ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

ΔΕΥΤΕΡΗ ΒΕΛΤΙΩΜΕΝΗ ΕΚΔΟΣΗ

Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

ΚΛΕΙΔΑΡΙΘΜΟΣ

Περιεχόμενα

Πρόλογος **7**

Πρόλογος της πρώτης έκδοσης **11**

Εισαγωγή **13**

Κεφάλαιο 1. Προπαρασκευαστική εισαγωγή **19**
1.1 Ας ξεκινήσουμε 20
1.2 Μεταβλητές και αριθμητικές παραστάσεις 23
1.3 Η εντολή for 30
1.4 Συμβολικές σταθερές 31
1.5 Είσοδος και έξοδος χαρακτήρων 32
1.6 Πίνακες 41
1.7 Συναρτήσεις 44
1.8 Ορίσματα – κλήση κατ' αξία 48
1.9 Πίνακες χαρακτήρων 49
1.10 Εξωτερικές μεταβλητές και εμβέλεια 53

Κεφάλαιο 2. Τύποι, τελεστές, και παραστάσεις **59**
2.1 Ονόματα μεταβλητών 60
2.2 Τύποι δεδομένων και μεγέθη 60
2.3 Σταθερές 62
2.4 Δηλώσεις 66
2.5 Αριθμητικοί τελεστές 67

2.6 Συσχετιστικοί και λογικοί τελεστές ... 68
2.7 Μετατροπές τύπων ... 69
2.8 Τελεστές μοναδιαίας αύξησης και μείωσης ... 74
2.9 Τελεστές πράξεων με bit ... 76
2.10 Τελεστές ανάθεσης τιμής και παραστάσεις ... 78
2.11 Παραστάσεις υπό συνθήκη ... 80
2.12 Προτεραιότητα και σειρά υπολογισμών ... 81

Κεφάλαιο 3. Η ροή του ελέγχου ... 85
3.1 Εντολές και μπλοκ ... 85
3.2 if-else ... 86
3.3 else-if ... 87
3.4 switch ... 89
3.5 Βρόχοι – while και for ... 91
3.6 Βρόχοι do-while ... 95
3.7 break και continue ... 97
3.8 goto και ετικέτες ... 98

Κεφάλαιο 4. Συναρτήσεις και δομή του προγράμματος ... 101
4.1 Τα βασικά στοιχεία των συναρτήσεων ... 102
4.2 Συναρτήσεις που επιστρέφουν μη ακέραιες τιμές ... 106
4.3 Εξωτερικές μεταβλητές ... 109
4.4 Κανόνες εμβέλειας ... 117
4.5 Αρχεία κεφαλίδας ... 119
4.6 Στατικές μεταβλητές ... 120
4.7 Μεταβλητές register ... 122
4.8 Δόμηση σε μπλοκ ... 122
4.9 Απόδοση αρχικών τιμών ... 123
4.10 Αναδρομικότητα ... 125
4.11 Ο προεπεξεργαστής της C ... 128

Κεφάλαιο 5. Δείκτες και πίνακες ... 135
5.1 Δείκτες και διευθύνσεις ... 136
5.2 Δείκτες και ορίσματα συναρτήσεων ... 138
5.3 Δείκτες και πίνακες ... 141
5.4 Αριθμητική διευθύνσεων ... 144
5.5 Δείκτες χαρακτήρα και συναρτήσεις ... 149
5.6 Πίνακες δεικτών, και δείκτες σε δείκτες ... 153
5.7 Πολυδιάστατοι πίνακες ... 157
5.8 Απόδοση αρχικής τιμής σε πίνακες δεικτών ... 160

5.9 Δείκτες και πολυδιάστατοι πίνακες 161
5.10 Ορίσματα γραμμής διαταγών 162
5.11 Δείκτες σε συναρτήσεις 167
5.12 Περίπλοκες δηλώσεις 171

Κεφάλαιο 6. Δομές 179
6.1 Τα βασικά για τις δομές 180
6.2 Δομές και συναρτήσεις 182
6.3 Πίνακες δομών 186
6.4 Δείκτες σε δομές 191
6.5 Αυτοαναφορικές δομές 193
6.6 Αναζήτηση πίνακα 200
6.7 typedef 203
6.8 Ενώσεις 205
6.9 Πεδία bit 207

Κεφάλαιο 7. Είσοδος και έξοδος 211
7.1 Τυπική είσοδος και έξοδος 212
7.2 Μορφοποιημένη έξοδος — printf 214
7.3 Λίστα με μεταβαλλόμενο πλήθος ορισμάτων 216
7.4 Μορφοποιημένη είσοδος — scanf 218
7.5 Προσπέλαση αρχείων 223
7.6 Χειρισμός σφαλμάτων — stderr και exit 227
7.7 Είσοδος και έξοδος γραμμών 228
7.8 Διάφορες συναρτήσεις 230

