1

Γλώσσες προγραμματισμού

1.1. Εισαγωγή

Τα προγράμματα των υπολογιστών είναι μαγεία εν δράσει. Το κάθε πρόγραμμα μοιάζει με ένα περίπλοκο, αλλά και καλαίσθητο, μαγικό ξόρκι· και το καλύτερο είναι ότι με την εκτέλεση ενός προγράμματος ανακαλύπτουμε ότι *τελικά τα μάγια πιάνουν*! Ακόμα και όταν το πρόγραμμά μας δεν κατορθώνει ακριβώς αυτό που είχαμε κατά νου, εντούτοις κάτι κατορθώνει – πράγμα που σίγουρα δεν ισχύει για τα περισσότερα ξόρκια. Και η απόλαυση αυτής της έμπρακτης μαγείας είναι η δύναμη που ωθεί πολλούς αρχάριους στον προγραμματισμό, και που συνεχίζει να ανταμείβει τους έμπειρους προγραμματιστές. Ο συγγραφέας, τουλάχιστον, βιώνει αμείωτη αυτήν την απόλαυση εδώ και 20 χρόνια ενασχόλησης με τον προγραμματισμό.

Το βιβλίο αυτό πραγματεύεται τις γλώσσες προγραμματισμού. Περιέχει διδακτικές εισαγωγές σε τρεις γλώσσες προγραμματισμού: την ML, την Java και την Prolog.[1] Οι γλώσσες αυτές είναι πολύ διαφορετικές μεταξύ τους, και αν γνωρίσετε έστω και ένα τμήμα καθεμίας από αυτές, θα έχετε τρία σαφώς διαφορετικά στίγματα στη διάθεσή σας για να «πλοηγηθείτε» στις αρχές των γλωσσών προγραμματισμού. Εάν θελήσετε να ασχοληθείτε και πρακτικά με αυτές τις γλώσσες, δεν θα δυσκολευτείτε καθόλου: υπάρχουν καλές και δωρεάν υλοποιήσεις και για τις τρεις τους, σε ποικιλία υπολογιστικών πλαισίων. Ανάμεσα στα διδακτικά κεφάλαια θα βρείτε κεφάλαια με περισσότερο φιλοσοφικό προσανατολισμό, τα οποία πραγματεύονται με πιο αφηρημένο τρόπο διάφορα σοβαρά ζητήματα που αφορούν τις γλώσσες προγραμματισμού. Αν και αυτά τα κεφάλαια έχουν πιο αφηρημένο χαρακτήρα, δεν απαιτούν ιδιαίτερο μαθηματικό υπόβαθρο. Οι γλώσσες προγραμματισμού στηρίζονται, βέβαια, σε πολύ ενδιαφέροντα και ιδιαίτερα κομψά μαθηματικά, αλλά προσφέρουν επίσης άφθονο υλικό μελέτης το οποίο δεν απαιτεί μαθηματικές γνώσεις δυσπρόσιτες ή και αδιάφορες σε πολλούς αναγνώστες.

---

1 Στα έντυπα εγχειρίδια των παλαιότερων γλωσσών, τα ονόματα των γλωσσών αναγράφονταν συνήθως με κεφαλαία γράμματα: FORTRAN, COBOL και BASIC. Στα νεώτερα εγχειρίδια, ακόμη και σε αυτά που αφορούν διαλέκτους των παλαιότερων γλωσσών, τα ονόματα αναγράφονται συνήθως σε μικτή γραφή: Fortran. Η σύμβαση που ακολουθούμε σε αυτό το βιβλίο είναι η εξής: για τα ονόματα που προφέρονται σαν λέξεις (π.χ. Java και Prolog) χρησιμοποιούμε μικτή γραφή, ενώ για τα ονόματα που προφέρονται σαν ακολουθίες γραμμάτων του αγγλικού αλφαβήτου (όπως η ML, που προφέρεται «εμ-ελ»), χρησιμοποιούμε κεφαλαιογράμματη γραφή.

Πριν συνεχίσετε όμως, θα πρέπει να λάβετε υπ' όψιν ότι το βιβλίο αυτό προϋποθέτει ικανοποιητική γνώση μίας τουλάχιστον γλώσσας προγραμματισμού, σε επίπεδο που αντιστοιχεί περίπου σε μια εισαγωγή διάρκειας δύο διδακτικών εξαμήνων. Δεν έχει σημασία ποια γλώσσα γνωρίζετε. Εάν όμως δεν έχετε προγραμματίσει ποτέ στη ζωή σας, τότε αυτό το βιβλίο δεν είναι ό,τι καταλληλότερο για να αρχίσετε.

