Προγραμματισμός Ι

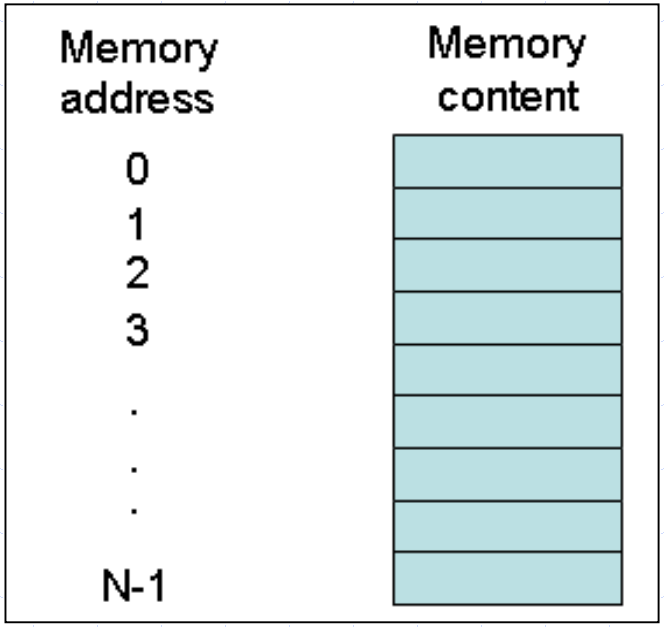
Δείκτες

Πανεπιστήμιο Πελοποννήσου
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Νικόλαος Δ. Τσελίκας

Μνήμη Υπολογιστή

- Η μνήμη RAM (Random Access Memory) ενός υπολογιστή αποτελείται από πολλές χιλιάδες θέσεις αποθήκευσης δεδομένων που έχουν **διαδοχική αρίθμηση**
- Κάθε **θέση ή κελί μνήμης** προσδιορίζεται από **μία μοναδική διεύθυνση**
- Η **διεύθυνση της κάθε θέσης μνήμης** είναι ένας αύξοντας αριθμός με τιμή που κυμαίνεται από το 0 έως μία μέγιστη τιμή (η οποία εξαρτάται από το μέγεθος της διαθέσιμης μνήμης στον συγκεκριμένο υπολογιστή). Για παράδειγμα, αν ο υπολογιστής διαθέτει **N** bytes τότε οι διευθύνσεις μνήμης κυμαίνονται από 0 έως **N-1**, όπως φαίνεται στο σχήμα
- Το **περιεχόμενο της κάθε θέσης** μνήμης είναι ένας ακέραιος αριθμός με μέγεθος 1 byte

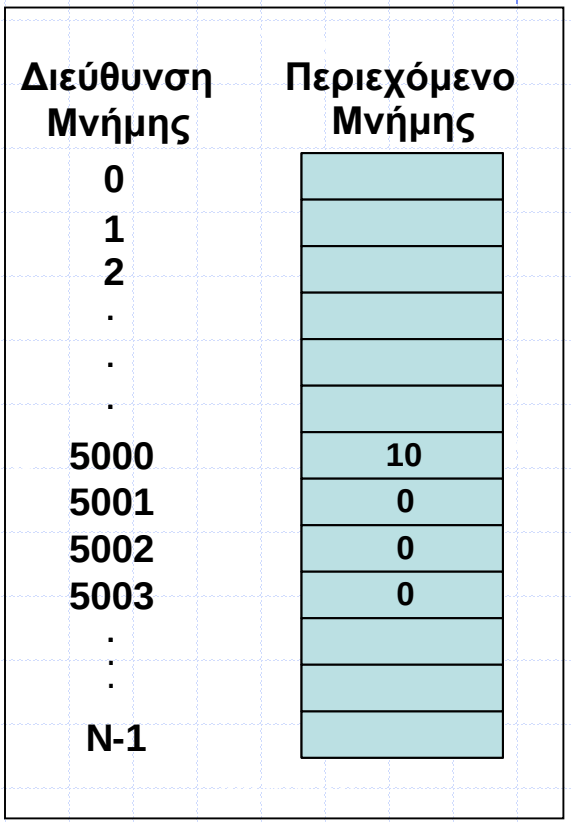


Μνήμη Υπολογιστή και Μεταβλητές

- Όταν δηλώνεται μία μεταβλητή, ο μεταγλωττιστής δεσμεύει τις απαραίτητες **συνεχόμενες** θέσεις (bytes) στη μνήμη, για να αποθηκεύσει την τιμή της
- Όπως ήδη ξέρουμε, κάθε τύπος μεταβλητής απαιτεί συγκεκριμένο χώρο στη μνήμη
- Π.χ. ο τύπος `char` απαιτεί 1 byte μνήμης, οι τύποι `float` και `int` απαιτούν 4 bytes, ο τύπος `double` απαιτεί 8 bytes, κ.ο.κ.
- Όταν μία μεταβλητή καταλαμβάνει πολλές θέσεις μνήμης (δηλ. περισσότερα από 1 byte), τότε ως **διεύθυνση της μεταβλητής** θεωρείται **η διεύθυνση της πρώτης θέσης μνήμης** (δηλ. του 1ου byte από τα bytes που καταλαμβάνει η μεταβλητή)

Παράδειγμα

- Έστω η δήλωση: `int a = 10;`
- Τότε: Ο μεταγλωττιστής ψάχνει και βρίσκει 4 συνεχόμενες θέσεις μνήμης στη RAM, οι οποίες δεν πρέπει να έχουν δεσμευτεί για άλλη μεταβλητή, και τις δεσμεύει για να αποθηκεύσει την τιμή της μεταβλητής `a`
- Στο διπλανό σχήμα θεωρούμε ότι η διεύθυνση της μεταβλητής `a` αρχίζει στη θέση 5000
- Έτσι, η τιμή της `a` (η τιμή 10) θα αποθηκευτεί στις θέσεις μνήμης από 5000 έως και 5003 ξεκινώντας από την οκτάδα με την χαμηλότερη διεύθυνση (*little-endian architecture*)
- Ο μεταγλωττιστής συσχετίζει το όνομα της μεταβλητής `a`, με τη διεύθυνση της μεταβλητής
- Όταν το πρόγραμμα χρησιμοποιεί το όνομα της μεταβλητής, ο μεταγλωττιστής προσπελαύνει αυτομάτως τη διεύθυνση της μεταβλητής
- Π.χ. με την εντολή `a = 80;` ο μεταγλωττιστής γνωρίζει ότι η διεύθυνση της `a` είναι η 5000 και θέτει το περιεχόμενό της ίσο με 80



Δήλωση Δείκτη

- Ο **δείκτης** είναι μία **μεταβλητή**, στην οποία αποθηκεύεται η διεύθυνση μνήμης μίας άλλης μεταβλητής

- Η γενική περίπτωση δήλωσης ενός δείκτη είναι:

```
τύπος_δεδομένων *όνομα_δείκτη;
```

Παρατηρήσεις

- Ο `τύπος_δεδομένων` μπορεί να είναι οποιοσδήποτε από τους τύπους μεταβλητών της C και δηλώνει τον τύπο της μεταβλητής στην οποία – συνηθίζουμε να λέμε – «δείχνει ο δείκτης»
- Το `όνομα_δείκτη` πρέπει να ακολουθεί τους κανόνες ονοματολογίας της C και να μην υπάρχει άλλη δήλωση με το ίδιο όνομα μέσα στο πρόγραμμα
- Ο τελεστής * χρησιμοποιείται για να δηλώσει ότι η μεταβλητή είναι δείκτης

Παραδείγματα Δήλωσης Δείκτη

```
int *ptr;
```

- Η μεταβλητή ptr είναι ένας δείκτης προς κάποια ακέραια μεταβλητή
- Αυτό σημαίνει, ότι στον δείκτη ptr θα αποθηκευτεί η διεύθυνση κάποιας ακέραιας μεταβλητής τύπου int

```
double *pt;
```

- Η μεταβλητή pt είναι ένας δείκτης προς κάποια πραγματική μεταβλητή (και μάλιστα, τύπου double)
- Αυτό σημαίνει, ότι στον δείκτη pt θα αποθηκευτεί η διεύθυνση κάποιας πραγματικής μεταβλητής (και πιο συγκεκριμένα, μιας μεταβλητής τύπου double)

Παρατηρήσεις (1/3)

- Με τη δήλωση:

```
int *ptr;
```

ο ptr δηλώνεται ως δείκτης σε μεταβλητές τύπου int, έτσι, ως τιμή του ptr μπορεί να αποθηκευτεί η διεύθυνση μνήμης οποιασδήποτε μεταβλητής τύπου int

- Γενικά, αν ο ptr είναι «δείκτης σε τύπο» T, τότε η έκφραση *ptr είναι τύπου T

- Π.χ. στην προηγούμενη δήλωση (int *ptr;), η έκφραση *ptr είναι τύπου int

- Οι μεταβλητές δείκτες, μπορούν να δηλωθούν ταυτόχρονα με άλλες μεταβλητές του ίδιου τύπου, π.χ.:

```
int *ptr, i, j, k;
```

Παρατηρήσεις (2/3)

- Σημειώνεται ότι επιτρέπεται να χρησιμοποιήσουμε τον τελεστή *, όχι δίπλα στο όνομα της μεταβλητής, αλλά δίπλα στον τύπο δεδομένων
- Π.χ.:

```
int* ptr;
```

- Αν και κάποιοι προγραμματιστές προτιμούν την παραπάνω γραφή, προτείνουμε να μην τη χρησιμοποιείτε, διότι είναι δυνατόν να προκαλέσει σύγχυση όταν δηλώνονται περισσότερες της μίας μεταβλητές
- Π.χ., με τη δήλωση: `int* p1, p2;`
  το p1 δηλώνεται ως «δείκτης σε ακέραιο» ενώ το p2 ως ακέραιος

  Σωστά??? Ή μήπως και οι δύο μεταβλητές είναι «δείκτες σε ακέραιο»???
- **Σωστά**, αλλά προτιμήστε τη δήλωση: `int *p1, p2;`
  που είναι περισσότερο ξεκάθαρη...

