Διάλεξη 3: Δείκτες (pointers) και Πίνακες

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:
Εισαγωγή στις έννοιες:

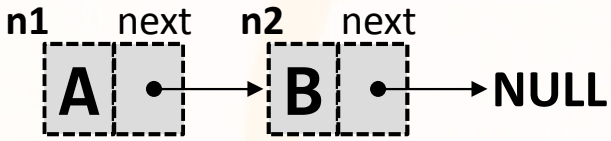
- Αριθμητική Δεικτών
- Δείκτες και Πίνακες
- Παραδείγματα

Διδάσκων: Παναγιώτης Ανδρέου

Δείκτες (Pointers)

Ορισμός Pointer: Μία μεταβλητή που περιέχει τη διεύθυνση μιας άλλης μεταβλητής. Δηλώνεται με τον **τύπο δεδομένων** και *, π.χ., `int * p`

- Δείκτες χρησιμοποιούνται στη C γιατί
  - Μερικές φορές είναι ο μόνος τρόπος για τη διατύπωση κάποιου υπολογισμού.
  - Συνήθως οδηγούν σε πιο περιεκτικό και αποτελεσματικό κώδικα.
- Η μνήμη είναι οργανωμένη ως μια σειρά κελιών, με διαδοχική αρίθμηση, ή διευθύνσεις, που μπορούν να χρησιμοποιηθούν ένα-ένα ή σε συνεχείς ομάδες.
  - 1 κελί τους ενός byte => π.χ., char, bool
  - 2 κελιά τους ενός byte => π.χ., short
  - 4 κελιά του ενός byte => π.χ., int
- Δείκτης είναι μια ομάδα κελιών 4 (x86) ή 8 (x64) που μπορούν να κρατήσουν μια διεύθυνση
- Παράδειγμα



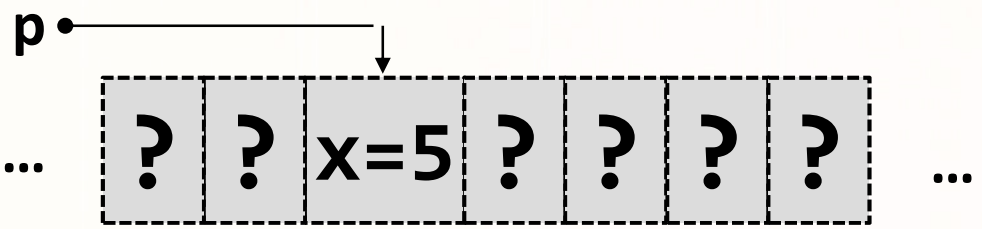
Node	Address	data	next
n2	3143928	B	0
	3143932		
n1	3143936	A	3143928

Βασικοί Τύποι Δεδομένων C - Περίληψη (για x86)

Τύπος	Μέγεθος	Πεδίο Τιμών	Μοναδικές
char	1byte	‘a’..‘z’ ‘A’..‘Z’‘0’..‘9’	28 ή 256
bool	1byte	true, false	2
int	4 bytes	-2^{31}… $2^{31}-1$	2^{32}
short int	2 bytes	-2^{15}… $2^{15}-1$	2^{16}
float	4 bytes	10^{-37}… 10^{38}	2^{32}
double	8 bytes	10^{-307}… 10^{308}	2^{64}
δείκτης	4 bytes	διευθύνσεις ($0..2^{32}-1$)	2^{32}

Δείκτες και Διευθύνσεις

- Έστω x ένας int, και p ένας δείκτης προς τον x. Στο επίπεδο της μνήμης, γραφικά, αυτό σημαίνει:



Variable	Address	data
x	2293544	5
p	2293562	2293544

```
int x=5;
int* p=&x;

printf("x=%d\t&x=%d\n", x, &x);
printf("p=%d\t*p=%d\n", p, *p);
```

Τυπώνει

```
x=5                 &x=2293544
p=2293544           *p=5
```

- Τελεστής διεύθυνσης: **&** (η διεύθυνση του αντικειμένου)
  - Δίνει τη διεύθυνση ενός αντικειμένου. π.χ. **&c** αποδίδει τη διεύθυνση του x.
  - **p = &x** : έχει ως αποτέλεσμα ανάθεση στο δείκτη p τη διεύθυνση του x.
- Τελεστής Έμμεσης Αναφοράς: * (το περιεχόμενο του δείκτη)
  - Όταν εφαρμόζεται σε δείκτη προσπελάζει το αντικείμενο που δείχνει ο δείκτης.
  - Το ***p**, έχει τιμή την τιμή του x.

