Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #7

Παρασκευή, 23 Απριλίου 2021

Παναγιώτης Παύλου

c-programming@allos.gr

Δείκτες/Pointers μεταβλητών

Πως αποθηκεύονται οι μεταβλητές στη μνήμη
και βάσει αυτού ποιες δυνατότητες ανοίγονται

Το μοντέλο της μνήμης

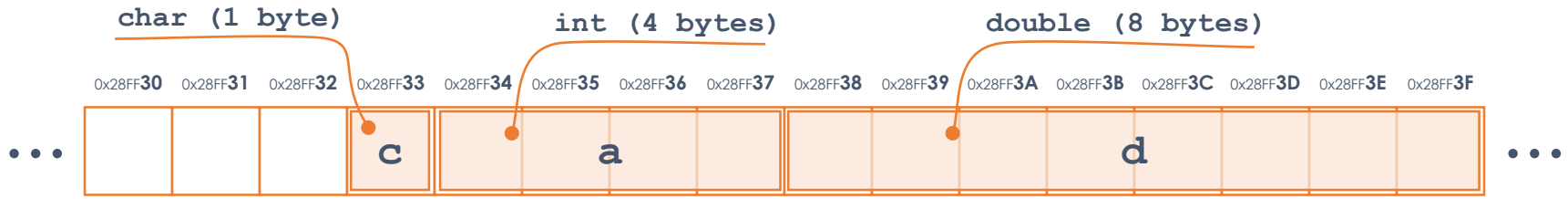
Η μνήμη του υπολογιστή μοντελοποιείται ως μία σειρά από "κουτάκια" που ονομάζονται **θέσεις μνήμης**. Η κάθε θέση μνήμης είναι μεγέθους 1byte, δηλαδή μπορεί να αποθηκεύσει μία τιμή των 8bits.

Οι θέσεις μνήμης έχουν συνεχόμενη αρίθμηση ξεκινώντας από το 0. Ο αριθμός που αντιστοιχεί στην κάθε θέση μνήμης ονομάζεται **διεύθυνση της θέσης μνήμης**.

Η διεύθυνση της θέσης μνήμης συνηθίζεται να γράφεται στο **δεκαεξαδικό** σύστημα αρίθμησης.

Αποθήκευση μεταβλητών στη μνήμη

Ήδη γνωρίζουμε ότι οι μεταβλητές αποθηκεύονται στη μνήμη του υπολογιστή, επίσης γνωρίζουμε ότι ανάλογα με τον τύπο δεδομένων η κάθε μεταβλητή χρειάζεται διαφορετικό αριθμό από bits για την αποθήκευσή της.

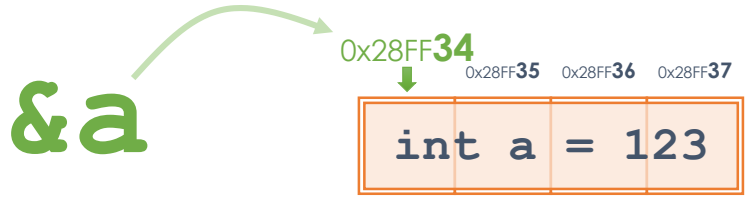


Οι μεταβλητές που απαιτούν περισσότερες από μία θέσεις μνήμης αποθηκεύονται σε διαδοχικά bytes, λέμε όμως ότι αποθηκεύονται στην **πρώτη** από αυτές.

Δηλαδή η μεταβλητή `a` δίπλα, είναι αποθηκευμένη στη θέση μνήμης `0x28FF34`.

Pointer σε μεταβλητή 1/4

Πως μπορούμε να προσδιορίσουμε αυτή την πρώτη θέση μνήμης που είναι αποθηκευμένη η μεταβλητή;



Γίνεται με τη χρήση του τελεστή **&** ο οποίος διαβάζεται και "address of".

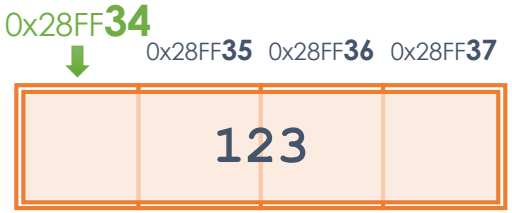
Η τιμή αυτή ονομάζεται **δείκτης** ή **pointer** στην μεταβλητή **a**.

Η πρώτη θέση μνήμης (από τον pointer) μαζί με το πλήθος των bytes (από τον τύπο δεδομένων) αρκούν για τον επακριβή εντοπισμό της περιοχής της μνήμης στην οποία είναι αποθηκευμένη η τιμή της μεταβλητής.