Παρατηρήσεις (3/3)

- Όταν δηλώνεται μία μεταβλητή-δείκτης, ο μεταγλωττιστής, όπως κάνει και για οποιαδήποτε μεταβλητή, δεσμεύει τις απαραίτητες θέσεις μνήμης για να αποθηκεύσει την τιμή του

- Το επόμενο πρόγραμμα εμφανίζει πόσα bytes μνήμης δεσμεύτηκαν για τον δείκτη `ptr` με χρήση του τελεστή `sizeof`

```
#include <stdio.h>
int main(void)
{
        int *ptr;

        printf("Bytes: %u\n", sizeof(ptr));
        return 0;
}
```

Τα bytes που δεσμεύονται για μία μεταβλητή-δείκτη είναι **4**, ανεξάρτητα από τον τύπο δεδομένων στον οποίο δείχνει ο δείκτης

- Δηλαδή, στο προηγούμενο παράδειγμα είτε έχουμε τη δήλωση
  `char *ptr;` είτε: `float *ptr;` είτε: `double *ptr;`
  το αποτέλεσμα είναι **4**

Απόδοση τιμής σε Δείκτη

- Ένας δείκτης, πριν χρησιμοποιηθεί, **πρέπει** να έχει σαν τιμή τη διεύθυνση κάποιας μεταβλητής ή, ισοδύναμα, **να «δείχνει»** σε κάποια **υπαρκτή μεταβλητή**
- Για να βρούμε τη διεύθυνση κάποιας μεταβλητής χρησιμοποιούμε τον **τελεστή διεύθυνσης** **&** πριν από το όνομα της μεταβλητής
- Υπενθυμίζεται ότι **η διεύθυνση** είναι **η θέση της μεταβλητής στη μνήμη** του υπολογιστή και **δεν έχει καμία σχέση με την τιμή της μεταβλητής**

```
#include <stdio.h>
int main(void)
{
        int *ptr, a;

        ptr = &a; /* ptr "points to" the memory address of a. */
        printf("Address = %p\n", ptr); /* Display the memory address of
a. */
        return 0;
}
```

Παρατηρήσεις

- Για την εμφάνιση στην οθόνη της διεύθυνσης μνήμης μίας μεταβλητής συνήθως χρησιμοποιείται το προσδιοριστικό `%p`, το οποίο εμφανίζει τη διεύθυνση σε **δεκαεξαδική μορφή** (μπορούμε να χρησιμοποιήσουμε και το προσδιοριστικό `%d`, για την εμφάνιση της διεύθυνσης σε **δεκαδική μορφή**)

- Όταν εκχωρείται η διεύθυνση μίας μεταβλητής σε έναν δείκτη, **ο δείκτης πρέπει να έχει δηλωθεί** σαν δείκτης **στον ίδιο τύπο** με **τη μεταβλητή**

  Προσοχή λοιπόν σε λάθη όπως αυτό του παρακάτω παραδείγματος

```
int *ptr;
float a;
ptr = &a;
```

- Η τιμή που εκχωρείται σε έναν δείκτη πρέπει να είναι η διεύθυνση κάποιας υπαρκτής μεταβλητής και όχι μία σταθερή αριθμητική τιμή

  Στο παρακάτω παράδειγμα ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους

```
int *ptr;
ptr = 1000;
```

Η ειδική τιμή NULL (Ι)

- Υπενθυμίζεται ότι, **όταν δηλώνεται μία μεταβλητή**, τότε ο μεταγλωττιστής αναθέτει στη μεταβλητή μία **τυχαία τιμή (τιμή «σκουπίδι»)**

- Επομένως, **όταν δηλώνεται μία μεταβλητή-δείκτης**, τότε η αρχική τιμή του δείκτη αυτού είναι μία **τυχαία διεύθυνση μνήμης**

- Στο παρακάτω παράδειγμα, εμφανίζεται στην οθόνη η τυχαία τιμή που εκχώρησε ο μεταγλωττιστής στον δείκτη `ptr`

```
#include <stdio.h>
int main(void)
{
        int* ptr;
        printf("Value = %p\n",ptr);
        return 0;
}
```

Η ειδική τιμή NULL (II)

- Όταν θέλουμε να δηλώσουμε ρητά ότι ένας δείκτης δεν δείχνει πουθενά (*null pointer*), τότε του αναθέτουμε την τιμή **NULL**
- Η τιμή **NULL** είναι μία ειδική τιμή ίση με το μηδέν (0)
- Επί της ουσίας πρόκειται για μια μακροεντολή με όνομα **NULL** και τιμή ίση με μηδέν (0) ή – ακόμα καλύτερα – με τιμή μηδέν (0) προσαρμοσμένη στον τύπο `void*`, δηλαδή τιμή `(void*)0`
- Η μακροεντολή **NULL** δηλώνεται σε διάφορα header files όπως π.χ. και στο **stdio.h**
- Επομένως, το επόμενο παράδειγμα εμφανίζει την τιμή 0

```
#include <stdio.h>
int main(void)
{
        int* ptr;
        ptr = NULL;
        printf("Value = %p\n",ptr);
        return 0;
}
```

Η ειδική τιμή NULL (III)

- Παρόλο που είναι δυνατόν να χρησιμοποιηθεί απευθείας κι η τιμή 0 αντί της NULL, σε περίπτωση που πρόκειται για μεταβλητή-δείκτη είναι προτιμότερο να χρησιμοποιείτε τη μακροεντολή NULL για να αποφεύγεται η παρακάτω σύγχυση:

- Π.χ., η ανάθεση `ptr = 0;` μπορεί να μας ξεγελάσει
  - Το `ptr` είναι δείκτης (λόγω ονόματος) ή μήπως απλή αριθμητική μεταβλητή (λόγω της ανάθεσης του μηδενός)?

- Αντιθέτως, η ανάθεση `ptr = NULL;` κάνει αμέσως ξεκάθαρο ότι ο `ptr` είναι δείκτης

- Η τιμή ενός δείκτη μπορεί να συγκριθεί έναντι της τιμής NULL, όπως παρακάτω:

```
if(ptr != NULL) /* Ισοδύναμο με if(ptr) */
if(ptr == NULL) /* Ισοδύναμο με if(!ptr) */
```

Χρήση Δείκτη

- Για να αποκτήσουμε **πρόσβαση στο περιεχόμενο** κάποιας διεύθυνσης μνήμης με χρήση δείκτη, χρησιμοποιούμε τον τελεστή * (**dereference operator** ή αλλιώς indirection operator) πριν από το όνομα του δείκτη
- Π.χ.

```
#include <stdio.h>
int main(void)
{
        int *ptr; /* Δήλωση δείκτη προς ακέραια μεταβλητή. */
        int a;

        a = 10;
        ptr = &a; /* Ο δείκτης ptr "δείχνει" στη διεύθυνση της
μεταβλητής a. */

        printf("Value = %d\n",*ptr); /* Εμφάνιση του περιεχομένου
της διεύθυνσης που δείχνει ο ptr. */
        return 0;
}
```

- Η έκφραση ***ptr** είναι ισοδύναμη με **το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο δείκτης ptr**
- Αφού ο **ptr** δείχνει στη διεύθυνση της μεταβλητής **a**, το ***ptr** είναι ένας διαφορετικός τρόπος έκφρασης του **a**, οπότε, το πρόγραμμα εμφανίζει **10**

Παρατηρήσεις (Ι)

- Πριν χρησιμοποιηθεί κάποια μεταβλητή-δείκτης θα πρέπει να έχει αρχικοποιηθεί, δηλαδή να της έχει εκχωρηθεί μία υπαρκτή διεύθυνση, δηλαδή ο δείκτης να δείχνει στη διεύθυνση κάποιας μεταβλητής
- Το επόμενο πρόγραμμα θα εμφανίσει μήνυμα λάθους κατά την εκτέλεσή του, γιατί στην εντολή `i = *ptr;` χρησιμοποιείται ο δείκτης `ptr`, ο οποίος δεν δείχνει στη διεύθυνση κάποιας μεταβλητής (δεν έχει αρχικοποιηθεί)

```
#include <stdio.h>
int main(void)
{
        int i;
        int *ptr;

        i = *ptr; /* Ο δείκτης ptr δεν δείχνει στη διεύθυνση
κάποιας μεταβλητής. */
        printf("Value = %d\n",i);
        return 0;
}
```

- Συνήθως, σε Unix/Linux περιβάλλον το παραπάνω λάθος υποδεικνύεται με το μήνυμα *"Segmentation fault"*

Παρατηρήσεις (ΙΙ)

- Το επόμενο πρόγραμμα λειτουργεί σωστά, γιατί τώρα ο δείκτης `ptr` δείχνει στη διεύθυνση κάποιας υπαρκτής μεταβλητής (της μεταβλητής `j`) πριν χρησιμοποιηθεί στην εντολή `i = *ptr;`
- Επομένως, αφού ο δείκτης `ptr` δείχνει στη διεύθυνση της μεταβλητής `j`, το `*ptr` θα είναι ίσο με την τιμή του `j`, δηλαδή `20`
- Άρα, με την εντολή `i = *ptr;` η τιμή του `i` θα γίνει ίση με `20`

```
#include <stdio.h>
int main(void)
{
        int *ptr, i, j;

        j = 20;
        ptr = &j;

        i = *ptr;
        printf("Val = %d\n", i);
        return 0;
}
```

Έξοδος: `Val = 20`

Παρατηρήσεις (III)

- Οι τελεστές * (περιεχόμενο διεύθυνσης μνήμης που δείχνει ο δείκτης) και & (διεύθυνση μνήμης μιας μεταβλητής) είναι μεταξύ τους **συμπληρωματικοί** ή **αντίστροφοι** (αλλιώς λέμε ότι **αλληλοαναιρούνται** ή **αλληλοεξουδετερώνονται** ή **ακυρώνει ο ένας τον άλλον**)

```c
#include <stdio.h>
int main(void)
{
        int a = 21;
        int *ptr;

        ptr = &a;

        printf("The address of a is %p \n", &a);
        printf("The value of ptr is %p \n\n", ptr);
        printf("The value of a is %d \n", a);
        printf("The value of *ptr is %d \n\n", *ptr);
        printf("Showing that * and & cancel each other\n");
        printf("&*ptr = %p \n", &*ptr);
        printf("*&ptr = %p \n", *&ptr);

        return 0;
}
```

Έξοδος (π.χ.):

```
The address of a is 0028F958
The value of ptr is 0028F958

The value of a is 21
The value of *ptr is 21

Showing that * and & cancel each other
&*ptr = 0028F958
*&ptr = 0028F958
```

Παρατηρήσεις (IV)

- Δεδομένου ότι οι χαρακτήρες /* σηματοδοτούν την αρχή ενός σχολίου, σε μία έκφραση σαν την παρακάτω:

```
a = b/*ptr;
```

ο,τιδήποτε ακολουθεί το /* έως και να συναντηθεί το */ στον κώδικά μας, θα θεωρηθεί σχόλιο και η μεταγλώττιση θα αποτύχει

- Σε αυτήν την περίπτωση να αφήνετε ένα κενό ή να κάνετε χρήση της παρένθεσης

- Π.χ.: `a = b/ *ptr;`

ή `a = b/(*ptr);`

Δεν ξεχνώ λοιπόν...

```c
#include <stdio.h>
int main(void)
{
        int a, i = 10;
        float b, j = 5.5;
        double c, pi = 3.14;

        int *ptr1;
        float *ptr2;
        double *ptr3;

        ptr1 = &i;
        ptr2 = &j;
        ptr3 = &pi;

        a = *ptr1;
        b = *ptr2;
        c = *ptr3;

        printf("%d\n", a);
        printf("%f\n", b);
        printf("%f\n", c);

        return 0;
}
```