Δήλωση Δεικτών

Δήλωση Pointer:

<τύπος δεδομένων> * <όνομα μεταβλητής>

Παραδείγματα:

```
int    *p;            //Δείκτης σε ακέραιο
int     foo(char *);  //Δείκτης σε χαρακτήρα
int    **p;           //Διπλός Δείκτης
```

- Επομένως ένας δείκτης δείχνει σε αντικείμενα κάποιου συγκεκριμένου τύπου δεδομένων. (εξαίρεση: δείκτης σε void)

- Προσοχή στη χρήση του * !!
  - `a*b;` (Τελεστής γινομένου)
  - `int *p;` (Δήλωση Δείκτη)
  - `x = *p + 1;` (Τελεστής Έμμεσης Αναφοράς)

Παραδείγματα με δείκτες

```
int    x = 1,   y = 2,   z[10];
int    *ip,  *iq, **dip;


ip    = &x;            /* ο ip δείχνει τώρα την x */
*ip   = *ip + 1;       /* η μεταβλητή που δείχει ο ip
                          (η x) αυξάνεται κατά 1 */
y     = *ip + 3;       /* η y παίρνει την τιμή 5 */
*ip   = 0;             /* η x παίρνει την τιμή 0 */
ip    = &z[6];???          /* ο ip δείχνει τώρα το
  z[6]*/
iq    = ip;            /* ο iq δείχνει εκεί που
                          δείχνει και ο ip */
dip   = &ip;           /* ο dip δείχνει τώρα την ip */
**dip = 13;            /* η x παίρνει την τιμή 13 */
```

Δείκτες και ορίσματα συναρτήσεων

- Ξέρουμε ότι αν καλέσουμε μια συνάρτηση με κάποιες παραμέτρους (δια τιμής) τότε αυτές οι τιμές δεν αλλάζουν (Εξαίρεση: πίνακες οι οποίοι περνάνε δια-αναφοράς εξ' ορισμού).

```
void increment(int c){
   c++;
}
```

```
int x=5;
printf("%d\n", x);     ➔ 5
increment(x);
printf("%d\n", x);     ➔ 5
```

- Παράδειγμα: Να γράψετε μια συνάρτηση η οποία να έχει σαν αποτέλεσμα την εναλλαγή δύο στοιχείων.

```
void swap(int *px, int *py){
   int temp;

   temp  = *px;
   *px   = *py;
   *py   = temp;
}
```

ΛΑΘΟΣ!
Καλώντας την συνάρτηση ως swap1(a,b) δεν θα έχει καμιά επίδραση στις τιμές των a και b.

Δείκτες και ορίσματα συναρτήσεων

```
void swap(int *px, int *py){
   int temp;

   temp  = *px;
   *px   = *py;
   *py   = temp;
}
```

ΣΩΣΤΟ!

```
int a=3;
int b=5;
printf("a=%d\tb=%d\n", a, b);
// Τυπώνει a=3   b=5
swap(&a,&b);

printf("a=%d\tb=%d\n", a, b);
// Τυπώνει a=5   b=3
```

- Η κλήση **swap(&a, &b)** θα έχει το επιθυμητό αποτέλεσμα:
  - Αφού ο τελεστής & δίνει τη διεύθυνση μιας μεταβλητής, οι &a και &b είναι δείκτες προς τις μεταβλητές a και b.
  - Οι παραμέτροι της swap δηλώνονται ως δείκτες και η προσπέλαση και ανταλλαγή των τιμών των a και b γίνονται μέσω των δεικτών αυτών.

Δείκτες και Πίνακες (συν.)

