ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

Κεφάλαιο 5: Δείκτες και πίνακες

Δείκτες

- Ένας δείκτης στην C είναι μία μεταβλητή στην οποία μπορούμε να καταχωρήσουμε κάποια διεύθυνση μνήμης. Στη διεύθυνση αυτή μπορούμε να φυλάξουμε κάποια τιμή συγκεκριμένου τύπου.
- Η απλή εκδοχή μίας δήλωσης μεταβλητής τύπου δείκτη είναι της μορφής:
  ⟨τύπος⟩ *⟨μεταβλητή⟩
  Με τη δήλωση αυτή ορίζουμε τη ⟨μεταβλητή⟩ να είναι δείκτης σε δεδομένα που ο τύπος τους είναι ⟨τύπος⟩.
- Μπορούμε όμως να ορίσουμε μία ⟨μεταβλητή⟩ να είναι δείκτης σε δεδομένα τύπου δείκτη που δείχνουν σε δεδομένα που ο τύπος τους είναι ⟨τύπος⟩. Αυτό μπορεί να γίνει με μία δήλωση της μορφής:
  ⟨τύπος⟩ **⟨μεταβλητή⟩
- Δεν υπάρχει αμφιβολία ότι η τακτική αυτή γενικεύεται και σε πιο πολύπλοκες δηλώσεις (δείκτης σε δείκτη σε δείκτη κλπ.), αλλά οι πραγματικά χρήσιμοι δείκτες, στη μεγάλη πλειοψηφία των προγραμμάτων που γράφουμε, είναι αυτοί που ορίζουμε με τις παραπάνω δηλώσεις.

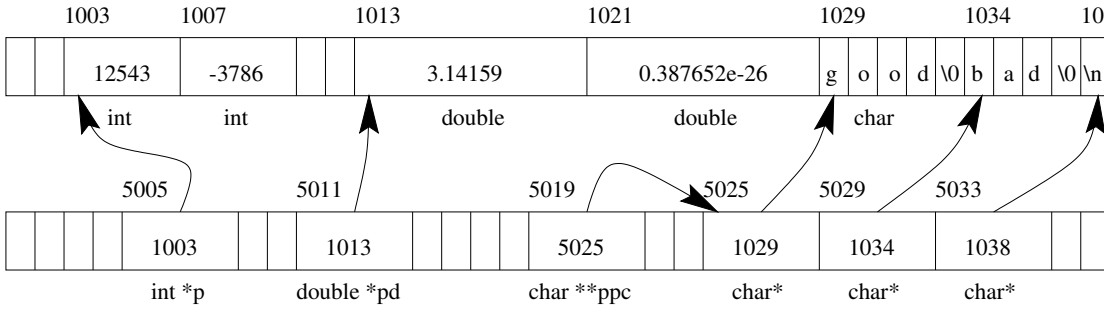
- Περί δεικτών, αλλά και πινάκων, που θα συζητήσουμε στη συνέχεια, το Κεφάλαιο 5 από το [KR] (σελ. 135–178) είναι μία ανεξάντλητη πηγή πληροφοριών. [1]

- Παράδειγμα:

```
int *p;
double *pd;
char **ppc;
```

Οι μεταβλητές p και pd ορίστηκαν σαν δείκτες σε ακεραίους και πραγματικούς διπλής ακρίβειας, αντίστοιχα.
Η μεταβλητή ppc είναι δείκτης σε θέση μνήμης που μπορούμε να φυλάξουμε δείκτη σε χαρακτήρες.

Δείτε και το σχήμα:



- Το σύμβολο * εφαρμοσμένο σε μεταβλητές τύπου δείκτη χρησιμοποιείται και στις εντολές του προγράμματος, ως τελεστής έμμεσης αναφοράς. Όταν έχουμε ορίσει μία ⟨μεταβλητή⟩ τύπου δείκτη, η έκφραση *⟨μεταβλητή⟩ παριστάνει το περιεχόμενο της θέσης μνήμης στην οποία δείχνει η ⟨μεταβλητή⟩.