Δήλωση μεταβλητών

Δήλωση «μεταβλητών δεικτών»

Ανάθεση τιμών στους Δείκτες (τις διευθύνσεις υπαρκτών μεταβλητών)

Απόδοση τιμής σε κάποια μεταβλητή, μέσω δείκτη (το περιεχόμενο της διεύθυνσης στην οποία δείχνει ο δείκτης)

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>
int main(void)
{
        int i,j,k;
        int* ptr1;
        int* ptr2;

        i = 10;
        j = 20;

        ptr1 = &i;
        ptr2 = &j;

        k = *ptr1 + *ptr2;
        printf("%d\n",k);
        return 0;
}
```

Έξοδος: 30

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *ptr, i = 10;

        ptr = &i;
        i += 20;
        printf("%d\n", *ptr);
        return 0;
}
```

Έξοδος: 30

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        int* ptr;

        ptr = &i;
        (*ptr)++;
        printf("Value = %d\n",i);
        return 0;
}
```

Έξοδος: 11

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        int* ptr;

        ptr = &i;

        for(i = 0; i < 3; i++)
                printf("%d ",*ptr);
        return 0;
}
```

Έξοδος: 0 1 2

Παραδείγματα (V)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10,j = 20,k;
        int* ptr1, *ptr2, *ptr3;

        ptr1 = &i;
        ptr2 = &j;
        ptr3 = &k;

        *ptr1 = *ptr2 = 100;
        k = i+j;

        printf("%d\n",*ptr3);
        return 0;
}
```

Έξοδος: 200

Παραδείγματα (VI)

- Γράψτε ένα πρόγραμμα το οποίο κάνοντας χρήση ενός δείκτη για να διαβάσει μία δεκαδική τιμή και να εμφανίζει την απόλυτη τιμή της τιμής αυτής.

```
#include <stdio.h>
int main(void)
{
        double *p, val;

        p = &val;
        printf("Enter number: ");
        scanf("%lf", p);

        if(*p >= 0)
                printf("%f\n", *p);
        else
                printf("%f\n", -*p);
        return 0;
}
```

Ο δείκτης void* (1/2)

- Ένας δείκτης σε τύπο void είναι ένας «γενικός» δείκτης, με την έννοια ότι μπορεί να δείξει σε μία μεταβλητή οποιουδήποτε τύπου
- Ένας δείκτης μπορεί να μετατραπεί σε τύπο void και πάλι πίσω στον αρχικό τύπο χωρίς απώλεια πληροφορίας
- Σημειώστε ότι, αν δεν είναι δείκτης σε void και οι τύποι είναι διαφορετικοί, πρέπει να γίνει προσαρμογή
- Π.χ.

```
char *p1;
int *p2;
void *p3;
...
p2 = p3;
p3 = p2;
p2 = p1; /* Wrong. */
p2 = (int*)p1; /* That's ok now. */
```

Ο δείκτης void* (2/2)

- Για να προσπελάσουμε τη μεταβλητή με χρήση ενός void* δείκτη πρέπει να προσαρμόσουμε τον τύπο του στον τύπο της μεταβλητής, ώστε ο μεταγλωττιστής να γνωρίζει το αντίστοιχο μέγεθος, όπως φαίνεται στο παρακάτω πρόγραμμα:

```
#include <stdio.h>
int main(void)
{
        void *ptr;
        int i = 10;

        ptr = &i;
        *(int*)ptr += 20;
        printf("%d\n", i);

        return 0;
}
```

- Για να αποκτήσουμε πρόσβαση στο περιεχόμενο της ακέραιας μεταβλητής i, προσαρμόζουμε τον τύπο του δείκτη σε int*
- Με αυτόν τον τρόπο, μπορούμε να αλλάξουμε την τιμή της μεταβλητής στην οποία δείχνει ο «γενικός» δείκτης
- Επομένως, το πρόγραμμα θα εμφανίσει: 30

Χρήση της λέξης const στη δήλωση ενός δείκτη

- Χρησιμοποιούμε τη δεσμευμένη λέξη const κατά τη δήλωση του δείκτη, όταν επιθυμούμε μία μεταβλητή-δείκτης :
  - είτε να μην μπορεί να αλλάξει την τιμή της μεταβλητής στην οποία δείχνει (χρήση της λέξης const πριν τον τύπο δεδομένων)
  - είτε να μην μπορεί να δείξει σε κάποια άλλη μεταβλητή (χρήση της λέξης const πριν το όνομα του δείκτη)

- Δείτε λοιπόν, τι επιτρέπεται και τι όχι, στα παρακάτω παραδείγματα

```
int j, i = 10;
const int *ptr;
ptr = &i;
*ptr = 30; /* Μη επιτρεπτή ενέργεια. */
ptr = &j; /* Επιτρεπτή ενέργεια. */
```

Ο δείκτης ptr δεν μπορεί να αλλάξει την τιμή της μεταβλητής στην οποία δείχνει (της i)

Ωστόσο, επιτρέπεται να "δείξει" σε κάποια άλλη μεταβλητή ίδιου τύπου (εδώ της j)

```
int i, j;
int* const ptr = &i;
ptr = &j; /* Μη επιτρεπτή ενέργεια. */
*ptr = 30; /* Επιτρεπτή ενέργεια.
              Η τιμή του i γίνεται 30. */
```

Ο δείκτης ptr δεν μπορεί να "δείξει" σε άλλη μεταβλητή (όπως π.χ. εδώ στην j), παρά μόνο στην i (ωστόσο, επιτρέπεται να αλλάξει την τιμή του i)

Χρήση της λέξης const στη δήλωση ενός δείκτη

- Χρησιμοποιώντας 2 φορές τη δεσμευμένη λέξη const κατά τη δήλωση του δείκτη, μπορούμε προφανώς να τον αναγκάσουμε και να μην μπορεί να αλλάξει την τιμή της μεταβλητής στην οποία δείχνει και (ταυτόχρονα) να μην μπορεί να δείξει σε κάποια άλλη μεταβλητή

- Π.χ.

```
int i, j;
const int* const ptr = &i; /* Initialize pointer. */
ptr = &j; /* Illegal action. */
*ptr = 30; /* Illegal action. */
```

Αριθμητική Δεικτών

- Η αριθμητική δεικτών αφορά στην εκτέλεση αριθμητικών πράξεων με δείκτες
- Σύμφωνα με το πρότυπο, η αριθμητική δεικτών παράγει αξιόπιστα αποτελέσματα όταν εφαρμόζεται σε στοιχεία του ίδιου πίνακα, αλλιώς το αποτέλεσμα είναι απροσδιόριστο
- Εξαίρεση στον παραπάνω κανόνα αποτελεί η διεύθυνση του πρώτου στοιχείου μετά το τέλος του πίνακα (το πρότυπο εγγυάται ότι μπορεί να χρησιμοποιηθεί η θέση αυτή στην αριθμητική δεικτών)
- Οι επιτρεπτές πράξεις είναι:
  - Η πρόσθεση ακεραίου σε δείκτη
  - Η αφαίρεση ακεραίου από δείκτη και
  - Η αφαίρεση δύο δεικτών
- Οι παραπάνω πράξεις έχουν ορισμένες ιδιαιτερότητες και για τον λόγο αυτό, απαιτείται ιδιαίτερη προσοχή
- Ως λειτουργίες, επίσης επιτρέπονται η σύγκριση δεικτών καθώς και η ανάθεση σε έναν δείκτη της τιμής 0 ή η σύγκρισή του με την τιμή 0

Δείκτες και Ακέραιοι (Αύξηση Δεικτών)

- Θεωρώντας ότι ο δείκτης `ptr` δείχνει σε ένα στοιχείο ενός πίνακα, η πρόσθεση ενός θετικού ακέραιου `n` στον `ptr`, π.χ.:

```
ptr = ptr + n;
```

αυξάνει την τιμή του δείκτη κατά `n * μέγεθος` του τύπου στον οποίο δείχνει ο `ptr` και **τον κάνει να δείχνει στη διεύθυνση του n-οστού στοιχείου μετά από αυτό που έδειχνε**

- Αν το αποτέλεσμα της πράξης είναι εκτός των ορίων του πίνακα, το αποτέλεσμα είναι απροσδιόριστο (εξαιρείται η διεύθυνση της πρώτης θέσης αμέσως μετά το τέλος του πίνακα, η οποία θεωρείται έγκυρη), π.χ, αν ο `ptr` έχει δηλωθεί σαν δείκτης σε:
  - `char`, τότε η τιμή του δείκτη αυξάνεται κατά `n*1 = n`, αφού το μέγεθος του τύπου `char` είναι `1` byte
  - `int` ή `float`, τότε η τιμή του δείκτη αυξάνεται κατά `n*4`, αφού το μέγεθος των τύπων `int` και `float` είναι `4` bytes
  - `double`, τότε η τιμή του δείκτη αυξάνεται κατά `n*8`, αφού το μέγεθος του τύπου `double` είναι `8` bytes

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος???

```
#include <stdio.h>
int main(void)
{
        int *ptr, arr[] = {10, 20, 30};

        ptr = &arr[0];
        printf("Addr:%d\n", ptr);
        ptr = ptr+2;
        printf("Addr:%d Val:%d\n", ptr, *ptr);
        return 0;
}
```

Ο `ptr` αυξάνεται κατά `8`, (όχι κατά `2`), δεδομένου ότι ο `ptr` έχει δηλωθεί ως δείκτης σε `int`

Έτσι, το πρόγραμμα εμφανίζει δύο διευθύνσεις (ως ακέραιο του δεκαδικού συστήματος) με τη δεύτερη να είναι κατά `8` θέσεις μεγαλύτερη απ' την πρώτη (π.χ. αν η πρώτη ήταν η διεύθυνση `89234834` η δεύτερη θα είναι η `89234842`) κι επειδή ο `ptr` δείχνει στο `arr[2]`, το πρόγραμμα εμφανίζει `30`, επίσης.

Δείκτες και Ακέραιοι (Μείωση Δεικτών)

- Όμοια με την πρόσθεση, η αφαίρεση ενός θετικού ακέραιου από έναν δείκτη σε μία ανάθεση όπως η παρακάτω :

```
ptr = ptr - n;
```

μειώνει την τιμή του δείκτη κατά n * μέγεθος του τύπου στον οποίο δείχνει ο δείκτης και κάνει τον δείκτη να δείχνει στη διεύθυνση του n-οστου στοιχείου πριν από αυτό που έδειχνε

- Π.χ.

Η δεύτερη διεύθυνση θα είναι κατά 8 θέσεις μικρότερη απ'την πρώτη και, όπως και κατά την πρόσθεση, το αποτέλεσμα θα είναι έγκυρο αν ο δείκτης δείχνει σε ένα στοιχείο εντός του πίνακα