- Στην C υπάρχει ισχυρός δεσμός ανάμεσα στους **δείκτες** και στους **πίνακες**. Οποιαδήποτε πράξη μπορεί να γίνει με **δείκτες πινάκων** (πχ α[i]) μπορεί να γίνει και με **δείκτες διευθύνσεων *pa**.

- Έστω πίνακας int a[10]. Η δήλωση αυτή ορίζει ένα μπλοκ 10 διαδοχικών αντικειμένων με ονόματα α[0], α[1], ..., α[9].
- Αν **int *pa;**

  τότε η ανάθεση **pa = &a[0];**

  κάνει τον pa να δείχνει το μηδενικό (πρώτο) στοιχείο του πίνακα.

- ***Αριθμητική δεικτών*** : Αν ο pa είναι δείκτης σε κάποια θέση του πίνακα, τότε οι pa ± i, pa++, pa-- είναι επίσης δείκτες
  - **pa + 1, pa++** δείκτης στο επόμενο στοιχείο του πίνακα από αυτό που δείχνει ο pa.
  - **pa + i , (pa – i)** δείκτης στο σημείο του πίνακα που βρίσκεται i θέσεις μετά (πριν) από αυτό που δείχνει ο pa.

Δείκτες και Πίνακες

- Έτσι, αν **pa = &a[0]**, τότε για παράδειγμα,

  pa + 2 δείχνει στη τρίτη θέση του πίνακα, δηλ. στο α[2].

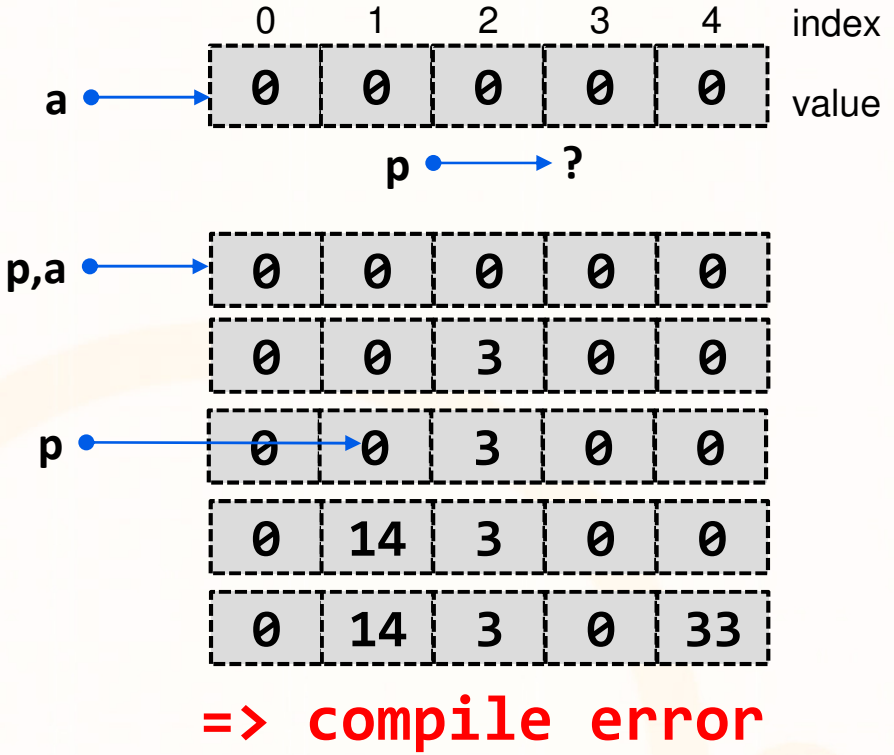
  ***(pa + 2)** έχει τιμή την τιμή της 3ης θέσης του πίνακα, δηλ. το α[2].

- Έξ'ορισμού, η τιμή μιας μεταβλητής τύπου πίνακα (int a[]) είναι η διεύθυνση του μηδενικού στοιχείου του πίνακα.
- Επομένως αντί **pa = &a[0]** μπορούμε να γράψουμε **pa = a**.
- Τότε οι pa και a έχουν τις ίδιες τιμές και μπορούν να χρησιμοποιούνται η μια στη θέση της άλλης:
  - Το **a[i]** είναι ταυτόσημο με τα ***(pa +i)** ή ***(a + i)** ή **pa[i]**.
  - Το **&a[i]** είναι ταυτόσημο με τα **a+i, pa+i**.