Στο υπόλοιπο αυτού του εισαγωγικού κεφαλαίου θα εξετάσουμε εκείνα τα στοιχεία που κάνουν τις γλώσσες προγραμματισμού ένα τόσο ενδιαφέρον θέμα: την εκπληκτική ποικιλία αυτών των γλωσσών, τις παράξενες αντιμαχίες που προκαλούν, την αξιοπερίεργη εξέλιξή τους, και τις πολλαπλές διασυνδέσεις τους με τους υπόλοιπους κλάδους της πληροφορικής.

1.2. Η εκπληκτική ποικιλία

Ένα από τα στοιχεία που κάνουν τις γλώσσες προγραμματισμού τόσο γοητευτικό αντικείμενο μελέτης είναι η ποικιλομορφία τους. Ας ρίξουμε μια ματιά σε τέσσερεις γλώσσες που ανήκουν σε εντελώς ανόμοια είδη· οι τρεις από αυτές είναι οι βασικές γλώσσες με τις οποίες θα ασχοληθούμε σε αυτό το βιβλίο.

Προστακτικές γλώσσες

Ας δούμε ένα παράδειγμα προστακτικής γλώσσας, της γλώσσας C. Πρόκειται για μια συνάρτηση `factorial(n)`, που υπολογίζει το παραγοντικό (factorial) ενός φυσικού αριθμού n:

```
int factorial(int n) {
  int sofar = 1;
  while (n > 0) sofar *= n--;
  return sofar;
}
```

Το παραπάνω παράδειγμα περιέχει τις δύο «σφραγίδες γνησιότητας» κάθε προστακτικής γλώσσας: την τιμοδότηση και την επανάληψη. Η εντολή «`sofar *= n--;`» της γλώσσας C τιμοδοτεί τη μεταβλητή `sofar`. Η μεταβλητή αυτή έχει κάποια τρέχουσα τιμή η οποία μεταβάλλεται κάθε φορά που πραγματοποιείται μια τιμοδότηση. Η εντολή επιδρά επίσης στη μεταβλητή `n`, μειώνοντας κάθε φορά την τιμή της κατά μία μονάδα. Ο βρόχος «`while`» επαναλαμβάνει διαρκώς την εντολή. Τελικά, σε κάποιο βήμα αυτής της επανάληψης, η τρέχουσα τιμή της μεταβλητής `n` θα γίνει μηδέν, και η επανάληψη θα σταματήσει. Καθώς οι τιμές των μεταβλητών αλλάζουν σε κάθε βήμα, η σειρά εκτέλεσης των εντολών του προγράμματος έχει καίρια σημασία.

Οι έννοιες που μόλις αναφέραμε είναι τόσο στοιχειώδεις, που περνούν απαρατήρητες από τους περισσότερους προγραμματιστές της γλώσσας C: για αυτούς είναι προφανές ότι η σειρά εκτέλεσης των εντολών παίζει καίριο ρόλο, και εξίσου προφανές ότι οι τιμές των μεταβλητών μεταβάλλονται. Υπάρχουν όμως πολλές γλώσσες προγραμματισμού για τις οποίες όλα τα παραπάνω δεν έχουν κανένα νόημα· γλώσσες στις οποίες δεν υπάρχουν ούτε τιμοδοτικές εντολές, ούτε επαναληπτικές εντολές, ούτε η έννοια της αλλαγής της «τρέχουσας τιμής» μιας μεταβλητής.

Συναρτησιακές γλώσσες

Ας δούμε την ίδια συνάρτηση (του παραγοντικού) υλοποιημένη στη γλώσσα ML:

```
fun factorial x =
  if x <= 0 then 1 else x * factorial(x-1);
```

Το παραπάνω παράδειγμα περιλαμβάνει δύο από τις «σφραγίδες γνησιότητας» των συναρτησιακών γλωσσών: την αναδρομή και τις μονότιμες μεταβλητές[2]. Η αναδρομή είναι μια προγραμματιστική τεχνική τόσο φυσική στους προγραμματιστές της ML, όσο φυσικές είναι οι επαναληπτικές εντολές στους προγραμματιστές της C.

Η ίδια συνάρτηση υλοποιημένη στη γλώσσα Lisp θα είχε ως εξής:

```
(defun factorial (x)
  (if (<= x 0) 1 (* x (factorial (- x 1)))))
```

Όπως βλέπετε, η Lisp έχει ιδιόρρυθμη σύνταξη. Αυτή η συντακτική διαφορά είναι όμως επιφανειακή. Σε βαθύτερο επίπεδο, η συνάρτηση `factorial` γραμμένη στη Lisp και η συνάρτηση `factorial` γραμμένη στην ML σχετίζονται μεταξύ τους πολύ περισσότερο απ' όσο σχετίζεται η καθεμία από αυτές με τη συνάρτηση `factorial` γραμμένη στη γλώσσα C: και οι δύο είναι γραμμένες στο συναρτησιακό ύφος, χωρίς τιμοδοτικές ή επαναληπτικές εντολές.