```
#include <stdio.h>
int main(void)
{
        int *ptr, arr[] = {10, 20, 30};

        ptr = &arr[2];
        printf("Addr:%d\n", ptr);
        ptr = ptr-2;
        printf("Addr:%d\n", ptr);
        return 0;
}
```

Δείκτες και τελεστές ++ ή -- (1/2)

- Η χρήση των τελεστών αύξησης ή μείωσης (++ και --) **επιφέρει έγκυρο αποτέλεσμα**, ακόμα κι αν ο δείκτης δεν δείχνει σε κάποιο στοιχείο ενός πίνακα
- Σε αυτή την περίπτωση ο δείκτης **αυξάνεται ή μειώνεται κατά το μέγεθος του τύπου στον οποίο δείχνει**

Π.χ.:

```
#include <stdio.h>
int main(void)
{
        double *ptr, i;

        ptr = &i;
        ptr++;
        printf("Addr:%d\n", ptr);
        ptr--;
        printf("Addr:%d\n", ptr);
        return 0;
}
```

Έξοδος: Το πρόγραμμα εμφανίζει δύο διευθύνσεις με την πρώτη να είναι κατά 8 θέσεις μεγαλύτερη από τη δεύτερη, αφού ο `ptr` δείχνει σε `double` τύπο δεδομένων

Δείκτες και τελεστές ++ ή -- (2/2)

- Η συνδυαστική χρήση των τελεστών ++ και -- με τον τελεστή * είναι πολύ συνηθισμένη στη διαχείριση πινάκων με χρήση δείκτη
- Το αποτέλεσμα της έκφρασης εξαρτάται από τη θέση των τελεστών με βάση τον πίνακα προτεραιοτήτων
- Ας δούμε τις διαφορετικές περιπτώσεις συνδυαστικής χρήσης του τελεστή * με τον τελεστή ++ (αντίστοιχα είναι και για τον --)

- `i = (*ptr)++;` πρώτα εκχωρείται η τιμή του `*ptr` στο `i` και μετά αυξάνεται κατά ένα η τιμή του `*ptr`
- `i = *ptr++;` πρώτα εκχωρείται η τιμή του `*ptr` στο `i` και μετά αυξάνεται η τιμή του `ptr`, ανάλογα με τον τύπο του
- `i = ++*ptr;` πρώτα αυξάνεται κατά ένα η τιμή του `*ptr` και μετά αυτή εκχωρείται στο `i`
- `i = *++ptr;` πρώτα αυξάνεται η τιμή του `ptr` και μετά εκχωρείται στο `i` η τιμή του `*ptr`

Αφαίρεση Δεικτών

- Το αποτέλεσμα της αφαίρεσης δεικτών είναι ο αριθμός των στοιχείων που μεσολαβούν μεταξύ τους
- Οι δείκτες πρέπει να δείχνουν σε στοιχεία του ίδιου πίνακα ή στην αμέσως επόμενη θέση από το τέλος του πίνακα
- Αν η τιμή του δείκτη που αφαιρείται είναι μεγαλύτερη, τότε το αποτέλεσμα είναι το ίδιο, απλά με αρνητικό πρόσημο
- Για παράδειγμα, αν ο `p1` δείχνει στο δεύτερο στοιχείο και ο `p2` στο πέμπτο στοιχείο του ίδιου πίνακα, το αποτέλεσμα της πράξης `p2-p1` είναι `3`, ενώ της πράξης `p1-p2` είναι `-3`

```c
#include <stdio.h>
int main(void)
{
      double *p1, *p2, x[] = {10, 20, 30, 40, 50, 60, 70};

      p1 = &x[1];
      p2 = &x[4];
      printf("%d", p2-p1);
      printf("%d", p1-p2);
      return 0;
}
```

Σύγκριση Δεικτών

- Το αποτέλεσμα της σύγκρισης δύο δεικτών με τον τελεστή == ή τον τελεστή != είναι πάντοτε αξιόπιστο
- Αντιθέτως, το αποτέλεσμα της σύγκρισης δύο δεικτών με τους τελεστές <, <=, >, ή >= θεωρείται αξιόπιστο μόνο αν οι δείκτες δείχνουν στο ίδιο αντικείμενο, π.χ. πίνακας ή δομή, αλλιώς είναι απροσδιόριστο
- Π.χ., αν θέλουμε να ελέγξουμε αν δύο δείκτες `ptr1` και `ptr2` δείχνουν στην ίδια διεύθυνση μνήμης (ή όχι) μπορούμε να γράψουμε:

  `if(ptr1 == ptr2)` ή αντίστοιχα: `if(ptr1 != ptr2)`

- Π.χ., αν ο `p1` δείχνει στο δεύτερο στοιχείο και ο `p2` στο πέμπτο στοιχείο του ίδιου πίνακα, το αποτέλεσμα της πράξης `p2>p1` είναι 1, ενώ της πράξης `p1>p2` είναι 0

```
#include <stdio.h>
int main(void)
{
        double *p1, *p2, x[] = {10, 20, 30, 40, 50, 60, 70};

        p1 = &x[1];
        p2 = &x[4];
        printf("%d", p2>p1);
        printf("%d", p1>p2);
        return 0;
}
```

Παρατηρήσεις

Εκτός από τις λειτουργίες που ήδη περιγράφησαν, **καμία άλλη αριθμητική πράξη δεν επιτρέπεται** να εκτελεστεί με τη συμμετοχή κάποιου δείκτη

- Π.χ. για τη δήλωση `double *ptr, *ptr1, *ptr2;`

οι εντολές:

πολλαπλασιασμού	`ptr *= 2;`
πρόσθεσης δεκαδικού	`ptr += 7.5;`
πρόσθεσης δεικτών	`ptr1 + ptr2;`

δεν είναι επιτρεπτές εκφράσεις ακόμα κι αν οι δείκτες αυτοί δείχνουν σε στοιχεία του ίδιου πίνακα

Παραδείγματα (Ι)

- Τι εμφανίζει η 2η `printf()` του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10, j;
        int *ptr;

        ptr = &j;
        printf("%d ", *ptr);
        ptr++;
        printf("%d ", *ptr);
        return 0;
}
```

Έξοδος: Το τυχαίο περιεχόμενο των επόμενων 4 bytes μετά τη θέση της μεταβλητής j στη μνήμη. Αν τύχει να είναι η διεύθυνση της μεταβλητής i, τότε το πρόγραμμα θα εμφανίσει 10, αλλιώς θα εμφανίσει μια τυχαία τιμή.

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10,j = 20,k = 30;
        int* ptr;

        ptr = &i;
        *ptr = 40;

        ptr = &j;
        *ptr += i;

        ptr = &k;
        *ptr += i + j ;

        printf("i = %d j = %d k = %d\n",i,j,k);
        return 0;
}
```

Έξοδος: i=40 j=60 k=130

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int* ptr1,*ptr2,*ptr3;
        int i = 10,j = 20,k = 30;

        ptr1 = &i;
        i = 100;

        ptr2 = &j;
        j = *ptr2 + *ptr1;

        ptr3 = &k;
        k = *ptr3 + *ptr2;

        printf("%d %d %d\n",*ptr1,*ptr2,*ptr3);
        return 0;
}
```

Έξοδος: 100 120 150

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        int* ptr1, *ptr2;

        ptr1 = &i;
        ptr2 = ptr1;

        *ptr1 = *ptr2 + *ptr1;
        printf("Value = %d\n",i);
        return 0;
}
```

Έξοδος: `Value = 20`

Παραδείγματα (V)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 20;
        int* ptr1, *ptr2;

        ptr1 = ptr2 = &i;

        *ptr2 += 40;
        i += *ptr1;

        printf("Value = %d\n",*ptr1);
        return 0;
}
```

Έξοδος: `Value = 120`

Παραδείγματα (VI)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 10,j = 20;
        int* ptr1, *ptr2;

        ptr1  = &i;
        *ptr1 = 150;

        ptr2 = &j;
        *ptr2 = 50;

        ptr2 = ptr1;
        *ptr2 = 250;

        ptr2 = &j;
        *ptr2 += *ptr1;

        printf("Value = %d\n",j);
        return 0;
}
```

Έξοδος: `Value = 300`

Παραδείγματα (VII)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *ptr, i = 0;

        for(ptr = &i; *ptr < 5; i++)
        {
                (*ptr)++;
                ++*ptr;
                printf("%d ", i);
        }
        return 0;
}
```

Έξοδος: 2 5

Παραδείγματα (VIII)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει δύο ακέραιους, τους οποίους να αποθηκεύει σε δύο ακέραιες μεταβλητές με χρήση δύο δεικτών σε ακέραιους. Με χρήση αυτών των δεικτών, αποθηκεύστε το άθροισμά τους σε μία τρίτη μεταβλητή με χρήση ενός επιπλέον δείκτη σε ακέραιο και εμφανίστε τα περιεχόμενα της κάθε μεταβλητής με χρήση των παραπάνω δεικτών.

```c
#include <stdio.h>
int main(void)
{
	int i,j,k;
	int* ptr1,*ptr2,*ptr3;

	/* A pointer must point to a valid address, before being
used. */
	ptr1 = &i;
	ptr2 = &j;
	ptr3 = &k;

	printf("Enter numbers: ");
	scanf("%d%d",ptr1,ptr2);  /*  Store  the  integers  in  the
memory addresses that the pointers point to. */
	*ptr3 = *ptr1 + *ptr2;

	printf("Values: %d %d %d\n",*ptr1,*ptr2,*ptr3);
	return 0;
}
```

Δείκτες και Πίνακες - Εισαγωγή

- Τα στοιχεία ενός πίνακα αποθηκεύονται σε διαδοχικές θέσεις μνήμης, με το πρώτο στοιχείο στη χαμηλότερη διεύθυνση
- Τα επόμενα στοιχεία του πίνακα αποθηκεύονται στις υψηλότερες διευθύνσεις
- Το πόσο υψηλότερα, εξαρτάται από τον τύπο δεδομένων του πίνακα (`char`, `int`, `float`, ..)
- Π.χ. σε έναν πίνακα χαρακτήρων (`char`), κάθε στοιχείο του πίνακα βρίσκεται 1 byte μετά από το προηγούμενο στοιχείο και η διεύθυνση κάθε στοιχείου είναι 1 θέση υψηλότερα από τη διεύθυνση του προηγούμενου στοιχείου
- Παρομοίως, σε έναν πίνακα ακεραίων (`int`), κάθε στοιχείο του πίνακα βρίσκεται 4 bytes μετά από το προηγούμενο στοιχείο και η διεύθυνση κάθε στοιχείου είναι 4 θέσεις υψηλότερα από τη διεύθυνση του προηγούμενου στοιχείου