Pointer σε μεταβλητή 2/4

Την τιμή του pointer σε τι τύπο μεταβλητής την αποθηκεύουμε;

```
int a = 123;
int *Ap = &a;
printf("%p\n", Ap);
```

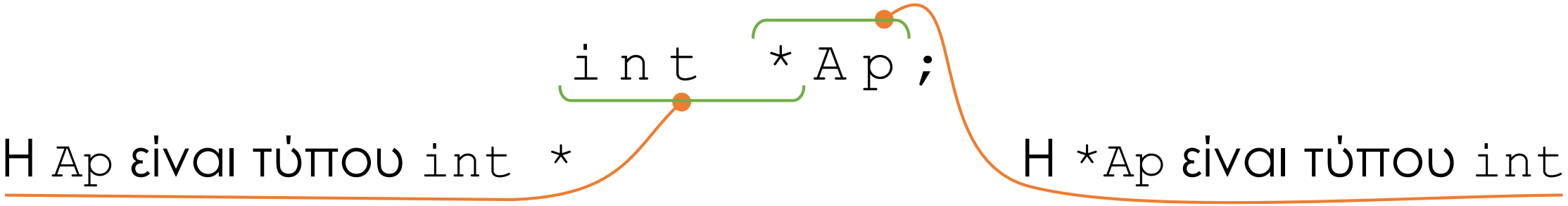


Η μεταβλητή `Ap` είναι τύπου pointer και για την ακρίβεια **δείκτη/pointer σε int** και συντάσσεται γράφοντας πρώτα τον τύπο δεδομένων της μεταβλητής στην οποία δείχνει ακολουθούμενο από τον μοναδιαίο τελεστή *****

Και εκτύπωσή της γίνεται με τη χρήση του format specifier **%p** για την εντολή printf, η οποία εκτυπώνει τη δεκαεξαδική τιμή της διεύθυνσης χωρίς το πρόθεμα **0x** και έχει νόημα μόνο στα πλαίσια του debugging.

Pointer σε μεταβλητή 3/4

Η δήλωση μιας τέτοιας μεταβλητής τύπου pointer (σε int) διαβάζεται με δύο τρόπους:

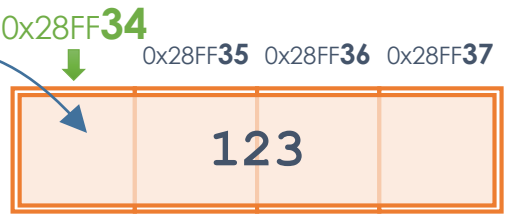


Και όντως η παράσταση `*Ap` αντιστοιχεί στην αντίστροφη λειτουργία από το `&a`, δηλαδή επιστρέφει την (ακέραια) τιμή εκεί που δείχνει ο pointer.

Άρα όταν **`Ap = &a`** τότε τα **`*Ap`** και **`a`** ταυτίζονται.

Παράδειγμα

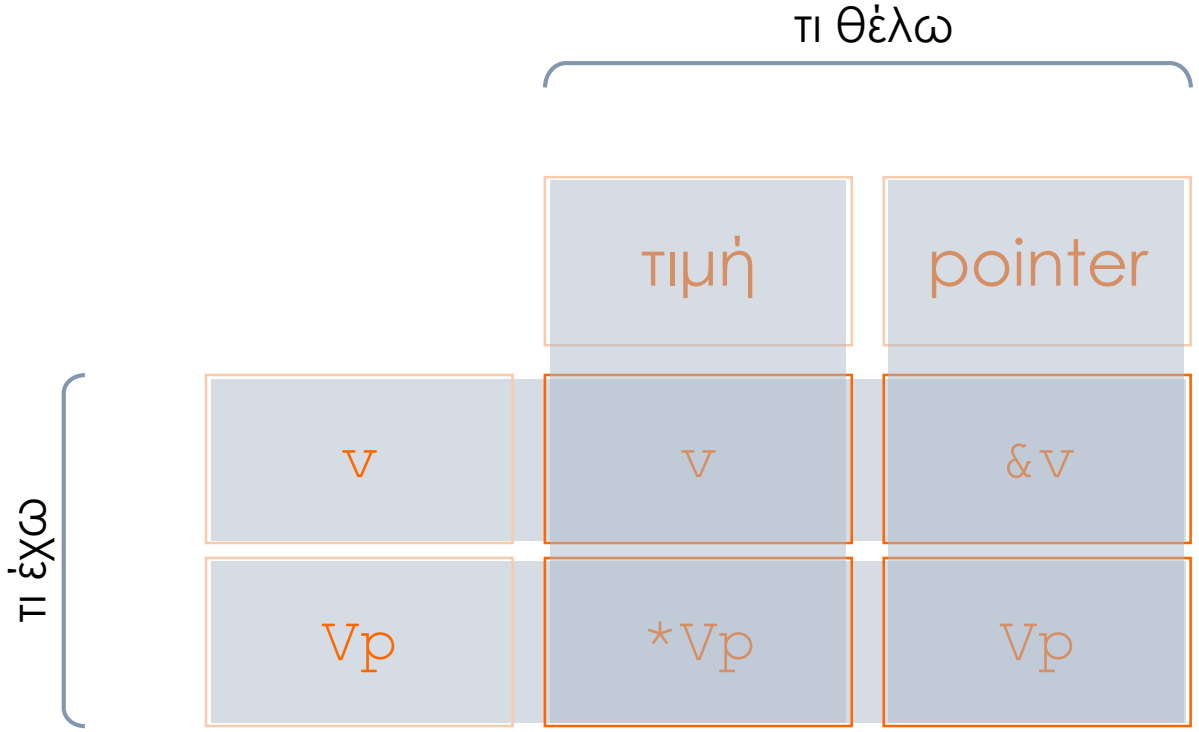
```
int a = 123;
int *Ap = &a;
printf("a = %d , *Ap = %d\n", a, *Ap);
*Ap = 234;
printf("a = %d , *Ap = %d\n", a, *Ap);
```




```
a = 123 , *Ap = 123
a = 234 , *Ap = 234
```

