 0 αν $s_1 < s_2$, > 0 αν $s_1 > s_2$.
- ❖ Παράδειγμα: $s_1 \rightarrow "abc"$, $s_2 \rightarrow "abcd"$

Σύγκριση - Εναλλακτικά

```
int strcmp2(char * s1, char * s2) {
    for (; *s1 == *s2; s1++, s2++) {
        if (*s1 == '\0') {
            return 0;
        }
    }
    return *s1 - *s2;
}

int main() {
    char A[N], B[N];
    ...
    strcmp2(A, B);
    ...
}
```

```
int strlen1(char s[]) {  
    int i = 0;  
    while (s[i] != '\0') i++;  
    return i;  
}  
  
int strlen2(char * s) {  
    char *p = s;  
    while (*p != '\0') p++; /* προχωράει 1 byte */  
    return p-s ;  
}
```

Πίνακες Δεικτών (Array of Pointers)

- ❖ Τι είναι;
 - Πίνακας που περιέχει δείκτες
- ❖ Πώς δηλώνεται;

```
<type> * <name> [size];
```
- ❖ Πώς αρχικοποιείται;
 - `int A[10];`
 - `int B[20];`
 - `int * C[2] = { A, B };`
- ❖ Αντί για πίνακα 2 διαστάσεων χρησιμοποιώ πίνακα δεικτών
 - Οικονομία μνήμης όταν η 2^η διάσταση μεταβάλλεται
 - Όταν η 2^η διάσταση είναι άγνωστη

- ❖ Πίνακας με strings διαφορετικών μεγεθών

```
char * months[12] = {"January", "February", "March", ..};
```



Παράδειγμα

```
char * get_month(int i) {  
    static char * months[12] = {"Jan", "Feb", ...};  
    return ( (i<12 && i >=0) ? months[i] : NULL );  
}
```

- ❖ Τι είναι η τιμή NULL που μπορεί να επιστρέψει η συνάρτηση?
- ❖ Τι θα συμβεί αν βγάλω το static;

Επανάληψη – υπενθύμιση: τι δουλεύει και τι όχι;

```
#include <stdio.h>
int main() {
    char buf[] = "hello";                                // Αντιγραφή από σταθερή συμβολοσειρά
    char * ptr = "hello";                                // Δείχνει στον 1ο χαρακτήρα της σταθερ. συμβολ.
    void test1(char s[]);
    void test2(char *s);

    buf = ptr;                                         // Δεν επιτρέπεται να αλλάξει ο «δείκτης» buf
    ptr = buf;                                         // OK, ο δείκτης ptr θα δείχνει όπου και ο buf
    buf = "dolly";                                     // Δεν επιτρέπεται να αλλάξει ο δείκτης buf
    ptr = "dolly";                                     // OK, θα δείχνει όπου είναι η αρχή του "dolly"
    *buf = 'D';                                       // OK, ισοδύναμο με buf[0] = 'D';
    *ptr = 'D';                                       // Δεν επιτρέπεται αλλαγή στη σταθερή συμβολοσ.
    printf("%d", sizeof(buf));                         // 6 (5 χαρακτήρες + το \0)
    printf("%d", sizeof(ptr));                         // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
    test1(buf);
    test1(ptr);
    test2(buf);
    test2(ptr);
    return 0;
}

void test1(char s[]) {                                // Πίνακας-παράμετρος περνιέται ως απλός δείκτης
    printf("%d, %c", sizeof(s), *(s+1));           // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
}

void test2(char * s) {                               // 4 (σε 32-μπιτο, 4 bytes για αποθήκευση δείκτη)
    printf("%d, %c", sizeof(s), s[1]);
```

Είναι όλα OK?

```
#include <stdio.h>          #include <stdio.h>
void max(int x, int y, int * res); void max(int x, int y, int * res);

int main() {
    int a, b, res;
    a = 5;
    b = 3;
    max(a, b, &res);
    return 0;
}

void max(int x, int y, int *res) {
    if (x > y) *res = x;
    else *res = y;
}

int main() {
    int a, b, *res;
    a = 5;
    b = 3;
    max(a, b, res);
    return 0;
}

void max(int x, int y, int *res) {
    if (x > y) *res = x;
    else *res = y;
}
```

Η γλώσσα C

Διαχείριση Συμβολοσειρών



ΜΥΥ502

Συμβολοσειρές (strings)

❖ Θυμόμαστε: τι είναι οι συμβολοσειρές;

- Πίνακας χαρακτήρων (με τη σύμβαση ότι στο τέλος έχει '\0').
 - ✧ char str[10]; /* Άρα 9 χαρακτήρες + το \0 */
- Με εύκολη αρχικοποίηση:
 - ✧ char str[10] = "This is";
- Και με διαχείριση που είναι πιο εύκολη από τη διαχείριση πινάκων, όπως θα δούμε.
 - ✧ Όμως μην ξεχνάμε ότι είναι πίνακες (άρα ισχύουν ότι και για αυτούς).

❖ Τι είναι τα παρακάτω:

char b[10] = "this";

Το b και το "this" είναι strings.

char *c;

Δεν είναι string! Είναι ένας ptr σε χαρακτήρα.

char *d = "that";

Το "that" είναι string (αποθηκευμένο κάπου στη μνήμη) και ο d είναι pointer στον πρώτο χαρακτήρα του d. Λέμε καταχρηστικά ότι και το d είναι string.

Αρχικοποίηση

```
int main() {
    char q[] = "First";
    char r[16] = { 'S', 'e', 'c', 'o', 'n', 'd', '\0' };
    char s[16] = "Third";
    char t[16];
    char *u = "Fifth";
    char *w;

    t = "Fourth"; /* Δεν επιτρέπεται! Πρέπει t[0]='F', t[1]='o' ... */
    w = "Sixth"; /* Επιτρέπεται, μιας και είναι pointer */
    return 0;
}
```

puts/fputs

❖ int puts(char *s);

- Η puts τυπώνει το string s και ένα χαρακτήρα νέας γραμμής ('\n') στην οθόνη.
- Επιστρέφει EOF αν συμβεί κάποιο λάθος, διαφορετικά επιστρέφει μη αρνητική τιμή.

❖ int fputs(char *s, FILE *fp);

- Η fputs τυπώνει το string s στο αρχείο fp (χωρίς επιπλέον \n).
- Για τύπωμα στην οθόνη, περάστε ως αρχείο το **stdout**.
- Επιστρέφει EOF αν συμβεί κάποιο λάθος, διαφορετικά επιστρέφει μη αρνητική τιμή.
- Π.χ. fputs("hi", stdout);



gets/fgets

- ❖ `char *gets(char *s);`
 - Η gets διαβάζει την επόμενη γραμμή εισόδου και την αποθηκεύει στο string s.
 - **Αντικαθιστά τον τερματικό χαρακτήρα νέας γραμμής με '\0'.**
 - Επιστρέφει s, ή NULL αν συναντηθεί το τέλος του αρχείου (EOF) ή αν συμβεί κάποιο λάθος.
 - **Δεν είναι ασφαλής συνάρτηση** (τι γίνεται αν το s[] δεν φτάνει?).

- ❖ `char *fgets(char *s, int num, FILE *fp);`
 - Η gets διαβάζει **μέχρι num-1 χαρακτήρες** από το αρχείο fp (το πληκτρολόγιο είναι το **stdin**) και την αποθηκεύει στο string s.
 - **Αν πληκτρολογήθηκε το newline, τότε γράφεται ΚΑΙ ο χαρακτήρας \n μέσα στο s και αμέσως μετά γράφεται και το '\0'.**
 - Επιστρέφει s, ή NULL αν συναντηθεί το τέλος του αρχείου (EOF) ή αν συμβεί κάποιο λάθος.
 - **Να την προτιμάτε έναντι της gets().**
 - Π.χ. `fgets(str,9,stdin); /* έως 8 χαρακτήρες από πληκτρολόγιο */`

printf / scanf

❖ Εκτύπωση ή διάβασμα με το "%s" στο format

❖ printf:

- "%s" : τυπώνει όλο το string
- "%Ns" τυπώνει τουλάχιστον N χαρακτήρες (με κενά αν χρειαστεί)
- "%.Ms" τυπώνει το πολύ M χαρακτήρες
- "%N.Ms" τυπώνει τουλάχιστον N και το πολύ M χαρακτήρες
- "%-Ns" τυπώνει τουλάχιστον N χαρακτήρες, με αριστερή στοίχιση
- Π.χ. printf("%-3.5s", str);

❖ scanf:

- "%s": διαβάζει μία συμβολοσειρά
- **ΠΡΟΣΟΧΗ:** η συμβολοσειρά τελειώνει όταν συναντηθεί χαρακτήρας κενού (*space, tab, newline*)



Παραδείγματα

```
/* gets example */
#include <stdio.h>
int main() {
    char string [256];
    printf ("Insert your full address: ");
    gets(string);
    printf("Your address is: %s\n", string);
    return 0;
}
```

```
/* puts example */
#include <stdio.h>
int main () {
    char string [] = "Hello world!";
    puts(string);
}
```

Παραδείγματα

```
#include <stdio.h>
int main() {
    char test1[5], test2[5];
    scanf("%s", test1);
    printf("test1=%s\n", test1);
    gets(test2);
    printf("test2=%s\n", test2);
    return 0;
}
```

Εκτέλεση:

\$./a.out

1313

test1=1313

test2=

\$...

To `test2` θα είναι ίσο με "end of line" από την προηγούμενη είσοδο διότι η `scanf()` ολοκληρώνει το διάβασμά της μόλις συναντήσει τον πρώτο κενό χαρακτήρα (space, tab, newline) – αλλά δεν τον «καταναλώνει»! Έτσι η `gets()` βρίσκει το newline και επιστρέφει αμέσως...

Παραδείγματα

```
#include <stdio.h>
int main() {
    char test1[5], test2[5];
    scanf("%s", test1);
    printf("test1=%s\n", test1);
    scanf("%s", test2);
    printf("test2=%s\n", test2);
    return 0;
}
```

Εκτέλεση:

\$./a.out

1313

test1=1313

1233

test2=1233

\$...

Χρησιμοποιείται η `scanf()` αντί για την `gets()`. Η `scanf(%s)` στην αρχή του διαβάσματος αγνοεί τα κενά (space, tab, newline) και άρα την αλλαγή γραμμής. Στη συνέχεια διαβάζει από το πρώτο μη-blank και σταματά να διαβάζει στο αμέσως επόμενο blank.

Παραδείγματα

```
#include <stdio.h>
int main() {
    char lula[]="lula";
    char *ptr = lula;
    puts(lula); puts(" > ");
    puts(ptr + 2);
    printf("%s > %s\n", lula, ptr+2);
    return 0;
}
```

Εκτέλεση:

```
$ ./a.out
Lula
>
La
Lula > La
$ ...
```

Η `puts` προσθέτει το \n

Παραδείγματα

- ❖ «Χειροποίητες» `puts` που δεν εκτυπώνουν το EOL.
 - Η `putchar(ch)` τυπώνει στην οθόνη τον χαρακτήρα `ch`
 - Είναι ισοδύναμη με το `printf("%c", ch);`

```
void myputs(char *string) {  
    int i = 0;  
    while (string[i] != '\0') putchar(string[i++]);  
}  
  
void myputs(char *string) {  
    while (*string != '\0') {  
        putchar(*string);  
        string++;  
    }  
}  
  
/* Το '\0' έχει ASCII κωδικό μηδέν (0) ! */  
void myputs(char *string) {  
    while (*string) putchar(*(string++));  
}
```

sprintf / sscanf

- ❖ `int sprintf(char *s, char *format, ...)`
 - Η συνάρτηση αυτή λειτουργεί ακριβώς όπως η printf() με τη διαφορά ότι **δεν τυπώνει στην οθόνη αλλά γράφει στο string s στο οποίο τοποθετείται επιπλέον και ο χαρακτήρας '\0'.**
 - Επιστρέφεται ο αριθμός των χαρακτήρων που γράφτηκαν στο s πλην του χαρακτήρα '\0'.
- ❖ `int sscanf(char *s, char *format, ...)`
 - Η συνάρτηση αυτή λειτουργεί ακριβώς όπως η scanf() με τη διαφορά ότι **η είσοδος των δεδομένων προέρχεται από το string s και όχι από το πληκτρολόγιο.**
 - Επιστρέφει είτε τον αριθμό των αντικειμένων που ενημερώθηκαν (αν όλα πάνε καλά), είτε EOF (σε περίπτωση λάθους).

Παράδειγμα

```
/* sprintf example */
#include <stdio.h>
int main () {
    char buffer[50];
    int n, a=5, b=3;

    n=sprintf(buffer, "%d plus %d is %d", a, b, a+b);
    printf("[%s] is a %d char string\n", buffer, n);
    return 0;
}
```

```
$ ./a.out
[5 plus 3 is 8] is a 13 char string
```

Παράδειγμα

```
#include <stdio.h>

int main() {
    int k, m;
    float f;
    char *x="2 minutes to 12.0";
    char y[20], z[20], w[80];

    sscanf(x, "%d%s%s%f", &m, y, z, &f) ;
    printf("%d\n%s\n%s\n%f\n", m, y, z, f) ;
    k = sprintf(w, "%d %s %d %s ",m, z, (int) f, y) ;
    printf("\nNew order: %s\n", w);
    printf("with %d characters (including spaces)\n", k);
    return 0;
}
```

\$./a.out

2

minutes

to

12.000000

New order: 2 to 12 minutes

with 15 characters (including spaces)

Επαναληπτική κλήση της sscanf

```
#include <stdio.h>

int main() {
    int k;
    char x[30] = "2 minutes to 12.0";
    char *p;
    char y[30];

    p = x;
    while (sscanf(p, "%s", y) > 0) {
        k = printf("%s\n", y); /* # chars in y, +1 */
        p = p + k;             /* skip k chars */
    }
    return 0;
}
```

Επαναληπτική μέτρηση λέξεων

```
#include <stdio.h>

int main() {
    int k, count;
    char x[30], y[30], *p;

    while (1) {
        count = 0;
        if (fgets(x, 30, stdin) == NULL) /* Ctrl-D */
            break;
        p = x;
        while (sscanf(p, "%s", y) > 0) {
            k = printf("%s\n", y);
            p = p + k;
            count++;
        }
        printf("Total number of words: %d\n", count);
    }
    return 0;
}
```

Επαναληπτική μέτρηση λέξεων (συντομότερη)

```
#include <stdio.h>

int main() {
    int count;
    char x[90], y[90], *p;

    while (fgets(x, 90, stdin) != NULL) {
        for (count = 0, p = x; sscanf(p,"%s", y) > 0; count++) {
            p = p + printf("%s\n", y);
        }
        printf("Total number of words: %d\n", count);
    }
    return 0;
}
```

Διαχείριση Συμβολοσειρών μέσω string.h

❖ Μια συμβολοσειρά (string) είναι ένας πίνακας χαρακτήρων στον οποίο τοποθετείται τελευταίος ο χαρακτήρας '\0', ως ένδειξη του τέλους της συμβολοσειράς

❖ Μπορούμε να διαχειριστούμε ένα string με δύο τρόπους

➤ Ως έναν πίνακα, το οποίο συνεπάγεται σχετική δυσκολία

```
char line[8];
line[0] = 'H'; line[1] = 'e'; line[2] = 'l'; line[3] =
'l'; line[4] = 'o'; line[5] = '\0';
```

➤ Μέσω της χρήσης ειδικών συναρτήσεων που παρέχει η C μέσω του αρχείου <string.h>

```
strcpy(line, "Hello");
```

Συναρτήσεις Διαχείρισης Συμβολοσειρών

❖ `char *strcpy(s, t);`

- Αντιγράφει το string `t` στο `s`, μαζί με τον χαρακτήρα '`\0`' και επιστρέφει το `s`

- Παράδειγμα

```
char s[6];
strcpy(s, "hello");
```

- Τι γίνεται αν στο `s` δεν χωράει το `t`;

❖ `char *strncpy(s, t, n);`

- Αντιγράφει το πολύ ο χαρακτήρες από το `t` στο `s`, το `t` μπορεί να έχει λιγότερους. Επιστρέφει το `s`

❖ `char *strcat(s, t);`

- Προσθέτει στο τέλος του `s` το string `t`. Επιστρέφει το `s`.

❖ `char *strncat(s, t, n);`

- Προσθέτει στο τέλος του `s` το πολύ ο χαρακτήρες του `t`, και τοποθετεί επίσης και τον χαρακτήρα '`\0`'. Επιστρέφει το `s`.

Συναρτήσεις Διαχείρισης Συμβολοσειρών

- ❖ **int strcmp(s, t);**
 - Συγκρίνει λεξικογραφικά τα δύο strings.
 - ✧ (βασικό κριτήριο) περιεχόμενο
 - ✧ (δευτερεύον κριτήριο) μήκος
 - Επιστρέφει:
 - ✧ Av ($s == t$) → 0
 - ✧ Av ($s > t$) → θετικό
 - ✧ Av ($s < t$) → αρνητικό
- ❖ **int strncmp(s, t, n);**
 - Όπως και παραπάνω αλλά συγκρίνει λεξικογραφικά το πολύ η χαρακτήρες
- ❖ **char *strstr(s, t);**
 - Επιστρέφει ένα δείκτη στην πρώτη εμφάνιση στο s του t, (διαφορετικά) επιστρέφει NULL αν το t δεν περιέχεται στο s.
- ❖ **int strlen(s);**
 - Επιστρέφει το μήκος της συμβολοσειράς s (χωρίς το \0).