Κεφάλαιο 8. Η διασύνδεση συστήματος του UNIX 235
8.1 Περιγραφείς αρχείων 236
8.2 Είσοδος/έξοδος χαμηλού επιπέδου — read και write 237
8.3 open, creat, close, unlink 239
8.4 Τυχαία προσπέλαση — lseek 242
8.5 Παράδειγμα — μια υλοποίηση των fopen και getc 244
8.6 Παράδειγμα — λίστες καταλόγων 248
8.7 Παράδειγμα — ένας κατανεμητής μνήμης 255

Παράρτημα Α. Εγχειρίδιο αναφοράς 263
Α.1 Εισαγωγή 263
Α.2 Λεκτικές συμβάσεις 263
Α.3 Η σημειογραφία του συντακτικού 268
Α.4 Η σημασία των αναγνωριστικών 268

Α.5 Αντικείμενα και α-τιμές ... 271
Α.6 Μετατροπές ... 272
Α.7 Παραστάσεις ... 276
Α.8 Δηλώσεις ... 290
Α.9 Εντολές ... 307
Α.10 Εξωτερικές δηλώσεις ... 311
Α.11 Εμβέλεια και σύνδεσμοι ... 315
Α.12 Προεπεξεργασία ... 316
Α.13 Η γραμματική ... 324

Παράρτημα Β. Η καθιερωμένη βιβλιοθήκη ... 333
Β.1 Είσοδος και έξοδος: <stdio.h> ... 334
Β.2 Έλεγχοι κατηγορίας χαρακτήρων: <ctype.h> ... 344
Β.3 Συναρτήσεις αλφαριθμητικών: <string.h> ... 344
Β.4 Μαθηματικές συναρτήσεις: <math.h> ... 346
Β.5 Βοηθητικές συναρτήσεις: <stdlib.h> ... 347
Β.6 Διαγνωστικά: <assert.h> ... 350
Β.7 Μεταβλητές λίστες ορισμάτων: <stdarg.h> ... 351
Β.8 Μη τοπικά άλματα: <setjmp.h> ... 352
Β.9 Σήματα: <signal.h> ... 352
Β.10 Συναρτήσεις ημερομηνίας και ώρας ... 353
Β.11 Όρια που καθορίζονται από την υλοποίηση: <limits.h> και <float.h> ... 355

Παράρτημα Γ. Σύνοψη των αλλαγών ... 357

Γλωσσάρι ... 361

Ευρετήριο ... 363

6

Δομές

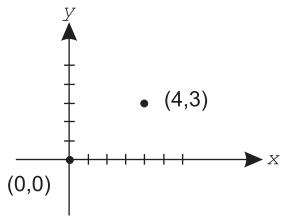
Δομή είναι συλλογή από μία ή περισσότερες μεταβλητές, πιθανώς διαφορετικών τύπων, που ομαδοποιούνται με ένα όνομα για ευκολία στο χειρισμό τους. (Οι δομές σε μερικές γλώσσες, ιδιαίτερα στην Pascal, ονομάζονται "εγγραφές", records.) Οι δομές βοηθούν στην οργάνωση περίπλοκων δεδομένων, ιδιαίτερα σε μεγάλα προγράμματα, επειδή επιτρέπουν τον ενιαίο χειρισμό μιας ομάδας σχετιζόμενων μεταβλητών, αντί να τις χειριζόμαστε ως ξεχωριστές οντότητες.

Ένα παραδοσιακό παράδειγμα δομής είναι η εγγραφή της μισθοδοσίας. Κάθε υπάλληλος περιγράφεται με ένα σύνολο χαρακτηριστικών όπως το όνομα, η διεύθυνση, ο αριθμός κοινωνικής ασφάλισης, ο μισθός, κ.λπ. Μερικά από τα στοιχεία αυτά θα μπορούσαν να είναι και αυτά με τη σειρά τους δομές: ένα όνομα έχει διάφορα στοιχεία, το ίδιο και η διεύθυνση ή ο μισθός. Ένα άλλο παράδειγμα, πιο τυπικό για τη C, προέρχεται από τα γραφικά: ένα σημείο είναι ένα ζεύγος συντεταγμένων, ένα ορθογώνιο είναι ένα ζεύγος σημείων, κ.ο.κ.

Η κύρια αλλαγή που έγινε από το πρότυπο ANSI είναι ότι όρισε την απόδοση μιας δομής ως τιμής — οι δομές μπορούν να αντιγράφονται και να αποδίδονται ως τιμές, να μεταβιβάζονται σε συναρτήσεις, και να επιστρέφονται από συναρτήσεις. Αυτό υποστηριζόταν από πολλούς μεταγλωττιστές εδώ και πολλά χρόνια, όμως τώρα καθορίζονται με ακρίβεια οι ιδιότητες της λειτουργίας αυτής. Επιπρόσθετα, μπορούν τώρα να παίρνουν αρχικές τιμές οι αυτόματοι πίνακες και οι δομές.