- Ο αντίστροφος του τελεστή * είναι ο &. Η έκφραση &⟨μεταβλητή⟩ παριστάνει τη διεύθυνση που φυλάσσεται η τιμή για τη ⟨μεταβλητή⟩, ό,τι τύπου και να είναι αυτή. Δηλαδή, οι τελεστές * και & διαβάζονται ως εξής:
&p = *η διεύθυνση μνήμης που είναι αποθηκευμένο το* p
*p = *το περιεχόμενο της θέσης μνήμης που δείχνει το* p

- Παράδειγμα:

```
int x = 5, y, *px, *py, *pz;
char c = 'F', *pc, d;
px = &x;
py = &y;
pc = &c;
pz = py;
*pz = (*px)++;
d = --(*pc);
pc = &d;
(*pc)--;
```

[1] Εκτός, ίσως, από τα θέματα της δυναμικής δέσμευσης μνήμης, που υπάρχει μία σχετική δυστοκία.

Ποιες οι τιμές των `x`, `y`, `c`, `d` μετά από αυτές τις εντολές; [2]

- Θα μπορούσαμε να είχαμε γράψει `--*pc` αντί για `--(*pc)`; [3]
- Επίσης, γράψαμε `(*px)++`. Οι παρενθέσεις χρειάζονται στην έκφραση αυτή, αν θέλουμε ο τελεστής μοναδιαίας αύξησης `++` να εφαρμοσθεί επάνω στο περιεχόμενο της θέσης μνήμης που δείχνει ο δείκτης `px`. Αν είχαμε γράψει `*px++`, λόγω της υψηλότερης προτεραιότητας του τελεστή `++` (αλλά και του `--`) έναντι του τελεστή `*`, αυτό θα σήμαινε το `*(px++)`.
- Τι σημαίνει όμως το `*(px++)` (ή το ισοδύναμό του `*px++`); [4]
- Όμως, ποιο είναι το νόημα να αυξήσουμε ή να μειώσουμε ένα δείκτη κατά 1, δηλαδή να κάνουμε τον δείκτη να δείχνει στην επόμενη ή προηγούμενη θέση μνήμης από αυτή που έδειχνε; Έχει νόημα αν αναφερόμαστε σε συνεχόμενες θέσεις μνήμης που, με κάποιο τρόπο, έχουμε δεσμεύσει και χρησιμοποιούμε.
- Εκτός από την αύξηση ή μείωση ενός δείκτη κατά 1 (μέσω των τελεστών μοναδιαίας αύξησης και μείωσης `++` και `--`), μπορούμε να προσθέσουμε σε ένα δείκτη ή να αφαιρέσουμε από αυτόν μία ακέραια σταθερά.
- Άλλες πράξεις μεταξύ δεικτών δεν επιτρέπονται, εκτός από την αφαίρεση δεικτών που δείχνουν σε στοιχεία ενός μπλοκ από δεδομένα του ίδιου τύπου, που έχουν δεσμευθεί, με κάποιο τρόπο, για ενιαία διαχείριση. Επίσης, επιτρέπονται και συγκρίσεις μεταξύ τέτοιων δεικτών.
- Αν οι `pi`, `pc` και `pd` είναι δείκτες (σε `int`, `char` και `double`, αντίστοιχα), οι παρακάτω εντολές είναι απολύτως νόμιμες:

  ```
  pi = pi+3;
  pc -= 4;
  pd = pd-2;
  ```

  Η πρόσθεση σε (αφαίρεση από) ένα δείκτη `p` μίας σταθεράς `n`, μεταβάλλει τον δείκτη ώστε να δείξει `n` θέσεις μνήμης (όχι `n` bytes), μεγέθους όσο αυτό του τύπου δεδομένων που δείχνει ο δείκτης, πιο μετά (πριν). Αν οι `int`, `char` και `double` είναι των 4, 1 και 8 bytes, αντίστοιχα, πόσο θα μεταβληθούν οι δείκτες `pi`, `pc` και `pd`; [5]
- Μία πολύ συνηθισμένη χρήση των δεικτών είναι στην περίπτωση που θέλουμε κάποια συνάρτηση να επιστρέψει κάποια τιμή (ή και περισσότερες), όχι σαν επιστρεφόμενη τιμή στο όνομά της, αλλά επιδρώντας στις μεταβλητές της συνάρτησης που την κάλεσε. Τότε,

[2] `6`, `5`, `'E'` και `'D'`, αντίστοιχα

[3] Ναι, γιατί όχι;