❖ Έστω:

- `char a[30] = "Kalimera, ";`
- `char b[20] = "Kalo mathima!" ;`

❖ Τότε:

- `strcpy(a,b)`
 - ✧ `printf("%s", a);` → *Kalo mathima!*
- `strncpy(a,b,4)`
 - ✧ `printf("%s", a);` → *Kalomera,*
- `strcat(a,b)`
 - ✧ `printf("%s", a);` → *Kalimera, Kalo mathima!*

❖ Έστω:

- int ret; char *p;
- char a[30] = "Kalimera, ";
- char b[20] = "Kalo mathima!";

❖ Τότε:

- ret = strcmp(a,b);
 - ✧ printf("%d", ret); → κάποια αρνητική τιμή
- p = strstr(a,"im");
 - ✧ printf("%s", p); → imera,
- printf("%d", strlen(a)); → 10

❖ Τι θα τυπώσει το: printf("%d, %d", sizeof(a), strlen(a));

- 30, 10

- ❖ `char *strtok(char *s, char *t);`
 - Ψάχνει στο s για κομμάτια (tokens) που διαχωρίζονται με τους χαρακτήρες που περιγράφονται στο t.
 - Κάθε διαφορετική κλήση της strtok επιστρέφει και ένα καινούργιο token (κανονικό string με χαρακτήρα τερματισμού).
 - Χρήση:
 - ✧ 1^o token: καλώ tok = strtok(s, t);
 - ✧ Επόμενα (συνήθως σε loop): καλώ tok = strtok(NULL, t);
 - Επιστρέφει NULL αν δεν υπάρχουν άλλα tokens στο s.

Προσοχή!

- **Η συνάρτηση τροποποιεί το πρώτο όρισμα της**
- **Δεν μπορεί να χρησιμοποιηθεί σε σταθερά strings**

Παράδειγμα strtok

```
char email[] = "zas@cse.uoi.gr";
char token[] = "@";
char *s;

s = strtok(email, token);
s = strtok(NULL, token);

...
```

Παράδειγμα strtok (1/2)

- ❖ Να γράψετε πρόγραμμα το οποίο λαμβάνει διευθύνσεις ηλεκτρονικού ταχυδρομείου και επιστρέφει τα πεδία από τα οποία αποτελούνται
- ❖ Σχετικό παράδειγμα εκτέλεσης:

```
$ myprog
```

```
type email address: zas@cse.uoi.gr
```

```
fields of email address: zas, cse, uoi, gr
```

Παράδειγμα strtok (2/2)

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

int main() {
    char email[80];
    char token[] = "@.";
    char *s;

    printf("type email address:");
    scanf("%s", email);
    printf("fields of email address:");
    s = strtok(email, token);
    if (s != NULL) {
        printf("%s", s);
    }
    while ((s = strtok(NULL, token)) != NULL) {
        printf(", %s", s);
    }
    return 0;
}
```

Συναρτήσεις Ελέγχου - Μετατροπής

- ❖ #include <ctype.h>
- ❖ Συναρτήσεις ελέγχου
 - int isalnum(int c); true για γράμμα ή ψηφίο
 - int isalpha(int c); true για γράμμα
 - int isdigit(int c); true για ψηφίο
 - int isspace(int c); true για κενό, tab, \n, ...
 - int islower(int c); true για γράμμα μικρό
 - int isupper(int c); true για γράμμα κεφαλαίο
- ❖ Συναρτήσεις μετατροπής
 - int tolower(int c); μετατροπή κεφαλαίου σε μικρό
 - int toupper(int c); μετατροπή μικρού σε κεφαλαίο

Συναρτήσεις Ελέγχου - Μετατροπής

❖ #include <stdlib.h>

- `int atoi(char *s);` μετατροπή string σε ακέραιο
 - ✧ Το string s πρέπει να ξεκινά με κενό ή κάποιον αριθμό
 - ✧ Η συνάρτηση σταματά να διαβάζει από το string μόλις βρει κάποιον μη-αριθμητικό χαρακτήρα
 - ✧ Αν η μετατροπή δεν μπορεί να συμβεί, επιστρέφει 0
- `long atol(char *s);` μετατροπή string σε long
- `double atof(char *s);` μετατροπή string σε double

❖ Πώς μετατρέπω αριθμούς σε strings;

- *Homework!*

Παραδείγματα

- ❖ `int i;`
`i = atoi("512");`
`i = atoi("512.035");`
`i = atoi(" 512.035");`
`i = atoi(" 512+34");`
`i = atoi(" 512 bottles of beer on the wall");`
- ❖ `int i = atoi(" does not work: 512"); // → i = 0`
- ❖ `long l = atol("1024.0001");`
- ❖ `double x = atof("42.0is_the_answer");`

- ❖ Βασικές συναρτήσεις εισόδου – εξόδου
 - `int printf(char *format, ...);`
 - `int scanf(char *format, ...);`
- ❖ Ειδικοί χαρακτήρες στο `format` τους
 - Ακέραιοι αριθμοί
 - ✧ `%d` στο δεκαδικό σύστημα
 - ✧ `%u` χωρίς πρόσημο στο δεκαδικό σύστημα
 - ✧ `%o` χωρίς πρόσημο στο οκταδικό σύστημα
 - ✧ `%x` χωρίς πρόσημο στο δεκαεξαδικό σύστημα
 - Αριθμοί κινητής υποδιαστολής
 - ✧ `%f` σε μορφή: [-]ddd.dddddd
 - ✧ `%e` σε μορφή: [-]ddd.dddddd e[+/-]ddd
 - ✧ `%g` σε μορφή %f ή %e

❖ Ειδικοί χαρακτήρες στο format τους

➤ Άλλοι τύποι

- ❖ %c χαρακτήρας
- ❖ %s συμβολοσειρά (string)
- ❖ %p δείκτης

❖ Παραλλαγές στο format

➤ Μέγεθος αριθμών

- ❖ %h αριθμοί short, π.χ. %hd, %hx
- ❖ %l αριθμοί long ή double, π.χ. %ld, %lf
- ❖ %L αριθμοί long double, π.χ. %Lf

❖ Παραλλαγές στο format

➤ Μήκος αποτελέσματος

- ✧ %8d αριθμός σε μήκος 8 χαρακτήρων
- ✧ %20s συμβολοσειρά σε μήκος 20 χαρακτήρων
- ✧ %+8d αριθμός σε μήκος 8 χαρακτήρων, τύπωσε και πρόσημο
- ✧ %08d αριθμός σε μήκος 8 χαρακτήρων, τα πρώτα εξ' αυτών 0
- ✧ %-8d όπως το %8d με στοίχιση αριστερά

printf demo (1/2)

```
#include <stdio.h>
int main() {
    int i;          // Number to print.
    double f;        // Number to print.

    printf("Enter an integer (use either + or -): ");
    scanf ("%d", &i);
    printf("This is the integer.....| %d|\n", i);
    printf("This is the integer in octal.....| %o|\n", i);
    printf("Octal with leading zero.....| %#o|\n", i);
    printf("This is the integer in hex.....| %x| or |%X|\n", i, i);
    printf("Hex with leading 0x.....| %#x| or |%#X|\n", i, i);
    printf("Forcing a plus or minus sign.....| +d|\n", i);
    printf("Include space before + numbers.....| % d|\n", i);
    printf("Field width of 3.....| %3d|\n", i);
    printf("Field width of 5.....| %5d|\n", i);
    printf("Field width of 7.....| %7d|\n", i);
    printf("Same as above with left justification.| %-7d|\n", i);
    printf("Field width of 7 with zero fill.....| %07d|\n", i);
    printf("At least 3 digits.....| %.3d|\n", i);
    printf("At least 5 digits.....| %.5d|\n", i);
    printf("Field width of 10, at least 7 digits..| %10.7d|\n", i);
```

printf demo (2/2)

```
printf("\nEnter a floating point number: ");
scanf ("%lf", &f);

printf("This is the number.....| %f|\n", f);
printf("Forcing a plus or minus sign.....| %+f|\n", f);
printf("Field width of 20.....| %20f|\n", f);
printf("0 decimal places.....| %.0f|\n", f);
printf("0 decimal places forcing decimal.....| %#.0f|\n", f);
printf("3 decimal places.....| %.3f|\n", f);
printf("20 decimal places.....| %.20f|\n", f);
printf("Field width of 20, 3 decimal places...| %20.3f|\n", f);
printf("\nHere is the number in e format:\n");
printf("3 decimal places.....| %.3e|\n", f);
printf("5 decimal places & big 'e'.....| %.5E|\n", f);

printf("\nHere is the number in g format:\n");
printf("No special requests.....| %g|\n", f);
printf("Maximum of 1 significant figure.....| %.1g|\n", f);
printf("Maximum of 4 significant figures.....| %.4g|\n", f);

return 0;
}
```

Συμπεριφορά `scanf()`

Τα παρακάτω περιγράφουν πλήρως την μερικές φορές «περίεργη» συμπεριφορά της `scanf()`:

- ❖ `scanf("%c" ...)`
 - Διαβάζει αμέσως όποιον χαρακτήρα βρει (ακόμα και κενό) και επιστρέφει
- ❖ `scanf("%<οτιδήποτε άλλο>" ...)`
 - Αγνοεί τα κενά (space, tab, newline) και αρχίζει και διαβάζει μόλις συναντήσει μη-κενό χαρακτήρα
 - Τελειώνει και επιστρέφει μόλις συναντήσει κενό χαρακτήρα. Όμως, δεν τον καταναλώνει!

- ❖ Διαχείριση χαρακτήρων/συμβολοσειρών
 - `int putchar(int c);`
 - `int getchar();`
 - `int puts(char *s); /* also: fputs() */`
 - `char *gets(char *s); /* unsafe, prefer: fgets() */`
- ❖ Διαχείριση συμβολοσειρών `<string.h>`
 - `size_t strlen(char *s);`
Μέτρηση αριθμού χαρακτήρων της συμβολοσειράς s
 - `char *strcpy(char *s1, const char *s2);`
Αντιγραφή της συμβολοσειράς s2 στην s1
 - `char *strcat(char *s1, const char *s2);`
Προσθήκη της συμβολοσειράς s2 στο τέλος της s1
 - `int strcmp(char *s1, const char *s2);`
Σύγκριση των συμβολοσειρών s1 και s2

- ❖ Μετατροπή συμβολοσειρών <stdlib.h>
 - `int atoi(char *s):`
Μετατροπή της συμβολοσειράς s σε int.
 - `long int atol(char *s):`
Μετατροπή της συμβολοσειράς s σε long int.
 - `double atof(char *s):`
Μετατροπή της συμβολοσειράς s σε double.

Προγραμματισμός σε C

Περίπτωση διαχείρισης συμβολοσειρών:
Ορίσματα στη main()



Ορίσματα στην main()

```
$ ls  
$ ls -l; ls -al; gcc myfile.c -lm
```

- ❖ Γενικά σε ένα πρόγραμμα μπορούμε να δώσουμε ως είσοδο δεδομένα/ορίσματα/επιλογές τη στιγμή που ξεκινάει η εκτέλεση του, από τη γραμμή εντολών
- ❖ Πώς μπορούμε να γνωρίζουμε τα δεδομένα/ορίσματα που δίνει ο χρήστης;
 - Απάντηση:
Παράμετροι στην main()! (την οποία μέχρι τώρα την ορίζαμε χωρίς παραμέτρους)
- ❖ Τα δεδομένα τα δέχεται η main() ως strings

Ορίσματα στην main()

- ❖ Μέχρι τώρα:

```
int main() { ... }
```

- ❖ Γενικά όμως, ο προγραμματιστής μπορεί να γράψει:

```
int main(int argc, char *argv[]) { ... }
```

ή ισοδύναμα:

```
int main(int argc, char **argv) { ... }
```

- ❖ Το argv είναι ένας πίνακας δεικτών σε συμβολοσειρές (strings)
- ❖ Το argc είναι το πλήθος των στοιχείων του πίνακα
- ❖ Πάντα το στοιχείο 0 είναι το όνομα του προγράμματος

Παράδειγμα ορισμάτων στην main()

\$ a.out hi there

- ❖ argc = 3
- ❖ argv[0]: όνομα προγράμματος, "a.out"
- ❖ argv[1]: πρώτο όρισμα προγράμματος "hi"
- ❖ argv[2]: δεύτερο όρισμα προγράμματος "there"

Παράδειγμα (εκτύπωση ορισμάτων)

```
$ a.out hello world
argc = 3
Program name: a.out
Arguments: hello, world
```

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int i;
    printf("argc = %d\n", argc);
    printf("Program name: %s\n", argv[0]);
    printf("Arguments: ");
    for (i = 1; i < argc; i++)
        printf("%s, ", argv[i]);
    printf("\n");
    return 0;
}
```

Επιπλέον παράδειγμα (πρόσθεση ορισμάτων)

```
$ myadd 5 10 15
```

Program name: myadd

Result: 30

```
#include <stdio.h>
#include <stdlib.h>      /* because of atoi() */
int main(int argc, char *argv[]) {
    int i, sum = 0;
    printf("Program name: %s\n", argv[0]);
    if (argc < 2) exit(1);    /* Nothing to add */
    for (i = 1; i < argc; i++) {
        sum = sum + atoi(argv[i]);
    }
    printf("Result: %d\n", sum);
    return 0;
}
```

Η γλώσσα C

Δείκτες και Διαχείριση Μνήμης
(memory management)



ΜΥΥ502

1 πείραμα = πολλές μετρήσεις

- ❖ Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από το χρήστη τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια ενός πειράματος
- ❖ Εν συνεχεία ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων σε ένα πίνακα
- ❖ Τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
 - μεταξύ άλλων υπολογίζει τον μέσο όρο των μετρήσεων



Άγνωστο μέγεθος πίνακα

```
#include <stdio.h>
int main() {
    int n;
    int i;
    float mo = 0;

    scanf("%d", &n);
    float measurements[n];
    πώς σας φαίνεται αυτή η υλοποίηση?

    for(i=0; i<n; i++)
        scanf("%f", &measurements[i]);

    for(i=0; i<n; i++)
        mo += measurements[i];

    mo = mo / n;
    ...
}
```

Άγνωστο μέγεθος πίνακα

```
#include <stdio.h>
int main() {
    int n;
    int i;
    float mo = 0;

    scanf("%d", &n);
    float measurements[n];

    for(i=0; i<n; i++)
        scanf("%f", &measurements[i]);

    for(i=0; i<n; i++)
        mo += measurements[i];

    mo = mo / n;
    ...
}
```

η υλοποίηση είναι λάθος!!

α) οι δηλώσεις πρέπει να γίνονται **ΜΟΝΟ** στην αρχή του μπλοκ { ... }

β) ΔΕΝ επιτρέπεται να είναι μεταβλητό το μέγεθος του πίνακα!

Σημείωση:

Η C99 τα επιτρέπει και τα δύο (δηλώσεις οπουδήποτε και “variable length arrays”).