Τα δύο παραπάνω παραδείγματα ίσως φαίνονται πιο κομψά από την εκδοχή της C, αλλά μια τέτοια σύγκριση δεν είναι δίκαιη. Το συναρτησιακό είδος προγραμματισμού ταιριάζει ιδιαίτερα σε συναρτήσεις όπως αυτή του παραγοντικού. Σε άλλα είδη προβλημάτων, όπως π.χ. ο πολλαπλασιασμός πινάκων, το πλεονέκτημα θα το είχαν οι προστακτικές γλώσσες προγραμματισμού.

Λογικοκεντρικές γλώσσες

Η συνάρτηση του παραγοντικού, ενώ είναι το καταλληλότερο παράδειγμα για τη γλώσσα ML, είναι ίσως το χειρότερο για την Prolog. Παρά ταύτα ας δούμε τι μορφή έχει στην Prolog:

```
factorial(X,1) :-
  X =:= 1.
factorial(X,F) :-
  X > 1,
  NewX is X - 1,
  factorial(NewX,NewF),
  F is X * NewF.
```

2 Σ.τ.Μ.: *Μεταβλητές* από το *variables* – όσο βέβαια τα συμφραζόμενα επιτρέπουν να ονομάζεται «μεταβλητή» κάτι που μένει σταθερό. Οι «μεταβλητές» των συναρτησιακών γλωσσών (όπως η ML) είναι ακριβέστερα *επώνυμες σταθερές*, γι' αυτό ο σχετικός όρος της ML, που θα δοθεί αργότερα, είναι *val από το value = τιμή* και όχι *var*, από το *variable = μεταβλητή*.

Οι πρώτες δύο γραμμές εκφράζουν έναν κανόνα που επιτρέπει στο σύστημα της Prolog να συμπεράνει ότι όταν το X ισούται με 1 το παραγοντικό του X είναι 1. Οι υπόλοιπες πέντε γραμμές κώδικα καθορίζουν έναν γενικό τρόπο για να διαπιστώνει κανείς ότι το παραγοντικό του X ισούται με κάποια δεδομένη τιμή. Συγκεκριμένα: «Για να αποδείξεις ότι το παραγοντικό του X ισούται με F, αρκεί να κάνεις τα εξής: να αποδείξεις ότι το X είναι μεγαλύτερο του 1· να αποδείξεις ότι το NewX είναι μικρότερο του X κατά 1· να αποδείξεις ότι το παραγοντικό του NewX ισούται με NewF· και τέλος να αποδείξεις ότι το F ισούται με X επί NewF». Η διατύπωση ενός προγράμματος μέσω κανόνων λογικού συμπερασμού είναι η «σφραγίδα γνησιότητας» του λογικού προγραμματισμού. Αν και αυτό το είδος προγραμματισμού δεν είναι το καταλληλότερο για τον υπολογισμό μαθηματικών συναρτήσεων, υπάρχουν κατηγορίες προβλημάτων στα οποία υπερέχει ξεκάθαρα. Παραδείγματα τέτοιων προβλημάτων θα δούμε από το Κεφάλαιο 19 και παρακάτω.

Οντοστρεφείς γλώσσες

Η συνάρτηση του παραγοντικού γραμμένη στη γλώσσα Java φαίνεται σχεδόν ίδια με την εκδοχή της στη γλώσσα C. Η Java όμως είναι μια οντοστρεφής γλώσσα, πράγμα που σημαίνει ότι αφ' ενός είναι προστακτικού τύπου, και αφ' ετέρου έχει σχεδιαστεί ώστε να διευκολύνει την επίλυση των διαφόρων προβλημάτων μέσω οντοτήτων (ή, αλλιώς, αντικειμένων). Ονομάζουμε *οντότητα* μια (συνήθως μικρή) δέσμη δεδομένων η οποία «γνωρίζει» πώς να χειρίζεται τον εαυτό της. Για παράδειγμα, ας δούμε τον ορισμό σε Java μιας οντότητας που φέρει έναν ακέραιο αριθμό, και «γνωρίζει» πώς να αναφέρει τόσο τον αριθμό αυτό, όσο και το παραγοντικό του.

```
public class MyInteger {
  private int value;
  public MyInteger(int value) {
    this.value = value;
  }
  public int getValue() {
    return value;
  }
  public MyInteger getFactorial() {
    return new MyInteger(factorial(value));
  }
  private int factorial(int n) {
    int sofar = 1;
    while (n > 1) sofar *= n--;
    return sofar;
  }
}
```