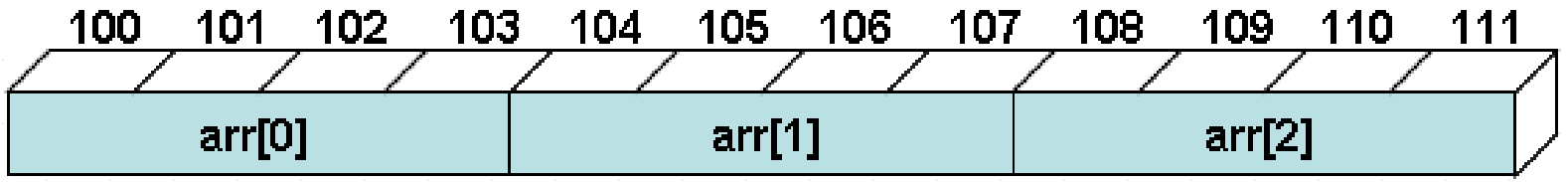
Παράδειγμα

- Έστω η δήλωση του πίνακα:

```
int arr[3];
```

- Αν θεωρήσουμε ότι η διεύθυνση του πρώτου στοιχείου είναι η θέση 100 στη μνήμη, τότε η διεύθυνση του δεύτερου στοιχείου είναι η 104 και του τρίτου η 108

- Αντίστοιχα, η τιμή του πρώτου στοιχείου του πίνακα (του arr[0]) αποθηκεύεται στις θέσεις 100 έως και 103, η τιμή του δεύτερου στοιχείου (του arr[1]) στις θέσεις 104 έως και 107 και η τιμή του τρίτου στοιχείου (του arr[2]) στις θέσεις 108 έως και 111

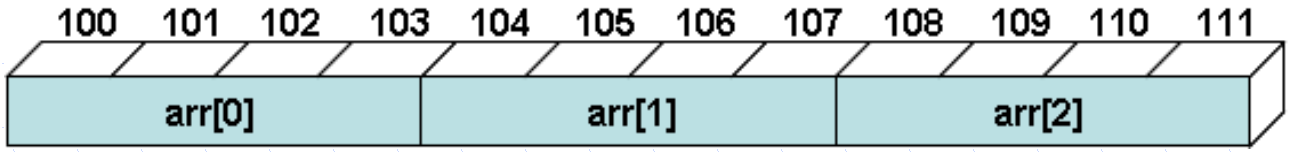


Δείκτες και Πίνακες (Ι)

- Το όνομα ενός πίνακα (χωρίς αγκύλες) μπορεί να χρησιμοποιηθεί ως δείκτης στο πρώτο του στοιχείο
- Με άλλα λόγια, η τιμή του ονόματος του πίνακα (χωρίς αγκύλες) ισούται με τη διεύθυνση του πρώτου στοιχείου του πίνακα
- Π.χ. αν έχει δηλωθεί ο πίνακας

```
int arr[50];
```

τότε η τιμή του `arr` είναι ίση με τη διεύθυνση του πρώτου στοιχείου του πίνακα (δηλ. ίση με `&arr[0]`) και αν η μνήμη του υπολογιστή ήταν όπως αυτή του παρακάτω σχήματος, η τιμή τους θα ήταν ίση με `100`



- Συμπερασματικά, οι εκφράσεις `arr` και `&arr[0]` είναι ισοδύναμες

Δείκτες και Πίνακες (ΙΙ)

- Υπενθυμίζεται από την αριθμητική δεικτών, ότι, όταν προστίθεται ένας ακέραιος αριθμός n σε έναν δείκτη (που δείχνει σε στοιχείο πίνακα), τότε ο δείκτης δείχνει σε μία νέα διεύθυνση που απέχει (σε bytes):

  n * μέγεθος του τύπου (στον οποίο δείχνει)

- Βάσει της λογικής αυτής, η έκφραση `arr+1` μπορεί να χρησιμοποιηθεί ως δείκτης που δείχνει στο δεύτερο στοιχείο του πίνακα, άρα οι εκφράσεις `arr+1` και `&arr[1]` είναι ισοδύναμες, αφού και οι δύο είναι ίσες με τη διεύθυνση του δεύτερου στοιχείου του πίνακα, η έκφραση `arr+2` μπορεί να χρησιμοποιηθεί ως δείκτης που δείχνει στο τρίτο στοιχείο του πίνακα κ.ο.κ.
- Δηλαδή, γενικά ισχύει ότι:

```
arr == &arr[0]
arr + 1 == &arr[1]
arr + 2 == &arr[2]
   ...
arr + n == &arr[n]
```

Δείκτες και Πίνακες (III)

- Έτσι, το παρακάτω πρόγραμμα εμφανίζει 4 φορές την ίδια τιμή

```
#include <stdio.h>
int main(void)
{
        int *ptr, arr[5];

        ptr = arr;
        printf("%p %p %p %p\n", ptr, &arr[0], arr, &arr);
        return 0;
}
```

Σημειώστε ότι παρόλο που η έκφραση **&arr** εμφανίζει την ίδια τιμή, είναι διαφορετική απ' τις υπόλοιπες

- Η έκφραση **&arr** είναι δείκτης σε ολόκληρο τον πίνακα, ενώ οι υπόλοιπες είναι δείκτες στο πρώτο στοιχείο του πίνακα (ως τιμές είναι ίδιες, αλλά διαφέρουν στον τύπο τους)
- Συγκεκριμένα, ο τύπος **&arr** είναι "δείκτης σε έναν πίνακα 5 ακεραίων", ενώ ο τύπος των υπολοίπων είναι "δείκτης σε ακέραιο". Ζόρικο???

- Και για να το δυσκολέψουμε ακόμα περισσότερο, αν γράψουμε **&arr+1** αντί για **&arr**, ποια θα είναι η διαφορά με τις άλλες τιμές (θα είναι ένα, τέσσερα, ή μήπως κάτι άλλο, ...)?

Για να σας δω... ☺

Δείκτες και Πίνακες (IV)

- Αφού το όνομα ενός πίνακα μπορεί να χρησιμοποιηθεί ως δείκτης στο πρώτο στοιχείο του, τότε το περιεχόμενό του θα είναι ίσο με την τιμή του πρώτου στοιχείου του
- Δηλαδή, ισχύει ότι το `*arr` είναι ίσο με `arr[0]`
- Αντίστοιχα, αφού το `arr+1` είναι δείκτης στο δεύτερο στοιχείο του πίνακα, τότε ισχύει ότι `*(arr+1)` είναι ίσο με `arr[1]`, κ.ο.κ.

Δηλαδή, γενικά ισχύει ότι (προσοχή στις παρενθέσεις):

```
*arr == arr[0]
*(arr + 1) == arr[1]
*(arr + 2) == arr[2]
    ...
*(arr + n) == arr[n]
```

- Ο τελεστής * έχει υψηλότερη προτεραιότητα απ΄τον τελεστή +
- Επομένως, οι εκφράσεις `*(arr+n)` και `*arr + n` δεν είναι ισοδύναμες (αφού δεν αποτιμούνται με τον ίδιο τρόπο)

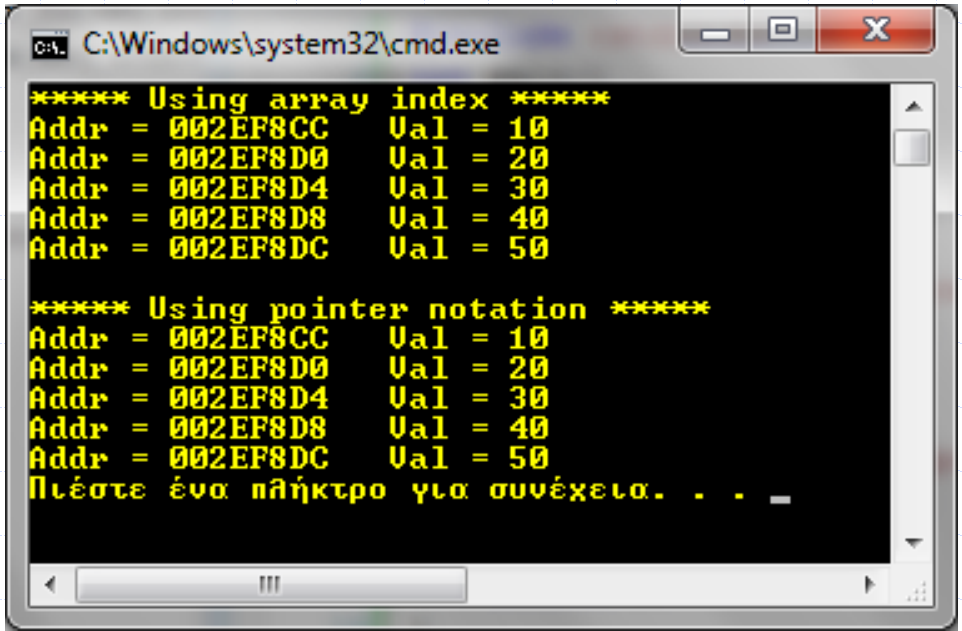
Παράδειγμα (Ι)

```c
#include <stdio.h>
int main(void)
{
	int i, arr[5] = {10, 20, 30, 40, 50};

	printf("***** Using array notation *****\n");
	for(i = 0; i < 5; i++)
		printf("Addr = %p   Val = %d\n", &arr[i], arr[i]);

	printf("\n***** Using pointer notation *****\n");
	for(i = 0; i < 5; i++)
		printf("Addr = %p   Val = %d\n", arr+i, *(arr+i));
	return 0;
}
```

Πιθανή Έξοδος:

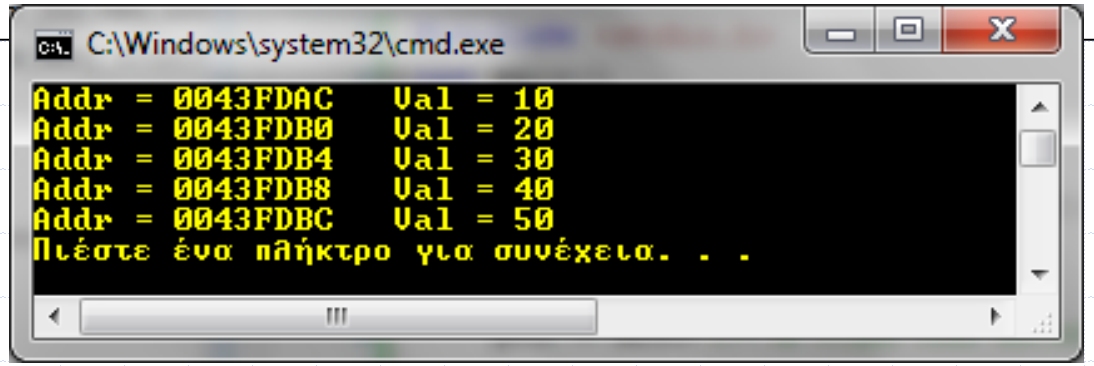


Παράδειγμα (ΙΙ)

```
#include <stdio.h>
int main(void)
{
	int i,arr[5] = {10,20,30,40,50};
	int *ptr;

	ptr = arr; /* The value of ptr pointer becomes equal to
the memory address of arr[0]. */
	for(i = 0; i < 5; i++)
	{
		printf("Addr = %p   Val = %d\n",ptr,*ptr);
		ptr++; /* The value of ptr pointer becomes equal to
the memory address of the next array element. Equivalently, we
could write ptr = &arr[i]; */
	}
	return 0;
}
```