Δυναμική δέσμευση μνήμης (dynamic memory allocation)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n; int i; float mo = 0;
    float *measurements;
    scanf("%d", &n);

    /* Μνήμη για n μετρήσεις */
    measurements = (float *) malloc(n * sizeof(float));
    if (measurements == NULL) exit(1);
    for(i=0; i<n; i++)
        scanf("%f", &measurements[i]);

    for(i=0; i<n; i++)
        mo += measurements[i];

    mo = mo / n;
    free (measurements);
    ...
}
```

- ❖ Η συνάρτηση malloc:

```
int *p;
```

```
p = (int *) malloc(N*sizeof(int));
```

- ❖ Η malloc δεσμεύει μνήμη δοσμένου μεγέθους και επιστρέφει μια διεύθυνση
- ❖ Όρισμα = το μέγεθος της αιτούμενης μνήμης (# bytes)
 - π.χ., στην πράξη θα χρειαστούμε μνήμη για αποθήκευση int, μνήμη για αποθήκευση float, μνήμη για αποθήκευση double,...
 - **Το μέγεθος της μνήμης δίνεται ως πολλαπλάσιο των sizeof(int), sizeof(float), sizeof(double),...**

Δυναμική δέσμευση μνήμης

- ❖ Η `malloc` δεν μπορεί να επιστρέφει διαφορετικούς τύπους από pointers
 - Επιστρέφει έναν «γενικό» τύπο pointer:
`void *malloc(int size);`
- ❖ Επομένως, είναι πάντα απαραίτητο να χρησιμοποιούμε το κατάλληλο casting, π.χ. (`int *`), για αυτό που επιστρέφει η `malloc`.
- ❖ Άρα αν για ένα μονοδιάστατο πίνακα τύπου `<T>` (όπου `<T>` είναι `int`, `float`, `double`, `char...`) δεν ξέρω τη διάσταση του πριν την εκτέλεση του προγράμματος,
- ❖ τότε ορίζω δείκτη `<T*>` `p` και χρησιμοποιώ την `malloc`
`p = (<T*>) malloc(N*sizeof(<T>));`
Π.χ.
`float *p;`
`p = (float *) malloc(N*sizeof(float));`

Παρένθεση - υπενθύμιση

- ❖ Το α και β παρακάτω είναι ίδια; Αν όχι υπάρχει κάποιο πρόβλημα;

(a)	(b)
<code>int x, *y = &x;</code>	<code>int x, *y; *y = &x;</code>

- ❖ Θυμηθείτε ότι άλλο σημαίνουν οι τελεστές της C μέσα σε μία ΔΗΛΩΣΗ και άλλο μέσα σε μία πράξη. Το (b) λοιπόν είναι διαφορετικό (και λάθος). Το (a) θα ήταν ισοδύναμο με το (c):

(a)	(c)
<code>int x, *y = &x;</code>	<code>int x, *y; y = &x;</code>

- ❖ Επομένως, η σωστή αρχικοποίηση ενός δείκτη με `malloc()` είναι μία από τις δύο παρακάτω:

(a)	(b)
<code>int *y; y = (int *) malloc(40);</code>	<code>int *y = (int *) malloc(40);</code>

- ❖ Η μνήμη που δεσμεύουμε μέσω της `malloc()`:
 - Δεν είναι αρχικοποιημένη (περιέχει τυχαίες τιμές)
 - Αποδεσμεύεται αυτόματα στο τέλος του προγράμματος και επιστρέφεται στο λειτουργικό σύστημα
 - Επομένως, όσο ζητάμε νέους χώρους μνήμης, τόσο μειώνεται η ελεύθερη μνήμη του συστήματος (θέλει προσοχή ώστε να μην φτάσουμε σε “out of memory”)
- ❖ Μπορούμε να αποδεσμεύσουμε / απελευθερώσουμε μνήμη με τη συνάρτηση `free`:

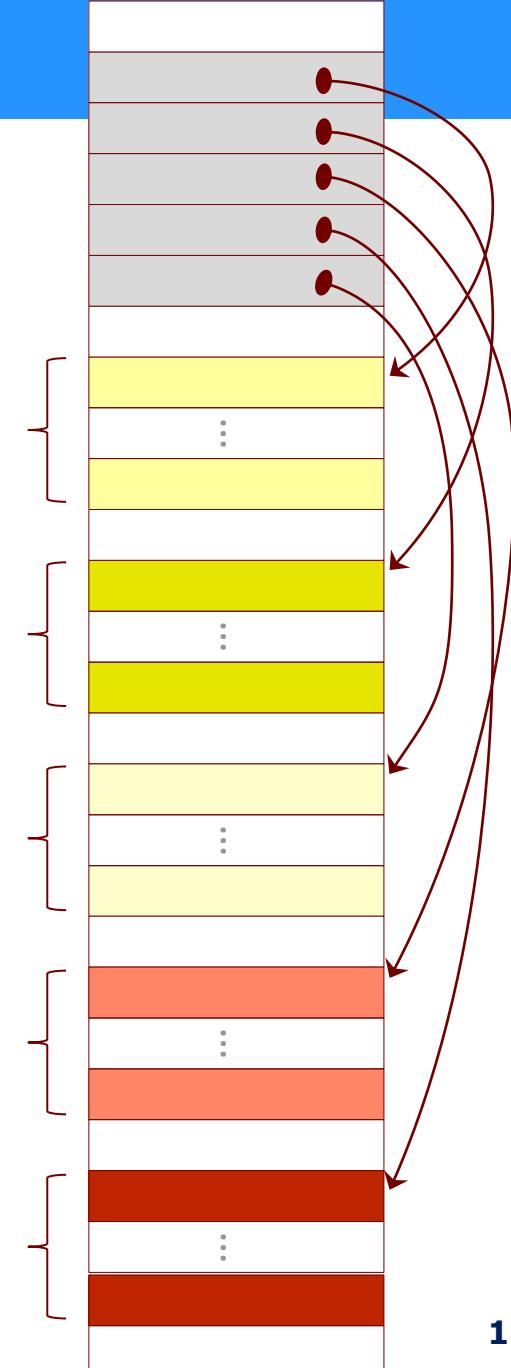
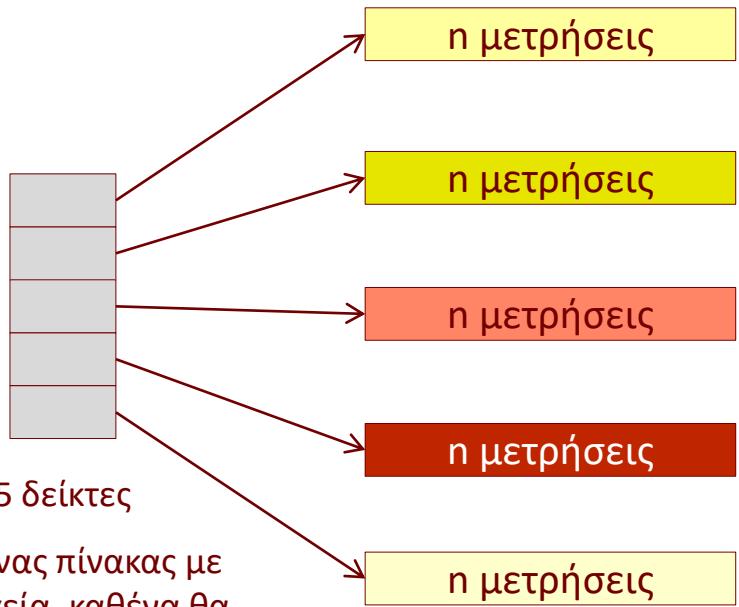
```
void free(void *ptr);
```

όπου ο `ptr` δείχνει στην αρχή του χώρου που πριν τον είχαμε δεσμέυσει με `malloc()`.

5 πειράματα με η μετρήσεις

- Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από το χρήστη τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια **5 πειραμάτων**
- εν συνεχεία ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων για κάθε πείραμα
- τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
 - μεταξύ άλλων υπολογίζει το μέσο όρο των μετρήσεων

Σχέδιο μνήμης



Δυναμική δέσμευση μνήμης & δείκτες

```
#include <stdio.h>
#include <stdlib.h>
#define M 5

int main() {
    int n, i,j; float mo;
    float *measurements[M]; /* Πίνακας από M δείκτες / ίδιο με float *(measurements[M]); */

    scanf("%d", &n);
    for(i=0; i<M; i++){ /* Για κάθε πείραμα, μνήμη για n μετρήσεις */
        measurements[i] = (float *) malloc(n*sizeof(float)); /* Δείχνει σε χώρο n αριθμών */
        if (measurements[i] == NULL) return 1;
    }

    for(i=0; i<M; i++)
        for(j=0; j<n; j++)
            scanf("%f", &measurements[i][j]); /* Χειρισμός σαν να είναι διδιάστατος πίνακας */
    for(i=0; i<M; i++) {
        for(j=0,mo=0.0; j<n; j++)
            mo += measurements[i][j];
        mo = mo / n;
        printf("%f\n", mo);
    }

    for(i=0; i<M; i++)
        free (measurements[i]); /* Απελευθέρωση μνήμης */
    return 0;
}
```

Ερωτήσεις κρίσεως στην προηγούμενη διαφάνεια

- ❖ Πώς μπορώ να γράψω την παρακάτω έκφραση κάνοντας χρήση μόνο δεικτών και πράξεων (όχι αγκυλών)

```
if (measurements[i] == NULL) return 1;
```

- ❖ Έτσι:

```
if (*measurements+i) == NULL) return 1;
```

- ❖ Το παρακάτω;

```
mo = measurements[i][j];
```

- ❖ Έτσι:

```
mo = *( *(measurements+i) + j );
```

- ❖ Διότι είναι ισοδύναμο με:

```
mo = *( measurements[i] + j );
```

- ❖ το οποίο είναι ισοδύναμο με:

```
mo = ( measurements[i] )[j];
```

Πολλά πειράματα, καθένα πολλές μετρήσεις

(Άγνωστος ο αριθμός των πειραμάτων εκ των προτέρων)

- ❖ Κατασκευάστε ένα πρόγραμμα το οποίο δέχεται ως είσοδο από τον χρήστη
 - τον συνολικό αριθμό πειραμάτων που έγιναν
 - τον συνολικό αριθμό μετρήσεων που έγιναν κατά τη διάρκεια ενός πειράματος
- ❖ εν συνεχείᾳ ο χρήστης εισάγει στο πρόγραμμα τις επιμέρους τιμές των μετρήσεων για κάθε πείραμα
- ❖ τέλος το πρόγραμμα επεξεργάζεται τα δεδομένα
 - μεταξύ άλλων υπολογίζει το μέσο όρο των μετρήσεων

Δυναμική δέσμευση μνήμης & δείκτες

- ❖ Στο προηγούμενο πρόβλημα γνωρίζαμε των αριθμό των πειραμάτων (`#define M 5`) και με βάση τον αριθμό των μετρήσεων (`int n`) κατασκευάσαμε έναν πίνακα από δείκτες, καθένας εκ των οποίων έδειχνε σε μια «γραμμή» από n μετρήσεις
 - Το χειριζόμαστε σαν να είναι διδιάστατος πίνακας `measurements[5][n]`
- ❖ Στο πρόβλημα αυτό πρέπει να κατασκευάσουμε δυναμικά και τον πίνακα από τους δείκτες, μιας και τόσο το πλήθος των μετρήσεων ανά πείραμα (n) όσο και το πλήθος των πειραμάτων (m) είναι μεταβλητές του προγράμματος.

Παράδειγμα

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

int main() {
    int n,m;
    int i,j; float mo;
float **measurements;

    scanf("%d", &m);      /* Μαθαίνουμε τις διαστάσεις */
    scanf("%d", &n);

    /* Μνήμη για τη δείκτες σε πειράματα */
measurements = (float **) malloc( m*sizeof(float *) );
    if (measurements == NULL)
        return 1;

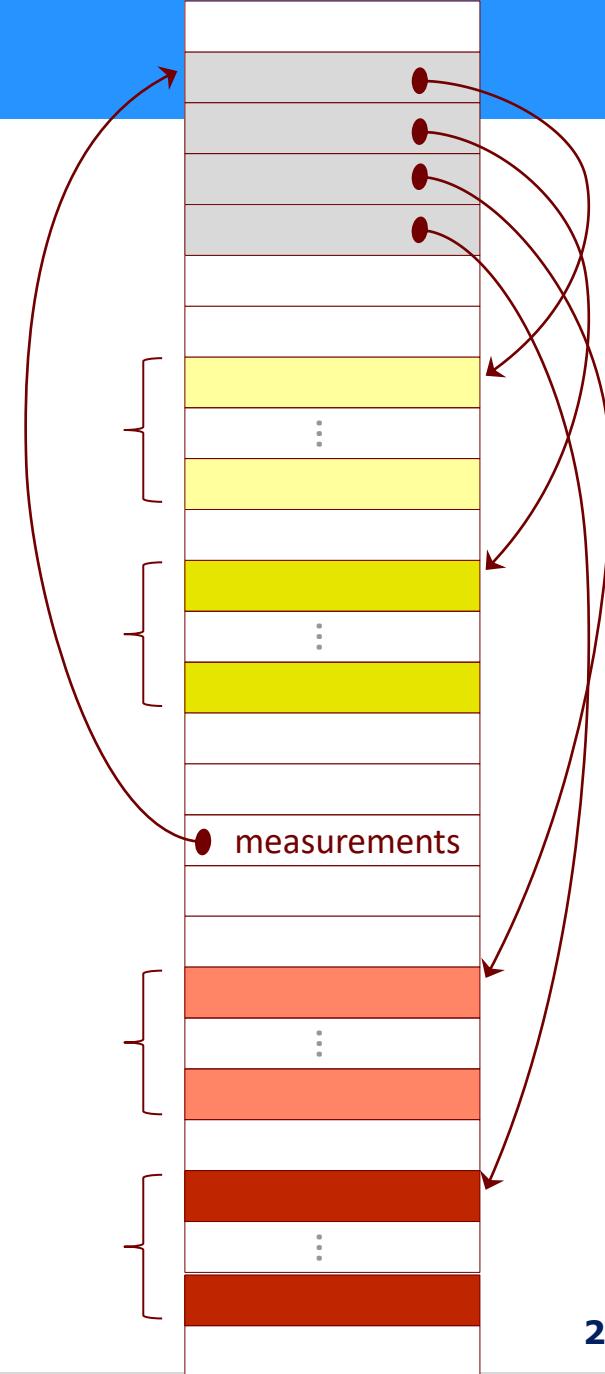
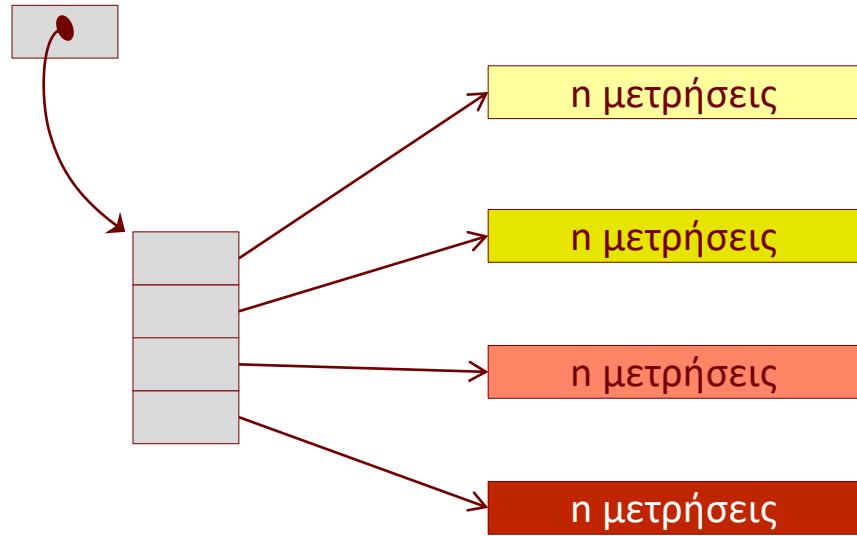
    for(i=0; i<m; i++) { /* Για κάθε πείραμα, μνήμη για n μετρήσεις */
        measurements[i] = (float *) malloc(n*sizeof(float));
        if (measurements[i] == NULL)
            return 1;
    }
}
```

Παράδειγμα

```
for(i=0; i<m; i++)  
    for(j=0; j<n; j++) {  
        scanf("%f", &measurements[i][j]);  
    }  
  
for(i=0; i<m; i++) {  
    for(j=0, mo=0.0; j<n; j++)  
        mo += measurements[i][j];  
    mo = mo / n;  
    printf("%f\n", mo);  
}  
  
for(i=0; i<m; i++)  
    free(measurements[i]); /* Απελευθέρωση γραμμής i */  
free(measurements); /* Απελευθέρωση του πίνακα δεικτών */  
  
return 0;  
}
```

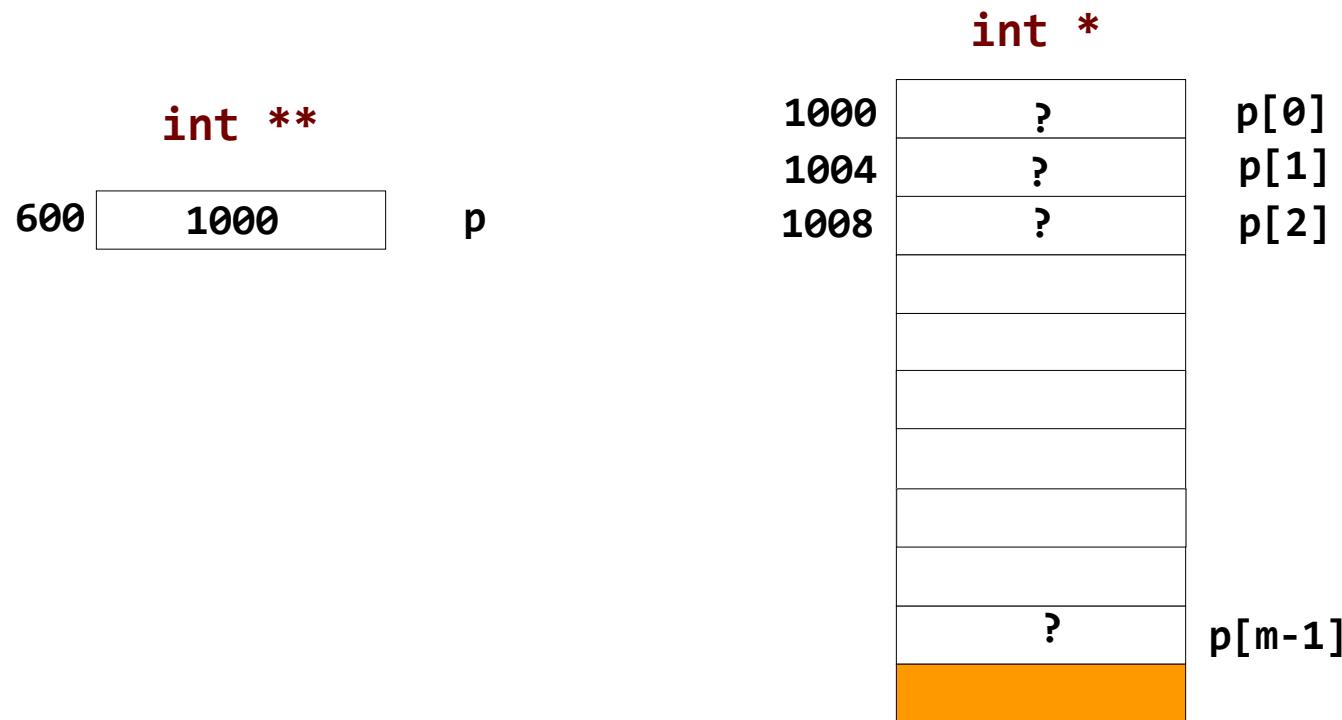
Σχεδιάγραμμα

Δείκτης σε δυναμικό πίνακα



Δυναμική δέσμευση και πίνακες δεικτών

```
int **p = (int **) malloc(m*sizeof(int *));
```

