Προγραμματισμός Ι

Δείκτες

Δημήτρης Μιχαήλ

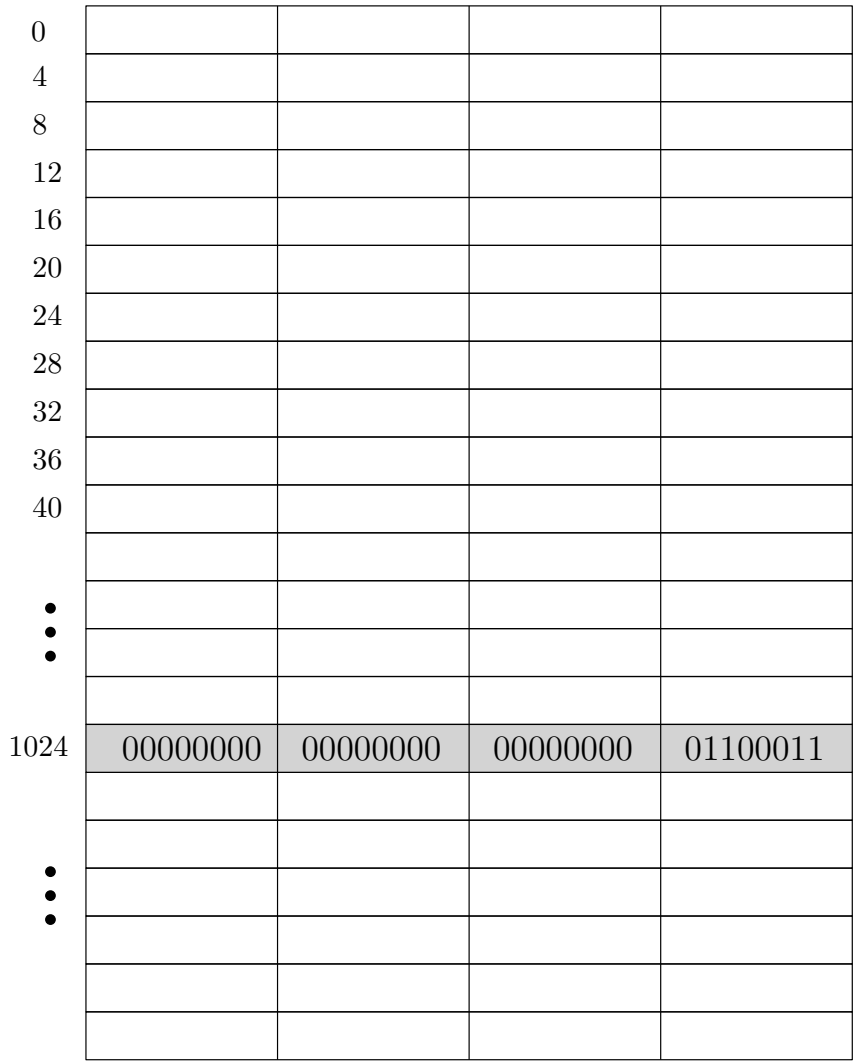
Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Τι είναι ο δείκτης

Ένας δείκτης είναι μια μεταβλητή που περιέχει μια διεύθυνση μνήμης.

Θυμηθείτε πως η μνήμη μοιάζει με έναν μονοδιάστατο πίνακα.

Στο παράδειγμα που βλέπετε η θέση μνήμης 1024 περιέχει την τιμή 99.



Δήλωση Δείκτη σε C

Το ”*” χρησιμοποιείται για να δηλώσει δείκτη:

```
int *p;
```

Η παραπάνω δήλωση δημιουργεί μια μεταβλητή τύπου δείκτη σε ακέραιο (pointer to int).

Δήλωση Δείκτη σε C

Το ”*” χρησιμοποιείται για να δηλώσει δείκτη:

```
int *p;
```

Η παραπάνω δήλωση δημιουργεί μια μεταβλητή τύπου δείκτη σε ακέραιο (pointer to int).

Το όνομα αυτής της μεταβλητής είναι `p`.

Δήλωση Δείκτη σε C

Το ”*” χρησιμοποιείται για να δηλώσει δείκτη:

```
int *p;
```

Η παραπάνω δήλωση δημιουργεί μια μεταβλητή τύπου δείκτη σε ακέραιο (pointer to int).

Το όνομα αυτής της μεταβλητής είναι `p`.

Η τιμή της μεταβλητής είναι μία διεύθυνση μνήμης.

Ανάθεση Τιμής σε Δείκτη

Όπως όλες τις άλλες μεταβλητές στην C πρέπει να αρχικοποιήσουμε έναν δείκτη.

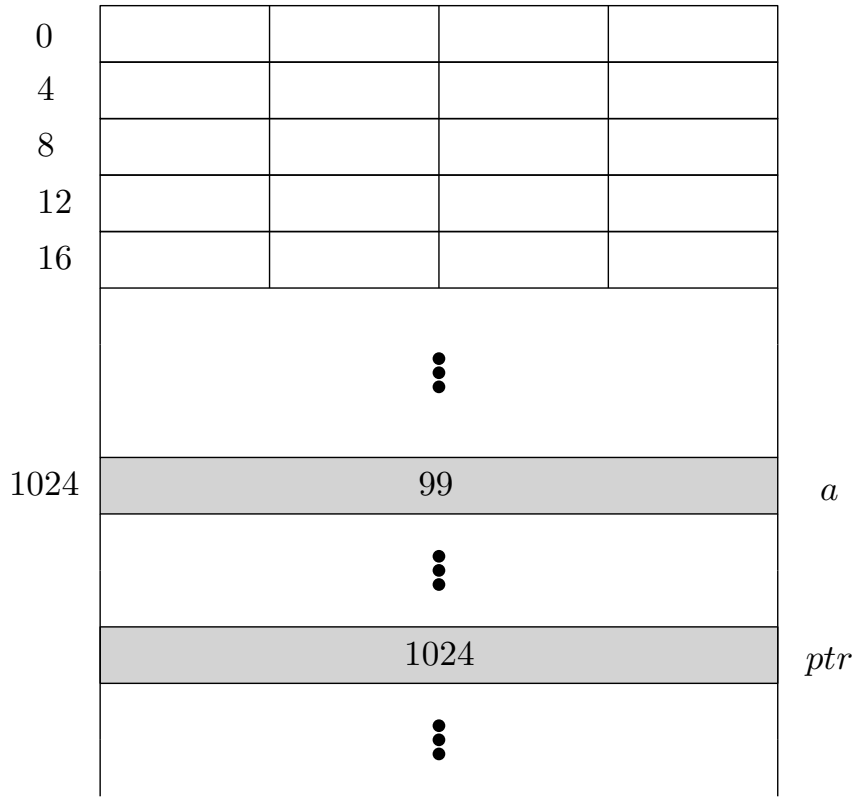
Για να βρούμε την διεύθυνση μνήμης μιας μεταβλητής χρησιμοποιούμε τον τελεστή &.