Pointer σε μεταβλητή 4/4

Άρα για μια μεταβλητή v οποιουδήποτε τύπου δεδομένων και για έναν pointer σε αυτή `vp =&v` ισχύει:



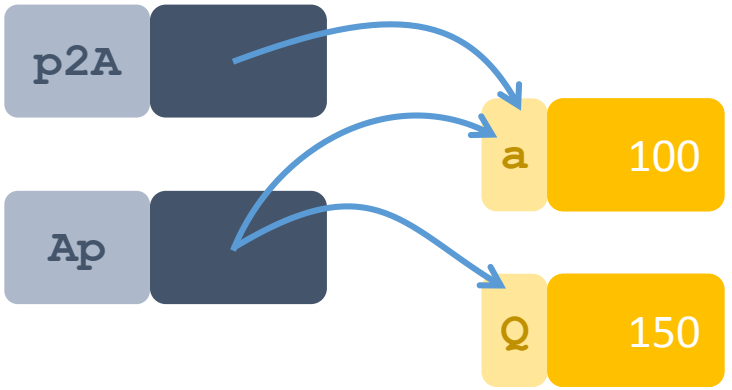
Call by reference

Η χρήση των pointers είναι πολύ εκτεταμένη και αποτελεί τον πυρήνα της C. Η απλούστερη εφαρμογή της είναι η παράκαμψη της προστασίας των ορισμάτων κατά την κλήση μιας συνάρτησης. Η πρακτική αυτή ονομάζεται **call by reference** σε αντίθεση με την τυπική κλήση της C που ονομάζεται call by value.

Ουσιαστικά αυτό που συμβαίνει είναι ότι αντί να δοθεί ως όρισμα η ίδια η μεταβλητή και να αντιγραφεί η τιμή της στη συνάρτηση, δίνεται ως όρισμα ο pointer στη μεταβλητή που μας ενδιαφέρει, οπότε και το αντίγραφο του pointer δείχνει στην ίδια μεταβλητή.

ΠΡΟΣΟΧΗ! Είναι πολύ εύκολο εκ παραδρομής ο δείκτης σε μια τοπική μεταβλητή να διατηρηθεί (π.χ. επιστρεφόμενος από μία συνάρτηση) έξω από την εμβέλεια της μεταβλητής, πράγμα το οποίο είναι προφανώς λανθασμένο.

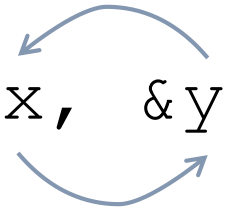
```
void changeTo100(int *Ap) {
    int Q = 123;
    *Ap = 100;
    Ap = &Q;
    *Ap = 150;
}
```




```
int main() {
    int a = 200;
    int *p2A = &a;
    changeTo100( p2A );
    return 0;
}
```

Εφαρμογή - Swap variables

```
void swap(double *a, double *b) {
     double x = *a;
     *a = *b;
     *b = x;
}

...
swap( &x, &y );
...
```



scanf - εισαγωγή από το πληκτρολόγιο

Όπως η εντολή `printf` εκτυπώνει στην οθόνη, έτσι και η εντολή `scanf` διαβάζει από το πληκτρολόγιο. Και για να μπορεί να αποδίδει σε μεταβλητές τις τιμές που διαβάζει, δέχεται τα ορίσματά της ως δείκτες στις μεταβλητές αυτές.