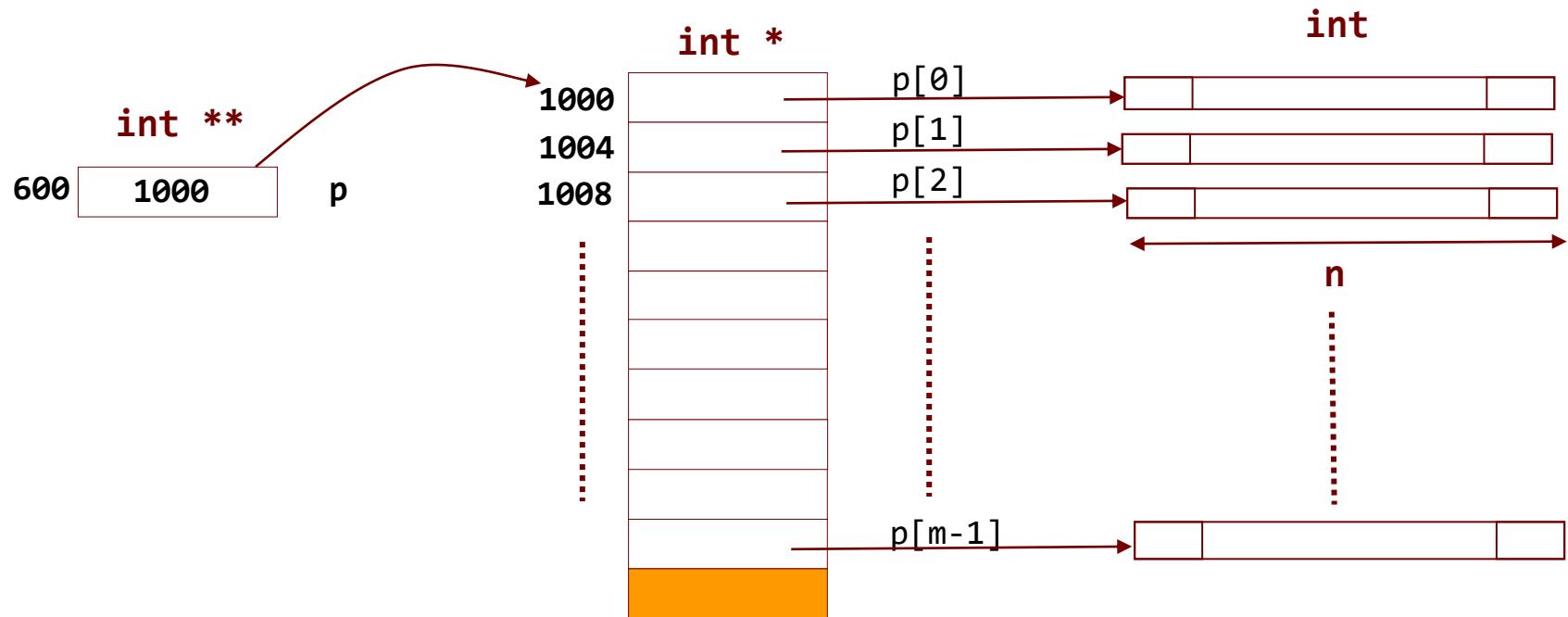


Δυναμική δέσμευση και πίνακες δεικτών

- ❖ Αν στη συνέχεια καλέσω

```
p[i] = (int *) malloc(n*sizeof(int));
```

- ❖ Έτσι έχω δημιουργήσει κάτι σαν «διδιάστατο» πίνακα (mxn), όπως τους φτιάχνει και η Java



Προηγούμενο παράδειγμα – κλήση σε συνάρτηση

```
for(i=0; i<m; i++)  
    for(j=0; j<n; j++) {  
        scanf("%f", &measurements[i][j]);  
    }  
  
show_array(m, n, measurements); /* Πώς ορίζεται; */  
  
for(i=0; i<m; i++) {  
    for(j=0, mo=0.0; j<n; j++)  
        mo += measurements[i][j];  
    mo = mo / n;  
    printf("%f\n", mo);  
}  
  
for(i=0; i<m; i++)  
    free(measurements[i]);           /* Απελευθέρωση γραμμής i */  
free(measurements);                /* Απελευθέρωση του πίνακα δεικτών */  
  
return 0;  
}
```

Παράμετροι σε συνάρτηση

```
void show_array(int r, int c, ?????)
    int i, j;
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++)
            printf("%f\n", arr[i][j]);
        printf("\n");
    }
}
```

```
void show_array(int r, int c, float arr[r][c])
```

- Στη C δεν επιτρέπονται πίνακες μεταβλητού μεγέθους

```
void show_array(int r, int c, float arr[][])
```

- Και δεν επιτρέπεται αλλά και δεν γνωρίζει το μέγεθος των γραμμών

```
void show_array(int r, int c, float *arr[])
```

- Μια χαρά! Πίνακας από pointers!

```
void show_array(int r, int c, float **arr)
```

- Μια χαρά! Pointer που δείχνει σε pointers (δηλ. στον πρώτο pointer ενός πίνακα από pointers).

```
void show_array(int r, int c, float (*arr)[])
```

- Λάθος! Pointer σε πίνακα από floats.

A) Δημιουργία **τριγωνικού** πίνακα με 10 γραμμές.

- Δηλαδή η γραμμή 0 θα έχει 10 στοιχεία, η γραμμή 1 θα έχει 9 στοιχεία, κλπ, και η γραμμή 9 θα έχει 1 στοιχείο.



B) Δημιουργία τριγωνικού πίνακα με το χρήστη να καθορίζει
κατά το χρόνο εκτέλεσης τον αριθμό των γραμμών

Τριγωνικός πίνακας 10x10 (I)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i, j, *a[10]; /* Πίνακας με 10 δείκτες, σε 10 γραμμές */

    for (i = 0; i < 10; i++)
        a[i] = (int *)malloc((10-i)*sizeof(int)); /* Γραμμή i */

    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10-i; j++) {
            a[i][j] = i + j;
            printf("a(%d,%d)=%d ", i, j, a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Τριγωνικός πίνακας 10xN (II)

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i, j, int *a[10], N;

    scanf("%d", &N);
    for (i = 0; i < 10; i++)
        a[i] = (int *)malloc((N-i)*sizeof(int));

    for (i = 0; i < 10; i++) {
        for (j = 0; j < N-i; j++) {
            a[i][j] = i + j;
            printf("a(%d,%d)=%d ", i, j, a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Τριγωνικός πίνακας NxN

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i, j, **a, N;
    scanf("%d", &N);
    a = (int **) malloc(N*sizeof(int *));
    for (i = 0; i < N; i++)
        a[i] = (int *)malloc((N-i)*sizeof(int));

    for (i = 0; i < N; i++) {
        for (j = 0; j < N-i; j++) {
            a[i][j] = i + j;
            printf("a(%d,%d)=%d ", i, j, a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Πώς δεσμεύω καθαρά διδιάστατους πίνακες;

- ❖ Δηλαδή όχι απλά πίνακα από pointers που δείχνουν σε Μ σκόρπιες γραμμές (τον οποίο τον χειρίζομαι μεν σαν να είναι διδιάστατος, όμως δεν καταλαμβάνει συνεχόμενο χώρο στη μνήμη)
- ❖ **αλλά** ένα πίνακα που όντως καταλαμβάνει συνεχόμενο χώρο $M*N$ στοιχείων στην μνήμη;

Απάντηση:

- ❖ Δεν γίνεται – η malloc() επιστρέφει πάντα ένα συνεχόμενο γραμμικό χώρο.
- ❖ Μπορώ όμως να τον χειριστώ εγώ «με το χέρι».
- ❖ Αν δεσμεύσω `a = (int *) malloc(M*N*sizeof(int));`; ποιο είναι το στοιχείο στην i γραμμή και j στήλη;
 - Το $a[i*N + j]$.

Χειρισμός διδιάστατου πίνακα με malloc()

```
void func() {  
    int i, j, M, N;  
    int *a;  
  
    scanf("%d", &M); /* Γραμμές */  
    scanf("%d", &N); /* Στήλες */  
    a = (int *) malloc(M*N*sizeof(int));  
  
    for (i = 0; i < M; i++)  
        for (j = 0; j < N; j++)  
            a[i*N+j] = 1;  
  
    free(a); /* 1 free για όλο το χώρο */  
}
```

Χειρισμός διδιάστατου πίνακα με malloc() - ευκολότερα

```
void func() {  
    int i, j, M, N;  
    int *a;  
    #define mat(r,c) a[r*N+c]      /* Preprocessor macro */  
  
    scanf("%d", &M);                /* Γραμμές */  
    scanf("%d", &N);                /* Στήλες */  
    a = (int *) malloc(M*N*sizeof(int));  
  
    for (i = 0; i < M; i++)  
        for (j = 0; j < N; j++)  
            mat(i,j) = 1;           /* Replaced by a[i*N+j] */  
  
    free(a);                      /* 1 free για όλο το χώρο */  
}  
  
/* Το macro δεν είναι τέλειο - θα τα δούμε αργότερα */
```

Επιλογή τύπου πίνακα 2D

- ❖ Όταν δεν μου ζητείται ρητά ο ένας τύπος ή ο άλλος (δηλαδή καθαρά διδιάστατος ή «γιαλαντζί» διδιάστατος – πίνακας από δείκτες σε σκόρπιες γραμμές), τι κάνω;
- ❖ Εξαρτάται από την εφαρμογή.
 - Π.χ. αν καλείται κάποια συνάρτηση που περιμένει ως παράμετρο έναν (πραγματικό) διδιάστατο τότε είμαι **αναγκασμένος** να δεσμεύσω χώρο για έναν τέτοιον πίνακα και να τον χειριστώ όπως δείξαμε.
 - Π.χ. αν οι γραμμές δεν έχουν ίδιο μέγεθος, αναγκαστικά θα πρέπει να έχω τον δεύτερο τύπο.
 - Αν δεν υπάρχει περιορισμός, είναι συνήθως πιο καλό / εύκολο / ευέλικτο να έχω έναν πίνακα από pointers σε σκόρπιες γραμμές,
 - παρά το γεγονός ότι ένας τέτοιος πίνακας έχει λίγο παραπάνω κόπο στο να δημιουργηθεί (πρέπει να κάνω `malloc` κάθε γραμμή ξεχωριστά) και να ελευθερωθεί (πρέπει να κάνω `free` κάθε γραμμή ξεχωριστά).

Ταξινόμηση strings (το πλήθος δίνεται ως όρισμα στη main) |

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void strBubbleSort(char *strings[], int num); /* Πρωτότυπο */
#define SIZE 30           /* Μέγιστο μήκος για κάθε string */

int main(int argc, char *argv[]) {
    int i, NumOfStrings;
    char **array;           /* Για αποθήκευση των strings */

    if (argc < 2) return 1;      /* ή exit(1) */
    NumOfStrings = atoi(argv[1]);
    if (NumOfStrings < 1) return 1;

    /* Μνήμη για τον πίνακα */
    array = (char **) malloc(NumOfStrings*sizeof(char *));
    if (array == NULL) {
        printf("no memory!\n");
        return 1;
    }
    /* Διάβασμα των string */
    for (i = 0; i < NumOfStrings; i++) { /* mallocs & αντιγραφή strings */
        array[i] = (char *) malloc(SIZE*sizeof(char));
        if (array[i] == NULL) {
            printf("no memory!\n");
            return 1;
        }
        fgets(array[i], 29, stdin);
    }
}
```

Ταξινόμηση strings (το πλήθος δίνεται ως όρισμα στη main) II

(Συνέχεια της main())

```
strBubbleSort(array, NumOfStrings);      /* Ταξινόμηση */
puts("\n\nThe sorted list in ascending order is:");
for (i = 0; i < NumOfStrings; i++)
    puts(array[i]);
return 0;
}

/* Προσέξτε ότι δεν μετακινούνται τα strings - μόνο οι pointers
 */
void strBubbleSort(char *strings[], int num) {
    char *temp;
    int top, seek;

    for (top = 0; top < num-1; top++) {
        for (seek = top+1; seek < num; seek++) {
            if (strcmp(strings[top], strings[seek]) > 0) {
                temp = strings[seek];          /* Εναλλαγή pointers */
                strings[seek]= strings[top];
                strings[top] = temp;
            }
        }
    }
}
```

Άλλες κλήσεις διαχείρισης μνήμης

```
void *calloc(size_t n, size_t size);
```

- Δεσμεύει χώρο στον οποίο αρχικοποιεί όλα τα bytes σε μηδέν
- Παίρνει 2 παραμέτρους: το πλήθος των στοιχείων και το μέγεθος του κάθε στοιχείου (σε bytes):
 - ✧ `int *p = (int *) calloc(n, sizeof(int));`

```
void *realloc(void *p, size_t size);
```

- Δεσμεύει νέο χώρο και ότι υπήρχε στον παλιό (όπου δείχνει ο p) αντιγράφεται στον νέο.
- Αν το νέο size είναι μικρότερο τότε χάνονται κάποια δεδομένα που υπήρχαν στον παλιό (p)

Παράδειγμα

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

int main() {
    int *p, *q;
    p = (int *) malloc(10*sizeof(int));
    if (p == NULL)
        return 0;
    q = (int *) realloc(p, 20*sizeof(int));
    if (q == NULL)
        return 0;
    printf("%p, %p\n", p, q);      /* μπορεί να διαφέρουν */
    return 0;
}
```

Αποδέσμευση μνήμης

```
void free(void *p);
```

Π.χ.

```
p = (int *) malloc(10*sizeof(int)); /* Δέσμευση */  
...  
free(p); /* Αποδέσμευση */
```

- ❖ Αποδεσμεύει τον χώρο στον οποίον δείχνει ο p. Ο χώρος αυτός πρέπει να έχει προέλθει από malloc/calloc/realloc.
- ❖ Προσοχή:
 - Ο χώρος αποδεσμεύεται (επιστρέφεται στο σύστημα και δεν μπορούμε να τον ξαναχρησιμοποιήσουμε), ΑΛΛΑ
 - ο pointer p ΣΥΝΕΧΙΖΕΙ ΝΑ ΔΕΙΧΝΕΙ ΕΚΕΙ (δεν άλλαξε η τιμή του)!!

- ❖ `#include <stdlib.h>`
- ❖ `void *malloc(size_t size);`
Δέσμευση μνήμης
- ❖ `void *calloc(size_t count, size_t size);`
Δέσμευση μνήμης αρχικοποιημένης σε μηδενικά bytes
- ❖ `void *realloc(void *ptr, size_t size);`
Επαναδέσμευση μνήμης (επέκταση/συρρίκνωση)
- ❖ `void free(void *ptr);`
Αποδέσμευση μνήμης

- ❖

```
int *p;
p = (int *) malloc(sizeof(int));
```
- ❖

```
int *p;
p = (int *) malloc(M*sizeof(int));
```
- ❖

```
int *p[10];
for (i = 0; i < 10; i++)
    p[i] = (int *) malloc(M*sizeof(int));
```
- ❖

```
int **p;
p = (int **) malloc(10*sizeof(int *));
for (i = 0; i < 10; i++)
    p[i] = (int *) malloc(M*sizeof(int));
```

Pointer (malloc) από συνάρτηση |

```
#include <stdio.h>

void makeArray(int * ip, int size) {
    int i;
    ip = (int *) malloc(size*sizeof(int));
    if (ip == NULL) return;
    for (i=0; i<size; i++){
        ip[i] = 3*i+45;
        printf("print from mA: %d\n",ip[i]);
    }
}
int main() {
    int * array = NULL;
    int i, N=12;

    makeArray(array, N);
    for (i=0;i<N;i++)
        printf("print from main: %d\n",array[i]);
    return 0;
}
```

ΔΟΥΛΕΥΕΙ ???? 

Pointer (malloc) από συνάρτηση II

```
#include <stdio.h>

void makeArray(int **ip, int size) {
    int i;
    *ip = (int *) malloc(size*sizeof(int));
    if (*ip == NULL) return;
    for (i=0; i<size; i++){
        (*ip)[i] = 3*i+45;      //try without ()
        printf("print from mA: %d\n",*((*ip)+i)); //aka (*ip)[i]
    }
}
int main() {
    int * array = NULL;
    int i, N=12;

    makeArray(&array, N);
    for (i=0;i<N;i++)
        printf("print from main: %d\n",array[i]);
    return 0;
}
```

ΔΟΥΛΕΥΕΙ !!!!!