Το παραπάνω παράδειγμα οντοστρεφούς προγραμματισμού φαίνεται φλύαρο σε σχέση με τα προηγούμενα, αλλά και πάλι η σύγκριση δεν είναι δίκαιη: ο οντοστρε-

φής προγραμματισμός έχει σχεδιαστεί ώστε να διευκολύνει την οργανωμένη σύνταξη προγραμμάτων πολύ μεγάλου μεγέθους, και ως εκ τούτου δεν φανερώνει τα προτερήματά του σε παραδείγματα μικρού μεγέθους.

Είδαμε λοιπόν παραδείγματα από τέσσερεις οικογένειες γλωσσών: τις προστακτικές (όπως η C), τις συναρτησιακές (όπως η ML), τις λογικοκεντρικές (όπως η Prolog), και τις οντοστρεφείς (όπως η Java). Κάθε γλώσσα προγραμματισμού μπορεί, με λίγη προσπάθεια, να καταταγεί σε κάποια από αυτές τις τέσσερεις κατηγορίες. Αυτές οι κατηγορίες δεν είναι όμως προσδιορισμένες με αυστηρότητα, και γι' αυτό δεν είναι πάντοτε σαφές ποιος είναι ο ορθότερος τρόπος ταξινόμησης μιας γλώσσας. Υπάρχουν πάμπολλες γλώσσες που κινούνται στα όρια μεταξύ των κατηγοριών αυτών. Στην πράξη συναντάμε περισσότερες κατηγορίες γλωσσών, όχι μόνο αυτές τις τέσσερεις. Οι διάφορες γλώσσες προγραμματισμού έχουν κατά καιρούς χαρακτηριστεί ως εφαρμοστικές, συγχρονικές, περιοριστικές, δηλωτικές, οριστικές, διαδικαστικές, σεναριογραφικές, μονοτιμοδοτικές και πάει λέγοντας.

Μάλιστα, ορισμένες από αυτές είναι τόσο ιδιόμορφες που η ένταξή τους σε μια κατηγορία στερείται νοήματος. Ας εξετάσουμε π.χ. τη γλώσσα Forth. Ο προγραμματισμός της συνάρτησης του παραγοντικού σε αυτή τη γλώσσα θα είχε ως εξής:

```
: FACTORIAL
   1 SWAP BEGIN ?DUP WHILE TUCK * SWAP 1- REPEAT ;
```

Η Forth είναι μια στοιβοστρεφής γλώσσα, όπως λ.χ. η PostScript είναι σελιδοστρεφής. Η μονολεκτική εντολή `SWAP` της Forth εναλλάσσει τα δύο κορυφαία στοιχεία της στοίβας που διατηρεί η γλώσσα αυτή. Θα μπορούσαμε να αποκαλέσουμε την Forth προστακτική γλώσσα, αλλά αυτό δεν θα είχε ιδιαίτερο νόημα, καθώς έχει ελάχιστα κοινά στοιχεία με τις περισσότερες άλλες προστακτικές γλώσσες.

Θεωρήστε επίσης την APL. Για να εκφραστεί η συνάρτηση παραγοντικού στη γλώσσα αυτή, αρκεί η παρακάτω έκφραση:

```
× / ⍳ X
```

Η APL φημίζεται για τη χρήση πάμπολλων ειδικών χαρακτήρων που απουσιάζουν από τα συνηθισμένα πληκτρολόγια. Η παραπάνω έκφραση δηλώνει ότι το `X` θα πρέπει να αναπτυχθεί σε μια ακολουθία διαδοχικών ακεραίων από το 1 έως το `X`, και όλοι αυτοί οι αριθμοί να πολλαπλασιαστούν μεταξύ τους. (Στην πράξη, δεν θα χρειαζόσασταν αυτήν την έκφραση, διότι στην APL μπορούμε να δηλώσουμε το παραγοντικό του `X` απλώς ως `!X`.) Θα μπορούσαμε να αποκαλέσουμε την APL συναρτησιακή γλώσσα, αλλά και πάλι δεν θα είχε νόημα: η APL έχει ελάχιστα κοινά σημεία με τις περισσότερες άλλες συναρτησιακές γλώσσες.

