ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ

ΔΕΥΤΕΡΗ ΑΜΕΡΙΚΑΝΙΚΗ ΕΚΔΟΣΗ

ΠΡΑΓΜΑΤΟΛΟΓΙΑ ΤΩΝ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Michael L. Scott

Μετάφραση/Επιστημονική επιμέλεια
Νικόλαος Σ. Παπασπύρου, *Εθνικό Μετσόβιο Πολυτεχνείο*

ΚΛΕΙΔΑΡΙΘΜΟΣ

Περιεχόμενα

Περιεχόμενα του CD

Στην αγγλική γλώσσα.
(Με τελεία σημειώνονται οι αντίστοιχες εισαγωγικές παράγραφοι στο βιβλίο.)

2 Σύνταξη των γλωσσών προγραμματισμού

Σε αντίθεση με τις φυσικές γλώσσες όπως τα Ελληνικά ή τα Αγγλικά, οι γλώσσες των υπολογιστών πρέπει να είναι ακριβείς. Τόσο η μορφή τους (σύνταξη) όσο και το νόημά τους (σημασιολογία) πρέπει να καθορίζονται χωρίς αμφισημίες, ώστε οι προγραμματιστές και οι υπολογιστές να μπορούν να καταλαβαίνουν τι πρέπει να κάνει ένα πρόγραμμα. Για να εξασφαλίζεται ο απαιτούμενος βαθμός ακρίβειας, οι σχεδιαστές και οι υλοποιητές των γλωσσών χρησιμοποιούν τυπικές συντακτικές και σημασιολογικές σημειογραφίες. Για να διευκολύνουμε τη μελέτη των δυνατοτήτων των γλωσσών στα επόμενα κεφάλαια, θα ασχοληθούμε πρώτα με αυτές τις σημειογραφίες: τη σύνταξη σε αυτό το κεφάλαιο και τη σημασιολογία στο Κεφάλαιο 4.

ΠΑΡΑΔΕΙΓΜΑ 2.1
Σύνταξη των αραβικών αριθμών

Θα χρησιμοποιήσουμε ως πρώτο παράδειγμα τη σύνταξη των Αραβικών αριθμών. Οι αριθμοί αυτοί αποτελούνται από ψηφία, που μπορούμε να απαριθμήσουμε ως εξής ('|' σημαίνει "ή"):

digit ⟶ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

Τα ψηφία (digits) είναι τα συντακτικά δομικά υλικά των αριθμών. Σύμφωνα με τη συνήθη σημειογραφία, λέμε ότι ένας φυσικός αριθμός αναπαρίσταται από μια (μη κενή) ακολουθία ψηφίων οποιουδήποτε μήκους, που αρχίζει με μη μηδενικό ψηφίο:

non_zero_digit ⟶ `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`
natural_number ⟶ *non_zero_digit digit* *

Εδώ το μετασύμβολο *, που ονομάζεται "άστρο του Kleene",[1] δείχνει καμία ή περισσότερες επαναλήψεις του συμβόλου που βρίσκεται στα αριστερά του. ■

Φυσικά, τα ψηφία είναι μόνο σύμβολα: κηλίδες από μελάνι στο χαρτί ή πίξελ στην οθόνη. Από μόνα τους δε μεταφέρουν κανένα νόημα. Προσθέτουμε σημασιολογία στα ψηφία όταν λέμε ότι αναπαριστούν τους φυσικούς αριθμούς από το μηδέν έως το εννέα, όπως αυτοί ορίζονται από τους μαθηματικούς. Εναλλακτικά, θα μπορούσαμε να πούμε ότι αναπαριστούν χρώματα, ή τις ημέρες της εβδομάδας σε ένα δεκαδικό ημερολόγιο. Αυτές θα ήταν διαφορετικές σημασιολογίες για την ίδια σύνταξη. Με παρόμοιο τρόπο, ορίζουμε τη σημασιολογία των φυσικών αριθμών αποδίδοντας στις ακολουθίες ψηφίων τη συνηθισμένη ερμηνεία του δεκαδικού συστήματος αρίθμησης. Παρόμοιοι συντακτικοί κανόνες και σημασιολογικές ερμηνείες μπορούν να οριστούν για τους ρητούς αριθμούς, τους πραγματικούς αριθμούς (περιορισμένης ακρίβειας), την αριθμητική, τις αναθέσεις τιμών, τον έλεγχο ροής, τις δηλώσεις, και γενικά για όλα τα στοιχεία που απαρτίζουν τις γλώσσες προγραμματισμού.