Πιθανή Έξοδος:



Παρατηρήσεις (Ι)

Όταν το όνομα ενός πίνακα χρησιμοποιείται ως δείκτης, η C τον μεταχειρίζεται ως `const` δείκτη, συνεπώς, δεν επιτρέπεται ούτε να αλλάξει την τιμή του ούτε να δείξει σε κάποια άλλη διεύθυνση

- Το όνομα ενός πίνακα δεν είναι δηλαδή μία τροποποιήσιμη *lvalue*; Η τιμή του θα είναι μόνιμα ίση με τη διεύθυνση του πρώτου του στοιχείου
- Έτσι, αν στο τελευταίο παράδειγμα γράφαμε `arr++;` ή `arr = &i;` ο μεταγλωττιστής θα εμφάνιζε μήνυμα λάθους για μη επιτρεπτή ενέργεια
- Συνεπώς, μάλλον τώρα μπορείτε να καταλάβετε γιατί δεν επιτρέπεται να γράψετε `b = a;` Για να αντιγράψετε τα στοιχεία ενός πίνακα `a` στον αντίστοιχο πίνακα `b`
- Παρόλα αυτά, επιτρέπεται να αντιγράψουμε την τιμή του ονόματος ενός πίνακα σε μία άλλη μεταβλητή δείκτη (όπως π.χ. κάναμε γράφοντας `ptr = arr;` στο τελευταίο παράδειγμα) και να χρησιμοποιήσουμε τον δείκτη `ptr` για να έχουμε πρόσβαση στα στοιχεία του πίνακα

Παρατηρήσεις (ΙΙ)

- Παρόλο που υπάρχει στενή σχέση μεταξύ πινάκων και δεικτών, πρέπει να είναι ξεκάθαρο ότι ένας πίνακας δεν είναι δείκτης καθώς και ένας δείκτης δεν είναι πίνακας...

- Π.χ., οι δηλώσεις `int a[50];` και `int *a;` είναι αρκετά διαφορετικές
  - Με την πρώτη δεσμεύεται μνήμη για 50 ακεραίους (π.χ. εδώ δεσμεύονται `50*4 = 200` bytes) και το όνομα `a` αναφέρεται πάντα στην ίδια θέση μνήμης. Όπως είπαμε, δεν μπορεί να αλλάξει η τιμή του, δηλαδή δεν μπορούμε να γράψουμε `a = arr;`
  - Με τη δεύτερη δήλωση δεσμεύεται μνήμη (τυπικά δεσμεύονται 4 bytes) για να αποθηκευτεί η τιμή του δείκτη. Η τιμή του μπορεί να αλλάζει, ώστε να δείχνει σε διαφορετικές διευθύνσεις, δηλαδή μπορούμε να γράψουμε `a = arr;`

Παρατηρήσεις (III)

Απλά να θυμάστε ότι, όταν το όνομα ενός πίνακα χρησιμοποιείται σε μία έκφραση, ο μεταγλωττιστής το μεταφράζει σε δείκτη στο πρώτο του στοιχείο

- Πάντα; Όχι πάντα, υπάρχουν κάποιες εξαιρέσεις...

a. Όταν αποτελεί τον τελεστέο του τελεστή `sizeof`, αφού τότε υπολογίζεται το μέγεθος όλου του πίνακα

b. Όπως ήδη αναφέραμε, όταν χρησιμοποιείται με τον τελεστή `&`, όπου τότε λαμβάνουμε τη διεύθυνση του πίνακα και όχι τη διεύθυνση του πρώτου του στοιχείου

c. Όταν ο πίνακας είναι ένα κυριολεκτικό αλφαριθμητικό που χρησιμοποιείται σαν αρχική τιμή σε μία δήλωση

Παρατηρήσεις (IV)

Αν και μία μεταβλητή δείκτης δεν είναι πίνακας, μπορεί να χρησιμοποιηθεί με σημειογραφία πίνακα

- Για παράδειγμα, το επόμενο πρόγραμμα χρησιμοποιεί τον δείκτη **ptr** σαν να ήταν πίνακας, για να εμφανίσει τις διευθύνσεις μνήμης και τις τιμές των στοιχείων του πίνακα **arr**

```
#include <stdio.h>
int main(void)
{
        int *ptr, i, arr[5] = {10, 20, 30, 40, 50};

        ptr = arr;
        for(i = 0; i < 5; i++)
                printf("Addr = %p  Val = %d\n", &ptr[i], ptr[i]);
        return 0;
}
```

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *ptr, arr[] = {10, 20, 30, 40, 50};

        ptr = arr;
        printf("Val1 = %d Val2 = %d\n", *ptr+2, *(ptr+2));
        return 0;
}
```

Έξοδος: `Val1 = 12  Val2 = 30`

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int* ptr;
        int i,j,arr[] = {10,20,30,40,50};

        ptr = arr;
        *ptr = 3;

        ptr += 2;
        *ptr = 5;

        printf("Val = %d\n",arr[0] + arr[2] + arr[4]);
        return 0;
}
```

Έξοδος: `Val = 58`

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int* ptr1, *ptr2;
        int arr[] = {10,20,30,40,50};

        ptr1 = &arr[0];
        ptr2 = &arr[3];

        printf("%d\n",ptr2-ptr1);
        return 0;
}
```

Έξοδος: 3

Παραδείγματα (IV)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int* ptr;
        int i,arr[5] = {10,20,30,40,50};

        ptr = arr+2;
        for(i = 0; i < 5; i++)
                printf("%d ",ptr[i]);
        return 0;
}
```

Έξοδος: 30 40 50 (και δύο τυχαίες τιμές)

Παραδείγματα (V)

- Υπάρχει κάποιο bug στο παρακάτω πρόγραμμα ???

```
#include <stdio.h>
int main(void)
{
        int i,arr[5] = {10,20,30,40,50};

                printf("%d\n",0[arr]);
                printf("%d\n",2[arr]);
                printf("%d\n",4[arr]);
        return 0;
}
```

Απάντηση: Όχι...

«σκονάκι No1»: θυμηθείτε ότι *(arr+i)= arr[i]

Κι άλλη υπόδειξη???

«σκονάκι No2»: *(arr+i)= *(i+arr)

Κι άλλο???

«σκονάκι No3»: arr[i]=*(arr+i)= *(i+arr)= i[arr]

Πίνακας Δεικτών

- Ένας πίνακας δεικτών είναι ένας πίνακας, όπου κάθε στοιχείο του είναι ένας δείκτης σε έναν συγκεκριμένο τύπο δεδομένων

- Για να δηλώσουμε έναν πίνακα δεικτών χρησιμοποιούμε τον τελεστή * πριν από το όνομα του πίνακα

Π.χ.

```
int *p[3];
```

Δήλωση ενός πίνακα δεικτών με όνομα p, ο οποίος περιέχει 3 στοιχεία και το καθένα από αυτά είναι ένας δείκτης σε μία ακέραια μεταβλητή τύπου int

Π.χ.

```
double *arr[5];
```

Δήλωση ενός πίνακα δεικτών με όνομα arr, ο οποίος περιέχει 5 στοιχεία και το καθένα από αυτά είναι ένας δείκτης σε έναν δεκαδικό αριθμό τύπου double

Παρατηρήσεις

- Για να μην τρομάζετε, το κάθε στοιχείο ενός πίνακα δεικτών, μπορεί να θεωρηθεί ως μία απλή μεταβλητή δείκτη
- Προσοχή, μόνο, όταν δηλώνεται ένας πίνακας δεικτών, το όνομα του πίνακα δεν πρέπει να περικλείεται σε παρενθέσεις
- Π.χ. με τη δήλωση:

```
int (*p)[3];
```

η μεταβλητή `p` δηλώνεται ως δείκτης προς έναν πίνακα τριών ακεραίων και όχι σαν πίνακας τριών δεικτών

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *p[3], i = 100, j = 200, k = 300;

        p[0] = &i;
        p[1] = &j;
        p[2] = &k;
        printf("%d %d %d\n", *p[0], *p[1], *p[2]);
        return 0;
}
```

Έξοδος: 100 200 300

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *p[3], i, arr[4] = {10, 20, 30, 40};

        for(i = 0; i < 3; i++)
        {
                p[i] = &arr[i]+1;
                printf("%d ", *p[i]);
        }
        return 0;
}
```

Έξοδος: 20 30 40

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int *p[3], i, num;

        for(i = 0; i < 3; i++)
        {
                printf("Enter number: ");
                scanf("%d", &num);
                p[i] = &num;
        }
        for(i = 0; i < 3; i++)
                printf("Num: %d\n", *p[i]);
        return 0;
}
```

Έξοδος: ο δεύτερος βρόχος εμφανίζει τρεις φορές την τελευταία εισαχθείσα τιμή.

Δείκτης σε Δείκτη

- Όταν δηλώνεται ένας δείκτης, ο μεταγλωττιστής, όπως κάνει για οποιαδήποτε μεταβλητή, δεσμεύει τις απαραίτητες θέσεις μνήμης για να αποθηκεύσει την τιμή του
- Επομένως, αφού έχει δεσμευτεί μία διεύθυνση μνήμης για έναν δείκτη μπορούμε να δηλώσουμε έναν άλλον δείκτη που να δείχνει σε αυτή τη διεύθυνση
- Για να δηλώσουμε έναν δείκτη σε κάποιον άλλον δείκτη χρησιμοποιούμε δύο φορές τον τελεστή *

- Παραδείγματα Δηλώσεων «Δείκτη σε Δείκτη»

```
int** ptr; /* Η μεταβλητή ptr δηλώνεται σαν δείκτης προς
κάποιον άλλον δείκτη, ο οποίος με τη σειρά του δείχνει στη
διεύθυνση μίας ακέραιας μεταβλητής. */

char** ptr; /* Η μεταβλητή ptr δηλώνεται σαν δείκτης προς
κάποιον άλλον δείκτη, ο οποίος με τη σειρά του δείχνει στη
διεύθυνση μίας μεταβλητής χαρακτήρα. */
```

Χρήση «Δείκτη σε Δείκτη»

- Αν έχουμε δηλώσει έναν δείκτη σε έναν δεύτερο δείκτη, τότε με τον τελεστή * έχουμε πρόσβαση στη διεύθυνση του δεύτερου δείκτη και με τον διπλό τελεστή ** έχουμε πρόσβαση στη μεταβλητή που δείχνει ο δεύτερος δείκτης