1.3. Οι παράξενες αντιμαχίες

Υπάρχουν κάποια γνωστικά αντικείμενα που φαίνονται εκ του φυσικού τους αμφιλεγόμενα: η βιολογική εξέλιξη, η αρχική Μεγάλη Έκρηξη του σύμπαντος, η ανθρώπινη σεξουαλικότητα – γενικά, οποιοδήποτε θέμα απασχολεί τακτικά την επιστημονική στήλη της εφημερίδας *The New York Times* είναι βέβαιο ότι θα προκαλέσει

πολλές αντιδικίες. Ισχύει όμως το ίδιο και για τις γλώσσες προγραμματισμού; Το δικό μας πεδίο μελέτης σπάνια διεισδύει μέχρι τα πρωτοσέλιδα των εφημερίδων, και είναι μάλλον ελάχιστοι αυτοί που το γνωρίζουν και που ενδιαφέρονται για αυτό. Παρά ταύτα, και παραδόξως, το θέμα των γλωσσών προγραμματισμού εγείρει συχνά θερμότατες διαμάχες.

Κατ' αρχάς, για κάθε γλώσσα προγραμματισμού θα βρείτε οπαδούς της πρόθυμους να υπερασπιστούν αυτήν την προτίμησή τους ενάντια σε κάθε άλλη γλώσσα. Μερικοί οπαδοί της ML «είναι στα μαχαίρια» με οπαδούς της γλώσσας Haskell. Οι οπαδοί της Forth θα πικραθούν, χωρίς πάντως να εκπλαγούν, όταν διαπιστώσουν ότι κρατούν άλλο ένα εγχειρίδιο που αγνοεί τη δική τους αγαπημένη γλώσσα. Διάφοροι οπαδοί της Prolog είναι αδύνατον να κατανοήσουν για ποιο λόγο ο λογικός προγραμματισμός δεν έχει υιοθετηθεί από όλους τους προγραμματιστές. Και υπάρχουν οπαδοί της Fortran πλήρως πεπεισμένοι ότι η δική τους προτίμηση δεν είναι μόνο η πρώτη ιστορικά, αλλά και η πρώτη σε σπουδαιότητα γλώσσα υψηλού επιπέδου.

Ανάμεσα στους θιασώτες της ίδιας γλώσσας ανακύπτουν διαμάχες άλλου είδους. Τα πρότυπα των γλωσσών προγραμματισμού αναπτύσσονται συχνά μέσω διεθνών επιτροπών. Ποιοι είναι οι «εγγυητές», και ποιοι συμμετέχουν τελικά σε τέτοιες διαδικασίες αποφάσεων; Τι πρόκειται και τι δεν πρόκειται να συμπεριληφθεί στην επόμενη επίσημη έκδοση μιας γλώσσας; Η ανάπτυξη των προδιαγραφών για κάποια γλώσσα προγραμματισμού συχνά είναι διαδικασία εντυπωσιακά αργή, περίπλοκη και... μνησίκακη.

Αυτό που ενδιαφέρει περισσότερο εμάς είναι οι συχνές διαφωνίες που αφορούν τους θεμελιώδεις ορισμούς. Έχουμε ήδη χρησιμοποιήσει τον όρο *οντοστρεφής*, ο οποίος είναι ένας από τους πιο έντονα αμφιλεγόμενους όρους. Ποιες ακριβώς ιδιότητες θα πρέπει να έχει μια γλώσσα ώστε να μπορεί να θεωρηθεί οντοστρεφής; Εμείς θα παρακάμψουμε αυτό το ερώτημα, παραθέτοντας μόνο μια άτυπη περιγραφή των οντοστρεφών γλωσσών. Θα αποφύγουμε γενικότερα να δώσουμε απολύτως αυστηρούς ορισμούς για τέτοιους αμφιλεγόμενους όρους, για δύο λόγους. Πρώτον, θα ήταν κάπως υποκριτικό να ισχυριστούμε ότι υπάρχει καν ορισμός, τη στιγμή που στην πραγματικότητα υπάρχουν πολλοί και ασύμβατοι ορισμοί. Και δεύτερον, διότι οι αυστηροί ορισμοί δεν θα είχαν πρακτική χρησιμότητα: ένας αυστηρός ορισμός λ.χ. για τον όρο «οντοστρεφής» απλώς θα πυροδοτούσε διαμάχες της μορφής: «η δική μου γλώσσα μου *είναι* οντοστρεφής ενώ η δική σου *δεν είναι*». Για τα ζητήματα που μας ενδιαφέρουν σε αυτό το βιβλίο, οι άτυπες περιγραφές είναι πιο χρήσιμες. Κάποιες γλώσσες είναι πράγματι πιο οντοστρεφείς από κάποιες άλλες, αλλά θα αφήσουμε τέτοια ζητήματα στη κρίση του αναγνώστη.[3]