6.1 Τα βασικά για τις δομές

Ας δημιουργήσουμε μερικές δομές κατάλληλες για γραφικά. Το βασικό αντικείμενο είναι το σημείο, το οποίο θα θεωρήσουμε ότι έχει μια συντεταγμένη *x* και μια συντεταγμένη *y*, που και οι δύο είναι ακέραιες.



Τα δύο στοιχεία μπορούν να τοποθετηθούν σε μια δομή που δηλώνεται με τον εξής τρόπο:

```
struct point {
   int x;
   int y;
};
```

Η δεσμευμένη λέξη `struct` ξεκινά μια δήλωση δομής, που είναι μια λίστα δηλώσεων οι οποίες περικλείονται από άγκιστρα. Η λέξη `struct` μπορεί να ακολουθείται προαιρετικά από ένα όνομα που αποκαλείται *ετικέτα δομής* (structure tag) (όπως συμβαίνει εδώ με τη λέξη `point`). Η ετικέτα "ονοματίζει" αυτό το είδος δομής, και στη συνέχεια μπορεί να χρησιμοποιείται ως συντομογραφία για το μέρος της δήλωσης που βρίσκεται μέσα στα άγκιστρα.

Οι μεταβλητές που κατονομάζονται σε μια δομή ονομάζονται *μέλη* (members). Ένα μέλος ή μια ετικέτα δομής μπορεί να έχει ίδιο όνομα με μια κανονική μεταβλητή (δηλαδή, μια μεταβλητή που δεν είναι μέλος) χωρίς να υπάρχει διένεξη, αφού τα ονόματα μπορούν πάντα να διακρίνονται από τα συμφραζόμενα. Επιπρόσθετα, μπορούν να εμφανίζονται τα ίδια ονόματα μελών σε διαφορετικές δομές, αν και για λόγους καλού στυλ προγραμματισμού κανονικά θα χρησιμοποιούμε ίδια ονόματα μόνο για στενά συνδεδεμένα αντικείμενα.

Η δήλωση `struct` ορίζει έναν τύπο δεδομένων. Το δεξιό άγκιστρο που τερματίζει τη λίστα των μελών μπορεί να ακολουθείται από μια λίστα μεταβλητών, όπως συμβαίνει και με όλους τους βασικούς τύπους δεδομένων. Με άλλα λόγια, η εντολή

```
struct { ... } x, y, z;
```

είναι συντακτικά ανάλογη με την

```
int x, y, z;
```

με την έννοια ότι κάθε εντολή δηλώνει τα x, y, και z ως μεταβλητές του κατονομαζόμενου τύπου δεδομένων και προκαλεί δέσμευση χώρου γι' αυτές.

Μια δήλωση δομής που δεν ακολουθείται από λίστα μεταβλητών δεν δεσμεύει χώρο στη μνήμη, αλλά απλώς περιγράφει το πρότυπο ή τη μορφή της δομής. Αν όμως η δομή έχει ετικέτα, η ετικέτα μπορεί να χρησιμοποιηθεί αργότερα σε ορισμούς στιγμιοτύπων της δομής. Για παράδειγμα, με δεδομένη την παραπάνω δήλωση της δομής point, η εντολή

```
struct point pt;
```

ορίζει μια μεταβλητή pt που είναι δομή τύπου struct point. Μπορούμε να αποδώσουμε αρχικές τιμές σε μια δομή τοποθετώντας αμέσως μετά τον ορισμό της μια λίστα αρχικών τιμών για τα μέλη, καθεμιά από τις οποίες είναι σταθερή παράσταση:

```
struct point maxpt = { 320, 200 };
```

Μια αυτόματη δομή μπορεί επίσης να πάρει αρχικές τιμές με ανάθεση τιμών, ή με την κλήση μιας συνάρτησης που επιστρέφει δομή του κατάλληλου τύπου.

Η αναφορά σε ένα μέλος μιας συγκεκριμένης δομής μέσα από μια παράσταση γίνεται με μια κατασκευή της μορφής

όνομα-δομής.μέλος

Ο τελεστής μέλους δομής "." συνδέει το όνομα της δομής με το όνομα του μέλους. Για παράδειγμα, για να εμφανίσουμε τις συντεταγμένες του σημείου pt γράφουμε

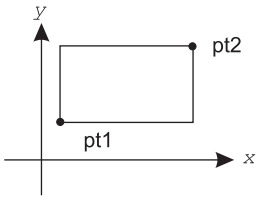
```
printf("%d,%d", pt.x, pt.y);
```

και για να υπολογίσουμε την απόσταση από την αρχή του συστήματος συντεταγμένων (0, 0) μέχρι το pt γράφουμε

```
double dist, sqrt(double);

dist=sqrt((double)pt.x * pt.x+(double)pt.y * pt.y);
```

Οι δομές μπορεί να είναι ένθετες. Μια αναπαράσταση ορθογωνίου είναι ένα ζευγάρι σημείων που αντιπροσωπεύουν τις διαγωνίως απέναντι κορυφές του:




```
struct rect {
   struct point pt1;
   struct point pt2;
};
```

Η δομή `rect` περιέχει δύο δομές `point`. Αν δηλώσουμε τη μεταβλητή `screen` ως

```
struct rect screen;
```

τότε το

```
screen.pt1.x
```

αναφέρεται στη συντεταγμένη *x* του μέλους `pt1` της `screen`.