Προβλήματα pointers / διαχ. μνήμης – Dangling pointers

```
int *p;  
printf("%d", *p);
```

Το p δεν δείχνει σε καμία θέση (σκουπίδια) - **dangling**

```
int *p;  
if (condition) {  
    int temp = 1;  
    p = &temp;  
    printf("%d", *p);  
}  
printf("%d", *p);
```

Μετά το μπλοκ του if {}, το temp ΔΕΝ ΥΦΙΣΤΑΤΑΙ. Επομένως το p δείχνει σε άκυρη / μη νόμιμη θέση - **dangling**

```
int *p;  
p = malloc(10*sizeof(int));  
if (p == NULL) ...  
p[5] = 100;  
...  
free(p);  
...  
p[5]--;
```

Μετά το free(), η δεσμευμένη μνήμη από το malloc() ΔΕΝ ΥΠΑΡΧΕΙ. Όμως, το p ΔΕΝ ΕΧΕΙ ΑΛΛΑΞΕΙ! Επομένως δείχνει σε άκυρη θέση - **dangling**

Κίνδυνοι από dangling pointers & προφυλάξεις

- ❖ Οι dangling pointers μπορεί να οδηγήσουν σε προσπέλαση άκυρων / παράνομων διευθύνσεων.
 - Ένα πρόβλημα που μπορούν να προκαλέσουν είναι το κρασάρισμα ή χωρίς αιτιολογία δυσλειτουργία του προγράμματος (εξαιρετικά δύσκολο debugging)
 - Ακόμα πιο σοβαρό είναι η πρόκληση ζημιάς / κρασάρισμα ΟΛΟΥ του συστήματος.

❖ Προφυλάξεις:

1. Πάντα αρχικοποιούμε τους *pointers* κατά τη δήλωσή τους (τουλάχιστον τους θέτουμε ίσους με *NULL*).
2. Μετά από *to free(p)*, θέτουμε *τον p = NULL*.

Έτσι τουλάχιστον αποφεύγουμε ζημιά στο σύστημα αλλά και διευκολύνουμε το debugging μιας και η προσπέλαση σε *NULL* «χτυπάει» αμέσως.

Προβλήματα pointers / διαχ. μνήμης – Memory leaks

(Δεν έχουν μπει οι έλεγχοι για NULL για απλότητα του κώδικα)

```
char *p;  
p = malloc(10*sizeof(char));  
scanf("%s", p);  
for ( ; p != '\0'; p++ )  
    printf("%c", *p);  
...
```

Κανένα πρόβλημα ορθότητας.

Απλά πλέον ο χώρος των 10 θέσεων που δεσμεύσαμε μέσω του p ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΠΡΟΣΠΕΛΑΣΤΕΙ ΜΕ ΤΙΠΟΤΕ.

```
int *p, *q;  
p = malloc(10*sizeof(int));  
q = malloc(20*sizeof(int));  
...  
if (condition) {  
    p = q;  
}  
printf("%d", p[5]);  
...
```

Ενώ συνεχίζει να υπάρχει στη μνήμη (δεν έχει αποδεσμευθεί), έχει χαθεί ο μοναδικός pointer που γνώριζε τη διεύθυνσή του – **memory leak (διαρροή μνήμης)**

MHN ΞΕΧΝΑΤΕ ΤΑ free() !!!

Αναλογία με αποθήκη προϊόντων

- ❖ Γνωστό κατάστημα διαθέτει αποθήκη με πολλά ράφια όπου σε κάθε ράφι υπάρχει ένα προϊόν (έπιπλο). Τα προϊόντα είναι συσκευασμένα έτσι ώστε ΔΕΝ ΜΠΟΡΕΙΣ να καταλάβεις τι είναι αν πας στα ράφια της αποθήκης. Για να πάρεις το έπιπλο πρέπει να επισκεφτείς την έκθεση του καταστήματος και να σημειώσεις σε ειδικά χαρτάκια το ΡΑΦΙ της αποθήκης που έχει το προϊόν που σε ενδιαφέρει. Με το συμπληρωμένο χαρτάκι πας στο σωστό ράφι και το παίρνεις – αλλιώς δεν μπορείς να βρεις το προϊόν.
- ❖ Αναλογία:
 - **Μνήμη** = αποθήκη
 - **Μεταβλητή** (κελί στη μνήμη) = ράφι
 - **Τιμή μεταβλητής** = προϊόν στο ράφι
 - **Δείκτης** = το χαρτάκι

Έννοιες & αναλογίες

Λειτουργία / έννοια	Αναλογία
<code>p = &x;</code>	Γράφω σε χαρτάκι το ράφι.
<code>*p</code>	Το προϊόν που υπάρχει στο ράφι (πάω εκεί και το βρίσκω).
<code>q = &y;</code>	Άλλο χαρτάκι που γράφει άλλο ράφι.
<code>p = q;</code>	Στο πρώτο χαρτάκι αλλάζω τι έγραψα και γράφω το ίδιο ράφι που γράφει το δεύτερο χαρτάκι.
<code>p++;</code>	Αλλάζω το χαρτάκι και σημειώνω το αμέσως επόμενο ράφι.
<code>temp = *a; *a = *b; *b = *a;</code>	Πάω στα ράφια που γράφουν τα χαρτάκια a και b και εναλλάσσω τα προϊόντα.
<code>t = malloc(10*sizeof(int));</code>	Δεσμεύω 10 συνεχόμενα άδεια ράφια στην αποθήκη. Στο χαρτάκι σημειώνω το ΠΡΩΤΟ από αυτά.
Διαρροή μνήμης	Σβήνω το μοναδικό χαρτάκι που έγραφε τα νέα ράφια. Δεν ξέρω πλέον που είναι τα νέα ράφια στην αποθήκη
<code>free(t);</code>	Αποδεσμεύω τα ράφια που δέσμευσα.
<code>v = malloc(10*sizeof(int)); w = v;</code>	Δεσμεύω 10 συνεχόμενα άδεια ράφια στην αποθήκη. Στο χαρτάκι σημειώνω το ΠΡΩΤΟ από αυτά και φτιάχνω και δεύτερο χαρτάκι με το ίδιο ράφι.
<code>free(v); v = NULL;</code>	Αποδεσμεύω τα ράφια που δέσμευσα. Σβήνω το πρώτο χαρτάκι.
Dangling pointer.	Το δεύτερο χαρτάκι (w) συνεχίζει να γράφει το πρώτο ράφι από τα αποδεμευμένα.

Η γλώσσα C

*Νέοι και σύνθετοι τύποι
(typedef & structs & unions)*



MYY502

typedef (I)

- ❖ Η C επιτρέπει να δώσουμε **οποιαδήποτε ονομασία μας αρέσει σε έναν υπάρχων τύπο**.
- ❖ Π.χ. δεν μου αρέσει το “int”, “char” κλπ.
 - Μπορώ να τα «βαφτίσω» με δεύτερο όνομα (ψευδώνυμο) με την typedef.

- ❖ Παραδείγματα:

```
typedef int Integer;      /* ψευδώνυμο του τύπου int */
typedef char Character; /* ψευδώνυμο του τύπου char */
int      n, m;
Integer x;
Character c;

x = n;
c = 'a';
```

typedef (II)

- ❖ Η C επιτρέπει να ορίσουμε **έναν νέο τύπο, με όποια ονομασία μας αρέσει.**
- ❖ Παράδειγμα: Θέλω να ορίσω τον τύπο «δείκτη σε χαρακτήρα» και να τον ονομάσω «pchar».
 - Το κάνω πάλι με την **typedef**.

```
typedef char * pchar;           /* Νέος τύπος, “pchar” */
char c, name[20];
pchar s, t;      /* pointer, αλλά χωρίς το αστεράκι!! */
s = &c;
t = name;        /* Δηλαδή, t = &name[0] */
*s = t[1];
```

Πολύπλοκα `typedef`

- ❖ Και πιο πολύπλοκοι τύποι!
- ❖ Π.χ. θέλω να ορίσω τον τύπο intarray, «πίνακας 10 ακεραίων».

```
typedef int intarray[10];
```

```
intarray A = { 0, 1, 2 };
```

```
A[5] = 15;
```

- ❖ Π.χ. τύπος πίνακα με 10 pointers σε χαρακτήρες

```
typedef char * pchar; /* Νέος τύπος, "pchar" */
```

```
typedef pchar pcarray[10]; /* Πίνακας με 10 pchar */
```

```
pcarray names;
```

```
names[0] = "Bill";
```

```
names[1] = "Demi";
```

```
names[2] = names[0];
```

Συνταγή για `typedef`

- ❖ Αν θέλετε να ορίσετε έναν νέο τύπο, αλλά δυσκολεύεστε στο πως να τον εκφράσετε, υπάρχει ο εξής απλός κανόνας:
 1. Σκεφτείτε ένα όνομα (π.χ. `mytype`)
 2. Ορίστε μία μεταβλητή `mytype` ακριβώς όπως θα την θέλατε
 3. Τοποθετήστε τη λέξη “`typedef`” μπροστά από τη δήλωση της μεταβλητής.
- ❖ Παράδειγμα:
 - Κάνω επεξεργασία εικόνων και για κάθε πίξελ χρησιμοποιώ ένα πίνακα 3 χρωμάτων (RGB – red, green, blue). Θέλω να ορίσω έναν νέο τύπο που να μου δίνει αυτόν τον πίνακα.
 1. Θα τον πω “`color`”.
 2. Είναι πίνακας 3 ακεραίων, άρα αν ήταν μεταβλητή θα οριζόταν ως εξής:
`int color[3];`
 3. Ολοκλήρωση με `typedef`:
`typedef int color[3];`
- ❖ Χρήση:

```
color c = {50, 50, 50}, pixels[100];
pixels[5][0] = c[0];
```

Η γλώσσα C

Δομές (*structs*)



MYY502

Εισαγωγή

- ❖ Οι δομές στη C επιτρέπουν τον ορισμό **σύνθετων τύπων δεδομένων** πέραν των βασικών τύπων που προσφέρει η γλώσσα
 - Με τον τρόπο αυτό επιτυγχάνουμε καλύτερη οργάνωση των δεδομένων
- ❖ Πολλές φορές θέλουμε να διατηρούμε συλλογή από πληροφορίες για **ΜΙΑ** οντότητα, π.χ. μία μεταβλητή «εργαζόμενος» θα θέλαμε να εμπεριέχει και το όνομα και το επώνυμο και την ηλικία του, αντί να ορίσουμε 3 ξεχωριστές μεταβλητές.
 - Μια δομή περιλαμβάνει μια συλλογή από **ανομοιογενή πεδία (διαφορετικών τύπων)** που ομαδοποιούνται με ένα μόνο όνομα που αντιστοιχεί στον νέο τύπο δεδομένων
- ❖ Ο ορισμός μιας **μεταβλητής** του νέου τύπου δεδομένων **αντιστοιχεί στον ορισμό μιας συλλογής από μεταβλητές** που αντιστοιχούν με τη σειρά τους **στα πεδία της δομής** που ορίσαμε.

```
struct <ετικέτα δομής> {  
    <δηλώσεις πεδίων>  
};
```

```
struct person {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};
```



Δηλώσεις μεταβλητών

❖ Προσοχή στη λέξη **struct**:

- Αρχικά χρησιμοποιείται **για να ορίσει την δομή** και την ετικέτα/όνομα της και
- Στη συνέχεια χρησιμοποιείται **για να ορίσει μεταβλητές** αυτού του τύπου δομής.

❖ Παράδειγμα:

```
struct person {      /* Ορισμός ονόματος & πεδίων δομής */
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
}
struct person x;      /* Ορισμός μεταβλητής τύπου struct person */
```

❖ Μπορεί κάποιος να κάνει και τα δύο σε ένα (όχι καλό!):

```
struct person {
    ...
} x;
```

❖ Αρχικοποίηση με αγκύλες:

```
struct person x = {"Στέλιος", "Μπέης", 'M', 40};
```

- Όπως και στους πίνακες, μπορεί να γίνει αρχικοποίηση μόνο μέχρι κάποιο πεδίο. Τα επόμενα πεδία αρχικοποιούνται αυτόματα στο 0.

Πίνακες από δομές

- ❖ Πίνακας από δομές:

```
struct person people[40];
```

```
struct person {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};
```

- ❖ Ορισμός δομής & μαζί δήλωση μεταβλητής:

```
struct person {  
    ...  
} people[40];
```

- ❖ Αρχικοποίηση:

```
struct person people[] = {  
    {"Στέλιος", "M", 'M', 40},  
    {"Κώστας", "A", 'M', 50}  
};
```

Πρόσβαση στα πεδία μίας δομής

❖ Χρήση: <δομή>.<πεδίο>

- x.firstName
- x.age

```
struct person {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};
```

❖ Π.χ.

```
struct person x = {"Στέλιος", "Μπέης", 'M', 40};  
printf("%c", x.gender);  
x.age ++;
```

❖ Π.χ.

```
struct person people[40];           /* Πίνακας με δομές */  
...  
printf("%c", people[3].gender);     /* Στοιχείο 3 */  
people[4].age++;                  /* Στοιχείο 4 */
```



Παράδειγμα I

```
struct person {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};  
  
int main() {  
    struct person x, y;  
  
    strcpy(x.firstName, "rivaldo");  
    strcpy(x.lastName, "unknown");  
    x.gender = 'M';  
    x.age = 37;  
  
    y = x; /* Προσέξτε αυτό! Αντιγράφονται ΌΛΑ ΤΑ BYTES */  
    return 0;  
}
```

Παράδειγμα II

```
struct person {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};  
  
int main() {  
    struct person x, y;  
  
    scanf("%s", x.firstName);  
    scanf("%s", x.lastname);  
    scanf("%c", &(x.gender));  
    scanf("%d", &(x.age));  
  
    y = x;  
    return 0;  
}
```

Προσοχή στα πεδία που χρησιμοποιείτε!! (I)

```
struct person {  
    char *firstName;           /* Αντί για πίνακες */  
    char *lastName;  
    char gender;  
    int age;  
};  
  
int main() {  
    struct person x, y;  
  
    scanf("%s", x.firstName);  
    scanf("%s", x.lastname);  
    scanf("%c", &(x.gender));  
    scanf("%d", &(x.age));  
  
    y = x;  
    return 0;  
}
```

Είναι όλα OK???

Προσοχή στα πεδία που χρησιμοποιείτε!! (I)

```
struct person {  
    char *firstName;           /* Αντί για πίνακες */  
    char *lastName;  
    char gender;  
    int age;  
};  
  
int main() {  
    struct person x, y;  
  
    x.firstname = (char *) malloc(20); /* Should check for NULL... */  
    x.lastname = (char *) malloc(20); /* Should check for NULL... */  
    scanf("%s", x.firstName);  
    scanf("%s", x.lastname);  
    scanf("%c", &(x.gender));  
    scanf("%d", &(x.age));  
  
    y = x;  
    return 0;  
}
```

QUIZ

Τι γίνεται στο τέλος
με το y???

typedef για ευκολία

- ❖ Μπορούμε να χρησιμοποιήσουμε την `typedef` ώστε κάνουμε τις δηλώσεις μας απλούστερες, π.χ.

```
struct person_s {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
};  
typedef struct person_s person_t;  
  
int main() {  
    person_t p;  
    p.age = 12;  
    return 0;  
}
```

Στυλ προγραμματισμού:

Το `struct` και το όνομα του τύπου από το `typedef` πρέπει να συνδέονται με συστηματικό τρόπο (ή ακόμα και συνώνυμο).

Συνήθης τακτική / σύμβαση:

Struct: person ή person_s

Typdef: Person, person_t

Συνδυασμός struct με typedef

Ορισμός struct και τύπου μαζί:

```
typedef struct person_s {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
} person_t;
```

```
int main() {  
    person_t p;  
    p.age = 12;  
    return 0;  
}
```

Κι άλλη (κακή) εκδοχή: μη-ονοματισμένο struct

```
typedef struct {  
    char firstName[20];  
    char lastName[20];  
    char gender;  
    int age;  
} person_t;
```

```
int main() {  
    person_t p;  
    p.age = 12;  
    return 0;  
}
```

Εμφωλευμένες δομές

- ❖ Οι δομές μπορεί να είναι ένθετες, δηλαδή να φωλιάζονται η μια μέσα στην άλλη, δημιουργώντας πιο πολύπλοκες δομές:

```
struct family {  
    struct person father;  
    struct person mother;  
    int     numofchildren;  
    struct person children[5];  
};
```

Εμφωλευμένες δομές – παράδειγμα

```
struct family {  
    struct person     father;  
    struct person     mother;  
    int               numofchildren;  
    struct person    children[5];  
};  
  
int main() {  
    struct person x, y, z;  
    struct family fml;  
  
    fml.numofchildren = 2;  
    strcpy(fml.father.firstName, "Joe");  
    strcpy(fml.children[0].firstName, "Marry");  
    return 0;  
}
```

Πράξεις/λειτουργίες σε δομές

- ❖ Επιτρεπτές πράξεις/λειτουργίες σε μια δομή είναι:
 - η αντιγραφή της
 - η απόδοση τιμής σ' αυτήν ως σύνολο
 - η εξαγωγή της διεύθυνσής της με &
 - η προσπέλαση των μελών της
- ❖ Επιτρέπονται
 - `<struct> = <struct>` (**αντιγράφονται τα πάντα**)
 - `&<struct>` (δείκτης στο χώρο που αποθηκεύεται το struct)
 - `<struct>.πεδίο`
- ❖ Ο δομές δεν μπορούν να συγκριθούν, δηλ. δεν επιτρέπεται:
 - `<struct> == <struct>`

Πράξεις σε δομές

- ❖ Μεταβλητές τύπου δομής μπορούν να μεταβιβαστούν ως ορίσματα σε συναρτήσεις όπως επίσης και να επιστραφούν ως αποτελέσματα συναρτήσεων

```
struct person inc_age(struct person x) {  
    x.age += 1;  
    return x;  
}  
  
int main() {  
    struct person x1, x2;  
    x1.age = 45;  
    x2 = inc_age(x1); /* Πέρασμα με τιμή (copy) */  
    return 0;  
}
```

Δομές και δείκτες

❖ Παράδειγμα δείκτη σε δομή

```
struct person p;
```

```
struct person *pp;
```

❖ Νόμιμες εκφράσεις:

```
pp = &p;
```

```
printf("%s", (*pp).firstName);
```

❖ Οι εκφράσεις τύπου **(*pp).firstName** απαιτούν πάντα

παρενθέσεις

❖ Εναλλακτικά: **pp->firstName**, δηλαδή

(*pp). ≡ pp->

Παράδειγμα - Μεταβίβαση με τιμή

```
#include <stdio.h>
struct person {
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
};

void initPerson(struct person p) {
    strcpy(p.firstName, "xxxxx");
    strcpy(p.lastName, "yyyyy");
    p.gender = 'M'; p.age = 40;
}

int main() {
    struct person q;
    q.age = 0;
    strcpy(q.firstName, "");
    strcpy(q.lastName, "");
    initPerson(q);          /* Με τιμή */
    printf("%s %s %d\n", q.firstName, q.lastName, q.age);
    return 0;
}
```

Παράδειγμα - Μεταβίβαση με αναφορά

```
#include <stdio.h>
struct person {
    char firstName[20];
    char lastName[20];
    char gender;
    int age;
};

void initPerson(struct person *p) {
    strcpy(p->firstName, "xxxxx");
    strcpy(p->lastName, "yyyyy");
    p->gender = 'M'; p->age = 40;
}

int main() {
    struct person q;
    q.age = 0;
    strcpy(q.firstName, "");
    strcpy(q.lastName, "");
    initPerson(&q);           /* Με αναφορά */
    printf("%s %s %d\n", q.firstName, q.lastName, q.age);
    return 0;
}
```

Για μεγάλες δομές, η μεταβίβαση με δείκτη είναι γενικά αποτελεσματικότερη, επίσης χρειάζεται όταν χρειάζεται να κάνουμε **πέρασμα με αναφορά**.