---

3 Στους μαθηματικούς κύκλους, η ανταλλαγή σκληρών αλλά και ασαφών επιχειρημάτων αποκαλείται συχνά «βατραχομυομαχία». Την έκφραση αυτή είχε χρησιμοποιήσει ως γνωστόν ο Albert Einstein για να χαρακτηρίσει μια έντονη διαμάχη μεταξύ των μαθηματικών David Hilbert και L.E.J. Brouwer, την οποία είχε αποκαλέσει στα γερμανικά «Frosch-Mäuse-Krieg» (δηλ. βατραχομυομαχία). Ο όρος προέρχεται από έναν γνωστό αρχαιοελληνικό μύθο: χρησιμοποιείται σε μια ελληνιστική παρωδία της *Ιλιάδας* που φέρει τον τίτλο *Βατραχομυομαχία*. Οι παράξενες αντιμαχίες στις γλώσσες προγραμματισμού ανήκουν συχνά στο είδος των βατραχομυομαχιών.

1.4. Η αξιοπερίεργη εξέλιξη

Οι ερευνητές του κλάδου επινοούν διαρκώς καινούργιες γλώσσες προγραμματισμού. Ίσως όχι ακριβώς *καινούργιες*, αφού οι σχεδιαστές γλωσσών στηρίζονται σε ιδέες από προηγούμενες γλώσσες. Ο σχεδιαστής μιας νέας γλώσσας, όμως, έχει λυμένα τα χέρια του, διότι δεν δεσμεύεται από προβλήματα συμβατότητας με τυχόντα προϋπάρχοντα προγράμματα. Κάποιες νέες γλώσσες αποκτούν ευρεία απήχηση, κάποιες άλλες παρακμάζουν. Είτε καθιερωθούν είτε όχι, οι καινούργιες γλώσσες διαμορφώνουν το πλαίσιο των ιδεών από το οποίο θα προέλθουν οι επόμενες γενεές γλωσσών.

Η διαδικασία σχεδίασης μιας νέας γλώσσας είναι αργή και έχει αυξητικό χαρακτήρα. Όλες σχεδόν οι γλώσσες, ακόμα και οι πιο νέες, αναπτύσσουν διάφορες «διαλέκτους». Λίγο πρωτύτερα μνημονεύσαμε τη σεβάσμια γλώσσα Fortran, στην πράξη όμως δεν υπάρχει *«η»* Fortran. Σε πρώτη φάση υπήρξαν τα αρχικά σχέδια και οι πρώτες υλοποιήσεις της Fortran από την εταιρεία IBM, που χρονολογούνται από τα μέσα της δεκαετίας του 1950. Ακολούθησε μια σειρά από πρότυπα: Fortran I, Fortran II, Fortran III, Fortran IV, Fortran 66, Fortran 77, Fortran 90, Fortran 95, και, πιθανόν Fortran 2000.[4] Τα νέα πρότυπα παραγκωνίζουν τα παλαιά με βραδύ ρυθμό – και σε ορισμένες περιπτώσεις ποτέ: πολλοί προγραμματιστές χρησιμοποιούν ακόμα την Fortran 77. Για κάθε διάλεκτο πιθανόν να υπάρχουν διαφορετικές υλοποιήσεις για τα διάφορα συστήματα λογισμικού και υλισμικού, καθεμία από τις οποίες διευκρινίζει το αρχικό πρότυπο με διαφορετικό τρόπο. Επιπλέον, εμφανίζονται διάλεκτοι που εξυπηρετούν εξειδικευμένους σκοπούς: π.χ. έχουν εμφανιστεί πάνω από δέκα διάλεκτοι της Fortran που προσθέτουν σε αυτήν ειδικές γλωσσικές δυνατότητες για παράλληλο προγραμματισμό.

Είτε απότομα είτε βαθμιαία, οι *γλώσσες προγραμματισμού μεταβάλλονται*. Και μάλιστα πολύ πιο γρήγορα απ᾽ ό,τι οι φυσικές γλώσσες. Εάν ασχοληθείτε με τον προγραμματισμό σε μακροπρόθεσμη βάση, τότε είναι σχεδόν βέβαιο ότι θα πρέπει διαρκώς να μαθαίνετε νέες διαλέκτους και νέες γλώσσες. Θα ήταν πιο εύκολο να είχαμε μια παντοτινά αναλλοίωτη γλώσσα, βολική όσο τα ήδη φορεμένα ρούχα μας. Αφού αυτό όμως δεν είναι εφικτό, ας απολαύσουμε τουλάχιστον την ιστορία των γλωσσών προγραμματισμού καθώς εκτυλίσσεται μέσα στον χρόνο.