6.2 Δομές και συναρτήσεις

Οι μόνες επιτρεπόμενες πράξεις με μια δομή είναι η αντιγραφή της, η απόδοση τιμής σε αυτή συνολικά, η εξαγωγή της διεύθυνσής της με τον τελεστή &, και η προσπέλαση των μελών της. Η αντιγραφή και η απόδοση τιμής περιλαμβάνουν τη μεταβίβαση ορισμάτων σε συναρτήσεις και την επιστροφή τιμών από συναρτήσεις. Οι δομές δεν μπορούν να συγκριθούν μεταξύ τους. Μια δομή μπορεί να πάρει αρχική τιμή με μια λίστα σταθερών τιμών για τα μέλη της, ενώ μια αυτόματη δομή μπορεί επίσης να πάρει αρχική τιμή με ανάθεση τιμών.

Ας εξετάσουμε τις δομές γράφοντας μερικές συναρτήσεις για το χειρισμό σημείων και ορθογωνίων. Υπάρχουν τουλάχιστον τρεις δυνατές προσεγγίσεις: η μεταβίβαση των στοιχείων ξεχωριστά, η μεταβίβαση μιας ολόκληρης δομής, ή η μεταβίβαση ενός δείκτη για αυτή. Η κάθε προσέγγιση έχει τα δυνατά και τα αδύνατα σημεία της.

Η πρώτη συνάρτηση, η `makepoint`, παίρνει δύο ακεραίους και επιστρέφει μια δομή `point`:

```
/* makepoint: δημιουργία σημείου
   από τα στοιχεία x και y */
struct point makepoint(int x, int y)
{
   struct point temp;

   temp.x = x;
   temp.y = y;
   return temp;
}
```

Προσέξτε ότι δεν υπάρχει διένεξη ανάμεσα στο όνομα του ορίσματος και το μέλος με το ίδιο όνομα. Αντίθετα, η χρησιμοποίηση των ίδιων ονομάτων τονίζει τη σχέση μεταξύ των στοιχείων.

Η συνάρτηση `makepoint` μπορεί τώρα να χρησιμοποιηθεί για τη δυναμική απόδοση αρχικών τιμών σε οποιαδήποτε δομή, ή για τη διάθεση ορισμάτων τύπου δομής σε μια συνάρτηση:

```
struct rect screen;
struct point middle;
struct point makepoint(int, int);

screen.pt1 = makepoint(0, 0);
screen.pt2 = makepoint(XMAX, YMAX);
middle = makepoint((screen.pt1.x + screen.pt2.x)/2,
                   (screen.pt1.y + screen.pt2.y)/2);
```

Το επόμενο βήμα είναι ένα σύνολο συναρτήσεων για την εκτέλεση αριθμητικών πράξεων με σημεία. Για παράδειγμα,

```
/* addpoint: πρόσθεση δύο σημείων */
struct point addpoint(struct point p1,
                      struct point p2)
{
   p1.x += p2.x;
   p1.y += p2.y;
   return p1;
}
```

Εδώ τόσο τα ορίσματα όσο και οι επιστρεφόμενες τιμές είναι δομές. Αυξήσαμε τα στοιχεία του `p1`, αντί να χρησιμοποιήσουμε μια προσωρινή μεταβλητή, για να δώσουμε έμφαση στο γεγονός ότι οι παράμετροι της δομής μεταβιβάζονται κατ' αξία, όπως και όλες οι άλλες παράμετροι.

Ως ένα ακόμα παράδειγμα, η συνάρτηση ptinrect ελέγχει αν ένα σημείο βρίσκεται μέσα σε ένα ορθογώνιο. Εδώ έχουμε υιοθετήσει τη σύμβαση ότι ένα ορθογώνιο περιλαμβάνει την αριστερή και την κάτω πλευρά του, δεν περιλαμβάνει όμως την επάνω και τη δεξιά πλευρά του:

```
/* ptinrect: επιστρέφει 1 αν το p είναι μέσα στο r,
    αλλιώς επιστρέφει 0 */
int ptinrect(struct point p, struct rect r)
  {
      return p.x >= r.pt1.x && p.x < r.pt2.x
          && p.y >= r.pt1.y && p.y < r.pt2.y;
  }
```

Αυτό προϋποθέτει ότι το ορθογώνιο αναπαριστάνεται με μια καθιερωμένη μορφή, όπου οι συντεταγμένες του pt1 είναι μικρότερες από τις συντεταγμένες του pt2. Η επόμενη συνάρτηση επιστρέφει ένα ορθογώνιο που έχει εγγυημένα κανονική μορφή:

```
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

/* canonrect: κανονικοποιεί τις συντεταγμένες
   ορθογωνίου */
struct rect canonrect(struct rect r)
{
   struct rect temp;

   temp.pt1.x = min(r.pt1.x, r.pt2.x);
   temp.pt1.y = min(r.pt1.y, r.pt2.y);
   temp.pt2.x = max(r.pt1.x, r.pt2.x);
   temp.pt1.y = max(r.pt1.y, r.pt2.y);
   return temp;
}
```