[4] Όχι, ΔΕΝ σημαίνει ότι πρώτα θα αυξηθεί ο δείκτης `px` και μετά θα πάρουμε το περιεχόμενο της θέσης μνήμης που δείχνει η νέα, αυξημένη, τιμή του δείκτη. Οι παρενθέσεις δεν δείχνουν τη σειρά που θα γίνουν οι υπολογισμοί, αλλά το πού εφαρμόζονται οι τελεστές. Στο προκείμενο παράδειγμα, ο τελεστής `++` είναι μεταθεματικός, άρα πρώτα θα πάρουμε το περιεχόμενο της θέσης μνήμης που δείχνει ο δείκτης `px` και μετά θα αυξηθεί ο δείκτης. Αν θέλαμε πρώτα να αυξήσουμε τον δείκτη και μετά να κάνουμε την αναφορά, έπρεπε να γράψουμε `*(++px)`.

[5] `12`, `-4` και `-16`, αντίστοιχα

η καλούσα συνάρτηση πρέπει να περάσει στην καλουμένη τη διεύθυνση μίας μεταβλητής της, στην οποία η καλουμένη θα γράψει την τιμή που πρέπει να επιστρέψει.

- Θεωρήστε το εξής πρόβλημα: Θέλουμε να γράψουμε μία συνάρτηση swap, η οποία να ανταλλάσσει τα περιεχόμενα δύο ακέραιων μεταβλητών.
- Έστω ότι για να ανταλλάξουμε τις τιμές των ακέραιων μεταβλητών a και b προτιθέμεθα να καλέσουμε τη συνάρτηση `swap(a,b)`, την οποία έχουμε ορίσει ως εξής:

```
void swap(int x, int y)
{ int temp;
  temp = x;
  x = y;
  y = temp; }
```

Γιατί είναι λάθος αυτό; [6]

- Πώς θα γράφαμε τη σωστή swap; Έτσι:

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

Και πώς θα την καλούσαμε για να ανταλλάξουμε τις τιμές των μεταβλητών a και b; [7]

- Και γιατί χρειαζόμαστε τη μεταβλητή `temp`; [8]

Περί ανάγνωσης ακεραίων (και όχι μόνο)

- Μέχρι στιγμής, δεν γνωρίζαμε κάποιο εύχρηστο τρόπο να διαβάσουμε έναν ακέραιο μέσα από ένα πρόγραμμα C. Τώρα, με τη χρήση δεικτών, αυτό είναι εφικτό.
- Η πρότυπη βιβλιοθήκη εισόδου/εξόδου της C παρέχει και τη συνάρτηση `scanf`, που είναι η " αδελφή" συνάρτηση της `printf`, και με την οποία μπορούμε να διαβάσουμε από την είσοδο ενός προγράμματος δεδομένα προς επεξεργασία.
- Η `scanf` συντάσσεται με αντίστοιχο τρόπο αυτού της `printf`, δηλαδή στην πρώτη παράμετρο δίνουμε μία συμβολοσειρά που περιγράφει τι τύπων δεδομένα σκοπεύουμε

[6] Έστω ότι πριν καλέσουμε την `swap(a,b)`, οι τιμές των μεταβλητών a και b είναι 5 και 8, αντίστοιχα. Τότε, ουσιαστικά καλούμε `swap(5,8)`, δηλαδή οι τυπικές παράμετροι x και y της swap, που είναι τοπικές/αυτόματες μεταβλητές για τη συνάρτηση, παίρνουν τις τιμές 5 και 8, αντίστοιχα, ανταλλάσσονται μέσα στην swap οι τιμές των x και y, αλλά οι μεταβλητές a και b της καλούσας συνάρτησης δεν επηρεάζονται καθόλου από την αλλαγή αυτή.

[7] `swap(&a,&b)`

[8] Για τον ίδιο λόγο που χρειαζόμαστε ένα τρίτο ποτήρι αν θέλουμε να ανταλλάξουμε τα περιεχόμενα δύο άλλων ποτηριών, εκτός κι αν είμαστε ταχυδακτυλουργοί `(:-)`.

να διαβάσουμε και με ποιον τρόπο. Οι επόμενες παράμετροι αναφέρονται στο πού θα φυλαχθούν οι τιμές που διαβάστηκαν.

