Δείκτες

Μνήμη Υπολογιστή

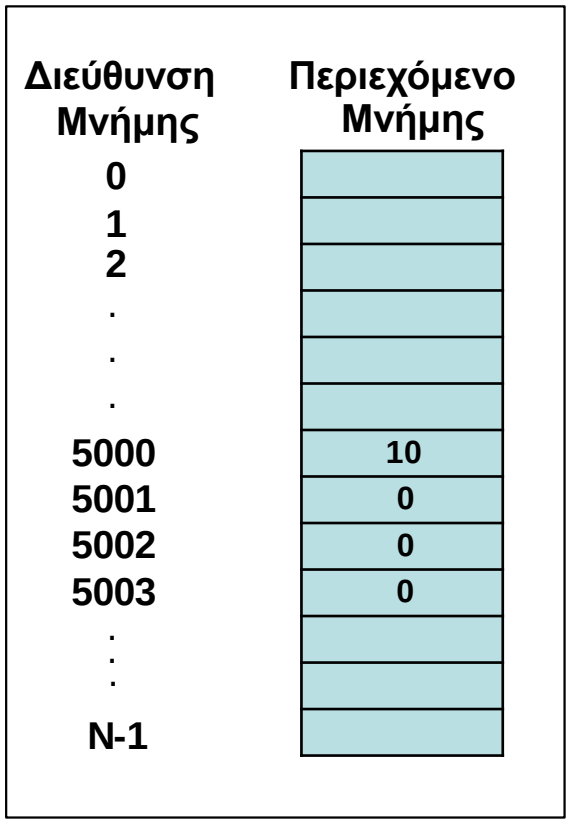
- Η μνήμη RAM (Random Access Memory) ενός υπολογιστή αποτελείται από πολλές χιλιάδες θέσεις αποθήκευσης δεδομένων που έχουν διαδοχική αρίθμηση
- Κάθε θέση ή κελί μνήμης προσδιορίζεται από μία μοναδική διεύθυνση
- Η διεύθυνση της κάθε θέσης μνήμης είναι ένας αύξοντας αριθμός με τιμή που κυμαίνεται από το 0 έως μία μέγιστη τιμή (η οποία εξαρτάται από το μέγεθος της διαθέσιμης μνήμης στον συγκεκριμένο υπολογιστή)
- 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;`
- 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
  - Σε αυτές τις θέσεις θα αποθηκεύεται η τιμή της μεταβλητής `a`
  - Στο διπλανό σχήμα θεωρούμε ότι η διεύθυνση της μεταβλητής `a` αρχίζει στη θέση `5000`
  - Η τιμή της `a` θα αποθηκευτεί στις θέσεις μνήμης από `5000` έως και `5003`
  - 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 = 10;` ο μεταγλωττιστής γνωρίζει ότι η διεύθυνση της `a` είναι η `5000` και θέτει το περιεχόμενό της ίσο με `10`, ξεκινώντας από την οκτάδα με την χαμηλότερη διεύθυνση



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`)

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()
{
      int* ptr; /* Δήλωση δείκτη προς ακέραια μεταβλητή. */
      int a;

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

      printf("Address = %p\n",ptr); /* Εμφάνιση της διεύθυνσης
μνήμης της μεταβλητής 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 (I)

- Υπενθυμίζεται ότι, όταν δηλώνεται μία μεταβλητή, τότε ο μεταγλωττιστής αναθέτει στη μεταβλητή μία τυχαία τιμή
- 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()
{
        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
- Επομένως, το επόμενο παράδειγμα εμφανίζει την τιμή `0`

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

Παρατηρήσεις

- 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()
{
        int* ptr;
        printf("Value: %d\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;` ή `double *ptr;` το αποτέλεσμα είναι `4`

Χρήση Δείκτη

- Για να αποκτήσουμε πρόσβαση στο περιεχόμενο κάποιας διεύθυνσης μνήμης με χρήση δείκτη, χρησιμοποιούμε τον τελεστή * πριν από το όνομα του δείκτη

- Π.χ.

```
#include <stdio.h>
int main()
{
	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;
}
```

```c
#include<stdio.h>
int main()
{
        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)

Παρατηρήσεις (Ι)

- Πριν χρησιμοποιηθεί κάποια μεταβλητή-δείκτης πρέπει 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

```
#include <stdio.h>
int main()
{
	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 `10`
- Άρα, με την εντολή `i = *ptr;` η τιμή του `i` θα γίνει ίση με `10`