Αν μια μεγάλη δομή πρόκειται να μεταβιβαστεί σε μια συνάρτηση, είναι γενικά πιο αποδοτική η μεταβίβαση ενός δείκτη, και όχι η αντιγραφή ολόκληρης της δομής. Οι δείκτες για δομές είναι όπως ακριβώς και οι δείκτες για κανονικές μεταβλητές. Η δήλωση

```
struct point *pp;
```

λέει ότι ο pp είναι δείκτης για μια δομή τύπου struct point. Αν ο pp δείχνει σε μια δομή point, τότε το *pp είναι η δομή, ενώ τα (*pp).x και (*pp).y είναι τα μέλη. Για να χρησιμοποιήσουμε το δείκτη pp, μπορούμε για παράδειγμα να γράψουμε,

```
struct point origin, *pp;

pp = &origin;
printf("Η αρχή είναι (%d,%d)\n", (*pp).x, (*pp).y);
```

Οι παρενθέσεις είναι απαραίτητες στο `(*pp).x` επειδή η προτεραιότητα του τελεστή μέλους δομής `"."` είναι υψηλότερη από εκείνη του τελεστή `*`. Η παράσταση `*pp.x` σημαίνει `*(pp.x)`, που εδώ δεν είναι αποδεκτή επειδή το `x` δεν είναι δείκτης.

Οι δείκτες για δομές χρησιμοποιούνται τόσο συχνά ώστε ως συντομογραφία παρέχεται ένας εναλλακτικός συμβολισμός. Αν ο `p` είναι δείκτης σε δομή, τότε το

`p->`*μέλος-της-δομής*

αναφέρεται στο συγκεκριμένο μέλος. (Ο τελεστής `->` είναι ένα σύμβολο πλην που ακολουθείται αμέσως μετά από το σύμβολο `>`.) Έτσι, στο παράδειγμά μας θα μπορούσαμε να γράψουμε

```
printf("Η αρχή είναι (%d,%d)\n", pp->x, pp->y);
```

Τόσο ο τελεστής `.` όσο και ο τελεστής `->` έχουν προσεταιριστικότητα από αριστερά προς τα δεξιά· έτσι, αν έχουμε

```
struct rect r, *rp = &r;
```

οι τέσσερις επόμενες παραστάσεις είναι ισοδύναμες:

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```

Οι τελεστές δομών `.` και `->`, μαζί με τις παρενθέσεις `()` για κλήσεις συναρτήσεων και τις αγκύλες `[]` για δείκτες θέσης πίνακα, βρίσκονται στην κορυφή της ιεραρχίας προτεραιότητας και επομένως συνδέονται ισχυρά με τους τελεστέους. Για παράδειγμα, με δεδομένη τη δήλωση

```
struct {
   int len;
   char *str;
} *p;
```

η εντολή

```
++p->len
```

αυξάνει το `len`, και όχι το `p`, επειδή οι εννοούμενες παρενθέσεις τη μετατρέπουν σε `++(p->len)`. Οι παρενθέσεις μπορούν να χρησιμοποιηθούν για να αλλάξουν τη σύνδεση: η παράσταση `(++p)->len` αυξάνει το `p` πριν προσπελάσει το `len`, και η παράσταση `(p++)->len` αυξάνει το `p` αφού προσπελάσει το `len`. (Το τελευταίο ζευγάρι παρενθέσεων δεν είναι απαραίτητο.)

Με τον ίδιο τρόπο, η παράσταση `*p->str` επιστρέφει αυτό στο οποίο δείχνει ο δείκτης `str`· η παράσταση `*p->str++` αυξάνει τον `str` μετά την προσπέλαση του στοιχείου στο οποίο δείχνει ο `str` (όπως ακριβώς στην παράσταση `*s++`), η παράσταση `(*p->str)++` αυξάνει αυτό στο οποίο δείχνει ο `str`, και η παράσταση `*p++->str` αυξάνει τον `p` μετά την προσπέλαση του στοιχείου στο οποίο δείχνει ο `str`.