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

    a = 99;
    ptr = &a;

    return 0;
}
```

Ανάθεση Τιμής σε Δείκτη

Για να βρούμε την διεύθυνση μνήμης μιας μεταβλητής χρησιμοποιούμε τον τελεστή &.

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

    a = 99;
    ptr = &a;

    return 0;
}
```



Με αυτόν τον τρόπο ο δείκτης `ptr` "**δείχνει**" στην μεταβλητή `a`.

Η τιμή της μεταβλητής `ptr` είναι η διεύθυνση μνήμης της μεταβλητής `a`.

Η Τιμή `0` και η Τιμή `NULL`

Για να μην δείχνει ένας δείκτης κάπου πρέπει να του αναθέσουμε την τιμή `0`. Η βιβλιοθήκη της C ορίζει μια σταθερά με όνομα `NULL` για να συμβολίσει το γεγονός πως ένας δείκτης δεν δείχνει πουθενά.

```
int main() {
    int *ptr = 0;

    // ...

    return 0;
}
```

ή

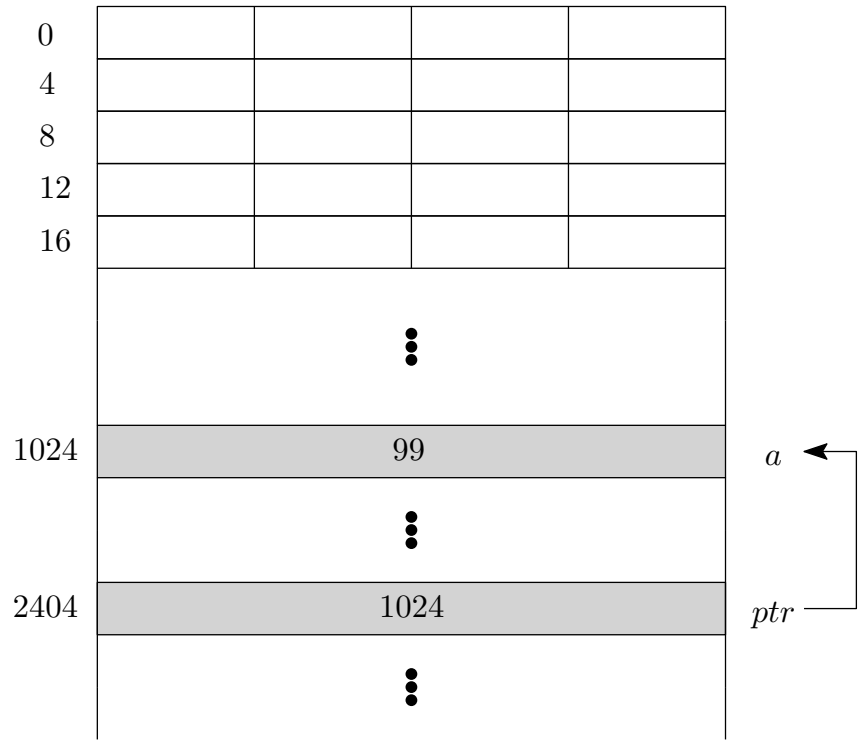
```
#include <stdio.h>

int main() {
    int *ptr = NULL;

    // ...

    return 0;
}
```

Τελεστής Έμμεσης Αναφοράς

Indirection/dereferencing operator

Μπορούμε να ακολουθήσουμε ένα δείκτη με την χρήση του τελεστή έμμεσης αναφοράς, "*";

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

    a = 99;
    ptr = &a;

    printf("%d\n", *ptr);

    return 0;
}
```



Η έκφραση `*ptr` μας δίνει την μεταβλητή στην θέση μνήμης που δείχνει ο δείκτης `ptr`.

Στην προκειμένη περίπτωση είναι η μεταβλητή `a`.

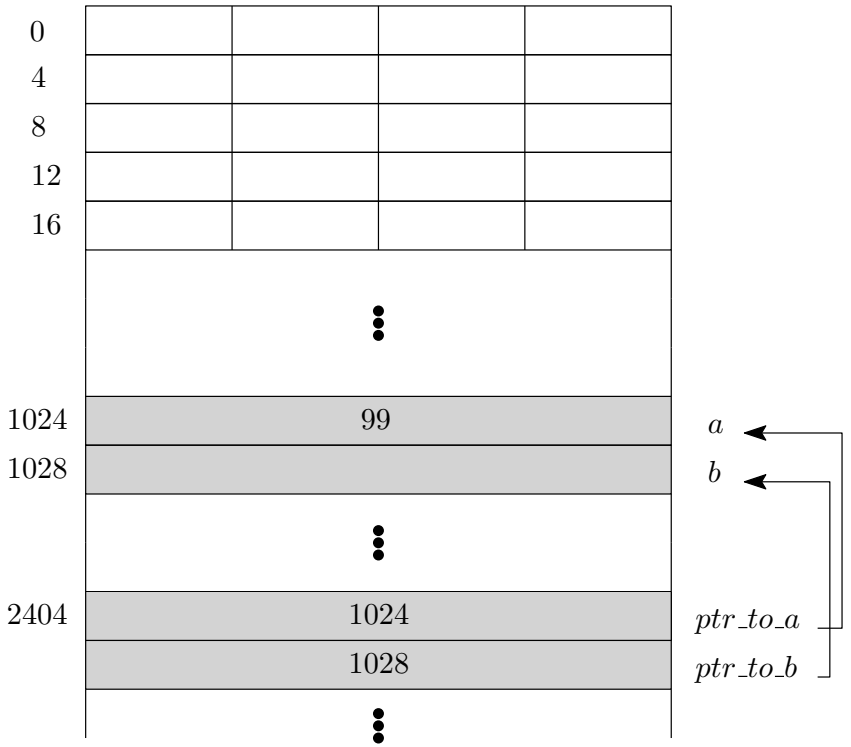
Προτεραιότητα Τελεστών

1. **παρενθέσεις:** () [] `expr`++ `expr`−−
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι τελεστές:** + − ++`expr` −−`expr` ! * &
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** * / %
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** + −
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** < > <= >=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** == !=
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **λογικό AND:** && Από αριστερά προς τα δεξιά.
8. **λογικό OR:** || Από αριστερά προς τα δεξιά.
9. **εκχώρησης:** = += −= *= /= %=
   Από δεξιά προς τα αριστερά.

Τελεστής Έμμεσης Αναφοράς

Indirection/dereferencing operator

```
1  int main() {
2      int a, b ;
3      int *ptr_to_a, *ptr_to_b;
4
5      a = 99;
6      ptr_to_a = &a;
7      ptr_to_b = &b;
8
9      *ptr_to_b = *ptr_t_a;
10     printf("%d\n", *ptr_to_b);
11
12     return 0;
13 }
```



Η γραμμή 10 είναι ουσιαστικά η γραμμή:

```
10 b = a;
```

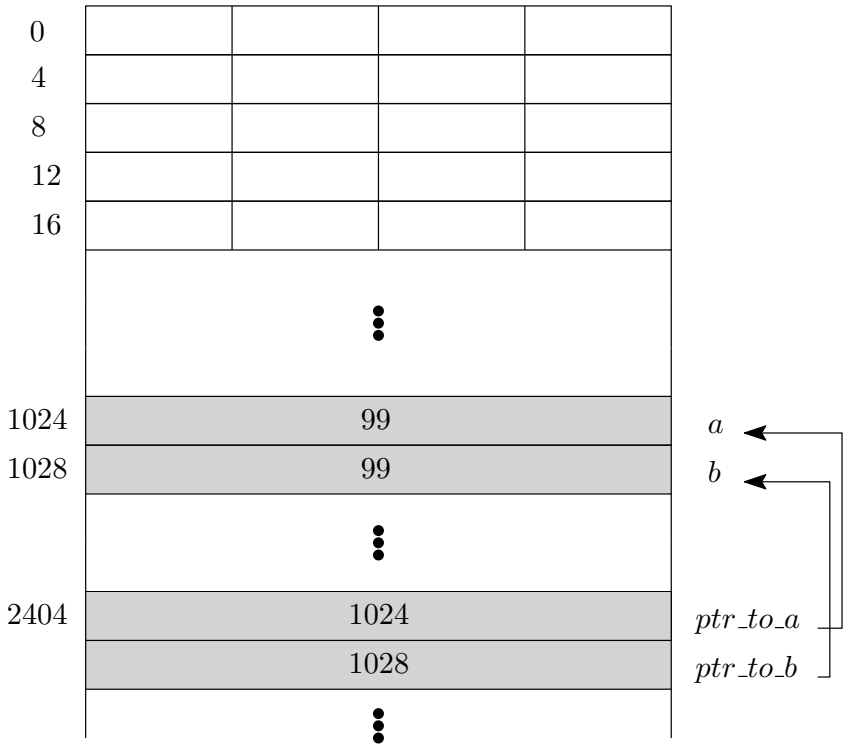
ενώ η 11 είναι

```
12 printf("%d\n", b);
```

Τελεστής Έμμεσης Αναφοράς

Indirection/dereferencing operator

```
1  int main() {
2      int a, b ;
3      int *ptr_to_a, *ptr_to_b;
4
5      a = 99;
6      ptr_to_a = &a;
7      ptr_to_b = &b;
8
9      *ptr_to_b = *ptr_t_a;
10     printf("%d\n", *ptr_to_b);
11
12     return 0;
13 }
```



Η γραμμή 10 είναι ουσιαστικά η γραμμή:

```
10 b = a;
```

ενώ η 11 είναι

```
12 printf("%d\n", b);
```

Δείκτες Άλλων Τύπων

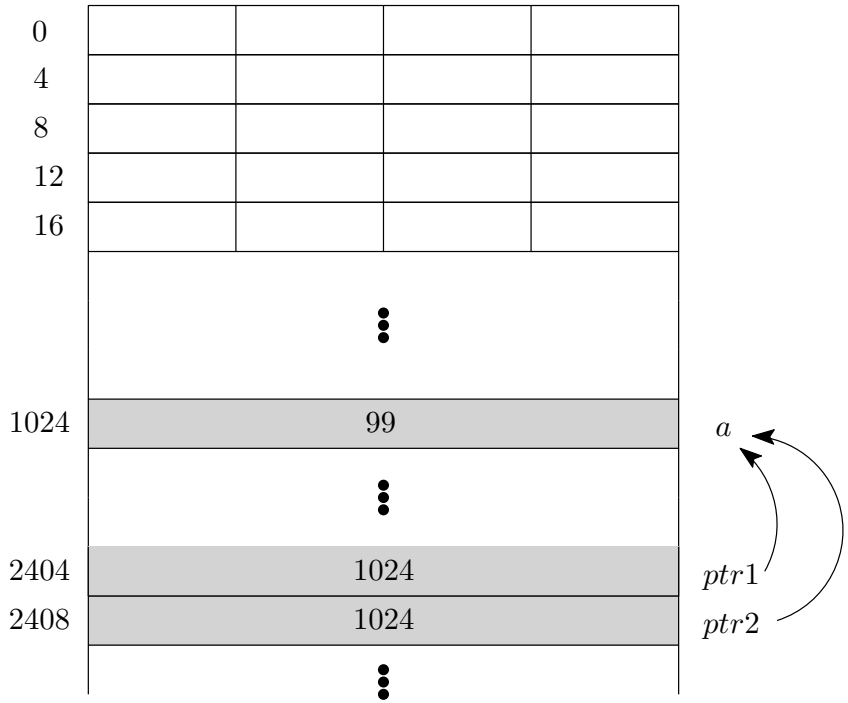
Στην γλώσσα C μπορούμε να έχουν δείκτες σε οποιαδήποτε μεταβλητή ανεξάρτητα από τον τύπο της. Πρέπει όμως να δηλώσουμε τον δείκτη ανάλογα.

```
1  #include <stdio.h>
2
3  int main() {
4      double pi;
5      double *ptr = &pi;
6
7      *ptr = 3.14159265;
8      printf("%lf\n", pi);
9
10     return 0;
11 }
```

Αντίστοιχα και με τους υπόλοιπους τύπους.

Πολλοί Δείκτες, σε μια Μεταβλητή

```
1  int main() {
2      int a;
3      int *ptr1, *ptr2;
4
5      a = 99;
6      ptr1 = &a;
7      ptr2 = &a;
8
9      (*ptr1)++;
10     (*ptr2)++;
11
12     printf("%d\n", a);
13     return 0;
14 }
```

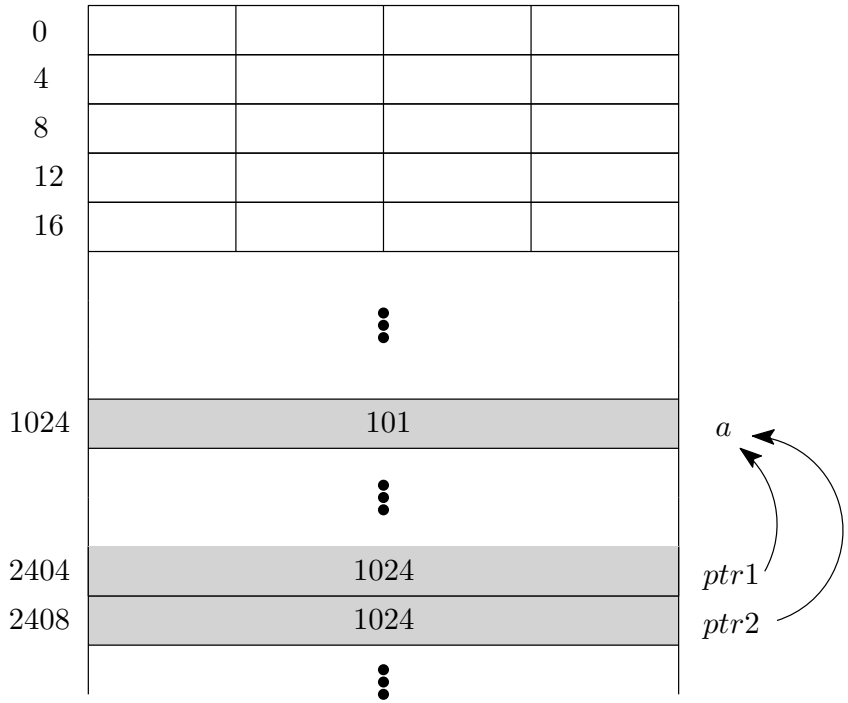


Η γραμμές 10 και 11 είναι ουσιαστικά η γραμμή:

```
a++;
```

Πολλοί Δείκτες, σε μια Μεταβλητή

```
1  int main() {
2      int a;
3      int *ptr1, *ptr2;
4
5      a = 99;
6      ptr1 = &a;
7      ptr2 = &a;
8
9      (*ptr1)++;
10     (*ptr2)++;
11
12     printf("%d\n", a);
13     return 0;
14 }
```



Η γραμμές 10 και 11 είναι ουσιαστικά η γραμμή:

```
a++;
```

Το πρόγραμμα θα εκτυπώσει 101.

Εκτύπωση της Τιμή Ενός Δείκτη

Η τιμή ενός δείκτη είναι μια διεύθυνση μνήμης. Για να τυπώσουμε την τιμή ενός δείκτη η `printf()` μας παρέχει την επιλογή `%p`.

```c
#include <stdio.h>

int main() {
    int a = 5;
    int *ptr = &a;

    printf("address of a is %p\n", ptr);

    return 0;
}
```

Στο σύστημα του ομιλητή το πρόγραμμα τυπώνει:

```
address of a is 0x7fff5bc2d5cc
```

Προσοχή όταν ορίζετε πολλούς δείκτες στην ίδια γραμμή.

```
int* ptr1, ptr2, ptr3;
```

Το αστεράκι ανήκει ουσιαστικά στο όνομα και όχι στον τύπο.

Η παραπάνω γραμμή ορίζει 1 δείκτη σε ακέραιο με όνομα `ptr1` και δύο ακεραίους με ονόματα `ptr2` και `ptr3`.

Για να ορίσουμε 3 δείκτες σε ακέραιο πρέπει να γράψουμε:

```
int *ptr1, *ptr2, *ptr3;
```

Συχνά Λάθη

Δείκτης και το Αντικείμενο

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 10, b = 20;
5      int *p1, *p2;
6
7      p1 = &a;
8      p2 = &b;
9
10     *p1 = *p2;
11     p1 = p2;
12     *p1 = *p2;
13
14     return 0;
15 }
```

- η γραμμή 11 είναι ισοδύναμη με `a = b`.
- η γραμμή 12 αναθέτει την διεύθυνση του `b` στον δείκτη `p1`. Οι δύο δείκτες δείχνουν στο `b`.
- η γραμμή 13 είναι ισοδύναμη με `b = b`.

Συχνά Λάθη

Προτεραιότητα Τελεστών

```
1  #include <stdio.h>
2
3  int main() {
4      int x, *p;
5
6      p = &x;          /* initialise pointer */
7      *p = 0;          /* set x to zero */
8      printf("x is %d\n", x);
9      printf("*p is %d\n", *p);
10
11     *p += 1;         /* increment what p points to */
12     printf("x is %d\n", x);
13
14     (*p)++;          /* increment what p points to */
15     printf("x is %d\n", x);
16
17     return 0;
18 }
```

Προσοχή η γραμμή 14 θέλει παρενθέσεις, αφού ο τελεστής postfix ++ έχει μεγαλύτερη προτεραιότητα από τον τελεστή *.

Κλήση Συνάρτησης Μέσω Αναφοράς

Με την χρήση δεικτών μπορούμε να κάνουμε κλήση μέσω αναφοράς στην C.

Θυμηθείτε πως στην C οι παράμετροι μιας συνάρτησης αντιγράφονται σε προσωρινές τοπικές μεταβλητές πριν κληθεί η συνάρτηση.

Θα κάνουμε μια μικρή παρένθεση και θα εξηγήσουμε τον τρόπο λειτουργίας των κλήσεων συναρτήσεων.

Κλήση Συναρτήσεων

Stack Frame

Για να μπορούμε να υποστηρίζουμε κλήση συναρτήσεων ο μεταγλωττιστής παράγει κώδικα που χρησιμοποιεί την λεγόμενη στοίβα κλήσεων.

Η στοίβα κλήσεων είναι υπεύθυνη για να αποθηκεύει διάφορες πληροφορίες όπως:

- την διεύθυνση επιστροφής μετά την συνάρτηση,
- τις παραμέτρους της συνάρτησης,
- τις τοπικές μεταβλητές της συνάρτησης,
- κ.τ.λ

Κλήση Συναρτήσεων

Stack Frame

Κάθε φορά που καλείται μια συνάρτηση προστίθεται στην κορυφή της στοίβας μια εγγραφή που περιέχει

- τις τοπικές μεταβλητές της συνάρτησης
- την διεύθυνση επιστροφής
- τις παραμέτρους της συνάρτησης

Μόλις τελειώσει η εκτέλεση της συνάρτησης αφαιρείται η τελευταία εγγραφή, και άρα καταστρέφονται όλες οι τοπικές μεταβλητές και παράμετροι.

Ο κώδικας δημιουργίας και καταστροφής των εγγραφών αυτών παράγεται από τον μεταγλωττιστή και προστίθεται σε κάθε σημείο του προγράμματος που γίνεται κλήση συνάρτησης.

Κλήση Συνάρτησης Μέσω Αναφοράς

Για να υλοποιήσουμε κλήση μέσω αναφοράς στην C χρησιμοποιούμε δείκτες.

- Δίνουμε ως παράμετρο σε μια συνάρτηση ένα δείκτη στην μεταβλητή που θέλουμε να περάσουμε στην συνάρτηση.
- Μέσω του δείκτη αλλάζουμε την τιμή της μεταβλητής που θέλουμε
- Μόλις επιστρέψει η συνάρτηση η παράμετρος (δείκτης) θα καταστραφεί αλλά η μεταβλητή μας θα έχει αλλάξει κανονικά τιμή.

Κλήση Συνάρτησης Μέσω Αναφοράς

```c
#include <stdio.h>

void increase(int *x) {
    (*x)++;
}

int main() {
    int a = 1;

    increase(&a);
    printf("%d\n", a);

    return 0;
}
```

Το παραπάνω πρόγραμμα τυπώνει 2.