Η διάκριση ανάμεσα στη σύνταξη και τη σημασιολογία είναι χρήσιμη τουλάχιστον για δύο λόγους. Πρώτον, διαφορετικές γλώσσες προγραμματισμού συχνά παρέχουν δυνατότητες με πολύ παρόμοια σημασιολογία αλλά πολύ διαφορετική σύνταξη. Γενικά, είναι

1 Ο Stephen Kleene (1909–1994), μαθηματικός στο Πανεπιστήμιο του Wisconsin, συνέβαλε στη θεμελίωση της θεωρίας υπολογισμού και για μεγάλο μέρος της ύλης που παρουσιάζεται στην Ενότητα 2.4.

πολύ ευκολότερο να μάθει κανείς μια καινούργια γλώσσα αν μπορεί να ξεχωρίσει τις κοινές (και μάλλον οικείες σε αυτόν) ιδέες που βρίσκονται κάτω από την άγνωστη σύνταξη. Δεύτερον, υπάρχουν πολύ αποδοτικοί και κομψοί αλγόριθμοι με τους οποίους ένας μεταγλωττιστής ή ένας διερμηνέας μπορεί να ανακαλύψει τη συντακτική δομή (αλλά όχι τη σημασιολογία!) ενός προγράμματος. Αυτοί οι αλγόριθμοι μπορούν να χρησιμοποιηθούν για να καθοδηγήσουν το υπόλοιπο της διαδικασίας μεταγλώττισης ή διερμηνείας.

Σε αυτό το κεφάλαιο εστιάζουμε στη σύνταξη: πώς καθορίζουμε τους δομικούς κανόνες μιας γλώσσας προγραμματισμού και πώς ο μεταγλωττιστής καταλαβαίνει τη δομή ενός δεδομένου αρχικού προγράμματος. Αυτές οι δύο εργασίες — ο καθορισμός των συντακτικών κανόνων και η διαπίστωση πώς (και κατά πόσο) η δομή ενός προγράμματος είναι σύμφωνη με αυτούς τους κανόνες — είναι διακριτές. Η πρώτη ενδιαφέρει κυρίως τους προγραμματιστές, που θέλουν να γράφουν έγκυρα προγράμματα. Η δεύτερη ενδιαφέρει κυρίως τους μεταγλωττιστές, που χρειάζεται να αναλύουν αυτά τα προγράμματα. Η πρώτη εργασία βασίζεται στις *κανονικές εκφράσεις* (regular expressions) και τις *γραμματικές χωρίς συμφραζόμενα* (context-free grammars) που καθορίζουν πώς σχηματίζονται έγκυρα προγράμματα. Η δεύτερη εργασία βασίζεται σε *λεκτικούς αναλυτές* (scanners), ή λεξικογραφικούς αναλυτές, και σε *συντακτικούς αναλυτές* (parsers), που αναγνωρίζουν τη συντακτική δομή των προγραμμάτων. Εξετάζουμε την πρώτη εργασία στην Ενότητα 2.1 και τη δεύτερη στις Ενότητες 2.2 και 2.3.