6.3 Πίνακες δομών

Ας γράψουμε ένα πρόγραμμα για τη μέτρηση των εμφανίσεων κάθε δεσμευμένης λέξης της C. Χρειαζόμαστε έναν πίνακα αλφαριθμητικών όπου θα αποθηκεύονται τα ονόματα, καθώς και έναν πίνακα ακεραίων για τις μετρήσεις. Μια πιθανότητα είναι να χρησιμοποιήσουμε δύο παράλληλους πίνακες, τους `keyword` και `keycount`, όπως εδώ

```
char *keyword[NKEYS];
int keycount[NKEYS];
```

Όμως, το ίδιο το γεγονός ότι οι πίνακες είναι παράλληλοι μας δίνει την ιδέα για μια διαφορετική οργάνωση: τον πίνακα δομών. Κάθε καταχώρηση δεσμευμένης λέξης είναι ένα ζεύγος:

```
char *word;
int count;
```

και υπάρχει ένας πίνακας ζευγών. Η δήλωση δομής

```
struct key {
   char *word;
   int count;
} keytab[NKEYS];
```

δηλώνει μια δομή τύπου `key`, ορίζει έναν πίνακα `keytab` με δομές αυτού του τύπου, και δεσμεύει μνήμη για τις δομές αυτές. Κάθε στοιχείο του πίνακα είναι δομή. Αυτό θα μπορούσε να γραφτεί και ως

```
struct key {
   char *word;
   int count;
};

struct key keytab[NKEYS];
```

Αφού ο πίνακας keytab περιέχει ένα σταθερό σύνολο ονομάτων, είναι ευκολότερο να τον κάνουμε εξωτερική μεταβλητή και να του αποδώσουμε μια και καλή αρχικές τιμές με τον ορισμό του. Η απόδοση αρχικών τιμών σε μια δομή γίνεται όπως και προηγουμένως — ο ορισμός ακολουθείται από μια λίστα αρχικών τιμών που περικλείονται μέσα σε άγκιστρα:

```
struct key {
   char *word;
   int count;
} keytab[] = {
     "auto", 0,
     "break", 0,
     "case", 0,
     "char", 0,
     "const", 0,
     "continue", 0,
     "default", 0,
     /* ... */
     "unsigned", 0,
     "void", 0,
     "volatile", 0,
     "while", 0
};
```

Οι αρχικές τιμές παραθέτονται σε ζεύγη που αντιστοιχούν στα μέλη της δομής. Πιο ακριβές θα ήταν να περικλείσουμε τις αρχικές τιμές για κάθε "γραμμή πίνακα" ή δομή σε άγκιστρα, όπως στο

```
{ "auto", 0 },
{ "break", 0 },
{ "case", 0 },
...
```

όμως τα εσωτερικά άγκιστρα δεν είναι αναγκαία όταν οι αρχικές τιμές είναι απλές μεταβλητές ή αλφαριθμητικά και υπάρχουν όλες οι τιμές. Όπως συνήθως, ο αριθμός των καταχωρίσεων στον πίνακα keytab θα υπολογιστεί με βάση τις αρχικές τιμές, και έτσι οι αγκύλες [] είναι κενές.

Το πρόγραμμα μέτρησης δεσμευμένων λέξεων ξεκινά με τον ορισμό του πίνακα keytab. Η κύρια ρουτίνα διαβάζει την είσοδο με επαναληπτικές κλήσεις της συνάρτησης getword, που προσκομίζει μία λέξη τη φορά. Κάθε λέξη αναζητείται στον πίνακα keytab με μια παραλλαγή της συνάρτησης δυαδικής αναζήτησης που γράψαμε στο Κεφάλαιο 3. Η λίστα των δεσμευμένων λέξεων πρέπει να είναι ταξινομημένη στον πίνακα κατά αύξουσα σειρά.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAXWORD 100

int getword(char *, int);
int binsearch(char *, struct key *, int);

/* μέτρηση των δεσμευμένων λέξεων της C */
main()
{
   int n;
   char word[MAXWORD];

   while (getword(word, MAXWORD) != EOF)
     if (isalpha(word[0]))
         if ((n = binsearch(word, keytab, NKEYS)) >= 0)
           keytab[n].count++;
   for (n = 0; n < NKEYS; n++)
     if (keytab[n].count > 0)
        printf("%4d %s\n",
           keytab[n].count, keytab[n].word);
   return 0;
}

/* binsearch: αναζήτηση λέξης στα tab[0]...tab[n-1] */
int binsearch(char *word, struct key tab[], int n)
{
   int cond ;
   int low, high, mid;

   low = 0;
   high = n - 1;
   while (low <= high) {
     mid = (low+high) / 2;
     if ((cond = strcmp(word, tab[mid].word)) < 0)
         high = mid - 1;
     else if (cond > 0)
         low = mid + 1;
     else
         return mid ;
```