1.5. Οι πολλαπλές διασυνδέσεις

Το βιβλίο αυτό πραγματεύεται τις ίδιες τις γλώσσες προγραμματισμού, και όχι τον καθαυτό προγραμματισμό, δηλαδή το πώς να γράφετε καλά προγράμματα. Μερικές φορές βέβαια θα σχολιάσουμε και ζητήματα που αφορούν το προγραμματιστικό ύφος, διότι οι γλώσσες δεν μένουν ουδέτερες σε αυτό το θέμα. Κάθε γλώσσα

4 Οι επιτροπές τυποποίησης συχνά είναι υπεραισιόδοξες ως προς τις ονομασίες που δίνουν στα σχέδιά τους: Το πρότυπο Fortran 90 ονομάστηκε διαδοχικά Fortran 82, 8X, και 88, πριν κυκλοφορήσει τελικά ως Fortran 90 (εντός του 1991!). Το πρότυπο Fortran 2000 είχε ανακοινωθεί ότι θα κυκλοφορήσει εντός του 2002.

υποστηρίζει κάποιο ιδιαίτερο ύφος προγραμματισμού, δηλαδή μια συγκεκριμένη προσέγγιση στο πώς να λύνουμε προβλήματα με αλγοριθμικό τρόπο. Η συσχέτιση γλωσσών προγραμματισμού και προγραμματιστικής πρακτικής έχει αντίκτυπο και στα δύο.

Από τη μία πλευρά, οι γλώσσες προγραμματισμού κατευθύνουν τον προγραμματιστή προς ένα συγκεκριμένο ύφος προγραμματισμού. Οι οντοστρεφείς γλώσσες, όπως η Java, ωθούν τους προγραμματιστές να χρησιμοποιούν οντότητες. Οι συναρτησιακές γλώσσες, όπως η ML, οδηγούν τους προγραμματιστές προς ένα ύφος προγραμματισμού που βασίζεται σε πολλές μικρές συναρτήσεις. Οι λογικοκεντρικές γλώσσες, όπως η Prolog, ενθαρρύνουν τους προγραμματιστές να εκφράζουν τη λύση των προβλημάτων ως αναζήτηση σε ένα λογικώς προσδιορισμένο σύνολο ενδεχόμενων λύσεων. Το να γράφετε προγράμματα σε ύφος αναντίστοιχο με τη γλώσσα που χρησιμοποιείτε είναι μεν εφικτό, αλλά σίγουρα δεν είναι καλή πρακτική. Μπορείτε να συντάξετε προστακτικά προγράμματα με βρόχους και τιμοδοτήσεις ακόμα και στην ML (αν και αυτό το βιβλίο δεν πρόκειται να σας εξηγήσει πώς γίνεται αυτό!). Μπορείτε να γράψετε προγράμματα Java τα οποία δεν δημιουργούν ούτε μία οντότητα και απλώς ενσωματώνουν έναν μεγάλο όγκο συνηθισμένου προστακτικού κώδικα στον ορισμό μιας μοναδικής κλάσης οντοτήτων. Στην Java, όπως και σε οποιαδήποτε άλλη γλώσσα που υποστηρίζει αναδρομή, είναι δυνατόν να γράψετε συναρτησιακά προγράμματα χωρίς επαναληπτικές εντολές ή τιμοδοτήσεις. Όλα τα παραπάνω όμως δεν είναι παρά οι εξαιρέσεις που επιβεβαιώνουν τον κανόνα: όταν προγραμματίζετε σε ύφος αφύσικο προς τη γλώσσα εργασίας, όταν δηλαδή αντιμάχεστε τον χαρακτήρα αυτής της γλώσσας, θα νοιώσετε ότι η ίδια η γλώσσα αντιστέκεται.

Από την άλλη πλευρά, η προγραμματιστική πρακτική οδηγεί συχνά τους προγραμματιστές προς νέες ιδέες όσον αφορά τις γλώσσες. Για παράδειγμα, ο John McCarthy εισήγαγε στη Lisp την αναδρομή και τις υποθετικές εκφράσεις, διότι διαπίστωσε ότι ήταν απαραίτητες για τις εφαρμογές τεχνητής νοημοσύνης που ανέπτυσσε. Οι «κλάσεις» και οι «οντότητες» εισήχθησαν στη Simula διότι οι σχεδιαστές της, Kristen Nygaard και Ole-Johan Dahl, τις χρειάζονταν για τις μεγάλες προσομοιώσεις που υλοποιούσαν. Στο Κεφάλαιο 24 θα αναφέρουμε και άλλα τέτοια περιστατικά από την ιστορία των γλωσσών προγραμματισμού.