- Η μεταβλητή `x` δείχνει στην μεταβλητή που θέλουμε να περάσουμε ως παράμετρο.
- Μέσα στην συνάρτηση αλλάζουμε την μεταβλητή μας χρησιμοποιώντας `*x`.
- Με το τέλος της συνάρτησης η τοπική μεταβλητή `x` θα καταστραφεί, αλλά η μεταβλητή `a` θα έχει αλλάξει τιμή.

Κλήση Συνάρτησης Μέσω Αναφοράς

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main() {
    int x = 1,
        y = 2;

    swap(&x, &y);

    printf("x = %d\n", x);
    printf("y = %d\n", y);

    return 0;
}
```

Το πρόγραμμα τυπώνει

```
x = 2
y = 1
```

Δείκτες και `const`

Θυμηθείτε πως με την δεσμευμένη λέξη `const` λέμε στον μεταγλωττιστή πως κάτι δεν πρέπει να αλλάξει.

Επειδή όταν ασχολούμαστε με δείκτες έχουμε ουσιαστικά 2 αντικείμενα:

1. ένα δείκτη και
2. εκεί που δείχνει ο δείκτης

η χρήση του `const` είναι λίγο πιο πολύπλοκη.

Δείκτες και `const`

Υπάρχουν 3 χρήσεις:

1. `const int *ptr;`

   Δείκτης σε σταθερό ακέραιο. Ο δείκτης μπορεί να αλλάξει τιμή, ο ακέραιος όχι.

2. `int *const ptr;`

   Σταθερός δείκτης σε ακέραιο. Ο δείκτης δεν μπορεί να αλλάξει τιμή, ο ακέραιος μπορεί.

3. `const int *const ptr;`

   Σταθερός δείκτης σε σταθερό ακέραιο. Ούτε ο δείκτης αλλά ούτε και ο ακέραιος μπορεί να αλλάξει τιμή.

Είναι εύκολο να διακρίνεται τις περιπτώσεις κοιτώντας δεξιά και αριστερά του αστερίσκου για την δεσμευμένη λέξη `const`.

Δείκτες και const

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      const int y = 5;
5      const int x = 3;
6
7      const int *ptr = &y;
8
9      (*ptr)++;    // NO!
10
11     ptr = &x; // ok
12 }
```

Ο μεταγλωττιστής δεν μας επιτρέπει να γράψουμε την γραμμή 9.

Δείκτες και const

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      int y = 5;
5      int x = 3;
6
7      int *const ptr = &y;
8
9      (*ptr)++;   // ok
10
11     ptr = &x;   // NO!
12 }
```

Ο μεταγλωττιστής δεν μας επιτρέπει να γράψουμε την γραμμή 11.

Δείκτες και const

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      const int y = 5;
5      const int x = 3;
6
7      const int *const ptr = &y;
8
9      (*ptr)++;   // NO!
10
11     ptr = &x;   // NO!
12 }
```

Ο μεταγλωττιστής δεν μας επιτρέπει να γράψουμε τις γραμμές 9 και 11.

Λάθη και casts

Προσοχή με τα casts.

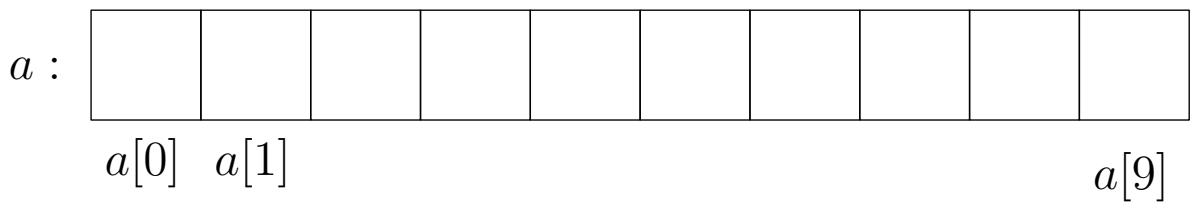
```c
#include <stdio.h>

int main() {
    int i;
    const int ci = 123;

    const int *cpi; // pointer to const
    int *ncpi;

    cpi = &ci;
    ncpi = &i;

    cpi = ncpi;      // allowed

    // this needs a cast - usually BIG MISTAKE
    ncpi = (int *)cpi;

    *ncpi = 0; // UNDEFINED BEHAVIOR !!
}
```

Δείκτες και Πίνακες

Οι πίνακες και οι δείκτες είναι πολύ στενά συνδεδεμένοι στην C.

Για να δηλώσουμε ένα πίνακα ακεραίων μεγέθους 10 με όνομα `a` γράφουμε:

```
int a[10];
```



Δείκτες και Πίνακες

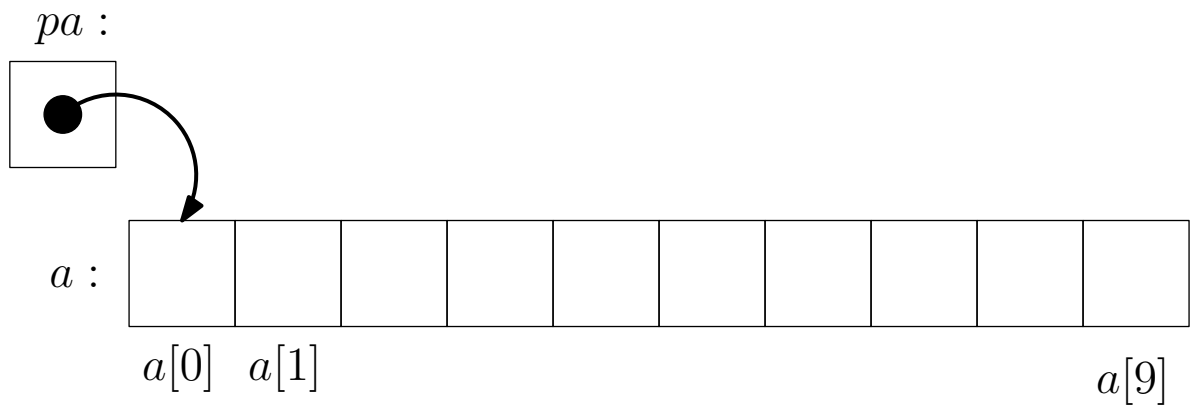
Εάν ο `pa` είναι ένας δείκτης σε ακέραιο

```
int *pa;
```

τότε η ανάθεση

```
pa = &a[0];
```

βάζει τον `pa` να δείχνει στο μηδενικό στοιχείο του πίνακα `a`.



Δείκτες και Πίνακες

Από τον ορισμό, η τιμή μιας μεταβλητής ή έκφρασης τύπου πίνακα είναι η διεύθυνση του μηδενικού στοιχείου του πίνακα. Έτσι μετά την ανάθεση

```
pa = &a[0];
```

ο `pa` και ο `a` έχουν ακριβώς την ίδια τιμή.