```
    }
    return -1;
}
```

Θα δείξουμε τη συνάρτηση getword σε λίγο· προς το παρόν, αρκεί να πούμε ότι κάθε κλήση στην getword βρίσκει μια λέξη, η οποία αντιγράφεται στον πίνακα που κατονομάζει το πρώτο της όρισμα.

Η ποσότητα NKEYS είναι ο αριθμός των δεσμευμένων λέξεων στον πίνακα keytab. Αν και θα μπορούσαμε να τις μετρήσουμε με το χέρι, είναι πολύ ευκολότερο και ασφαλέστερο να το κάνουμε με το μηχάνημα, ειδικά αν υπάρχει περίπτωση να αλλάξει η λίστα. Μια δυνατότητα θα ήταν να τερματίζουμε τη λίστα των αρχικών τιμών με ένα μηδενικό δείκτη, και μετά να διατρέχουμε τον keytab μέχρι να συναντήσουμε το τέλος.

Αλλά αυτό είναι περισσότερο απ' ό,τι χρειάζεται εδώ, αφού το μέγεθος του πίνακα καθορίζεται πλήρως κατά τη διάρκεια της μεταγλώττισης. Το μέγεθος του πίνακα είναι το μέγεθος μιας καταχώρισης επί το πλήθος των καταχωρίσεων, έτσι ο αριθμός των καταχωρίσεων είναι ακριβώς

μέγεθος του keytab / *μέγεθος του* struct key

Η C διαθέτει ένα μοναδιαίο τελεστή χρόνου μεταγλώττισης που ονομάζεται sizeof, ο οποίος μπορεί να χρησιμοποιείται για τον υπολογισμό του μεγέθους οποιουδήποτε αντικειμένου. Οι παραστάσεις

sizeof *αντικείμενο*

και

sizeof(*όνομα τύπου*)

δίνουν έναν ακέραιο ίσο σε μέγεθος με τα byte του καθοριζόμενου αντικειμένου ή τύπου δεδομένων. (Αυστηρά, ο sizeof παράγει μια απρόσημη ακέραια τιμή της οποίας ο τύπος size_t ορίζεται στο αρχείο κεφαλίδας <stddef.h>.) Το αντικείμενο μπορεί να είναι μια μεταβλητή, ένας πίνακας, ή μια δομή. Το όνομα τύπου μπορεί να είναι το όνομα ενός βασικού τύπου δεδομένων όπως int ή double, ή ένας παραγόμενος τύπος δεδομένων όπως μια δομή ή ένας δείκτης.

Στην περίπτωσή μας ο αριθμός των δεσμευμένων λέξεων είναι το μέγεθος του πίνακα δια το μέγεθος ενός αντικειμένου. Ο υπολογισμός αυτός χρησιμοποιείται σε μια εντολή #define για τον καθορισμό της τιμής του NKEYS:

```
#define NKEYS (sizeof keytab / sizeof(struct key))
```

Ένας άλλος τρόπος για να το γράψουμε αυτό είναι να διαιρέσουμε το μέγεθος του πίνακα με το μέγεθος ενός συγκεκριμένου στοιχείου:

```
#define NKEYS (sizeof keytab / sizeof keytab[0])
```

Αυτό έχει το πλεονέκτημα ότι δεν χρειάζεται τροποποίηση αν αλλάξει ο τύπος δεδομένων.

Ο τελεστής sizeof δεν μπορεί να χρησιμοποιείται σε γραμμές #if, επειδή ο προεπεξεργαστής δεν αναλύει ονόματα τύπων. Όμως, η παράσταση στην εντολή #define δεν υπολογίζεται από τον προεπεξεργαστή, έτσι ο κώδικας αυτός είναι έγκυρος.

Ας δούμε τώρα τη συνάρτηση getword. Έχουμε γράψει μια συνάρτηση getword που είναι πιο γενική απ' ό,τι χρειάζεται για το συγκεκριμένο πρόγραμμα, όμως δεν είναι περίπλοκη. Η getword φέρνει την επόμενη "λέξη" της εισόδου, όπου λέξη είναι μια σειρά γραμμάτων και ψηφίων που ξεκινά από γράμμα, ή ένας μεμονωμένος χαρακτήρας μη λευκού διαστήματος. Τιμή της συνάρτησης είναι ο πρώτος χαρακτήρας της λέξης, το EOF για το τέλος του αρχείου, ή ο ίδιος ο χαρακτήρας, εφόσον δεν είναι αλφαβητικός.

```
/* getword: φέρνει την επόμενη λέξη ή
   τον επόμενο χαρακτήρα της εισόδου */
int getword(char *word, int lim)
{
   int c, getch(void);
   void ungetch(int);
   char *w = word;