Οι γλώσσες προγραμματισμού διαπλέκονται όχι μόνον με την προγραμματιστική πρακτική, αλλά και με πολλούς άλλους κλάδους της επιστήμης των υπολογιστών. Η εξέλιξη των γλωσσών καθοδηγεί, και επίσης καθοδηγείται από, την εξέλιξη του υλισμικού. Η θεωρία τυπικών γλωσσών και αυτομάτων, ένας από τους πιο μαθηματικοποιημένους κλάδους της επιστήμης των υπολογιστών, έχει πολλές εφαρμογές στον ορισμό και την υλοποίηση γλωσσών προγραμματισμού. Τα λειτουργικά συστήματα αλληλεπιδρούν στενά με τις γλώσσες προγραμματισμού. Όλοι οι τομείς εφαρμογών (τεχνητή νοημοσύνη, δίκτυα, βάσεις δεδομένων, επιχειρηματικές εφαρμογές, αριθμητικοί υπολογισμοί, κ.ο.κ.) καταθέτουν τη δική τους άποψη για το ζήτημα της σχεδίασης των γλωσσών προγραμματισμού.

1.6. Σύντομο σχόλιο για τις Διεπαφές για Προγραμματισμό Εφαρμογών

Οι σύγχρονες εμπορικές γλώσσες προγραμματισμού υποστηρίζονται από μεγάλες και τυποποιημένες βιβλιοθήκες έτοιμου κώδικα, τις λεγόμενες *Διεπαφές για Προγραμματισμό Εφαρμογών* (ΔιΠΕ). Μια ΔιΠΕ είναι δυνατόν να περιέχει κώδικα που υλοποιεί βασικές δομές δεδομένων (στοίβες, ουρές, πίνακες διασποράς, κ.λπ.), δι-διάστατα και τριδιάστατα γραφικά, γραφιστικές διεπαφές εργασίας, υποστήριξη δικτύου, διαχείριση αρχείων, κρυπτογράφηση και ασφάλεια, και πολλές άλλες υπηρεσίες. Η αφομοίωση των περιεχομένων και του τρόπου χρήσης κάποιας ΔιΠΕ είναι από τα βασικά καθήκοντα των μαχόμενων προγραμματιστών. Η έντυπη περιγραφή των προδιαγραφών μιας γλώσσας συχνά έχει μικρότερο μέγεθος από την έντυπη τεκμηρίωση μιας ΔιΠΕ.

Αναφέρουμε εδώ τις Διεπαφές για Προγραμματισμό Εφαρμογών για να τις αγνοήσουμε. Αν και είναι σημαντικότατες, δεν είναι το αντικείμενό μας. Στα διδακτικά κεφάλαια αυτού του βιβλίου περί ML, Java και Prolog θα βρείτε αρκετές πληροφορίες ώστε να είστε σε θέση να λύνετε απλά προβλήματα. Για την ανάπτυξη εφαρμογών μεγάλης κλίμακας θα χρειαστείτε βαθύτερη γνώση των αντίστοιχων ΔιΠΕ, η οποία δεν εμπίπτει στους στόχους αυτού του βιβλίου.

1.7. Ανακεφαλαίωση

Στο κεφάλαιο αυτό εξηγήσαμε ορισμένους λόγους για τους οποίους ο συγγραφέας είναι γοητευμένος από τις γλώσσες προγραμματισμού. Πολλοί αναγνώστες ενός βιβλίου παραλείπουν να διαβάσουν το εισαγωγικό κεφάλαιο, φοβούμενοι ότι θα είναι ασαφές και βαρετό, γεμάτο γενικότητες γύρω από ένα θέμα για το οποίο δεν γνωρίζουν ακόμα τίποτα συγκεκριμένο. Το δικό μας εισαγωγικό κεφάλαιο δεν αποτελεί εξαίρεση, αλλά τουλάχιστον ήταν σύντομο.

Στο κεφάλαιο αυτό περιγράψαμε επίσης τη δομή αυτού του βιβλίου: είναι ένα μείγμα από πρακτικά-διδακτικά και θεωρητικά κεφάλαια. Όλα τα κεφάλαια αυτού του βιβλίου –εκτός από το πρώτο και το τελευταίο– έχουν στο τέλος τους κάποιες ασκήσεις. Πολλά περιλαμβάνουν και μια ενότητα με βιβλιογραφικές υποδείξεις για περαιτέρω μελέτη.

Το επόμενο κεφάλαιο είναι το πρώτο από τα θεωρητικά. Ασχολείται με το πώς ορίζουμε τη σύνταξη μιας γλώσσας προγραμματισμού. (Το πρώτο πρακτικό-διδακτικό κεφάλαιο είναι το 5, και αφορά τη συναρτησιακή γλώσσα ML.)