Επειδή το όνομα ενός πίνακα είναι συνώνυμο με την τοποθεσία του μηδενικού του στοιχείου, η τελευταία ανάθεση μπορεί να γραφεί και ως

```
pa = a;
```

Δείκτες και Πίνακες

Επειδή το όνομα ενός πίνακα είναι συνώνυμο με την τοποθεσία του μηδενικού του στοιχείου, μπορούμε να κάνουμε και dereference.

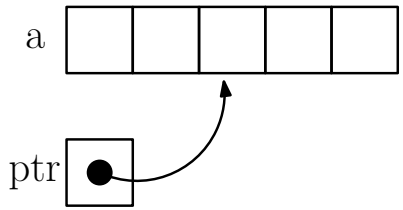
```
1  #include <stdio.h>
2
3  int main() {
4      int a[] = {9, 8, 7, 6, 5, 4, 3, 2, 1};
5
6      printf("%d\n", *a);
7      *a = 11;
8      printf("%d\n", a[0]);
9
10     return 0;
11 }
```

Το παραπάνω πρόγραμμα τυπώνει

```
9
11
```

Δείκτες και Πίνακες

Ένας δείκτης μπορεί να δείχνει σε μια μεταβλητή ενός πίνακα.




```c
#include <stdio.h>

int main() {
    int a[] = {9, 8, 7, 6, 5};
    int *ptr = &a[2];

    printf("%d\n", *ptr);

    return 0;
}
```

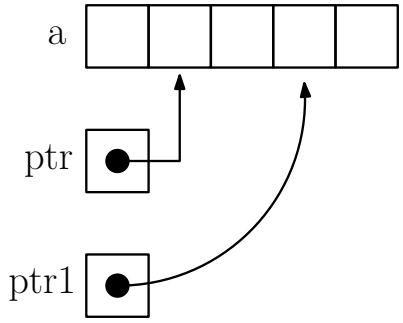
Το παραπάνω πρόγραμμα τυπώνει

```
7
```

Αριθμητική Δεικτών

Έχοντας ένα δείκτη σε ένα στοιχείο ενός πίνακα, μπορούμε να κάνουμε **αριθμητική δεικτών**.

```c
#include <stdio.h>

int main() {
    int a[] = {9, 8, 7, 6, 5};

    int *ptr  = &a[1];
    int *ptr1 = ptr + 2;

    printf("%d\n", *ptr);
    printf("%d\n", *ptr1);

    return 0;
}
```



Το παραπάνω πρόγραμμα τυπώνει

```
8
6
```

Αριθμητική Δεικτών

Στην αριθμητική δεικτών μπορούμε να προσθέσουμε ή να αφαιρέσουμε.

Όταν γράφουμε `p`+1 όπου `p` είναι ένας δείκτης, το αποτέλεσμα είναι ένας δείκτης που δείχνει μια θέση μετά την θέση που δείχνει ο `p`.

Πόσο μακρύτερα δείχνει ο `p`+1 από τον `p` εξαρτάται από τον τύπο δεδομένων που δείχνει ο `p`.

Εαν ο `p` είναι ένας δείκτης σε ακεραίους τότε η θέση μνήμης `p`+1 θα είναι 4 bytes μετά την θέση `p`. Εαν όμως ο `p` δείχνει σε char τότε θα είναι μόλις 1 byte μετά.

Δείκτες και Πίνακες

Όταν γράφουμε `a[i]` στην C, γίνεται η μετατροπή σε `*(a+i)`. Οι δύο αυτές μορφές είναι ισοδύναμες.

Χρησιμοποιώντας τον τελεστή `&` και στις δύο μορφές προκύπτει πως τα

- `&a[i]`
- `a+i`

είναι επίσης ισοδύναμα. `a+i` είναι η διεύθυνση του *i* στοιχείου μετά το *a*.

Παρόμοια έχοντας ένα δείκτη `pa = &a[0]` η έκφραση `pa[2]` είναι ισοδύναμη με `*(pa + 2)`. Άρα μπορούμε να χρησιμοποιήσουμε ένα δείκτη με τον τελεστή `[]`.

Δείκτες και Πίνακες

Επειδή γράφοντας `a[i]` στην C, γίνεται η μετατροπή σε `*(a+i)`, το ίδιο γίνεται και γράφοντας `i[a]`.

```
a[i]
*((a) + (i))      (ορισμός)
*((i) + (a))      (αντιμεταθετικότητα πρόσθεσης)
i[a]              (ορισμός)
```

Αυτό το γεγονός επιτρέπει να γράφουμε ”περίεργο” κώδικα σε διαγωνισμούς ”περίεργου” κώδικα C.

Για παράδειγμα η έκφραση

```
5["abcdef"]
```

είναι απόλυτα σωστή και είναι ουσιαστικά ο χαρακτήρας `'f'`.

Αριθμητική Δεικτών

Αντιγραφή Συμβολοσειρών

```
1  void mystrcpy(char *dest, const char *src) {
2      char *dp = &dest[0],
3           *sp = &src[0];
4
5      while(*sp != '\0')
6          *dp++ = *sp++;
7
8      *dp = '\0';
9  }
```

Η έκφραση `*dp++` κάνει dereference τον δείκτη `dp` και μετά κάνει `dp=dp+1`.
Για να αυξήσει την μεταβλητή που δείχνει ο `dp` πρέπει να γραφεί ως `(*dp)++`.

Αριθμητική Δεικτών

Αντιγραφή Συμβολοσειρών

Όταν κάνουμε αριθμητική δεικτών πρέπει να προσέχουμε να μην ξεφύγουμε εκτός πίνακα.

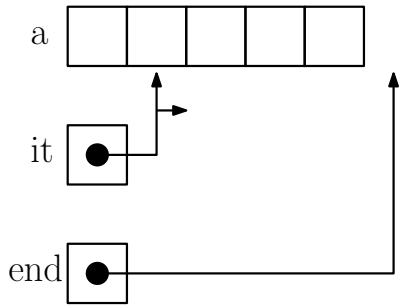
```
1  #include <stdio.h>
2
3  int main() {
4      int a[] = {1, 2, 3, 4, 5};
5      int *p = &a[2];
6
7      printf("%d\n", *(p + 4));
8
9      return 0;
10 }
```

Επειδή δεν γνωρίζουμε τι περιέχει και σε ποιον ανήκει η μνήμη εκτός του πίνακα, μπορεί να συμβεί οτιδήποτε.

Αριθμητική Δεικτών

Μια μικρή εξαίρεση

Όταν κάνουμε αριθμητική δεικτών μπορούμε να χρησιμοποιήσουμε για σύγκριση (όχι όμως και να κάνουμε dereference) την θέση που ακολουθεί αμέσως μετά το τέλος ενός πίνακα.