Η σύνταξή της είναι ανάλογη της printf:

```
scanf("%lf %d", &d, &c);
```

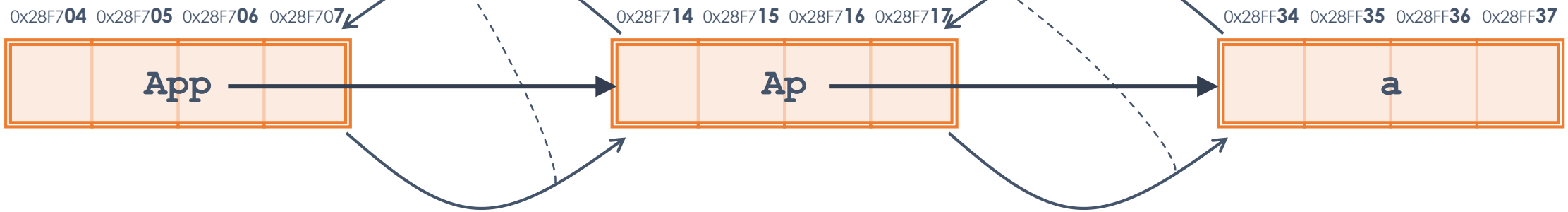
Δηλαδή υπάρχει το format string που δηλώνει τη μορφή των δεδομένων που αναμένεται να διαβαστούν και ακολουθούν ένα όρισμα για κάθε "όρο", του format string σε αντιστοιχία. Τα ορίσματα αυτά είναι (αναγκαστικά) δείκτες στις μεταβλητές οι οποίες θα υποδεχθούν τις τιμές.

Προσέξτε τις διαθέσιμες επιλογές για τους όρους του format string, δεν είναι το σύνολο των υποστηριζόμενων για την printf. Συμβουλευτείτε τα σχετικά παραρτήματα των σημειώσεων. Δείτε ιδιαιτέρως το `%i`

Pointer σε Pointer

Εφόσον μια μεταβλητή pointer είναι ... μια μεταβλητή, αποθηκεύεται και αυτή κάπου στη μνήμη. Άρα μπορώ να γράψω:

```
int a = 123;
int *Ap = &a;
int **App = &Ap;
**App = 888;          // αλλάζει η τιμή του a
// K.O.K.
```



Δείκτες σε δομές δεδομένων

Πως εφαρμόζονται οι δείκτες στις δομές δεδομένων

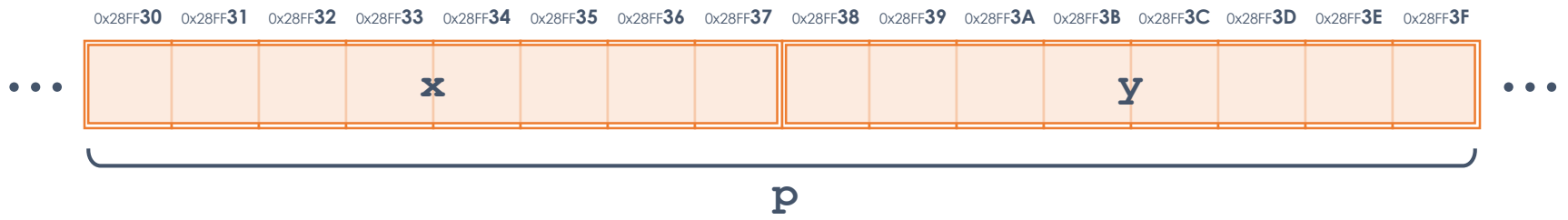
Αποθήκευση δομών δεδομένων στη μνήμη

Οι δομές δεδομένων (struct) αποτελούν ομαδοποιήσεις μεταβλητών σχετικές μεταξύ τους. Έτσι όταν δηλώνεται μια μεταβλητή τέτοιου τύπου, τα πεδία της βρίσκονται αποθηκευμένα συνεχόμενα στη μνήμη.

Δείτε το παράδειγμα:

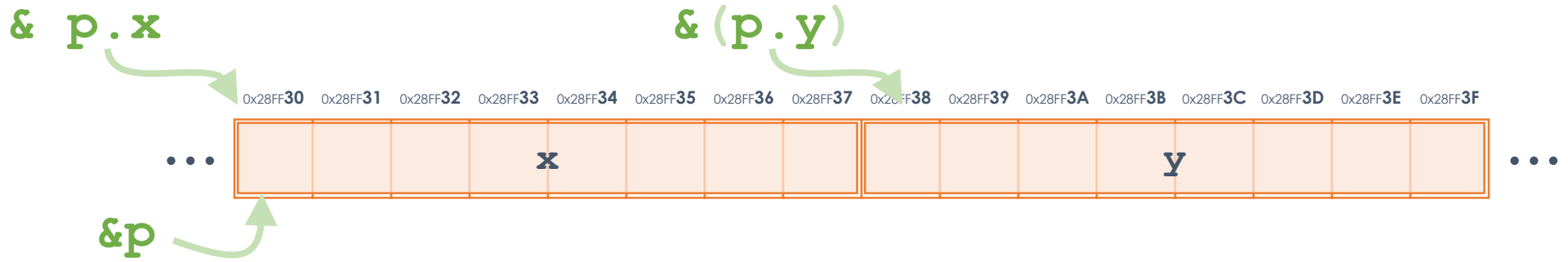
```
typedef struct _point2d {
    double x;
    double y;
} Point2D ;
...
Point2D p;
```

Η σειρά με την οποία αποθηκεύονται συμπίπτει με τη σειρά την οποία δηλώνονται.