- Μόνο ΠΡΟΣΟΧΗ! Επειδή η συνάρτηση `scanf` πρόκειται να επιστρέψει στη συνάρτηση που την κάλεσε κάποιες τιμές, στις αντίστοιχες παραμέτρους ΔΕΝ βάζουμε τις μεταβλητές στις οποίες θέλουμε να φυλαχθούν οι τιμές αυτές, αλλά δείκτες στις μεταβλητές αυτές. Είναι ο ίδιος ακριβώς λόγος για τον οποίο τη συνάρτηση `swap` την καλούμε σαν `swap(&a,&b)`, όπου `a` και `b` είναι ακέραιες μεταβλητές.

- Παράδειγμα:

```
int id, rank, *prank;
float salary;
prank = &rank;
printf("Please give id and rank: ");
scanf("%d %d", &id, prank);
printf("Please give salary: ");
scanf("%f", &salary);
```

- Η ανάγνωση τιμών για την ακέραια μεταβλητή `id` και τη μεταβλητή κινητής υποδιαστολής `salary` γίνεται περνώντας στις συναρτήσεις `scanf` δείκτες στις μεταβλητές αυτές. Όμως στην πρώτη `scanf` περάσαμε την ίδια τη μεταβλητή `prank`, αφού αυτή έχει ήδη ορισθεί σαν δείκτης σε ακέραιο. Βέβαια, στη συνέχεια, για να αναφερθούμε στην τιμή που διαβάσαμε, αυτό πρέπει να γίνει μέσω του τελεστή έμμεσης αναφοράς, δηλαδή `*prank`, ή μέσω της μεταβλητής `rank`.

- Οι προδιαγραφές `%d` και `%f` στις `scanf` υποδεικνύουν ότι οι τιμές που θα διαβαστούν είναι ακέραιες και κινητής υποδιαστολής, αντίστοιχα.

- Υπάρχουν και άλλες προδιαγραφές που μπορεί να χρησιμοποιηθούν για ανάγνωση τιμών από την `scanf`, καθώς και κάποιες ενδιαφέρουσες δυνατότητες που παρέχονται από τη συνάρτηση. Για περισσότερα, "`man scanf`" ή όταν θα αναφερθούμε αναλυτικότερα στις συναρτήσεις της πρότυπης βιβλιοθήκης εισόδου/εξόδου της `C`.

Πίνακες

- Όταν θέλουμε στην C να χειριστούμε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο, ορίζουμε ένα πίνακα συγκεκριμένου μεγέθους για τη φύλαξη των δεδομένων αυτών.

- Με τη δήλωση

```
int myarray[20];
```

ορίζουμε ένα μονοδιάστατο πίνακα ακεραίων με όνομα `myarray` στον οποίο μπορούν να φυλαχθούν το πολύ 20 ακέραιες τιμές. Το `20` είναι η διάσταση του πίνακα.

- Ένας πίνακας αποθηκεύεται στη μνήμη σε συνεχόμενες πάντα θέσεις.

- Μέσα στο πρόγραμμά μας, μπορούμε να αναφερόμαστε στα στοιχεία του πίνακα `myarray`, που έχει οριστεί με διάσταση `20`, σαν `myarray[0]`, `myarray[1]`, ..., `myarray[19]`. Γενικά, με το `myarray[i]` αναφερόμαστε στο στοιχείο του πίνακα `myarray` με δείκτη[9] `i`, όπου το `i` μπορεί να είναι κάποια παράσταση, όχι μόνο σταθερά ή μεταβλητή, η οποία πρώτα θα αποτιμηθεί και μετά θα γίνει η προσπέλαση του στοιχείου `myarray[i]`.

- Αν η διάσταση ενός πίνακα είναι `N`, οι δείκτες των στοιχείων του κυμαίνονται από `0` έως `N-1`.

- Προσοχή στο κλασικό λάθος που γίνεται στον προγραμματισμό με τη διαχείριση πινάκων! Απόπειρα πρόσβασης εκτός των ορίων ενός πίνακα δεν είναι δυνατόν να διαγνωσθεί από τον μεταγλωττιστή, με συνέπεια να προκύψει σοβαρό πρόβλημα κατά την εκτέλεση του προγράμματος.