```
#include <stdio.h>
int main()
{
    int i = 20;
    int* ptr1;
    int** ptr;
    ptr1 = &i;
    ptr = &ptr1;
    printf("Value = %d\n", **ptr);
    return 0;
}
```

Δήλωση Μεταβλητής (int)

Δήλωση Δείκτη (ptr1) σε int

Δήλωση Δείκτη (ptr) σε Δείκτη (που δείχνει σε int)

Ο δείκτης ptr1 δείχνει στον i

Ο δεύτερος δείκτης ptr δείχνει στον πρώτο δείκτη ptr1

**ptr = * (ptr1) = i

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        int i = 20;
        int* ptr1;
        int** ptr;

        ptr1 = &i;
        ptr = &ptr1;

        **ptr += 100;
        printf("%d %d %d\n", **ptr, *ptr1, i);
        return 0;
}
```

Έξοδος: `120 120 120`

Δείκτες και Διδιάστατοι Πίνακες (Ι)

- Όπως και στους μονοδιάστατους πίνακες, έτσι και στους πολυδιάστατους, η κάθε διάσταση δηλώνεται μέσα σε αγκύλες `[]`
- Π.χ. με την εντολή: `int arr[2][3];`

  δηλώνεται ένας διδιάστατος πίνακας, ο οποίος αποτελείται από 2 γραμμές και 3 στήλες (δηλ. συνολικά περιέχει 6 ακέραιες μεταβλητές) και σχηματικά απεικονίζεται όπως παρακάτω

`arr[0][0]`	`arr[0][1]`	`arr[0][2]`
`arr[1][0]`	`arr[1][1]`	`arr[1][2]`

- Τα στοιχεία του πίνακα αποθηκεύονται σε διαδοχικές θέσεις στη μνήμη ξεκινώντας από τα στοιχεία της 1η γραμμής, συνεχίζοντας με τα στοιχεία της 2ης γραμμής, κ.ο.κ.
- Άρα, η σειρά αποθήκευσης των στοιχείων του παραπάνω πίνακα `arr` στη μνήμη είναι: `arr[0][0]`, `arr[0][1]`, `arr[0][2]`, `arr[1][0]`, `arr[1][1]` και `arr[1][2]`

Δείκτες και Διδιάστατοι Πίνακες (ΙΙ)

- Θυμηθείτε, επίσης, ότι η C χειρίζεται έναν διδιάστατο πίνακα σαν μονοδιάστατο πίνακα, όπου το κάθε στοιχείο του είναι ένας μονοδιάστατος πίνακας
- Στο προηγούμενο παράδειγμα, τα στοιχεία του `arr` είναι τα `arr[0]` και `arr[1]`, όπου το καθένα από αυτά είναι μονοδιάστατος πίνακας τριών ακεραίων
- Για να χειριστούμε έναν διδιάστατο πίνακα με χρήση δεικτών, έστω `arr[N][M]`, μπορούμε να θεωρήσουμε ότι ο πίνακας `arr` αποτελείται από έναν πίνακα δεικτών `N` στοιχείων, `arr[0]`, `arr[1]`, ..., `arr[N-1]`, όπου καθένα από αυτά είναι δείκτης σε έναν πίνακα `M` στοιχείων
- Π.χ. με την εντολή:

```
int arr[2][3];
```

το `arr[0]` μπορεί να χρησιμοποιηθεί σαν δείκτης προς έναν μονοδιάστατο πίνακα 3 ακεραίων που περιέχει τα στοιχεία της πρώτης γραμμής, δηλαδή τα `arr[0][0]`, `arr[0][1]` και `arr[0][2]`

Δείκτες και Διδιάστατοι Πίνακες (ΙΙΙ)

- Συγκεκριμένα, το `arr[0]` είναι δείκτης στο πρώτο στοιχείο του πίνακα, δηλαδή στο `arr[0][0]`, άρα, η τιμή του `*arr[0]` είναι ίση με το `arr[0][0]`
- Επίσης, σύμφωνα με την αριθμητική δεικτών:
  - το `arr[0]+1` είναι δείκτης στο δεύτερο στοιχείο του πίνακα, δηλαδή στο `arr[0][1]`
  - το `arr[0]+2` είναι δείκτης στο τρίτο στοιχείο του πίνακα, δηλαδή στο `arr[0][2]`, κ.ο.κ.
  - ...
  - συνεπώς, στη γενική περίπτωση ισχύει ότι το `arr[0]+κ` είναι δείκτης στο στοιχείο `arr[0][κ]` της πρώτης γραμμής του διδιάστατου πίνακα

- Δηλαδή, ισχύει ότι:
  - το `arr[0]+κ` είναι ισοδύναμο με `&arr[0][κ]`
  - η τιμή του `*(arr[0]+κ)` είναι ίση με `arr[0][κ]`

Δείκτες και Διδιάστατοι Πίνακες (IV)

- Αντίστοιχα, το `arr[1]` μπορεί να χρησιμοποιηθεί σαν δείκτης προς έναν πίνακα 3 ακεραίων που περιέχει τα στοιχεία της δεύτερης γραμμής, δηλαδή τα `arr[1][0]`, `arr[1][1]` και `arr[1][2]`

- Συγκεκριμένα:
  - το `arr[1]` είναι δείκτης στο πρώτο στοιχείο του πίνακα, δηλαδή στο `arr[1][0]`
  - Άρα, η τιμή του `*arr[1]` είναι ίση με το `arr[1][0]`

- Παρομοίως με πριν, ισχύει ότι το `arr[1]+κ` είναι δείκτης στο στοιχείο `arr[1][κ]` της δεύτερης γραμμής του διδιάστατου πίνακα

- Δηλαδή, ισχύει ότι:
  - το `arr[1]+κ` είναι ισοδύναμο με `&arr[1][κ]`
  - η τιμή του `*(arr[1]+κ)` είναι ίση με `arr[1][κ]`

Δείκτες και Διδιάστατοι Πίνακες (V)

- Γενικά, θεωρούμε ότι τα στοιχεία ενός πίνακα `arr[N][M]`, είναι τα `arr[0]`, `arr[1]`, ..., `arr[N-1]` τα οποία είναι **δείκτες σε πίνακες** που περιέχουν τα `M` στοιχεία της αντίστοιχης γραμμής

- Δηλαδή,
  - το **πρώτο** στοιχείο του πίνακα `arr[N][M]` είναι το `arr[0]`, το οποίο είναι δείκτης σε έναν πίνακα που περιέχει τα `M` στοιχεία της **πρώτης** γραμμής
  - το **δεύτερο** στοιχείο του πίνακα `arr[N][M]` είναι το `arr[1]`, το οποίο είναι δείκτης σε έναν πίνακα που περιέχει τα `M` στοιχεία της **δεύτερης** γραμμής
  - ...
  - ενώ το **τελευταίο** στοιχείο είναι το `arr[N-1]`, το οποίο είναι δείκτης σε έναν πίνακα που περιέχει τα `M` στοιχεία της **τελευταίας** (της `N`-οστής) γραμμής

Παράδειγμα

- Τι κάνει το παρακάτω πρόγραμμα ???

```
#include <stdio.h>
int main()
{
	int i,k,arr[2][3] = {10,20,30,40,50,60};

	for(i = 0; i < 2; i++)
		for(k = 0; k < 3; k++)
			printf("Value of [%d][%d] element is: %d\n",
i,k,*(arr[i] + k));
	return 0;
}
```

Εμφανίζει τις τιμές όλων των στοιχείων του πίνακα με χρήση δείκτη !!!

Χειρισμός Διδιάστατου Πίνακα με «δείκτη σε δείκτη» (Ι)

- Ένας εναλλακτικός τρόπος για να διαχειριστούμε έναν διδιάστατο πίνακα με χρήση δείκτη, είναι χρησιμοποιώντας **το όνομα του πίνακα**
- Θυμηθείτε ότι **το όνομα** ενός πίνακα χωρίς τις αγκύλες είναι ισοδύναμο με **τη διεύθυνση του πρώτου στοιχείου** του πίνακα
- Π.χ. αν θεωρήσουμε την παρακάτω δήλωση:

```
int arr[2][3];
```

το όνομα του πίνακα `arr` είναι δείκτης στο πρώτο στοιχείο του πίνακα, δηλ. στο `arr[0]`

- Όμως, όπως είδαμε προηγουμένως, το πρώτο στοιχείο του πίνακα (το `arr[0]`) είναι με τη σειρά του δείκτης σε έναν πίνακα που περιέχει τα 3 στοιχεία της **πρώτης** γραμμής
- Συγκεκριμένα, το `arr[0]` είναι δείκτης στο πρώτο στοιχείο του πίνακα, δηλαδή στο `arr[0][0]`

Χειρισμός Διδιάστατου Πίνακα με «δείκτη σε δείκτη» (ΙΙ)

- Άρα, ισχύει ότι το `arr` είναι δείκτης στο `arr[0]` και το `arr[0]` είναι δείκτης στο `arr[0][0]`
- Επομένως, πώς μπορούμε να χειριστούμε το όνομα του πίνακα `arr` ???
- Η απάντηση είναι ότι μπορούμε να το χειριστούμε σαν **δείκτη προς δείκτη**
- Π.χ. αφού το `arr` είναι δείκτης σε έναν δείκτη που δείχνει στη διεύθυνση του στοιχείου `arr[0][0]`, τότε το `**arr` είναι ίσο με την τιμή `arr[0][0]`
- Παρομοίως, το `arr+1` είναι δείκτης στο `arr[1]` και το `arr[1]` είναι δείκτης που δείχνει στη διεύθυνση του στοιχείου `arr[1][0]`
- Άρα, το `**(arr+1)` είναι ίσο με την τιμή `arr[1][0]`

- Στη γενική περίπτωση, ισχύει ότι:
  - το `arr+κ` είναι ισοδύναμο με `&arr[κ]`
  - το `*(arr+κ)` είναι ισοδύναμο με `arr[κ]`, δηλαδή με `&arr[κ][0]`
  - το `**(arr+κ)` είναι ισοδύναμο με `arr[κ][0]`

Παράδειγμα

- Τι κάνει το παρακάτω πρόγραμμα ???

```
#include <stdio.h>
int main()
{
	int i,k,arr[2][3] = {10,20,30,40,50,60};

	for(i = 0; i < 2; i++)
		for(k = 0; k < 3; k++)
			printf("Value of [%d][%d] element is: %d\n",
i,k,*(*(arr + i) + k));
	return 0;
}
```

Εμφανίζει τις τιμές όλων των στοιχείων του πίνακα με χρήση του ονόματος του πίνακα ως «δείκτη σε δείκτη» !!!