Pointers σε δομές δεδομένων

Τα πεδία μιας δομής δεδομένων λειτουργούν ως απλές μεταβλητές. Άρα μπορεί με τη χρήση του τελεστή & να ληφθεί ο pointer σε αυτές.



Με βάση το παράδειγμα της προηγούμενης διαφάνειας, οι pointers στα πεδία x και y γράφονται απλά όπως παραπάνω. Λόγω της υψηλότερης προτεραιότητας που έχει ο τελεστής . ως προς τον & οι παρενθέσεις που παρουσιάζονται περισσεύουν.

Όσον αφορά την ίδια τη μεταβλητή `p`, όπως και στις απλές μεταβλητές, ο δείκτης σε αυτή ταυτίζεται με την πρώτη διεύθυνση της μνήμης η οποία χρησιμοποιείται. Οπότε η παρακάτω ισότητα ισχύει πάντα για το πρώτο πεδίο της (εδώ το x).

```
&p == &p.x
```

Χρήση pointers δομών δεδομένων

Ο δείκτης στη δομή δεδομένων δηλώνεται όπως και αυτός των απλών μεταβλητών. Άρα μπορούμε να γράψουμε:

```
Point2D p;
Point2D *Pp = &p;
```

Όμως η αναφορά σε ένα στοιχείο με δεδομένο των pointer θέλει προσοχή στη γραφή της:

~~`*Pp.x`~~ `(*Pp).x`

καθώς ο τελεστής **.** έχει υψηλότερη προτεραιότητα από τον * επειδή όμως αυτό δυσκολεύει την αναγνωσιμότητα του κώδικα υπάρχει ο ισοδύναμος τελεστής ->

`Pp->x` *ή* `Pp->y`

Παράδειγμα χρήσης

Όταν γίνεται κλήση μιας συνάρτησης με όρισμα μία δομή, επειδή η κλήση γίνεται με call by value, όλα τα πεδία της δομής του ορίσματος αντιγράφονται στα αντίστοιχα πεδία της παραμέτρου (και επειδή τα δεδομένα μιας δομής μπορεί να είναι πάρα πολλά) αυτό μπορεί να προκαλέσει μη αμελητέα καθυστέρηση.

Πέρα από αυτόν τον λόγο υπάρχουν και άλλοι σπουδαιότεροι που θα δούμε αργότερα, όμως όταν μία συνάρτηση χρειάζεται πρόσβαση στα στοιχεία μιας δομής συνηθίζεται να δίνεται στη συνάρτηση ο δείκτης στη δομή.

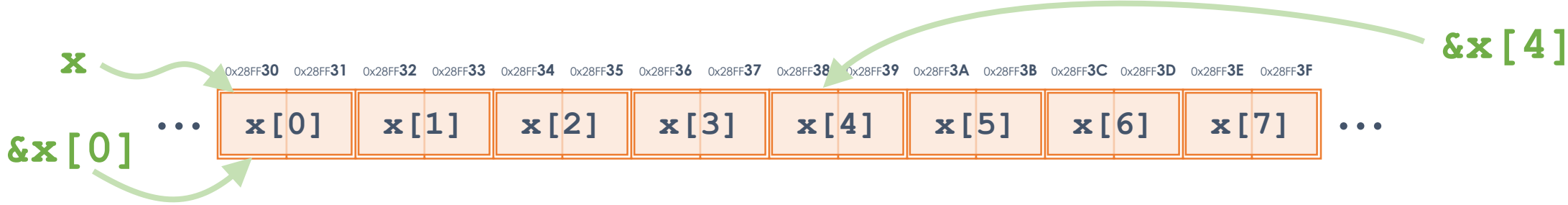
```
double distance(
    Point2D *A,
    Point2D *B
) {
    double dx = B->x - A->x;
    double dy = B->y - A->y;
    return sqrt(
        dx*dx + dy*dy
    );
}
```

Δείκτες σε πίνακες

Πως εφαρμόζονται οι δείκτες στους πίνακες - Μια ιδιαίτερη σχέση

Αποθήκευση πινάκων στη μνήμη

Οι πίνακες, όπως έχει ήδη αναφερθεί από το σχετικό μάθημα, έχουν τα στοιχεία τους αποθηκευμένα στη μνήμη διαδοχικά. Δείτε το παράδειγμα του `short int x[8]` στο σχήμα:



Βλέπουμε ότι οι δείκτες στα μεμονωμένα στοιχεία, τα οποία λειτουργούν (όπως ξέρουμε) ως μεταβλητές, λαμβάνονται κατά τον γνωστό τρόπο.