- Είναι ευθύνη του προγραμματιστή να εξασφαλίσει ότι όταν στο πρόγραμμά του κάνει πρόσβαση στο στοιχείο `x[i]`, όπου ο πίνακας `x` έχει ορισθεί με διάσταση, έστω, `100`, τότε το `i` δεν θα έχει τιμή μικρότερη από το `0` ή μεγαλύτερη από το `99`, όταν γίνεται η αναφορά στο `x[i]`. Αλλιώς ...
  Στο Unix, για παράδειγμα, θα πάρουμε ένα μεγαλοπρεπές "`Segmentation fault`" ή κανένα "`Bus error`" κατά την εκτέλεση του προγράμματος. Γενικώς, αυτά τα μηνύματα μας υποδεικνύουν ότι έχουμε κάνει στο πρόγραμμά μας κάποιο λάθος διαχείρισης μνήμης.[10] Σε περιβάλλοντα Microsoft Windows, το πιο πιθανό αποτέλεσμα όταν υπάρχει λάθος διαχείρισης μνήμης είναι ο βίαιος τερματισμός του προγράμματος.

- Υπάρχει πολύ στενή σχέση μεταξύ των πινάκων και των δεικτών διευθύνσεων στην C. Ουσιαστικά, το όνομα ενός πίνακα είναι ένας σταθερός δείκτης στο πρώτο στοιχείο του πίνακα.

- Για παράδειγμα, αν έχουμε ορίσει έναν πίνακα `a`, τότε η έκφραση `*(a+i)` είναι ισοδύναμη με την `a[i]`. Ομοίως, και οι εκφράσεις `a+i` και `&a[i]` είναι ισοδύναμες.

- Αν σ' ένα δείκτη `pa`, που δείχνει σε στοιχεία ίδιου τύπου με τον τύπο των στοιχείων ενός πίνακα `a`, αναθέσουμε σαν τιμή τη διεύθυνση του πρώτου, ή κάποιου άλλου, στοιχείου του πίνακα, οι εκφράσεις `pa+`⟨N⟩, όπου ⟨N⟩ είναι ακέραια παράσταση, είναι νόμιμες ως διευθύνσεις για προσπέλαση των στοιχείων του πίνακα, αρκεί να έχουμε εξασφαλίσει ότι δείχνουν μέσα στα όρια του πίνακα.

- Επίσης, μπορούμε τον δείκτη `pa` να τον αυξάνουμε ή να τον μειώνουμε, και, γενικά, να τον

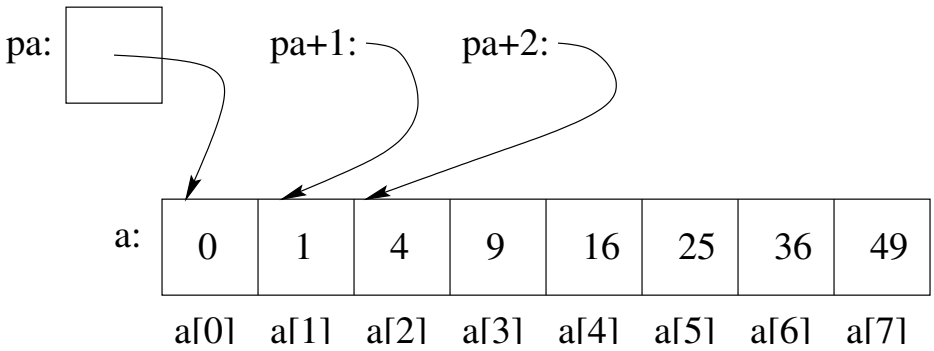
[9] Προσοχή στην ορολογία! Τον όρο "δείκτης" εδώ, τον χρησιμοποιούμε σαν ελληνική απόδοση του Αγγλικού "index". Πρόκειται για δείκτη πίνακα. Οι δείκτες όμως που είδαμε στην προηγούμενη ενότητα είναι δείκτες διευθύνσεων και αντιστοιχούν στον Αγγλικό όρο "pointer". Για τη συνέχεια, δεν θα κάνουμε ιδιαίτερη αναφορά, όταν χρησιμοποιούμε τον όρο "δείκτης", σε ποια εκδοχή αναφερόμαστε. Θα προκύπτει αυτό εύκολα από τα συμφραζόμενα.