Παράδειγμα 1

```
#include <stdio.h>
struct person {
    char name[20];
    int age;
};

struct person inc_age(struct person x) {
    x.age += 1;
    return x;
}

int main() {
    struct person a = {"Me", 10};
    struct person c;

    c = inc_age(a);
    printf("C:%d A:%d\n", c.age, a.age);
    return 0;
}
```

\$./a.out

C:11 A:10

Παράδειγμα 2 – δομές και δείκτες

```
#include <stdio.h>
#include <string.h>
struct person {
    char name[20];
    int age;
};

struct person inc_age(struct person x) {
    x.age += 1;
    return x;
}

void inc_age_ptr(struct person *x) {
    x->age += 1;
}

int main() {
    struct person a = {"XXX", 10}, b = {"YYY", 20}, c;

    c = inc_age(a);
    inc_age_ptr(&b);
    printf("C:%d A:%d B:%d \n", c.age, a.age, b.age);
    return 0;
}
```

\$./a.out

C:11 A:10 B:21

Παράδειγμα 3 (1/2)

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

struct name{
    char *fname;
    char *lname;
    int letters;      /* Πλήθος γραμμάτων επώνυμου + μικρού ονόματος */
};

void getinfo(struct name *pst) {
    char temp[81];

    printf("First name ? ");
    fgets(temp, 80, stdin);                      /* Διάβασμα */
    pst->fname = (char *) malloc(strlen(temp) + 1); /* Δέσμευση μνήμης */
    if (pst->fname == NULL) exit(1);
    strcpy(pst->fname, temp);                    /* Αντιγραφή */

    printf("Last name ? ");
    fgets(temp, 80, stdin);
    pst->lname = (char *) malloc(strlen(temp) + 1));
    if (pst->fname == NULL) exit(1);
    strcpy(pst->lname, temp);
}
```

Παράδειγμα 3 (2/2)

```
/* Συνέχεια */

void computeLen(struct name *pst) {
    pst->letters = strlen(pst->fname) + strlen(pst->lname);
}

void cleanup(struct name *pst) {
    free(pst->fname);
    free(pst->lname);
}

int main() {
    struct name x;

    getinfo(&x);
    computeLen(&x);
    printf(" %s %s %d\n", x.fname, x.lname, x.letters);
    cleanup(&x);
    return 0;
}
```

Παράδειγμα – πίνακες και δομές (1/2)

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

typedef struct student { /* Θα φυλάμε βαθμολογίες μαθητών */
    int AM;
    int grade;
} Student;

void calculate_stats(Student class[], int studNo);
void print_student_details(Student class[], int studNo);

int main(int argc, char *argv[]) {
    Student *class;
    int i, n;

    n = atoi(argv[1]); /* Εκτέλεση του προγράμματος με ορίσματα */
    class = (Student *) malloc(n*sizeof(Student));
    for (i = 0; i < n; i++) /* Διάβασμα η φοιτητών-βαθμών */
        scanf("%d%d", &class[i].AM, &class[i].grade);

    print_student_details(class, n);
    calculate_stats(class, n);
    return 0;
}
```

Παράδειγμα – πίνακες και δομές (2/2)

```
void calculate_stats(Student class[], int studNo) {  
    int i;  
    int max = 0;  
  
    for (i = 0; i < studNo; i++) {  
        if (class[i].grade > max)  
            max = class[i].grade;  
    }  
    printf("MAX GRADE = %d\n", max);  
}  
  
void print_student_details(Student class[], int studNo) {  
    int i;  
  
    printf("AA \t AM \t GRADE\n");  
  
    for (i = 0; i < studNo; i++) {  
        printf("%d \t %d \t %d\n", i, class[i].AM, class[i].grade);  
    }  
}
```

Σύνθετο παράδειγμα

```
struct Name { char first[20]; char last[20]; };
struct An8rwpos { struct Name gname; float income; }

int main() {
    struct An8rwpos people[2] = { {"A", "B"}, 123.4}, {"C", "D"}, 345.67} };
    struct An8rwpos *p, me;

    p = &people[0];      /* Δείχνει στο 0 στοιχείο του people */
    me = *(p+1);        /* = p[1] (το πρώτο στοιχείο του p) */
    printf("%s %s %f \n\n", /* A B 123.400000 */
           p->gname.first, p->gname.last, p->income);
    printf("%s %s %f \n", people[1].gname.first,
           people[1].gname.last, people[1].income);
    printf("%s %s %f \n",
           me.gname.first, me.gname.last, me.income);
    printf("%s %s %f \n\n", (*(p+1)).gname.first,
           (*(p+1)).gname.last, (*(p+1)).income);

    strcpy(p[1].gname.first, "X");
    strcpy(p[1].gname.last, "Y");
    p[1].income = 0.0;

    printf("%s %s %f \n", people[1].gname.first,
           people[1].gname.last, people[1].income);
    printf("%s %s %f \n\n",
           me.gname.first, me.gname.last, me.income);
}
```

\$./a.out

A	B	123.400000
C	D	345.670000
C	D	345.670000
C	D	345.670000
X	Y	0.000000
C	D	345.670000

Ισοδύναμες εκφράσεις στο προηγούμενο παράδειγμα

```
/* struct An8rwpos people[2] = { {"A", "B"}, 123.4},  
                                {"C", "D"}, 345.67} };  
  
struct An8rwpos *p, me;  
p = &people[0];  
me = *(p+1);  
*/
```

- ❖ Υπάρχουν δύο σύνολα ισοδύναμων εκφράσεων
- ❖ 1^ο σύνολο:
 - `printf("%s\n", p->gname.first);`
 - `printf("%s\n", people[0].gname.first);`
 - `printf("%s\n", (*p).gname.first);`
- ❖ 2^ο σύνολο:
 - `printf("%s\n", me.gname.first);`
 - `printf("%s\n", people[1].gname.first);`
 - `printf("%s\n", (*(p+1)).gname.first);`

Η γλώσσα C

Αναφορές σε δομές



ΜΥΥ502

Αναφορές (προς ίδιου τύπου δομή)

❖ Δεν επιτρέπεται

```
struct Employee {  
    char name[20];  
    int age;  
    struct Employee manager;  
};
```

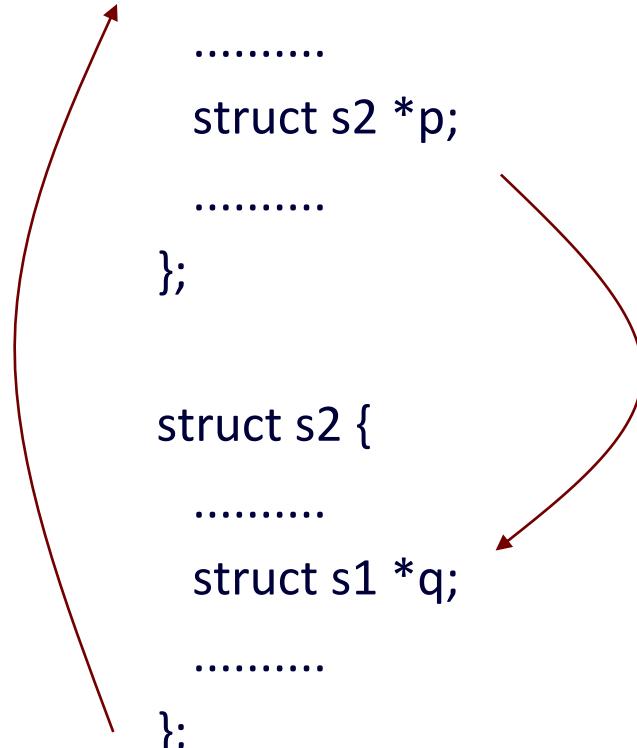
❖ Επιτρέπεται

```
struct Employee {  
    char name[20];  
    int age;  
    struct Employee *manager;  
};
```

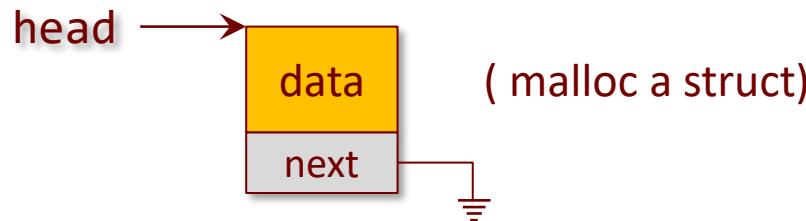
Αναφορές

- ❖ Επίσης επιτρέπεται (κυκλική αναφορά)

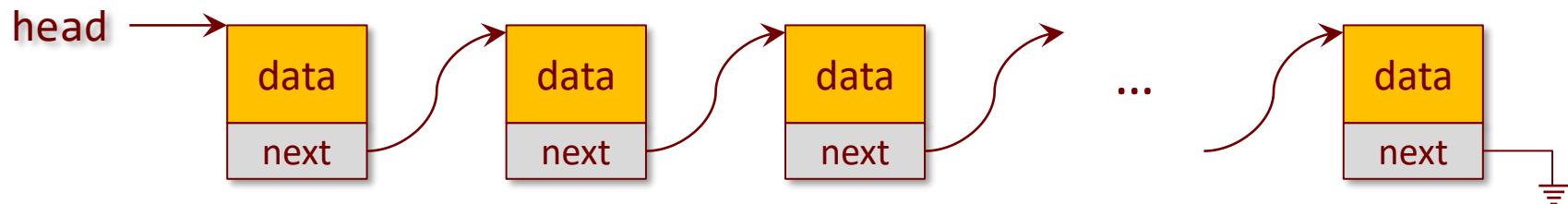
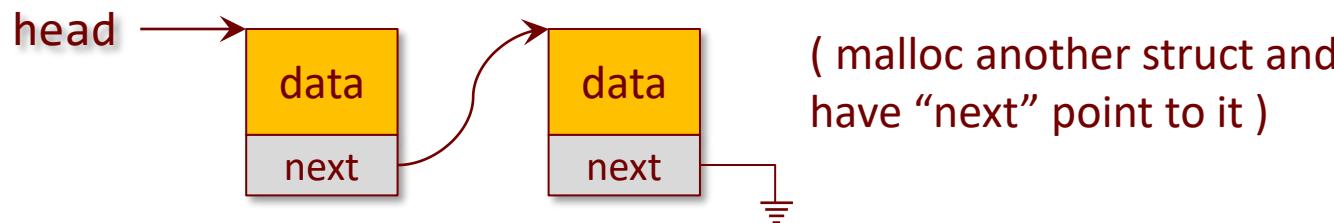
```
struct s1 {  
    .....  
    struct s2 *p;  
    .....  
};  
  
struct s2 {  
    .....  
    struct s1 *q;  
    .....  
};
```



Παράδειγμα – αναφορές / δυναμική λίστα (1/3)



Στον τελευταίο κόμβο πρέπει πάντα το “next” να είναι ίσο με NULL ώστε να βρίσκουμε που τελειώνει η λίστα.



Παράδειγμα – αναφορές / δυναμική λίστα (2/3)

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

struct listnode {                  /* Define our simple struct */
    int value;                      /* We store our data (a number) here */
    struct listnode *next;          /* Pointer to the next node in the list */
};

typedef struct listnode listnode_t;  /* for simplicity & clarity */

/* Calculate the sum of all numbers in the list */
void find_sum(listnode_t *node)
{
    int sum = 0;

    while (node != NULL) {          /* Traverse the list till its end */
        sum += node->value;         /* Value of current node */
        node = node->next;          /* Point to the next node */
    }
    printf("sum of numbers = %d\n", sum);
}
```

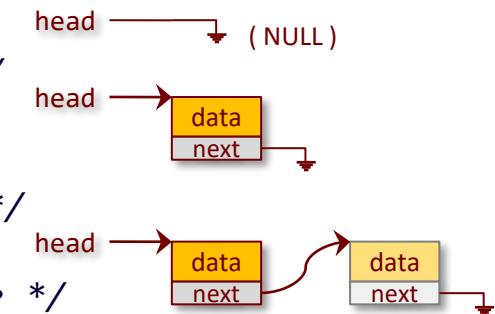
Παράδειγμα – αναφορές / δυναμική λίστα (3/3)

```
int main()
{
    listnode_t *node, *head = NULL; /* head: pointer to start of list (empty now) */
    int n;

    printf("How many numbers? ");
    scanf("%d", &n); /* This should be positive... */
    for (; n > 0; n--) {
        node = (listnode_t *) malloc(sizeof(listnode_t)); /* Malloc 1 node (struct) */
        if (node == NULL) return (1);

        scanf("%d", &node->value); /* Read & store number */

        if (head == NULL) { /* Empty list */
            node->next = NULL; /* No more nodes after this one */
            head = node; /* The only node in the list */
        }
        else { /* Non-empty list; insert new node */
            node->next = head; /* Inserted in the beginning */
            head = node; /* Head now points to the new node */
        }
    }
    find_sum(head); /* Call by passing the starting node of the list */
    return (0);
}
```



Η γλώσσα C

Ενώσεις (unions)



MYY502

Ενώσεις (Unions)

- ❖ Οι ενώσεις είναι μια ειδική περίπτωση «δομών» οι οποίες επιτρέπουν να κάνουμε οικονομία στη μνήμη.
 - Σε αντίθεση με τα structs, **δεν αποθηκεύονται όλα τα πεδία αλλά μόνο ένα από αυτά!!**
- ❖ Για παράδειγμα, ένα πρόγραμμα δέχεται ως είσοδο έναν int ή έναν double (ανάλογα με το τι επιλέγει ο χρήστης) – όχι και τα δύο μαζί. Πώς το αποθηκεύουμε αυτό;

```
int main() {                                     /* Απλή λύση: 2 μεταβλητές */
    int ival;
    float fval;
    char choice;
    scanf("%c", &choice);
    if (c == 'i') scanf("%d", &ival);
    if (c == 'f') scanf("%f", &fval);
    ...
}
```

- ❖ Μια ένωση αποτελείται από μια συλλογή πεδίων εκ των οποίων το πρόγραμμά μας μπορεί να επιλέξει ένα από όλα κάθε φορά.
- ❖ **Τα στοιχεία/πεδία μιας ένωσης μοιράζονται τον ίδιο χώρο μνήμης!!**
- ❖ Χειρισμός σαν να είναι structs.