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

        j = 10;
        ptr = &j;

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

Έξοδος: `Value = 10`

Παρατηρήσεις (ΙΙΙ)

- 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)

```c
#include <stdio.h>

int main()
{
   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 & are complements of each other\n");
   printf("&*ptr = %p\n", &*ptr);
   printf("*&ptr = %p\n", *&ptr);

return 0;
}
```

Έξοδος:

```
The address of a is 0028F958

The value of ptr is 0028F95  8

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

Showing that * and & are co  mplements of each other
&*ptr = 0028F958
*&ptr = 0028F958
```

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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()
{
        int i = 5;
        int* ptr;

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

Έξοδος: 25

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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()
{
        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()
{
        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

Ο δείκτης `void*`

- Ένας δείκτης σε τύπο `void*` είναι ένας «γενικός» δείκτης, με την έννοια ότι μπορεί να δείξει σε μία μεταβλητή οποιουδήποτε τύπου

- Για να ανακτήσουμε το περιεχόμενο μίας διεύθυνσης με χρήση ενός `void*` δείκτη πρέπει να προσαρμόσουμε τον τύπο του δείκτη στον τύπο της μεταβλητής, όπως φαίνεται στο παρακάτω πρόγραμμα

```c
#include <stdio.h>
int main()
{
        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`)

Αριθμητική Δεικτών

- Οι μόνοι τελεστές που μπορούν να χρησιμοποιηθούν στην αριθμητική δεικτών είναι οι:
  - ++
  - --
  - + και
  - –
- Αντίστοιχα, οι μαθηματικές πράξεις που επιτρέπονται με δείκτες είναι οι:
  - πρόσθεση ακεραίου σε δείκτη
  - αφαίρεση ακεραίου από δείκτη και
  - αφαίρεση δύο δεικτών που δείχνουν στον ίδιο τύπο δεδομένων
- Οι παραπάνω πράξεις έχουν ορισμένες ιδιαιτερότητες και για τον λόγο αυτό, απαιτείται ιδιαίτερη προσοχή

Δείκτες και Ακέραιοι (Αύξηση Δεικτών)

- Όταν προστίθεται ο αριθμός 1 σε έναν δείκτη, τότε η τιμή του δείκτη αυξάνει σύμφωνα με το μέγεθος του τύπου δεδομένων στον οποίο δείχνει

Π.χ. αν ο δείκτης έχει δηλωθεί σαν δείκτης σε:

- `char`, τότε η τιμή του δείκτη αυξάνεται κατά 1, αφού ο τύπος `char` δεσμεύει 1 byte από τη μνήμη
- `int` ή `float`, τότε η τιμή του δείκτη αυξάνεται κατά 4, αφού οι τύποι `int` και `float` δεσμεύουν 4 bytes από τη μνήμη
- `double`, τότε η τιμή του δείκτη αυξάνεται κατά 8, αφού ο τύπος `double` δεσμεύει 8 bytes από τη μνήμη

- Στη γενική περίπτωση, όταν προστίθεται ένας ακέραιος αριθμός n σε έναν δείκτη, τότε η τιμή του αυξάνεται κατά το γινόμενο:

n * μέγεθος του τύπου (στον οποίο δείχνει)

Παράδειγμα

- Τι εμφανίζει η τελευταία `printf()` του προγράμματος ???

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

	ptr = &i;
	printf("Address = %p\n",ptr); /* Έστω ότι εμφανίζεται το
1000. */
	ptr++;
	printf("Address = %p\n",ptr);
	return 0;
}
```

Έξοδος: `1004 (σε δεκαεξαδική μορφή)`

- **Αν είχα αντί για `ptr++;` Την εντολή `ptr+=5;` ???**

Έξοδος: `1020 (σε δεκαεξαδική μορφή)`

Δείκτες και Ακέραιοι (Μείωση Δεικτών)

- Ισχύει ακριβώς ό,τι ισχύει και κατά την αύξηση ενός δείκτη
- Δηλ. στη γενική περίπτωση, όταν αφαιρείται ένας ακέραιος αριθμός n από έναν δείκτη, τότε η τιμή του μειώνεται κατά το γινόμενο:

n * μέγεθος του τύπου (στον οποίο δείχνει)

Π.χ.