[10] Ευτυχώς, δηλαδή, γιατί έτσι μαθαίνουμε ότι το πρόγραμμά μας έχει σφάλματα, οπότε μπορούμε να μπούμε στη διαδικασία να τα διορθώσουμε. Όμως, αυτό δεν συμβαίνει πάντοτε. Υπάρχουν περιπτώσεις που να έχουμε κάνει λάθος διαχείρισης μνήμης, να μην πάρουμε κάποια ένδειξη γι' αυτό και να νομίζουμε ότι το πρόγραμμά μας δουλεύει σωστά, ενώ αυτό θα κάνει άλλα αντ' άλλων.

τροποποιούμε, κατά βούληση (π.χ. pa++, --pa, κλπ.), πάντα υπό την προϋπόθεση ότι μέσω του τροποποιημένου δείκτη θα προσπελαύνουμε στοιχεία του πίνακα εντός των ορίων του.

- Παράδειγμα:

```
int i, a[8], *pa;
for (i=0 ; i<8 ; i++)
  a[i] = i*i;
pa = &a[0];
a[6] = *(a+4);
*(pa+3) = a[5];
a[0] = *((pa++)+2);
*((++pa)+5) = a[1];
*(&a[5]-1) = *(--pa);
```



Ποια θα είναι τα περιεχόμενα του πίνακα a μετά την εκτέλεση των παραπάνω εντολών; [11]

- Προσέξτε ότι ενώ στο προηγούμενο παράδειγμα, τροποποιούσαμε τον δείκτη pa, αυτό δεν μπορεί να γίνει για το όνομα του πίνακα, παρότι είναι επιτρεπτό να χρησιμοποιηθεί σαν δείκτης (π.χ. *(a+4)). Δηλαδή, απαγορεύεται να γράψουμε a++.
- Μερικές φορές, θέλουμε να ορίσουμε μία συνάρτηση, η οποία να επεξεργάζεται τα στοιχεία ενός πίνακα. Φυσικά, μία λύση είναι να έχει δηλωθεί ο πίνακας εξωτερικός, οπότε μπορεί να γίνει η επικοινωνία με την καλούσα συνάρτηση πολύ απλά.
- Είναι πιθανό, όμως, για διάφορους λόγους, κυρίως για να είναι η συνάρτησή μας γενικής χρήσης, να θέλουμε να περάσουμε τον πίνακα στη συνάρτηση σαν παράμετρο. Στην περίπτωση αυτή, απλώς περνάμε ένα δείκτη στο πρώτο στοιχείο του πίνακα και, μέσω αυτού, η συνάρτηση μπορεί να προσπελάσει οποιοδήποτε στοιχείο του.
- Παράδειγμα:

```
#define N 50

int main(void)
{  int x[N], *myp;
```

[11] 4, 1, 4, 25, 1, 25, 16, 1

```
    ..............
    myfun(&x[0], N);
    ..............
}

void myfun(int *px, int n)
{  int y;
    ..............
    y = *(px+2);
    ..............
}
```

- Στο προηγούμενο παράδειγμα, η κλήση της συνάρτησης `myfun` από την `main` θα μπορούσε να είχε γίνει και σαν `myfun(x, N)`, δηλαδή να περάσουμε το όνομα του πίνακα, αφού, όπως γνωρίζουμε, το όνομα αυτό είναι ουσιαστικά ένας δείκτης στο πρώτο στοιχείο του πίνακα.
- Αν είχαμε αναθέσει στον δείκτη `myp` τη διεύθυνση του πρώτου στοιχείου του πίνακα (`myp = &x[0]` ή, ισοδύναμα, `myp = x`), θα μπορούσαμε να καλέσουμε τη συνάρτηση και σαν `myfun(myp, N)`.
- Αν θέλουμε να περάσουμε στη συνάρτηση έναν υποπίνακα του αρχικού πίνακα, από ένα στοιχείο και μετά, θα μπορούσαμε να την καλέσουμε με παράμετρο τη διεύθυνση του στοιχείου αυτού. Για παράδειγμα, αν έχουμε κάνει πρώτα την ανάθεση `myp = &x[2]` με την κλήση `myfun(myp, N-2)` (ή απ' ευθείας `myfun(&x[2], N-2)`, η συνάρτηση `myfun` θα "δει" έναν πίνακα με πρώτο στοιχείο το τρίτο του αρχικού.
- Επίσης, στον ορισμό της συνάρτησης, θα μπορούσαμε για τυπική παράμετρο να βάλουμε το `int px[]`, αντί για το `int *px`. Είναι απολύτως ισοδύναμα.