Επίσης ως δείκτης σε πίνακα ορίζεται - κατ' αναλογία με τα προηγούμενα - η πρώτη διεύθυνση στην οποία είναι αποθηκευμένο το πρώτο στοιχείο του πίνακα, δηλαδή ο δείκτης στο πρώτο στοιχείο του πίνακα.

Προσοχή όμως, το όνομα του πίνακα **είναι** ο δείκτης στον πίνακα, έτσι δεν πρέπει να χρησιμοποιηθεί ο τελεστής & και κατά συνέπεια ισχύει πάντα αυτή η έκφραση:

```
x == &x[0]
```

Σημείο προσοχής!

Αφού το όνομα του πίνακα ταυτίζεται με τον δείκτη του πίνακα, αυτό σημαίνει ότι και ο τύπος δεδομένων του ταυτίζεται με δείκτη σε κατάλληλο τύπο δεδομένων. Π.χ. για έναν πίνακα ακεραίων:

```
int a[10];
int *b;

b = a;

b[4] = 888;
printf("%d\n", a[4]); // εμφανίζει 888
```

άρα ο δείκτης σε κάποιο τύπο δεδομένων συμπίπτει ως χρήση με πίνακα στον ίδιο τύπο δεδομένων.

Προσοχή όμως! Είναι ευθύνη του προγραμματιστή να μην τα χρησιμοποιήσει λανθασμένα.

Ως συνεπακόλουθο είναι οι δύο παρακάτω δηλώσεις να είναι (σχεδόν) ισοδύναμες:

```
int a[];
int *a;
```

Αριθμητική Pointers 1/2

Μέχρι αυτό το σημείο δεν έχουμε αναφερθεί παρά μόνο σε εκχωρήσεις τιμών σε pointers και χρήση τους.

Μια μεταβλητή τύπου pointer δεν διαφέρει στη λογική της από άλλες μεταβλητές, είναι δηλαδή αρχικά ακαθόριστη, άρα δείχνει δηλαδή σε άγνωστο κάθε φορά σημείο της μνήμης.

Εάν κατά τη δήλωση ενός pointer δεν είναι δυνατόν να εκχωρηθεί τιμή, τότε θα πρέπει να δίνεται η μηδενική.

```
int *A = NULL;
```

Η ειδική τιμή NULL είναι το μηδέν μαζί με έναν τύπο δεδομένων pointer. Αυτή ορίζεται στο stddef.h (το οποίο γίνεται έμμεσα include από σχεδόν όλα τα άλλα γνωστά includes) και στο σχετικό σημείο του γράφει:

```
#define NULL ((void *)0)
```

Η πρακτική αυτή βοηθά με τον εξής τρόπο:

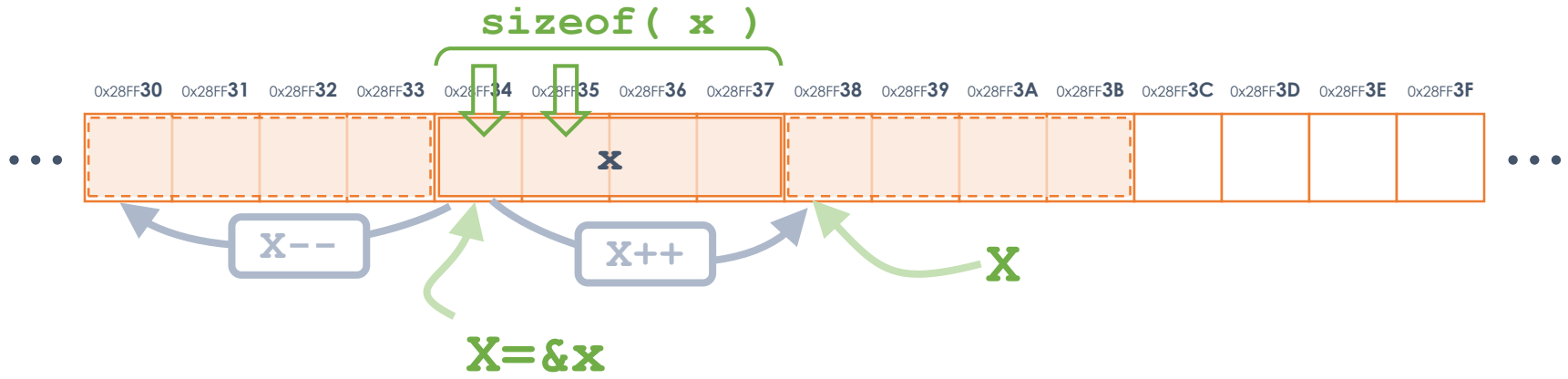
- Εάν χρησιμοποιηθεί η αρχική, ακαθόριστη τιμή του δείκτη τότε θα προκύψει ένα επίσης ακαθόριστο αποτέλεσμα, που στην **καλύτερη** περίπτωση θα είναι να τερματιστεί ανώμαλα το πρόγραμμα (να crashάρει όπως λέμε).
- Ενώ με την ειδική τιμή **NULL** είναι δεδομένο ότι θα προκύψει ανώμαλος τερματισμός και ενδεχομένως με την ένδειξη *NULL pointer exception*. Άρα τουλάχιστον το πρόγραμμα δεν θα συμπεριφέρεται αλλοπρόσαλλα παράγοντας τυχαία, διαφορετικά σφάλματα σε κάθε εκτέλεση.