```
typedef union trick {  
    int ival;  
    float fval;  
} trick;  
  
int main() {  
    trick value;      /* sizeof(value): 4 bytes, not 8 bytes! */  
    char choice;  
    scanf("%c", &choice);  
    if (c == 'i') scanf("%d", &value.ival);  
    if (c == 'f') scanf("%f", &value.fval);  
    ...  
}
```

Παράδειγμα – Ενώσεις

```
#define INT_TYPE 1
#define REAL_TYPE 2
struct item {
    int type;
    union {
        int ival;
        float fval;
    } value;
};

int main() {
    struct item x;

    x.type = INT_TYPE;
    x.value.ival = 4;
    printf("%d %f\n", x.value.ival, x.value.fval);
    x.type = REAL_TYPE;
    x.value.fval = 28000.5;
    printf("%d %f\n", x.value.ival, x.value.fval);
    return 0;
}
```

\$./a.out
4 0.000000
1188741376 28000.500000

Παράδειγμα – Ενώσεις

```
#define INT_TYPE 1
#define REAL_TYPE 2
struct item {
    int type;
    union {
        int ival;
        float fval;
    } value;
};

void print_item(struct item x) {
    if (x.type == INT_TYPE)
        printf("value = %d\n", x.value.ival);
    if (x.type == REAL_TYPE)
        printf("value = %f\n", x.value.fval);
}

int main() {
    struct item x = {INT_TYPE, { 4 }};
    print_item(x);
    return 0;
}
```

```
$ ./a.out
value = 4
```

Προγραμματισμός σε C

*Αρχεία κειμένου
(Text files)*



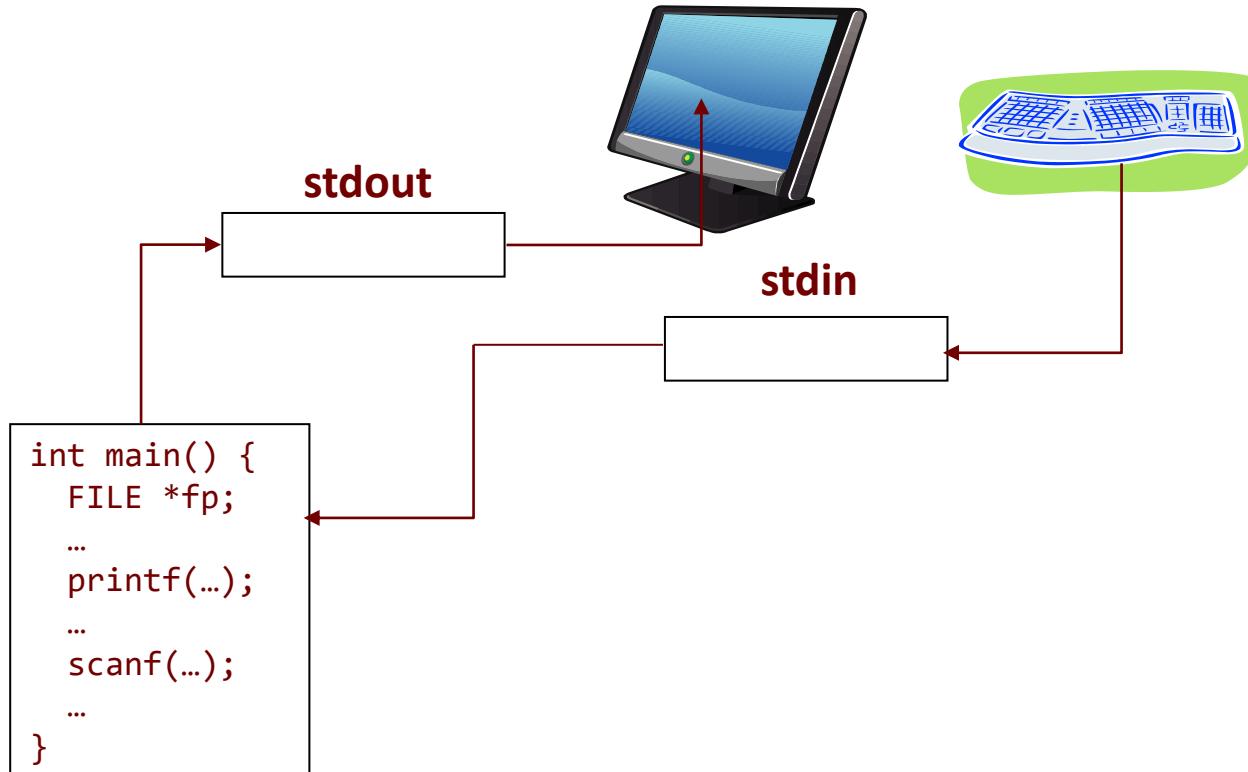
MYY502

Εισαγωγή

- ❖ Στη C έχουμε ειδικές συναρτήσεις για να επεξεργαζόμαστε αρχεία κειμένου που αποθηκεύονται στο δίσκο
 - Τα αρχεία είναι σημαντικά για μόνιμη αποθήκευση "κειμένου"
 - Αποθηκεύουν "απεριόριστη" πληροφορία.
 - Μπορείτε να τα θεωρήσετε ως ένα μεγάλο string που φυλάσσεται στον δίσκο και όχι στη μνήμη
 - Όμως ΔΕΝ είναι σαν τα strings διότι π.χ.
 - ✧ Το μέγεθός του μόνο αυξάνεται, δεν αφαιρείται τίποτε...
 - ✧ Δεν μπορείς να πας άμεσα σε όποιο στοιχείο (χαρακτήρα) θέλεις – συνήθως ξεκινάς από την αρχή και προχωράς ...
 - ✧ Επομένως δεν μπορείς να το χειριστείς με δείκτες.
- ❖ Προκειμένου να αποθηκευτεί / διαβαστεί πληροφορία σε αρχείο, απαιτείται η χρήση **ρευμάτων εισόδου/εξόδου (streams)**
 - ένα **ρεύμα εξόδου** μπορείτε να το φαντάζεστε σαν χώρο μνήμης στον οποίο το πρόγραμμα μόνο αποθηκεύει δεδομένα (χρησιμοποιώντας γνωστές συναρτήσεις – printf, fprintf, puts, fputs) τα οποία εν συνεχείᾳ τα παραλαμβάνει το λειτουργικό και τα στέλνει στο δίσκο.
 - ένα **ρεύμα εισόδου** μπορείτε να το φαντάζεστε σαν χώρο μνήμης στον οποίο το λειτουργικό αποθηκεύει δεδομένα από το δίσκο και εν συνεχείᾳ το πρόγραμμα μπορεί να παραλάβει αυτά τα δεδομένα για επεξεργασία χρησιμοποιώντας γνωστές συναρτήσεις (scanf, gets, fscanf, fgets...).

Ειδικά ρεύματα (streams)

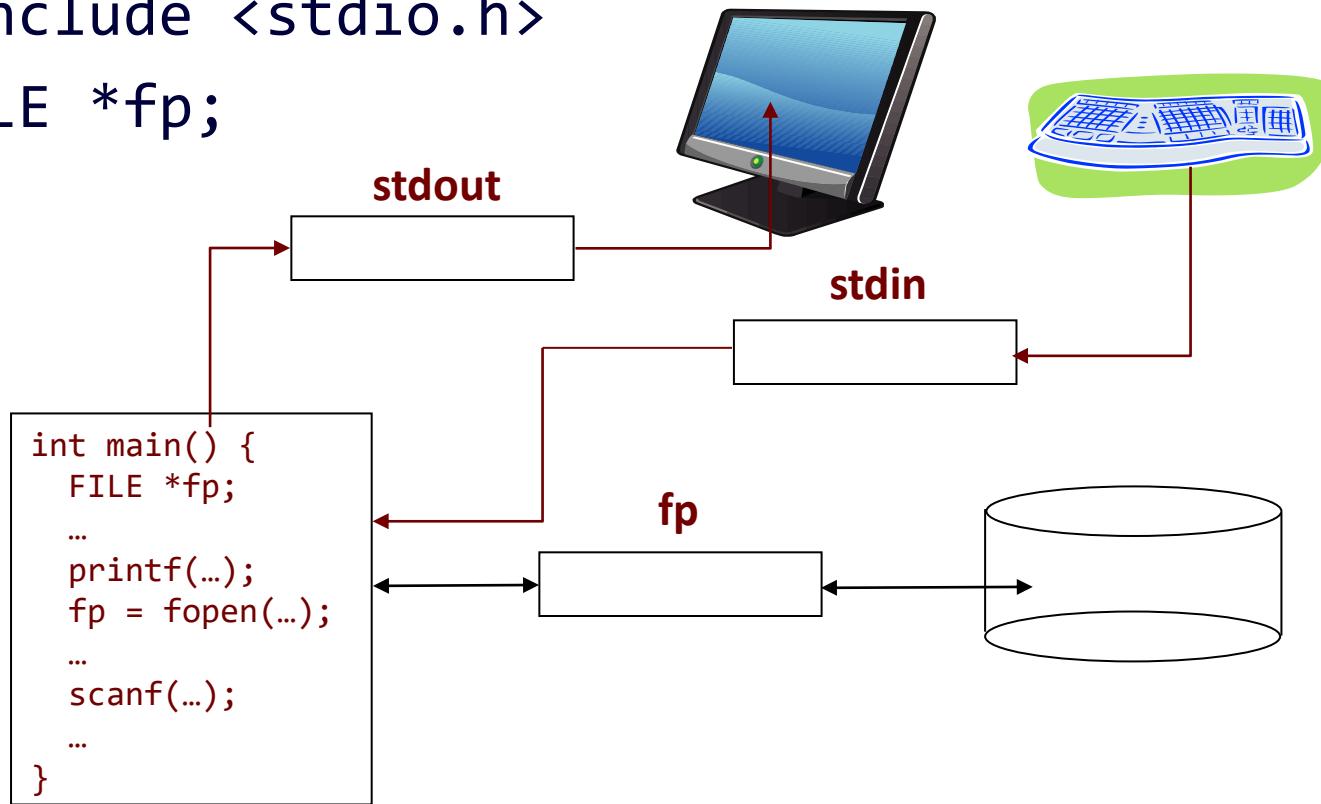
- ❖ **stdout** → γράφω σε ρεύμα που γίνεται flush στην οθόνη
- ❖ **stdin** → διαβάζω από ρεύμα στο οποίο γίνονται flush τα δεδομένα που περνάω από το πληκτρολόγιο



Ρεύματα που κατευθύνονται στο δίσκο

- ❖ Για να χρησιμοποιήσω από ένα πρόγραμμα κάποιο αρχείο χρειάζομαι ένα ρεύμα εισόδου/εξόδου το οποίο το δημιουργώ μέσω ενός δείκτη σε FILE.

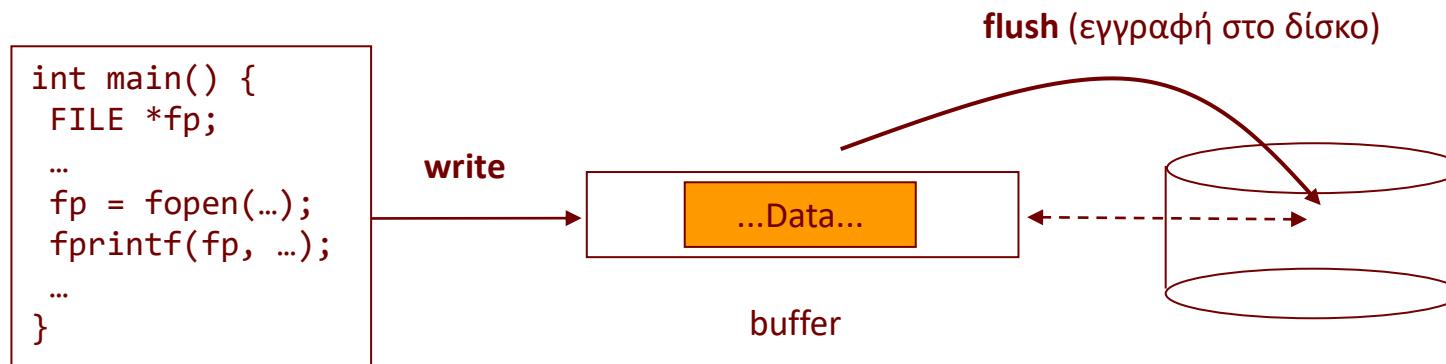
```
#include <stdio.h>
FILE *fp;
```



- ❖ FILE *fopen(char *name, char *mode);
 - name: όνομα αρχείου
 - mode:
 - ✧ "r" για ανάγνωση (αν δεν ξ, η fopen επιστρέφει NULL)
 - ✧ "w" για εγγραφή (αν ξ ήδη, τότε αδειάζουν τα περιεχόμενα)
 - ✧ "a" για επέκταση (αν ξ ήδη, τα περιεχόμενα διατηρούνται)
 - Άλλες επιλογές για mode
 - ✧ Η πρόσθεση του + μετά από έναν χαρακτήρα σημαίνει ότι επιτρέπεται και η αντίθετη χρήση ταυτόχρονα
 - ✧ "r+" για ανάγνωση / εγγραφή στην αρχή (αν δεν ξ, επιστρέφει NULL)
 - ✧ "w+" για ανάγνωση / εγγραφή με απόρριψη (αν ξ ήδη, άδειασμα)
 - ✧ "a+" για ανάγνωση / εγγραφή στο τέλος
- ❖ ΣΗΜΕΙΩΣΗ: τα ειδικά ρεύματα *stdout* και *stdin* είναι πάντα αρχικοποιημένα αυτόματα όταν ξεκινά ένα πρόγραμμα.

Αρχικοποίηση

- ❖ Η συνάρτηση fopen επιστρέφει έναν δείκτη (file pointer) σε ένα **ρεύμα** που συσχετίζεται με το αρχείο.
 - Άρα δεν μπορεί κανείς να προσβεί το αρχείο άμεσα μέσω του δείκτη (π.χ. δεν γίνεται, `*fp = 'a';`).
- ❖ Αν δεν υπάρχει αρχείο και ανοίγω με mode "w" ή "a", τότε δημιουργείται νέο αρχείο
- ❖ Αν κάτι πάει στραβά, η συνάρτηση επιστρέφει NULL



Βασικές Συναρτήσεις

- ❖ `int fscanf(FILE *fp, char *format, ...)`
 - Λειτουργεί όπως η `scanf`
 - Επιστρέφει αριθμό στοιχείων που διαβάστηκαν
 - Επιστρέφει **EOF** σε περίπτωση τέλους αρχείου ή λάθους
- ❖ `int fprintf(FILE *fp, char *format, ...)`
 - Λειτουργεί όπως η `printf` μόνο που γράφει στο `fp`
 - Επιστρέφει αριθμό χαρακτήρων που γράφτηκαν
 - Επιστρέφει < 0 σε περίπτωση λάθος
- ❖ Η `scanf` διαβάζει από τον ειδικό δείκτη αρχείου `stdin`, άρα:
`scanf("%d", &i);` ≡ `fscanf(stdin, "%d", &i);`
- ❖ Η `printf` γράφει στον ειδικό δείκτη αρχείου `stdout`, επομένως:
`printf("%d", i);` ≡ `fprintf(stdout, "%d", i);`

fgets

- ❖ `char *fgets(char *line, int maxline, FILE *fp);`
 - Διαβάζει το πολύ **maxline-1** χαρακτήρες
 - Αν βρει πρώτα αλλαγή γραμμής ('\n') διαβάζει μέχρι εκεί και η αλλαγή γραμμής ('\n') τοποθετείται στο line
 - Το line τερματίζει με το χαρακτήρα τερματισμού '\0'
 - Η συνάρτηση επιστρέφει **NULL** όταν τελειώσει το αρχείο ή συμβεί σφάλμα
- ❖ `char *gets(char *s);`
 - Η gets διαβάζει 1 γραμμή από το stdin ενώ αντικαθιστά τον τερματικό χαρακτήρα νέας γραμμής με '\0'.
 - Αν size of line < μέγεθος της γραμμής που διαβάζω από το stdin , τότε έχω undefined behavior
 - Συνιστάται η χρήση της fgets αντί της gets, όπως έχουμε πει
- ❖ `int fputs(char *s, FILE *fp);`
 - Γράφει το string s στο fp
 - Η συνάρτηση επιστρέφει τον αριθμό των χαρακτήρων που έγραψε και EOF σε περίπτωση λάθους
- ❖ `int puts(char *s);`
 - Γράφει το string s μαζί με τον χαρακτήρα αλλαγής γραμμής στο stdout

❖ `int fclose(FILE *fp);`

- Κλείνει το αρχείο (γίνεται flush o buffer)
 - Επιστρέφει 0 σε περίπτωση επιτυχίας
 - Επιστρέφει EOF σε περίπτωση λάθους
 - Το σύστημα εκτέλεσης κλείνει όλα τα ανοικτά αρχεία με τον τερματισμό του κυρίου προγράμματος. Αποτελεί καλή προγραμματιστική πρακτική, παρ' όλα αυτά, να κλείνουμε ρητά τα αρχεία που ανοίγουμε.
-
- ❖ Μπορεί να εκτελεστεί η εντολή `fclose(stdout)`, μετά όμως δεν μπορώ να γράψω – η `fprintf(stdout,...)` θα επιστρέφει **EOF**.

Λεπτομέρειες

- ❖ Ο δείκτης FILE *fp, δεν μας δίνει άμεση πρόσβαση στα περιεχόμενα του αρχείου (π.χ. ΔΕΝ ΜΠΟΡΟΥΜΕ ΝΑ ΚΑΝΟΥΜΕ ΑΡΙΘΜΗΤΙΚΗ ΜΕ AYTON). Άρα δεν μπορούμε να γράψουμε στο αρχείο με εντολές τύπου:
 - ~~*fp = 'a'; /* Ο δείκτης δεν δείχνει στα περιεχόμενα */~~
 - ~~strcpy(fp, "abc"); /* Το ίδιο */~~
- ❖ Τα περιεχόμενα τα διαβάζουμε / γράφουμε **ΜΟΝΟ** μέσω των fgets/fputs/fscanf/fprintf.
- ❖ Το αρχείο είναι σαν **TAINIA** ΑΠΟ ΚΑΣΕΤΑ. Όταν διαβάζουμε ή γράφουμε κάτι, **ΠΡΟΧΩΡΑΕΙ Η TAINIA ΑΠΟ ΜΟΝΗ ΤΗΣ.**
 - Επομένως, δεν προχωράμε εμείς τον δείκτη fp
 - Υπάρχει κλήση που μας επιστρέφει στην αρχή, ή σε κάποιο άλλο σημείο της "ταινίας".

Παράδειγμα 1 – fgets/fputs (διάβασμα γραμμή-γραμμή)