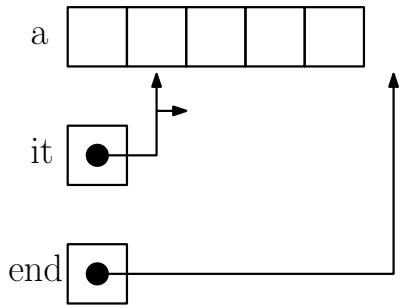



```
1  #include <stdio.h>
2
3  int main() {
4      int a[] = {1, 2, 3, 4, 5};
5
6      int *it  = &a[0],
7          *end = it + 5;
8
9      while(it < end) {
10         printf("%d\n", *it++);
11     }
12
13     return 0;
14 }
```

Αριθμητική Δεικτών

Μια μικρή εξαίρεση

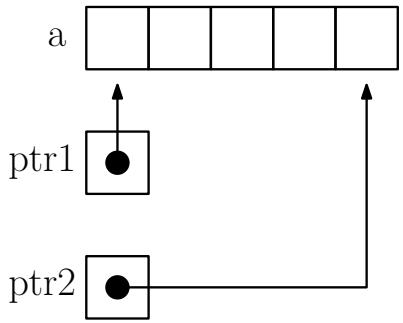
```
1  #include <stdio.h>
2
3  int main() {
4      int a[] = {1, 2, 3, 4, 5};
5
6      int *it  = &a[0],
7          *end = it + 5;
8
9      while(it < end) {
10         printf("%d\n", *it++);
11     }
12
13     return 0;
14 }
```



Στον παραπάνω κώδικα που τυπώνει όλα τα στοιχεία ενός πίνακα, ο δείκτης `end` δείχνει αμέσως μετά το τέλος του πίνακα. Είναι επιτρεπτό να κάνουμε την σύγκριση `it < end`. Ο κώδικας είναι σωστός αρκεί να μην κάνουμε ποτέ `*end`.

Αριθμητική Δεικτών

Μπορούμε να κάνουμε και αφαίρεση δεικτών.




```
1  #include <stdio.h>
2
3  int main() {
4      int a[] = { 1, 2, 3, 4, 5 };
5      int *ptr1 = &a[0],
6          *ptr2 = &a[4];
7
8      printf("%d", ptr2 - ptr1);
9
10     return 0;
11 }
```

Τι εκτυπώνει ο παραπάνω κώδικας;

Δείκτες και Πίνακες

Υπάρχει μια διαφορά μεταξύ του ονόματος ενός πίνακα και ενός δείκτη που πρέπει πάντα να θυμόμαστε.

Ένας δείκτης είναι μια μεταβλητή και άρα εκφράσεις όπως

```
pa = a;
```

ή

```
pa++;
```

είναι σωστές.

Το όνομα ενός πίνακα όμως δεν είναι μεταβλητή και άρα οι αντίστοιχες

```
a = pa;
```

ή

```
a++;
```

δεν επιτρέπονται.

Δείκτες, Πίνακες και Συναρτήσεις

Όταν ένας πίνακας δίνεται σε μια συνάρτηση, ουσιαστικά δίνεται η θέση του πρώτου στοιχείου του πίνακα. Μέσα στην συνάρτηση, η παράμετρος είναι τοπική μεταβλητή, και άρα είναι ένας δείκτης.

Οι παρακάτω δύο δηλώσεις είναι λοιπόν ισοδύναμες

```
int strlen(char s[]);
```

και

```
int strlen(char *s);
```

και καλό είναι να χρησιμοποιούμε τον δεύτερο τρόπο μιας και εξηγεί το γεγονός πως η παράμετρος `s` είναι δείκτης και όχι πίνακας.

Δείκτες, Πίνακες και Συναρτήσεις

Είναι δυνατό να περάσουμε *μέρος ενός πίνακα* σε μια συνάρτηση, περνώντας ένα δείκτη στην αρχή του υποπίνακα.

Για παράδειγμα εάν `a` είναι ένας πίνακας, τότε

```
f(&a[2])
```

και

```
f(a+2)
```

περνάνε στην συνάρτηση την διεύθυνση του υποπίνακα που ξεκινά στο `a`[2].

Δείκτες, Πίνακες και Συναρτήσεις

Η συνάρτηση μπορεί να γραφεί ως

```
f(int arr[]) { ... }
```

ή

```
f(int *arr) { ... }
```

Η συνάρτηση `f` δεν διαφοροποιείται επειδή η παράμετρος αναφέρεται σε κομμάτι μεγαλύτερου πίνακα.

Εάν κάποιος είναι σίγουρος πως τα στοιχεία υπάρχουν, μπορεί να χρησιμοποιήσει και αρνητικές διευθύνσεις όπως `p`$[-1]$ ή `p`$[-2]$. Είναι φυσικά λάθος να βγεί κάποιος εκτός πίνακα.

Πίνακες Δεικτών

Όπως μπορούμε να έχουμε πίνακες διαφόρων τύπων, με τον ίδιο τρόπο μπορούμε να έχουμε και πίνακες δεικτών. Η σύνταξη είναι λίγο πιο πολύπλοκη.

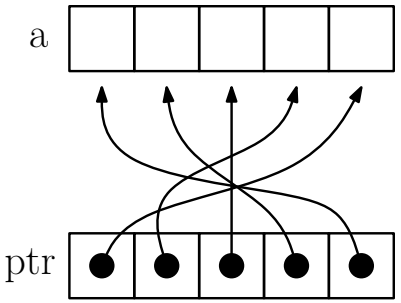
Η παρακάτω εντολή δηλώνει ένα πίνακα με 10 δείκτες σε ακεραίους.

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

Πίνακες Δεικτών

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      int i;
5      int a[5] = {0, 1, 2, 3, 4};
6      int *ptr[5];
7
8      for(i = 0; i < 5; i++)
9          ptr[i] = &a[4-i];
10
11     for(i = 0; i < 5; i++)
12         printf("%2d", *ptr[i]);
13
14     return 0;
15 }
```



Ο κώδικας τυπώνει

```
4  3  2  1  0
```

Δείκτες σε Συναρτήσεις

Η C εκτός από δείκτες σε τύπους μας επιτρέπει να έχουμε και δείκτες σε συναρτήσεις. Αυτό είναι επιτρεπτό αφού και οι συναρτήσεις είναι κάπου φορτωμένες στην μνήμη.

Για να δηλώσουμε ένα δείκτη σε συνάρτηση, ο τύπος του δείκτη πρέπει να περιγράφει τις παραμέτρους και τον τύπο επιστροφής της συνάρτησης.

```
int (*ptr)(int, double);
```

Η παραπάνω εντολή δηλώνει ένα δείκτη σε συνάρτηση που πέρνει ως παραμέτρους έναν `int` και έναν `double` και επιστρέφει `int`. Το όνομα του δείκτη είναι `ptr`.

Δείκτες σε Συναρτήσεις

Παρακάτω δηλώνουμε ένα δείκτη με όνομα `ptr` που μπορεί να δείχνει σε συνάρτηση που πέρνει μια παράμετρο τύπου `int` και επιστρέφει `void`.