Αριθμητική Pointers 2/2

Επιτρέπονται αριθμητικές πράξεις με τους δείκτες; Ναι, αλλά μόνο οι προσθετικές. Όμως από αυτές οι σχεδόν αποκλειστικά χρησιμοποιούμενες είναι οι ++ και −− και αυτές μόνο θα δούμε παρακάτω.

Η αύξηση (ή η μείωση) της τιμής ενός δείκτη κατά μία θέση μνήμης δεν έχει πρακτική χρήση, καθώς στην πλειονότητα των περιπτώσεων θα έδειχνε πλέον ο δείκτης σε θέση μνήμης που δεν έχει κάποια χρήσιμη πληροφορία.

Έτσι οι τελεστές ++ και −− μεταβάλουν την τιμή του δείκτη κατάλληλα, ώστε πλέον να δείχνει στην αμέσως επόμενη ή προηγούμενη θέση μιας τέτοιας μεταβλητής.



Αυτό έχει νόημα κυρίως όταν ο δείκτης χρησιμοποιείται στα πλαίσια ενός πίνακα, όπου με την κάθε αύξηση (++) ο δείκτης δείχνει στο επόμενο στοιχείο του πίνακα και αντίθετα με τη μείωση (−−) στο προηγούμενο.

Η ποσότητα της μεταβολής αυτής σε bytes είναι όσο το μέγεθος του τύπου δεδομένων. Το μέγεθος ενός τύπου δεδομένων δίνεται με τον τελεστή **`sizeof`**`( )` ο οποίος εφαρμόζεται είτε σε μεταβλητές αυτού του τύπου, είτε στον ίδιο τον τύπο δεδομένων.

`sizeof``( x )` ή **`sizeof`**`( int )`

Προτεραιότητα των νέων τελεστών

Στον πίνακα προτεραιότητας των τελεστών μπορείτε να δείτε τις προτεραιότητες όλων των τελεστών. Εδώ βλέπουμε τις σχετικές προτεραιότητες των τελεστών αυτής της ενότητας.

Οι προτεραιότητες συχνά μας μπερδεύουν έτσι καλό είναι να χρησιμοποιούμε παρενθέσεις αφού δεν κοστίζουν υπολογιστικά τίποτα.

Τελεστής	Προτεραιότητα
++ -- (ως επίθεμα)	1
[ ]	
.	
->	
++ -- (ως πρόθεμα)	2
*	
&	
sizeof	

π.χ. η έκφραση `*somePointer++` μπορεί να μας προβληματίζει, οπότε ανάλογα το τι θέλουμε καλύτερα να γράφουμε:

```
(*somePointer)++
```

ή

```
*(somePointer++)
```

Εφαρμογές δεικτών - 1ο μέρος

Ποιός ο λόγος που μας ενδιαφέρουν οι δείκτες;

Κείμενα και pointers

Καθώς η αναπαράσταση των κειμένων γίνεται μέσω πινάκων στη C, αλλά και οι πίνακες είναι ισοδύναμοι με τους pointers, όλες οι συναρτήσεις που αφορούν κείμενα έχουν τις παραμέτρους τους δηλωμένες ως `char *` και όχι ως `char []` που βλέπαμε στο προηγούμενο μάθημα. Ακριβώς αυτό το `char *` είναι που μας έκρυβε η `smString` της smLib.

Δείτε για παράδειγμα πόσο πιο απλός είναι ο κώδικας στη συνάρτηση που επιστρέφει το κείμενο με τους περισσότερους χαρακτήρες.

Δείτε επίσης την `stringLength` που είναι παραλλαγή της `calculateLength` του προηγούμενου μαθήματος με pointer.