   while (isspace(c = getch()))
        ;
   if (c != EOF)
        *w++ = c;
   if (!isalpha(c)) {
        *w = '\0';
        return c;
   }
   for ( ; --lim > 0; w++)
        if (!isalnum(*w = getch())) {
            ungetch(*w);
            break;
        }
   *w = '\0';
   return word[0];
}
```

Η getword χρησιμοποιεί τις συναρτήσεις getch και ungetch που γράψαμε στο Κεφάλαιο 4. Όταν σταματάει η περισυλλογή ενός αλφαριθμητικού συμβόλου, η getword έχει προχωρήσει κατά ένα χαρακτήρα περισσότερο απ' ό,τι πρέπει. Η κλήση της ungetch επαναπροωθεί το χαρακτήρα στην είσοδο, για την επόμενη κλήση. Η getword χρησιμοποιεί επίσης την isspace για να παρα-

βλέπει τα λευκά διαστήματα, την isalpha για να αναγνωρίζει τα γράμματα, και την isalnum για να αναγνωρίζει γράμματα και αριθμητικά ψηφία. Όλες αυτές οι συναρτήσεις προέρχονται από το αρχείο κεφαλίδας <ctype.h>.

Άσκηση 6.1 Η εκδοχή μας της getword δεν χειρίζεται σωστά τις υπογραμμίσεις, τις αλφαριθμητικές σταθερές, τα σχόλια, ή τις γραμμές ελέγχου του προεπεξεργαστή. Γράψτε μια καλύτερη εκδοχή.

6.4 Δείκτες σε δομές

Για να παρουσιάσουμε μερικά από τα σημεία που πρέπει να λαμβάνονται υπόψη στη χρήση δεικτών για δομές και πινάκων με δομές, θα ξαναγράψουμε το πρόγραμμα μέτρησης δεσμευμένων λέξεων χρησιμοποιώντας αυτή τη φορά δείκτες διευθύνσεων, αντί για αριθμοδείκτες θέσης πίνακα.

Η εξωτερική δήλωση του keytab δεν χρειάζεται τροποποίηση, όμως η συνάρτηση main και η binsearch χρειάζονται τροποποίηση.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAXWORD 100

int getword(char *, int);
struct key *binsearch(char *, struct key *, int);

/* μέτρηση των δεσμευμένων λέξεων της C -έκδοση με δείκτες */
main()
{
   char word[MAXWORD];
   struct key *p;

   while (getword(word, MAXWORD) != EOF)
     if (isalpha(word[0]))
        if ((p=binsearch(word, keytab, NKEYS)) != NULL)
         p->count++;
   for (p = keytab; p < keytab + NKEYS; p++)
     if (p->count > 0)
        printf("%4d %s\n", p->count, p->word);
   return 0;
}

/*
binsearch: αναζήτηση λέξης στα tab[0]...tab[n-1] */
struct key *binsearch(char *word,struct key *tab,int n)
{
   int cond;
   struct key *low = &tab[0];
   struct key *high = &tab[n];
```

Η Γλώσσα Προγραμματισμού C

Δεύτερη Βελτιωμένη Έκδοση

Brian W. Kernighan / Dennis M. Ritchie

Το βιβλίο *Η γλώσσα προγραμματισμού C* των Dennis Ritchie και Brian Kernighan θεωρείται ένα από τα πιο έγκυρα βιβλία αναφοράς για τη γλώσσα C και έχει μεταφραστεί σε περισσότερες από 20 γλώσσες. Η γλώσσα προγραμματισμού C δεν είναι μεγάλη και γι' αυτό δεν εξυπηρετείται καλά από ένα μεγάλο βιβλίο. Το βιβλίο καλύπτει πλήρως τη γλώσσα C με εύληπτο τρόπο, όπως αυτή ορίζεται από το πρότυπο ANSI. Στα παραδείγματα περιλαμβάνονται πλήρη προγράμματα, παρόμοια με αυτά που θα συναντήσει ο αναγνώστης κατά τη χρήση της γλώσσας, ενώ δίνεται έμφαση στις καλές πρακτικές προγραμματισμού. Η παρουσίαση των χαρακτηριστικών που είναι σημαντικά για τον προγραμματισμό σε PC έχει βελτιωθεί, ενώ τα αρχικά παραδείγματα έχουν αναθεωρηθεί και έχουν προστεθεί νέα σε αρκετά κεφάλαια.

Οι συγγραφείς

Ο **Dennis M. Ritchie** σχεδίασε και εφάρμοσε τη γλώσσα προγραμματισμού C, ενώ είναι από τους δημιουργούς του λειτουργικού συστήματος Unix, η ανάπτυξη του οποίου συνδέεται στενά με την ανάπτυξη της γλώσσας C. Συμμετείχε επίσης στη δημιουργία των λειτουργικών συστημάτων Multics, Plan 9, και Inferno, καθώς και της γλώσσας προγραμματισμού Limbo. Έλαβε τα βραβεία Turing (1983) και National Medal of Technology (1998).

Ο **Brian W. Kernighan** εργάστηκε στα εργαστήρια Bell και συνεργάστηκε με τους δημιουργούς του λειτουργικού συστήματος Unix, Ken Thompson και Dennis Ritchie, συμβάλλοντας σημαντικά στην ανάπτυξή του. Έχει επίσης συμμετάσχει στη δημιουργία των γλωσσών προγραμματισμού AWK και AMPL. Έχει γράψει πολλά προγράμματα για Unix. Έχει δηλώσει ότι: "*...αν ναυαγούσα σε κάποιο ερημονήσι και είχα τη δυνατότητα να έχω στη διάθεσή μου μια γλώσσα προγραμματισμού, αυτή θα ήταν η C*".

ISBN 978-960-461-132-4

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ

Στοιχνάρη 27Β, 10682 Αθήνα, Τηλ. 210-3832044

Επισκεφθείτε μας στο Internet:
www.klidarithmos.gr

9 789604 611324