```
1  #include <stdio.h>
2
3  void print(int x) {
4      printf("%d\n", x);
5  }
6
7  int main() {
8      void (*ptr)(int);
9
10     ptr = &print;
11
12     return 0;
13 }
```

- Για να δείξει κάπου ο δείκτης χρησιμοποιούμε τον τελεστή διεύθυνσης &. Για παράδειγμα `ptr = &print`.
- Η C για ευκολία μας επιτρέπει να γράψουμε και `ptr = print` με το ίδιο αποτέλεσμα.

Δείκτες σε Συναρτήσεις

```c
#include <stdio.h>

void print(int x) {
    printf("%d\n", x);
}

int main() {
    void (*ptr)(int);

    ptr = &print;
    (*ptr)(5);

    return 0;
}
```

- Για να χρησιμοποιήσουμε ένα δείκτη σε συνάρτηση τον κάνουμε dereference και δίνουμε παραμέτρους, π.χ (*`ptr`)(5).
- Η C για ευκολία μας επιτρέπει να γράψουμε και `ptr`(5) με το ίδιο αποτέλεσμα.

Δείκτες σε Συναρτήσεις

```
1  #include <stdio.h>
2
3  void foo(int x) {
4      printf("foo %d\n", x);
5  }
6
7  void bar(int y) {
8      printf("bar %d\n", y);
9  }
10
11 int main() {
12     void (*ptr)(int);
13
14     ptr = &foo;
15     (*ptr)(5);
16
17     ptr = &bar;
18     (*ptr)(5);
19
20     return 0;
21 }
```

Ο κώδικας καλεί μέσω ενός δείκτη τις 2 συναρτήσεις `foo()` και `bar()`.

Δείκτες σε Συναρτήσεις

Παράμετροι Συναρτήσεων

```c
#include <stdio.h>

void print(int x, void (*ptr)(int)) {
    (*ptr)(x);
}

void foo(int x) {
    printf("foo %d\n", x);
}

void bar(int y) {
    printf("bar %d\n", y);
}

int main() {
    print(5, &foo);
    print(5, &bar);

    return 0;
}
```

Μπορούμε να περάσουμε ένα δείκτη συνάρτησης ως παράμετρο σε άλλη συνάρτηση.

Ο κώδικας καλεί πρώτα την `foo()` και μετά την `bar()`.

Επιστροφή Δείκτη Συνάρτησης

Είναι λίγο πολύπλοκη η σύνταξη αλλά μπορούμε να γράψουμε το εξής:

```
float (*getFunction(const char code))(float, float)
{
    // code here
}
```

Που ορίζει μια συνάρτηση με όνομα `getFunction` η οποία πέρνει ως παράμετρο ένα χαρακτήρα τύπου `const char`, επιστρέφει ένα δείκτη σε συνάρτηση που έχει 2 `float` παραμέτρους και με την σειρά της επιστρέφει τύπο `float`.

Επιστροφή Δείκτη Συνάρτησης

Παράδειγμα

```
1  #include <stdio.h>
2
3  float minus(float f1, float f2) { return f1-f2; }
4
5  float plus(float f1, float f2) { return f1+f2; }
6
7  float (*getFunction(const char code))(float, float)
8  {
9      if (code == '+')
10         return &plus;
11     else if (code == '-')
12         return &minus;
13     return NULL;
14 }
15
16 int main() {
17     float f;
18     f = (*getFunction('+'))(2.0, 1.0);
19     f = (*getFunction('-'))(f, 1.0);
20     printf("%f\n", f);
21
22     return 0;
23 }
```

Πίνακες Δεικτών Συναρτήσεων

Μπορούμε να έχουμε και πίνακες με δείκτες συναρτήσεων. Η σύνταξη είναι πάλι κάπως παράξενη.

Παρακάτω φαίνεται πως μπορούμε να δηλώσουμε ένα πίνακα 10 θέσεων που κάθε θέση είναι ένας δείκτης σε μια συνάρτηση που επιστρέφει `int` και πέρνει 3 παραμέτρους `float`, `char` και `char`.

```
int (*funcArr[10])(float, char, char);
```

Το όνομα του πίνακα είναι **`funcArr`**.

Πίνακες Δεικτών Συναρτήσεων

```c
#include <stdio.h>

void foo1(int x) { printf("foo1 %d\n", x); }
void foo2(int x) { printf("foo2 %d\n", x); }
void foo3(int x) { printf("foo3 %d\n", x); }
void foo4(int x) { printf("foo4 %d\n", x); }

int main() {
    int i, j;
    void (*farray[4])(int) = {NULL};

    farray[0] = &foo1;
    farray[1] = &foo2;
    farray[2] = &foo3;
    farray[3] = &foo4;

    for(i = 0; i < 100; i++)
        for(j = 0; j < 4; j++)
            (*farray[j])(i);

    return 0;
}
```

Δείκτες σε `void`

Στο παρακάτω πρόγραμμα

```
1  #include <stdio.h>
2
3  int main() {
4      int c;
5      double *ptr;
6
7      ptr = &c;
8
9      return 0;
10 }
```

ο μεταγλωττιστής μας προειδοποιεί πως

```
test.c: In function ‘main’:
test.c:8: warning: assignment from incompatible pointer type
```

Δείκτες σε `void`

Η C μας παρέχει ένα δείκτη τύπου `void` που μπορεί να δείξει οπουδήποτε.

```
1  #include <stdio.h>
2
3  int main() {
4      int c;
5      double k;
6      void *ptr;
7
8      ptr = &c;
9      ptr = &k;
10
11     return 0;
12 }
```

Θα δούμε που χρησιμεύει σε επόμενες διαλέξεις.

Δείκτες σε Δείκτες

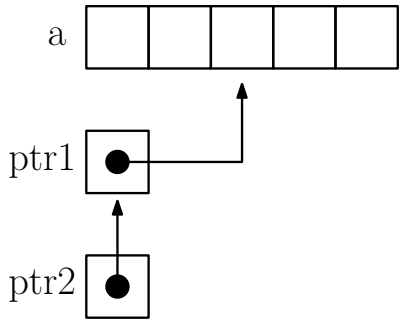
Για να δηλώσουμε ένα δείκτη που να δείχνει σε μια μεταβλητή που είναι και αυτή δείκτης πρέπει να χρησιμοποιήσουμε την παρακάτω σύνταξη.

```
int **x;
```

Θα ήταν πιο ευδιάκριτο εαν γράφαμε `int* *x` για να καταλάβουμε πως ο `x` είναι ένας δείκτης που δείχνει σε ένα δείκτη ακεραίων.

Δείκτες σε Δείκτες

```c
#include <stdio.h>

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    int *ptr1  = &a[2];
    int **ptr2 = &ptr1;

    printf("%d\n", **ptr2);

    return 0;
}
```



Ο κώδικας τυπώνει

```
3
```