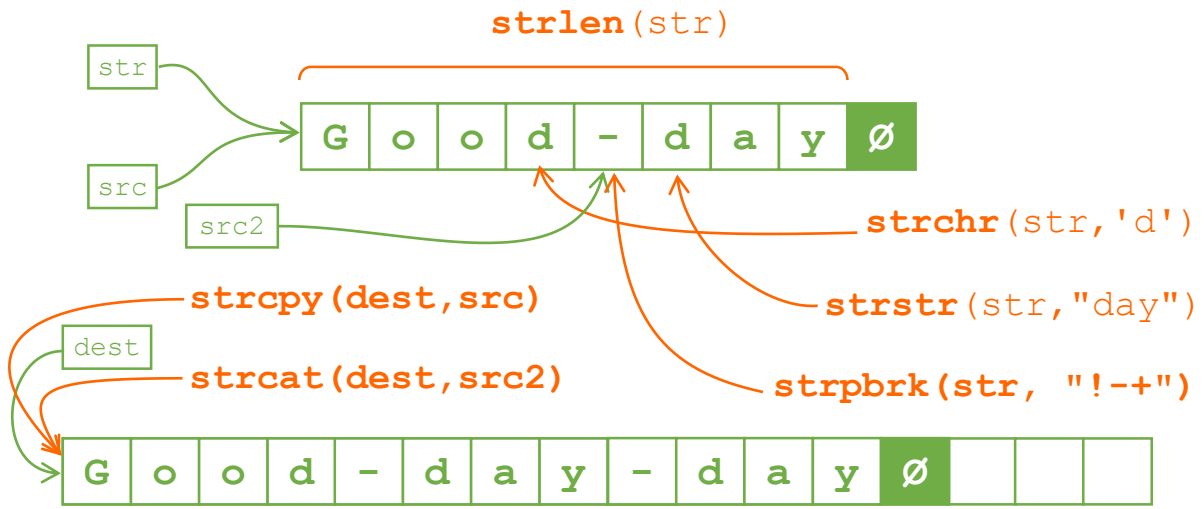
```
int stringLength(char *text) {
    int len=0;
    while (*text != 0) {
        text++;
        len++;
    }
    return len;
}

char *longestText(char *text1, char *text2) {
    if (stringLength(text2) > stringLength(text1))
        return text2;
    return text1;
}

int main() {
    char *t1 = "Goodday!", *t2 = "Goodnight!";
    printf("A. %s %d\n", t1, stringLength(t1));
    printf("B. %s %d\n", t2, stringLength(t2));
    printf("Longer : %s\n", longestText(t1,t2));
    return 0;
}
```

string.h

Η βιβλιοθήκη string έχει αρκετές έτοιμες συναρτήσεις για τη διαχείριση των κειμένων.

Πλήρη λίστα μπορείτε να δείτε εδώ και τις πιο σημαντικές θα τις βρείτε στο σχετικό παράρτημα των σημειώσεων.

Πολύ συχνά χρησιμοποιούμενες είναι αυτές που παραθέτουμε στα δεξιά.



`size_t strlen(char *str)`	Επιστρέφει το μήκος της συμβολοσειράς που δίνεται ως παράμετρος.
`char *strchr(char *str, int c)`	Βρίσκει τον χαρακτήρα c στο κείμενο str ή επιστρέφει NULL εάν δεν βρεθεί
`char *strstr( char *haystack, char *needle)`	Βρίσκει την πρώτη εμφάνιση του δεύτερου κειμένου μέσα στο πρώτο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή NULL εάν δεν υπάρχει.
`char *strpbrk( char *str1, char *str2)`	Βρίσκει τον πρώτο χαρακτήρα στο πρώτο κείμενο (str1) από αυτούς που υπάρχουν στο δεύτερο κείμενο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή NULL αν δεν υπάρχει κανένας.
`char *strcpy(char *dest, char *src)`	Αντιγράφει το 2ο κείμενο πάνω στο 1ο (υποθέτοντας ότι υπάρχει επαρκής χώρος)
`char *strcat(char *dest, char *src)`	Ενώνει δύο κείμενα στη θέση του πρώτου. Δηλαδή μετά το τέλος του 1ου κειμένου αντιγράφει το 2ο κείμενο.
`int strcmp(char *str1, char *str2)`	Συγκρίνει τα δύο κείμενα και επιστρέφει 0 εάν είναι ίσα, θετική τιμή εάν το 1ο κείμενο είναι λεξικογραφικά μεγαλύτερο από το 2ο και αρνητική εάν είναι μικρότερο.

strcmp(str, dest) ⟶ -1

Σημαντικά σημεία

Μετά από τη σημερινή διάλεξη θα πρέπει να γνωρίζετε:

- Το μοντέλο της μνήμης
- Δείκτες σε μεταβλητές, σε δομές και σε μονοδιάστατους πίνακες
- Τον δυισμό πίνακα/δείκτη
- Την κλήση με αναφορά / call-by-reference
- Βασική αριθμητική των δεικτών
- Τον τελεστή sizeof
- Τη χρήση των κειμένων μέσω pointers

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  c-programming@allos.gr

- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε μαζί τον κώδικα και τα μηνύματα λάθους από το CLion ως κείμενα με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!