Παρατηρήσεις Ι

- Προφανώς, η διαχείριση των στοιχείων ενός διδιάστατου πίνακα με χρήση δεικτών (είτε απλού δείκτη είτε «δείκτη σε δείκτη») οδηγεί σε δυσνόητο και μη ευανάγνωστο κώδικα

- Για τον λόγο αυτό, προτείνουμε η διαχείριση των στοιχείων να γίνεται με τη χρήση των αγκυλών `[][]` και των αντιστοίχων θέσεων στον πίνακα

Παρατηρήσεις ΙΙ

- Έστω ότι είχαμε τις ακόλουθες δηλώσεις:

```
int a[2][3], b[10];
```

- Ερώτηση: Όπως λέμε ότι το b μπορεί να χρησιμοποιηθεί σαν δείκτης στο b[0], μπορούμε να πούμε ότι και το a είναι δείκτης στο a[0][0];

  Η απάντηση είναι ΟΧΙ, επειδή η C χειρίζεται τον a σαν μονοδιάστατο πίνακα με στοιχεία πίνακες, οπότε το a είναι δείκτης στο πρώτο στοιχείο του που είναι το a[0], άρα, αν θέλαμε να εκχωρήσουμε το a σε έναν δείκτη, ποιος θα έπρεπε να είναι ο τύπος του δείκτη; Θα έπρπε να είναι δείκτης σε πίνακα, π.χ.:

```
int (*p)[3]; /* Οι παρενθέσεις είναι απαραίτητες, γιατί
αλλιώς το p θα μεταφραζόταν σαν πίνακας τριών δεικτών σε
ακεραίους. */
p = a;
```

και τώρα το p δείχνει στο πρώτο στοιχείο της πρώτης γραμμής του a, δηλαδή στο a[0][0]

Παρατηρήσεις ΙΙΙ

Να θυμάστε: αν έχουμε τη δήλωση `int x[5]` ο πίνακας `x` μπορεί να «υποβιβαστεί» σε δείκτη, ενώ αν έχουμε τη δήλωση `int y[5][3]` ο πίνακας `y` δεν «υποβιβάζεται» σε δείκτη σε δείκτη, αλλά σε δείκτη σε πίνακα

Επίσης σημειώστε: με δεδομένες τις δηλώσεις `int x[5]` και `int y[5][3]` η έκφραση `&x` είναι δείκτης σε έναν πίνακα 5 ακεραίων, ενώ η έκφραση `&y` είναι δείκτης σε έναν πίνακα 5 πινάκων που ο καθένας έχει 3 ακεραίους.

Παράδειγμα

- Τι κάνει το παρακάτω πρόγραμμα ???

```
#include <stdio.h>
int main(void)
{
      int i, arr[2][5] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

      for(i = 0; i < 2; i++)
            *(arr[i]+3) = 0;
      return 0;
}
```

Σε κάθε επανάληψη, το `arr[i]` δείχνει στο πρώτο στοιχείο της γραμμής `i`. Η έκφραση `arr[i]+3` είναι ένας δείκτης στο τέταρτο στοιχείο της γραμμής `i`. Συνεπώς, το `*(arr[i]+3)` είναι ισοδύναμο με `arr[i][3]`, οπότε, το πρόγραμμα μηδενίζει τα στοιχεία της τέταρτης στήλης του πίνακα, δηλαδή τα `arr[0][3]` και `arr[1][3]` γίνονται 0

Δείκτης προς Συνάρτηση

- **ΠΡΟΣΟΧΗ:** Για να γίνει κατανοητή η συγκεκριμένη υποενότητα, θα πρέπει να έχει διδαχθεί η ενότητα των Συναρτήσεων
- Όταν εκτελείται ένα πρόγραμμα, ο κώδικας για κάθε συνάρτηση που περιέχει το πρόγραμμα φορτώνεται στη μνήμη αρχίζοντας σε μία συγκεκριμένη διεύθυνση
- Ένας **δείκτης προς μία συνάρτηση δείχνει στη διεύθυνση μνήμης, στην οποία είναι αποθηκευμένος ο κώδικας της συνάρτησης**
- Η γενική μορφή της δήλωσης ενός δείκτη προς μία συνάρτηση είναι:

```
τύπος_επιστροφής (*όνομα_δείκτη) (τύπος_παραμ_1 όνομα_1,
τύπος_παραμ_2 όνομα_2, ..., τύπος_παραμ_ν όνομα_ν);
```

- Ο `τύπος_επιστροφής` καθορίζει τον τύπο δεδομένων που επιστρέφει η συνάρτηση στο πρόγραμμα που την κάλεσε
- Οι μεταβλητές `όνομα_1, όνομα_2,..., όνομα_ν` αποτελούν τις παραμέτρους της συνάρτησης (εφόσον η συνάρτηση δέχεται παραμέτρους)

Παραδείγματα δήλωσης Δείκτη προς Συνάρτηση

```
int (*ptr)(int arr[], int size); /* Η μεταβλητή ptr
δηλώνεται σαν δείκτης προς μία συνάρτηση, η οποία
δέχεται σαν παραμέτρους έναν πίνακα ακεραίων και
έναν ακέραιο και επιστρέφει μία ακέραια τιμή. */

void (*ptr)(double *arr[]); /* Η μεταβλητή ptr
δηλώνεται σαν δείκτης προς μία συνάρτηση, η οποία
δέχεται σαν παράμετρο έναν πίνακα δεικτών σε
πραγματικούς και δεν επιστρέφει τίποτα. */

int test(void (*ptr)(int a)); /* Η συνάρτηση test()
επιστρέφει μία ακέραια τιμή και δέχεται σαν
παράμετρο έναν δείκτη προς μία άλλη συνάρτηση, η
οποία δέχεται μία ακέραια παράμετρο και δεν
επιστρέφει τίποτα. */
```

Παρατηρήσεις

- Το όνομα του δείκτη **πρέπει** να βρίσκεται ανάμεσα σε **παρενθέσεις**, γιατί ο τελεστής * έχει χαμηλότερη προτεραιότητα από τις παρενθέσεις που περιβάλλουν τη λίστα παραμέτρων της συνάρτησης

- Π.χ. αν γράψουμε:

```
int *ptr(int a);
```

αντί για

```
int (*ptr)(int a);
```

τότε δηλώνεται μία συνάρτηση με όνομα `ptr`, η οποία δέχεται μία ακέραια παράμετρο και επιστρέφει έναν δείκτη σε μία ακέραια μεταβλητή

Χρήση Δείκτη προς Συνάρτηση

- Η μόνη απαίτηση για να δείξει ένας δείκτης σε μία συνάρτηση είναι **ο τύπος επιστροφής** της συνάρτησης και **η λίστα παραμέτρων** της **να είναι ίδια** με **τον τύπο επιστροφής** και τη **λίστα παραμέτρων** της δήλωσης του δείκτη
- Παράδειγμα:

```
#include <stdio.h>

void test(int a);

int main(void)
{
      void (*ptr)(int a); /* Η μεταβλητή ptr δηλώνεται σαν
δείκτης προς μία συνάρτηση, η οποία δέχεται μία ακέραια
παράμετρο και δεν επιστρέφει τίποτα. */

      ptr = test; /* Ο δείκτης ptr δείχνει στη διεύθυνση μνήμης
της συνάρτησης test(). */
      (*ptr)(10); /* Κλήση της συνάρτησης στην οποία δείχνει ο
δείκτης ptr. */
      return 0;
}

void test(int a)
{
      printf("%d\n", 2*a);
}
```

Παράδειγμα

Δημιουργήστε μία συνάρτηση που να δέχεται σαν παραμέτρους τους βαθμούς δύο φοιτητών και να επιστρέφει τον μεγαλύτερο από αυτούς. Στη συνέχεια γράψτε ένα πρόγραμμα το οποίο να διαβάζει δύο βαθμούς και να χρησιμοποιεί έναν δείκτη για να καλέσει τη συνάρτηση και να εμφανίσει τον μεγαλύτερο βαθμό.

```c
#include <stdio.h>

float test(float a, float b);

int main(void)
{
	float (*ptr)(float a, float b); /* Η μεταβλητή ptr δηλώνεται σαν δείκτης προς μία συνάρτηση, η οποία δέχεται δύο πραγματικούς αριθμούς και επιστρέφει μία πραγματική τιμή. */
	float i, j, max;

	printf("Enter grades: ");
	scanf("%f%f", &i, &j);

	ptr = test;
	max = (*ptr)(i, j); /* Κλήση της συνάρτησης στην οποία δείχνει ο δείκτης ptr. */
	printf("Max = %f\n", max);
	return 0;
}

float test(float a, float b)
{
	if(a > b)
		return a;
	else
		return b;
}
```

Πίνακας Δεικτών σε Συναρτήσεις

- Ένας πίνακας δεικτών σε συναρτήσεις είναι ένας πίνακας όπου κάθε στοιχείο του είναι δείκτης σε κάποια συνάρτηση

- Η δήλωσή του είναι παρόμοια με τη δήλωση ενός δείκτη σε συνάρτηση με τη διαφορά ότι αντί για ένας δείκτης δηλώνεται ένας πίνακας δεικτών

- Π.χ. η εντολή:

```
void (*ptr[20])(int a);
```

δηλώνει έναν πίνακα δεικτών με όνομα ptr, του οποίου τα στοιχεία είναι 20 δείκτες προς συναρτήσεις, οι οποίες δέχονται μία ακέραια παράμετρο και δεν επιστρέφουν τίποτα

Παράδειγμα

- Τι κάνει το παρακάτω πρόγραμμα ???

```c
#include <stdio.h>

int test_1(int a, int b);
int test_2(int a, int b);
int test_3(int a, int b);

int main(void)
{
      int (*ptr[3])(int a, int b);
      int i, j, k;

      ptr[0] = test_1; /* Ο δείκτης ptr[0] δείχνει στη διεύθυνση της συνάρτησης test_1. */
      ptr[1] = test_2;
      ptr[2] = test_3;

      printf("Enter numbers: ");
      scanf("%d%d", &i, &j);

      if(i > 0 && i < 10)
            k = ptr[0](i, j); /* Κλήση της συνάρτησης στην οποία δείχνει ο δείκτης ptr[0]. */
      else if(i >= 10 && i < 20)
            k = ptr[1](i, j); /* Κλήση της συνάρτησης στην οποία δείχνει ο δείκτης ptr[1]. */
      else
            k = ptr[2](i, j); /* Κλήση της συνάρτησης στην οποία δείχνει ο δείκτης ptr[2]. */
      printf("Val = %d\n", k);
      return 0;
}
```


(συνέχεια προγράμματος) →

Παράδειγμα

```
int test_1(int a, int b)
{
      return a+b;
}

int test_2(int a, int b)
{
      return a-b;
}

int test_3(int a, int b)
{
      return a*b;
}
```

Το πρόγραμμα ανάλογα με την τιμή του **i** καλεί την αντίστοιχη συνάρτηση μέσω του δείκτη που δείχνει στη διεύθυνσή της.
Η τιμή επιστροφής της συνάρτησης εμφανίζεται στην οθόνη.