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

	ptr = &i;
	printf("Address = %p\n",ptr); /* Έστω ότι εμφανίζεται το
1000. */
	ptr-=5;
	printf("Address = %p\n",ptr);
	return 0;
}
```

Έξοδος: 980 (σε δεκαεξαδική μορφή)

Αφαίρεση Δεικτών

- Η αφαίρεση δεικτών, επιτρέπεται μόνο μεταξύ δεικτών, οι οποίοι δείχνουν σε ίδιο τύπο δεδομένων

- Το αποτέλεσμα της αφαίρεσης είναι ένας ακέραιος αριθμός που δεν δηλώνει πόσο απέχουν μεταξύ τους (δηλ. πόσες θέσεις μνήμης), αλλά πόσα στοιχεία του τύπου που δείχνουν οι δείκτες μεσολαβούν μεταξύ τους

Παράδειγμα

- Τι εμφανίζει η τελευταία `printf()` του προγράμματος ???

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

    ptr1 = &i;
    printf("Address = %p\n",ptr1); /* Έστω ότι εμφανίζεται το 1000. */
    ptr2 = &j;
    printf("Address = %p\n",ptr2); /* Έστω ότι εμφανίζεται το 1040. */

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

Έξοδος: 10 (και όχι 40)

Αφαίρεση Δεικτών

- Στη γενική περίπτωση, το αποτέλεσμα της αφαίρεσης δεικτών είναι το πηλίκο της διαίρεσης της αριθμητικής διαφοράς των δύο δεικτών με το μέγεθος του τύπου, στον οποίο δείχνουν
- Στο προηγούμενο παράδειγμα το αποτέλεσμα προκύπτει ως:

```
(1040 - 1000) / sizeof(int) = 40/4 = 10
```

- Προφανώς, αν αφαιρέτης είναι ο δείκτης με τη χαμηλότερη διεύθυνση στη μνήμη και αφαιρετέος ο δείκτης με την υψηλότερη διεύθυνση αντίστοιχα, τότε το αποτέλεσμα της αφαίρεσης των δεικτών θα είναι το ίδιο αριθμητικά, με αρνητικό όμως πρόσημο
- Στο προηγούμενο παράδειγμα αν στην τελευταία `printf()` ζητούσαμε τη διαφορά `ptr1-ptr2`, το αποτέλεσμα θα ήταν:

```
(1000 - 1040) / sizeof(int) = -40/4 = -10
```

Σύγκριση Δεικτών

- Η σύγκριση δεικτών έχει νόημα μόνο αν και οι δύο δείκτες δείχνουν σε μέλη της ίδιας δομής δεδομένων (π.χ. σε πίνακα)

- Οι δείκτες μπορούν να συγκριθούν με χρήση των τελεστών σύγκρισης ==, !=, >, <, >= και <=

- Π.χ., αν θέλουμε να ελέγξουμε αν δύο δείκτες `ptr1` και `ptr2` δείχνουν στην ίδια διεύθυνση μνήμης μπορούμε να γράψουμε:

```
if(ptr1 == ptr2)
```

ή αντίστοιχα:

```
if(ptr1 != ptr2)
```

- Π.χ., αν θέλουμε να ελέγξουμε αν ο δείκτης `ptr1` δείχνει σε κάποια μεταβλητή με «μεγαλύτερη» διεύθυνση από την αντίστοιχη που δείχνει ο δείκτης `ptr2` μπορούμε να γράψουμε:

```
if(ptr1 > ptr2)
```

Παρατηρήσεις

- Ο μεταγλωττιστής, κατά την αφαίρεση και την πρόσθεση ενός ακέραιου αριθμού σε έναν δείκτη, αλλάζει την τιμή του δείκτη με βάση το μέγεθος του τύπου δεδομένων που δείχνει ο δείκτης

Εκτός από την πρόσθεση ή την αφαίρεση ακεραίων αριθμών σε/από έναν δείκτη, τη σύγκριση και την αφαίρεση δεικτών ίδιου τύπου, καμία άλλη αριθμητική πράξη δεν επιτρέπεται

- Π.χ. οι εντολές:

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

δεν είναι επιτρεπτές εκφράσεις ακόμα κι αν οι αυτοί δείκτες δείχνουν στον ίδιο τύπο δεδομένων

Παραδείγματα (Ι)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

        ptr = &i;
        printf("Value = %d\n",*ptr);

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

Έξοδος: Το τυχαίο περιεχόμενο των επόμενων 4 bytes μετά τη θέση της μεταβλητής i στη μνήμη

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
        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()
{
        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()
{
        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()
{
        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()
{
        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`