Δείκτες και Πίνακες (συν.)

- **ΠΡΟΣΟΧΗ:** ο δείκτης είναι μεταβλητή και έτσι pa = a και pa++ είναι **έγκυρες εκφράσεις**. Το **όνομα ενός πίνακα όμως δεν είναι μεταβλητή**, επομένως κατασκευές όπως α = pa και α++ δεν είναι έγκυρες!
- Ποιες από τις πιο κάτω εντολές είναι έγκυρες;

```
int   a[5] = {},
      i = 4, *pa;

pa    = a;
pa[2] = 3;
++pa;
*pa   = 14;
*(a+i) = 33;
++a;
```



Παράδειγμα – Αντιγραφή String <strcpy>

- **Ζητούμενο:** να γράψετε συνάρτηση που αντιγράφει ένα αλφαριθμητικό t στο αλφαριθμητικό s.
- **Ερώτηση:** Γιατί δεν είναι αρκετό να γράψουμε s=t ;;;
- Εκδοχή με πίνακες:

```
void mystrcpy(
 char * dest,
 const char * source ){
   int i=0;
   while(dest[i]!='\0'){
       dest[i]=source[i];
       i++;
   }
}
```

```
void mystrcpy(
 char * dest,
 const char * source ){
   int i=0;
   while(dest[i]=source[i++]);
}
```

Παράδειγμα – Αντιγραφή String <strcpy> (συν.)

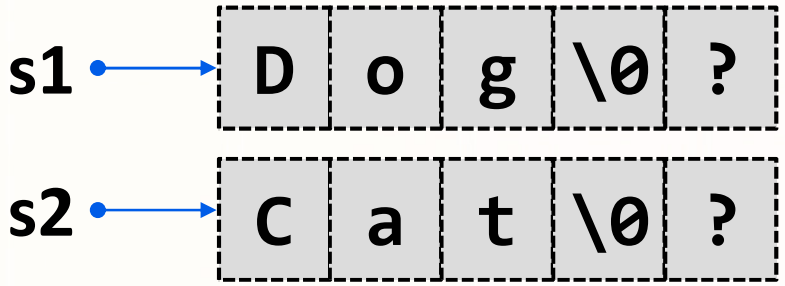
- Εκδοχή με Δείκτες:

```
void mystrcpy_p(
 char * dest,
 const char * source ){

   while(*dest!='\0'){
       *dest=*source;
       *dest++;
       *source++;
   }
}
```

```
void mystrcpy_p(
 char * dest,
 const char * source ){

   while(*dest++=*source++);
}
```

Παράδειγμα – Σύγκριση Strings <strcmp>

- **Ζητούμενο:** να γράψετε συνάρτηση να συγκρίνει τα αλφαριθμητικά s1 και s2 και επιστρέφει «1» αν το s1 μεγαλύτερο λεξικογραφικά, “0” αν είναι ίσα και «-1» αν s2 μεγαλύτερο λεξικογραφικά
- Παράδειγμα



s1>s2, ‘D’ > ‘C’

➔ strcmp(s1,s2)=1

- Εκδοχή με πίνακες

```
int mystrcmp(const char * dest,
const char * source ){
   int i=0;
   while(dest[i]==source[i]){
        if(dest[i]=='\0') return 0;
        i++;
   }
   return dest[i]>source[i]? 1:-1;
}
```

Παράδειγμα – Σύγκριση Strings <strcmp>

- Εκδοχή με δείκτες:

```
int mystrcmp_p(
 const char * dest,
 const char * source ){
   while(*dest==*source){
      if(*dest=='\0') return 0;
      *dest++;
      *source++;
   }
   return (*dest>*source)? 1:-1;
}
```

- Εκδοχή με δείκτες 2:

```
int mystrcmp_p2(
 const char * dest,
 const char * source ){
   while(*dest++==*source++){
      if(!(*(dest-1))) return 0;
   }
   return (*(--dest)>*(--source))? 1:-1;
}
```