Ιστόγραμμα συχνοτήτων γραμμάτων στην είσοδο

```c
/* File: histogram.c */
#include <stdio.h>
#define YAXISLEN 12                                  /* Y-axis length */

int main(void)
{ int i, j, ch, total;
  int letfr[26];         /* Letter occurrences and frequencies array */
  for (i=0 ; i < 26 ; i++)
    letfr[i] = 0;                                  /* Initialize array */
  total = 0;       /* Initialize counter of total letter occurrences */
  while ((ch = getchar()) != EOF) {/* Well-known read-character loop */
    if (ch >= 'A' && ch <= 'Z') {
      letfr[ch-'A']++;                    /* Found upper case letter */
      total++;
```

```
    }
    if (ch >= 'a' && ch <= 'z') {
      letfr[ch-'a']++;                    /* Found lower case letter */
      total++;
    }
  }
  printf("  |");  /* Start histogram printing - first Y-axis segment */
  for (i=0 ; i < 26 ; i++) {           /* Convert letter occurrences */
                         /* to frequencies rounded to nearest integer */
    letfr[i] = (int) ((100.0*letfr[i])/total+0.5);
    printf("%s", (letfr[i] > YAXISLEN) ? "^^" : "  ");    /* If i-th */
               /* letter frequency exceeds Y-axis length, print "^^" */
  }
  printf("\n");
  for (j=YAXISLEN ; j > 0 ; j--) {/* Print line at j-value of Y-axis */
    printf("%2d|", j);   /* Print frequency label and Y-axis segment */
    for (i=0 ; i < 26 ; i++)
      printf("%s", (letfr[i] >= j) ? "xx" : "  "); /* If i-th letter */
              /* frequency is greater than or equal to j, print "xx" */
    printf("\n");
  }                                /* Print X-axis and letter labels */
  printf("  +--------------------------------------------------\n");
  printf("   AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz\n");
  return 0;
}
```

```
% gcc -o histogram histogram.c
% ./histogram < histogram.c
  |
12|
11|        xx                            xx
10|        xx      xx                    xx
 9|        xx      xx                xx  xx
 8|        xx      xx        xx      xx  xx
 7|        xx      xx        xx      xx  xx
 6|xx      xx      xx    xx  xx      xx  xx
 5|xx      xxxx    xx    xx  xxxx    xx  xx
 4|xx  xx  xxxx    xx    xx  xxxx    xxxxxx
 3|xx  xx  xxxx  xxxx    xx  xxxx    xxxxxx
 2|xx  xxxxxxxxxxxxx     xx  xxxxxx  xxxxxxxx    xxxx
 1|xxxxxxxxxxxxxxxxxxxx  xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  +--------------------------------------------------
   AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
% ./histogram < /usr/share/dict/words
```

```
  |
12|
11|        xx
10|        xx
 9|xx      xx
 8|xx      xx      xx
 7|xx      xx      xx        xxxx    xx  xx
 6|xx      xx      xx    xx  xxxx    xxxxxx
 5|xx  xx  xx      xx    xx  xxxx    xxxxxx
 4|xx  xx  xx      xx    xx  xxxx    xxxxxxxx
 3|xx  xxxxxx    xxxx    xxxxxxxxxx  xxxxxxxx
 2|xxxxxxxxxx  xxxxxx    xxxxxxxxxx  xxxxxxxx      xx
 1|xxxxxxxxxxxxxxxxxx  xxxxxxxxxxx  xxxxxxxxxxx  xx
  +--------------------------------------------------
   AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
% ./histogram < /usr/include/stdio.h
  |        ^^
12|        xx
11|        xx
10|        xx                xx
 9|        xx      xx        xx
 8|        xxxx    xx        xx          xx
 7|        xxxx    xx        xx          xx
 6|      xxxxxx    xx        xx        xxxx
 5|      xxxxxx    xx        xxxx    xxxxxx
 4|      xxxxxx    xx    xx  xxxxxx  xxxxxx
 3|xx  xxxxxxxx    xx    xx  xxxxxx  xxxxxxxx    xx
 2|xx  xxxxxxxxxx  xx    xx  xxxxxx  xxxxxxxx    xx
 1|xxxxxxxxxxxxxxxxxx  xxxxxxxxxxx  xxxxxxxx    xx
  +--------------------------------------------------
   AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
%
```