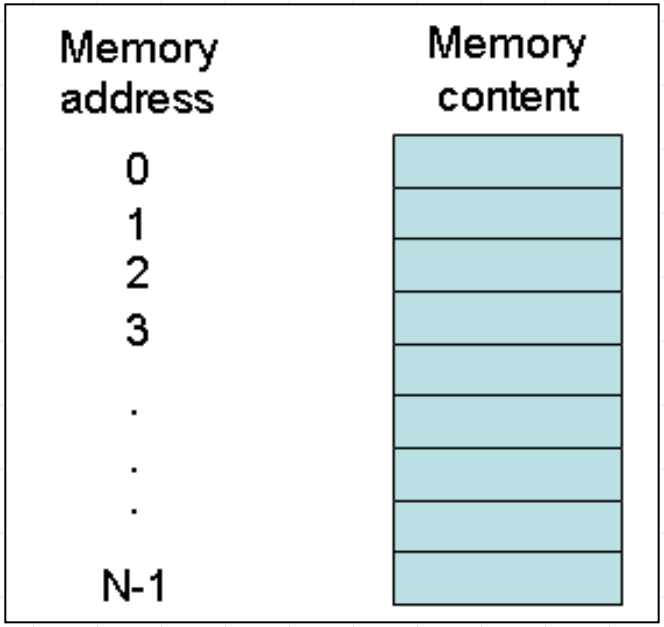
C: Από τη Θεωρία στην Εφαρμογή

Κεφάλαιο 8ο
Δείκτες

Γ. Σ. Τσελίκης – Ν. 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

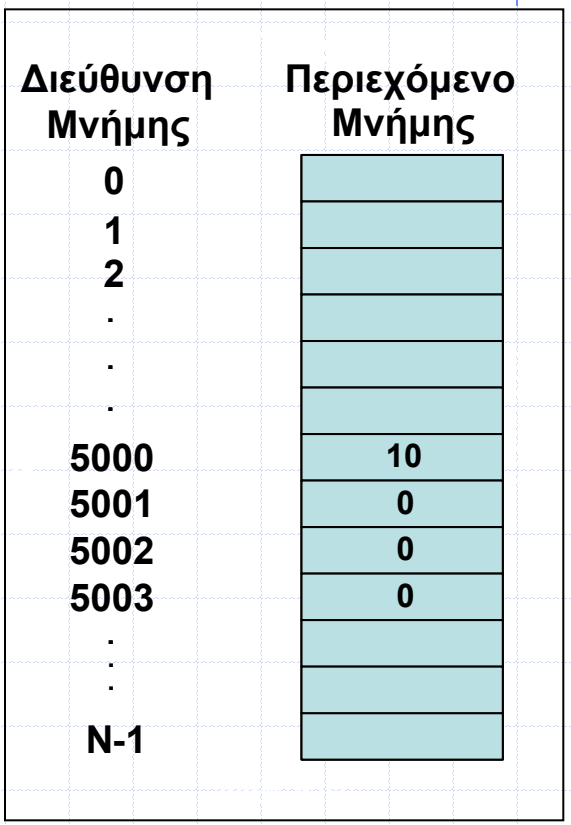


Μνήμη Υπολογιστή και Μεταβλητές

- 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**

```c
#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 ???

```
#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

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

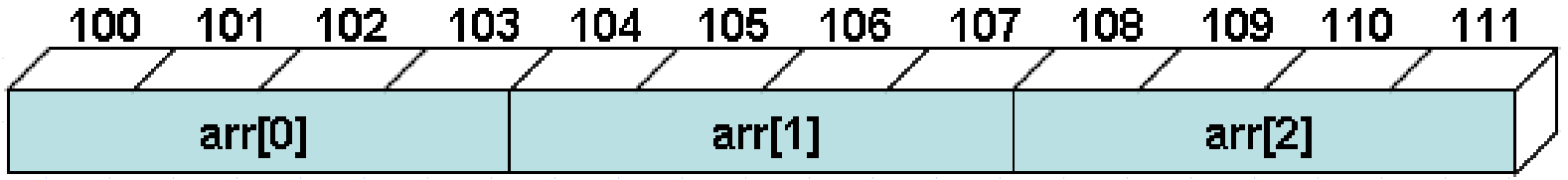
Παράδειγμα

- 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

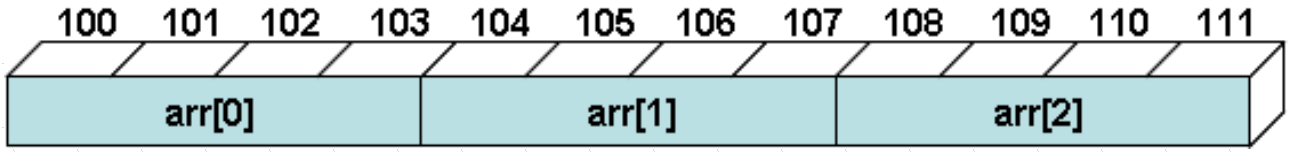


Δείκτες και Πίνακες (Ι)

- 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`



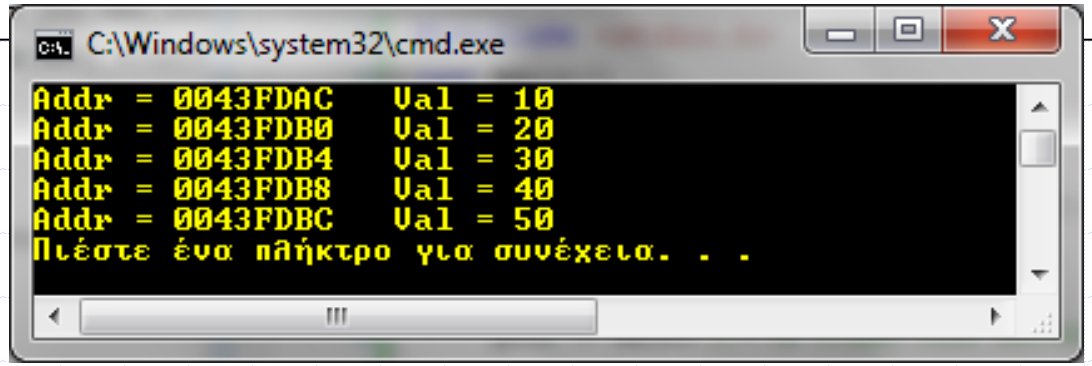
- Συμπερασματικά, οι εκφράσεις `arr` και `&arr[0]` είναι ισοδύναμες

Παράδειγμα (ΙΙ)

```c
#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 ???

```c
#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 (III)

- 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]