```
#include <stdio.h>

int main() {
    FILE *infile, *outfile;
    char buf[10];

    if ((infile = fopen("original.txt", "r")) == NULL)
        return 1;
    if ((outfile = fopen("copy.txt", "w")) == NULL)
        return 1;
    while (fgets(buf, 10, infile) != NULL) {
        fputs(buf, outfile);
    }
    fclose(infile);
    fclose(outfile);
    return 0;
}
```

Παράδειγμα 2 – fscanf/fprintf (διάβασμα λέξη-λέξη)

```
#include <stdio.h>

int main() {
    FILE *infile, *outfile;
    char buf[81];

    if ((infile = fopen("original.txt", "r")) == NULL)
        return 1;
    if ((outfile = fopen("copy.txt", "w")) == NULL)
        return 1;
    while (fscanf(infile, "%s", buf) != EOF) {
        fprintf(outfile, "%s", buf); /* Όλες οι λέξεις κολλητά */
        fprintf(stdout, "%s", buf); /* Εμφάνιση και στην οθόνη */
    }
    fclose(infile);
    fclose(outfile);
    return 0;
}
```

Παράδειγμα 3 – fflush/fclose

```
int main() {  
    FILE *fp;  
    fp = fopen("file", "w"); /* Should check for NULL */  
    fprintf(fp, "%s", "xxx");  
    while(1) ;  
    return 0;  
}
```

\$./a.out

^C

- ❖ Η εκτέλεση της εντολής "cat file" κατά την ώρα εκτέλεσης του προγράμματος δεν θα δείξει δεδομένα
- ❖ Το ίδιο θα συμβεί και μετά τον μη ομαλό τερματισμό του προγράμματος (με Control-C)
- ❖ Για την εγγραφή των δεδομένων πρέπει να προστεθεί η **fflush(fp);** πριν το while loop, η οποία **ολοκληρώνει** ότι **εγγραφές έχουν γίνει μέχρι εκείνη τη στιγμή στον δίσκο.**
- ❖ Η **fclose(fp);** κάνει ακριβώς το ίδιο (μόνο που επιπλέον κλείνει και το αρχείο).



Παράδειγμα 4 – Εύρεση μέγιστης λέξης

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

int main() {
    FILE *infile;
    char buf[101], maxWord[101];
    int maxLength = 0;

    if ((infile=fopen("testWords.ascii", "r")) == NULL) return 1;

    strcpy(maxWord, "");
    while (fscanf(infile, "%s", buf) != EOF) {
        if (strlen(buf)>maxLength) {
            strcpy(maxWord, buf);
            maxLength = strlen(buf);
        }
    }
    fprintf(stdout, "Max string is: %s with length %d\n",
            maxWord, maxLength);
    return 0;
}
```

Παράδειγμα 4 – Β' έκδοση, με χρήση stdin

```
/* Άμεση πληκτρολόγηση γραμμών κειμένου και τερματισμός με τον συνδυασμό Control-D (^D), που σηματοδοτεί το EOF */
#include <stdio.h>
#include <string.h>

int main() {
    char buf[101], maxWord[101];
    int maxLength = 0;

    strcpy(maxWord, "");
    while (fscanf(stdin, "%s", buf) != EOF) {
        if (strlen(buf)>maxLength) {
            strcpy(maxWord, buf);
            maxLength = strlen(buf);
        }
    }
    fprintf(stdout, "Max string is:%s with length %d\n",
            maxWord, maxLength);
    return 0;
}
```

Παράδειγμα 5 – fgets/sscanf (διάβασμα γραμμών + sscanf)

```
#include <stdio.h>

int main() {
    FILE *infile;
    int k;
    char s1[20], s2[20];
    float f;
    char buf[81];

    if ((infile = fopen("testSscanf.txt", "r")) == NULL) return 1;
    while (fgets(buf, 81, infile) != NULL) {
        puts(buf); /* show original line */
        sscanf(buf, "%d %s %s %f", &k, s1, s2, &f);
        printf("%d\t%s\t%s\t%f\n", k, s1, s2, f);
    }
    fclose(infile);
    return 0;
}
```

```
$cat testSscanf.txt
2 minutes to 12.00
4 days to 1.0 breakdown
3 months to 5
$./a.out
2 minutes to 12.00

2      minutes to      12.000000
4 days to 1.0 breakdown

4      days      to      1.000000
3 months to 5

3      months   to      5.000000
```

Παράδειγμα 6 – μίνι grep

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

int main(int argc, char *argv[]) {
    FILE *infile;
    char buf[101];

    if ((infile = fopen("testWords.ascii", "r"))==NULL) {
        return 1;
    }

    if (argc != 2) {
        printf("Usage: %s searchString\n", argv[0]);
        return 1;
    }

    while (fgets(buf, 101, infile) != NULL) {
        if (strstr(buf, argv[1]) != NULL)
            printf("%s", buf); /* No "%s\n" here ?? */
    }
    fclose(infile);
    return 0;
}
```

Παράδειγμα 7 – Εκτύπωση λέξεων (fgets + strtok)

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

int main() {
    FILE *infile;
    int c;
    char buf[81];
    char *p, tokens[]={ "\n\t";

    if ((infile = fopen("testWords.ascii", "r"))==NULL) return 1;

    while (fgets(buf, 81, infile) != NULL) {
        c = buf[strlen(buf)-1];
        p = strtok(buf, tokens); /* εκτύπωση λέξεων γραμμής στην οθόνη */
        while (p) {
            printf("%s ", p); /* ένα κενό μεταξύ των λέξεων */
            p = strtok(NULL, tokens);
        }
        if (c=='\n') printf("\n");
    }
    fclose(infile);
    return 0;
}
```

Παράδειγμα 8 – Μετρητής λέξης

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

void CheckDiomedes(char *X, int *counter) {
    char *c = strstr(X, "Diomhdh");

    while (c) {
        (*counter)++;
        c += strlen("Diomhdh");
        c = strstr(c, "Diomhdh");
    }
    printf("%s %d\n", X, *counter);
}

int main() {
    FILE *infile;
    char buf[81];
    int nOcc = 0;

    if ((infile = fopen("iliada.txt", "r"))==NULL)
        return 1;
    while (fscanf(infile, "%s", buf) != EOF)
        CheckDiomedes(buf, &nOcc);
    printf("Diomhdh: %d\n", nOcc);
    fclose(infile);
    return 0;
}
```

Παράδειγμα 9 – Ένωση αρχείων

```
/* Παράδειγμα εκτέλεσης: ./a.out f1.txt f2.txt bothfiles.txt */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    FILE *inp, *outp;
    char line[200];
    int i;

    if ((outp = fopen(argv[argc-1], "w")) == NULL) return 1;
    for (i = 1; i < argc - 1; i++) {
        if ((inp = fopen(argv[i], "r")) == NULL) continue;
        while (fgets(line, 200, inp) != NULL)
            fputs(line, outp);
        fclose(inp);
    }
    fclose(outp);
    return 0;
}
```

Θέση σε αρχείο

- ❖ Η δομή FILE διαχειρίζεται τη θέση του αρχείου στην οποία βρισκόμαστε αυτή τη στιγμή
 - Η αρίθμηση είναι από το 0 (αρχή του αρχείου, «πριν» τον πρώτο χαρακτήρα του αρχείου) μέχρι N, όπου N είναι το μέγεθος του αρχείου (θέση αμέσως μετά τον τελευταίο χαρακτήρα).
- ❖ Η θέση αυτή περιγράφεται από έναν ακέραιο long int.
 - Ουσιαστικά περιγράφει πόσους χαρακτήρες απέχουμε από την αρχή του αρχείου.
- ❖ Όπως γνωρίζουμε, αν καλέσουμε μια συνάρτηση εισόδου-εξόδου (π.χ. fscanf/fprintf), η τρέχουσα θέση αλλάζει αυτόματα μετά την ολοκλήρωσή της (προχωρά η "ταινία").
- ❖ Για να μάθουμε την τρέχουσα θέση:
 - long ftell(FILE *fp);
Επιστρέφει την τρέχουσα θέση στο αρχείο
- ❖ Μπορούμε να την τροποποιήσουμε με συγκεκριμένες εντολές



fseek / rewind

- ❖ `int fseek(FILE *fp, long offset, int pos);`
 - Η fseek ξεκινάει από την θέση **pos** και προσθέτει ή αφαιρεί **offset** θέσεις
 - Δυνατές τιμές για την pos
 - ✧ SEEK_CUR: τρέχουσα θέση στο αρχείο
 - ✧ SEEK_SET: αρχή αρχείου
 - ✧ SEEK_END: τέλος αρχείου
- ❖ `void rewind(FILE *fp);`
 - Τρέχουσα θέση = αρχή του αρχείου
 - `rewind(fp) ≡ fseek(fp, 0L, SEEK_SET);`
- ❖ Οι συναρτήσεις εγγραφής και ανάγνωσης (π.χ. fputs, fgets) αλλάζουν την τρέχουσα θέση.



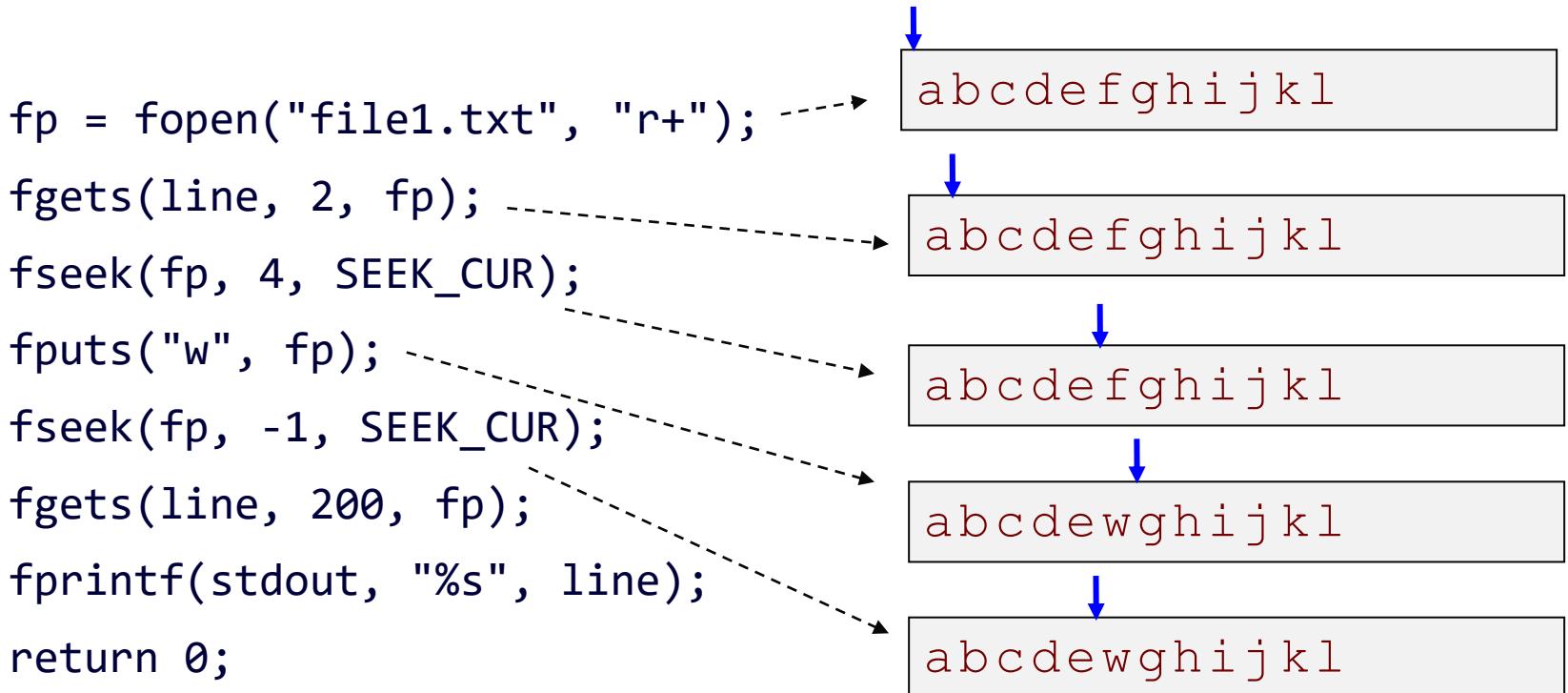
Παραδείγματα fseek

- ❖ `fseek(fp, 0L, SEEK_SET)`
 - Στην αρχή του αρχείου
- ❖ `fseek(fp, 0L, SEEK_END)`
 - Στο τέλος του αρχείου (μετά τον τελευταίο χαρακτήρα)
- ❖ `fseek(fp, (long) -1, SEEK_END)`
 - Πριν τον τελευταίο χαρακτήρα. Η επόμενη ανάγνωση θα διαβάσει τον τελευταίο χαρακτήρα.
- ❖ `fseek(fp, ftell(fp), SEEK_SET)`
`fseek(fp, 0L, SEEK_CUR)`
 - Μείνε εδώ που είσαι (ισοδύναμα)
- ❖ Κατά το άνοιγμα του αρχείου με:
 - `FILE *fp = fopen("file", "r+");`
Η αρχική θέση θα είναι στην αρχή του αρχείου
 - `FILE *fp = fopen("file", "a+");`
Η αρχική θέση θα είναι στο τέλος του αρχείου

Παράδειγμα fseek

```
#include <stdio.h>
int main() {
    FILE *fp;
    char line[200];
    fp = fopen("file1.txt", "r+");
    fgets(line, 2, fp);
    fseek(fp, 4, SEEK_CUR);
    fputs("w", fp);
    fseek(fp, -1, SEEK_CUR);
    fgets(line, 200, fp);
    printf(stdout, "%s", line);
    return 0;
}
```

```
$./a.out
wghijkl
$cat file1.txt
abcdewghijkl
```



Παράδειγμα ftell

```
#include <stdio.h>
int main() {
    FILE *fp;
    char line[200];

    fp = fopen("file2.test", "r+");
    printf("%ld\n", ftell(fp));
    fseek(fp, ftell(fp)+4, SEEK_SET);
    fputs("w", fp);
    fseek(fp, ftell(fp)-1, SEEK_SET);
    fgets(line, 200, fp);
    fprintf(stdout, "%s", line);
    printf("%ld\n", ftell(fp));
    fseek(fp, 4, SEEK_SET);
    fgets(line, 200, fp);
    fprintf(stdout, "%s", line);
    fclose(fp);
    return 0;
}
```

\$cat file2.test

abcdefghijkl

\$./a.out

0

wabcdefghijkl

13

wabcdefghijkl

\$cat file2.test

abcdwabcdefghijkl

Προσθήκη δεδομένων ("a+")

```
#include <stdio.h>

int main() {
    FILE *fp;
    char buf[81] = "22 hours ago";

    fp = fopen("testMode.txt", "a+");
    if (fp == NULL) return 1;
    fputs(buf, fp);      /* γράφει στο τέλος του αρχείου */

    rewind(fp);
    while (fgets(buf, 81, fp) != NULL)
        fputs(buf, stdout);    /* εκτύπωση στην οθόνη */
    fclose(fp);
    return 0;
}
```

```
$ cat testMode.txt
2 minutes to 12.00
4 days to 1.0 breakdown
3 months to 5
$ ./a.out
...
$ cat testMode.txt
2 minutes to 12.00
4 days to 1.0 breakdown
3 months to 5
22 hours ago
```

Προσθήκη δεδομένων ("r+")

```
#include <stdio.h>

int main() {
    FILE *fp;
    char buf[81] = "22 hours ago";

    fp = fopen("testMode.txt", "r+");
    if (fp == NULL) return 1;
    fputs(buf, fp);      /* γράφει στη τρέχουσα θέση, δηλαδή στην
                           αρχή του αρχείου */

    rewind(fp);
    while (fgets(buf, 81, fp) != NULL)
        fputs(buf, stdout);    /* εκτύπωση στην οθόνη */
    fclose(fp);
    return 0;
}
```

```
$ cat testMode.txt
2 minutes to 12.00
4 days to 1.0 breakdown
3 months to 5
$ ./a.out
...
$ cat testMode.txt
22 hours ago 12.00
4 days to 1.0 breakdown
3 months to 5
```

Προσθήκη δεδομένων ("w+")

```
#include <stdio.h>

int main() {
    FILE *fp;
    char buf[81] = "22 hours ago";

    fp = fopen("testMode.txt", "w+");
    if (fp == NULL) return 1;
    fputs(buf, fp);      /* γράφει στην αρχή του αρχείου, όλα τα
                           προηγούμενα δεδομένα χάνονται */
    rewind(fp);
    while (fgets(buf, 81, fp) != NULL)
        fputs(buf, stdout);    /* εκτύπωση στην οθόνη */
    fclose(fp);
    return 0;
}
```

```
$ cat testMode.txt
2 minutes to 12.00
4 days to 1.0 breakdown
3 months to 5
$ ./a.out
...
$ cat testMode.txt
22 hours ago
```