Στην Ενότητα 2.4 (κατά το μεγαλύτερο μέρος στο ΠΓΠ CD) εξετάζουμε σε μεγαλύτερο βάθος την τυπική θεωρία πίσω από τη λεκτική και τη συντακτική ανάλυση. Στη θεωρητική ορολογία, ο λεκτικός αναλυτής είναι ένα *ντετερμινιστικό πεπερασμένο αυτόματο* (ΝΠΑ) (deterministic finite automaton, DFA) που αναγνωρίζει τις λεκτικές μονάδες μιας γλώσσας προγραμματισμού. Ο συντακτικός αναλυτής είναι ένα ντετερμινιστικό *αυτόματο στοίβας* (ΑΣ) (push-down automaton, PDA) που αναγνωρίζει τη χωρίς συμφραζόμενα σύνταξη της γλώσσας. Αποδεικνύεται ότι λεκτικοί και συντακτικοί αναλυτές μπορούν να κατασκευαστούν αυτόματα από κανονικές εκφράσεις και γραμματικές χωρίς συμφραζόμενα, με τη χρήση εργαλείων όπως τα `lex` και `yacc` του Unix.[2] Ίσως πουθενά αλλού στην επιστήμη των υπολογιστών δεν είναι η σύνδεση μεταξύ θεωρίας και πράξης τόσο ορατή και συναρπαστική.

2.1 Περιγραφή της σύνταξης: κανονικές εκφράσεις και γραμματικές χωρίς συμφραζόμενα

Η τυπική περιγραφή της σύνταξης χρειάζεται ένα σύνολο από κανόνες. Η πολυπλοκότητα (η εκφραστικότητα) της σύνταξης εξαρτάται από το είδος των κανόνων που χρησιμοποιούνται για την περιγραφή της. Αποδεικνύεται πως όλες οι λεκτικές μονάδες που μπορεί κανείς να φανταστεί είναι δυνατόν να κατασκευαστούν από μεμονωμένους χαρακτήρες με μόνο τρία είδη τυπικών κανόνων: τη συνένωση (concatenation) την εναλλαγή (alternation), δηλαδή την επιλογή από ένα πεπερασμένο σύνολο εναλλακτικών μορφών, και το επονομαζόμενο "κλείσιμο του Kleene", δηλαδή την επανάληψη κατά αυθαίρετα πολλές φορές. Για να περιγράψουμε σχεδόν ό,τι άλλο μπορούμε να φανταστούμε σχετικά με τη σύνταξη, απαιτείται μόνο ένας ακόμη κανόνας: η αναδρομή, δηλαδή η δημιουργία μιας συντακτικής δομής από απλούστερες εμφανίσεις της ίδιας συντακτικής δομής. Οποιοδήποτε σύνολο συμβολοσειρών που μπορεί να οριστεί με τη χρήση των τριών πρώτων κανόνων ονομάζεται *κανονικό σύνολο* (regular set). Τα κανονικά σύνολα παράγονται από *κανονικές εκφράσεις* και αναγνωρίζονται από λεκτικούς αναλυτές. Κάθε σύνολο συμβολοσειρών που μπορεί να οριστεί αν προσθέσουμε και την αναδρομή ονομάζεται *γλώσσα χωρίς συμφραζόμενα* (context-free language, CFL). Οι γλώσσες χωρίς συμφραζόμενα παράγονται από *γραμματικές χωρίς συμφραζόμενα* και αναγνωρίζονται από συντα-

2 Σε πολλούς υπολογιστές, τα `lex` και `yacc` έχουν αντικατασταθεί από τα εργαλεία GNU `flex` και `bison`. Αυτά τα εργαλεία δεν είναι εμπορικά, έχουν αναπτυχθεί ανεξάρτητα, και είναι διαθέσιμα δωρεάν από το Ίδρυμα Ελεύθερου Λογισμικού (Free Software Foundation) στη διεύθυνση `www.gnu.org/software`. Παρέχουν ένα υπερσύνολο των λειτουργιών των `lex` και `yacc`.

κτικούς αναλυτές. (Η ορολογία μπορεί μερικές φορές να προκαλέσει σύγχυση. Το νόημα της λέξης *γλώσσα* ποικίλει σημαντικά, ανάλογα με το αν μιλούμε για "τυπικές γλώσσες" [π.χ., κανονικές ή χωρίς συμφραζόμενα] ή για γλώσσες προγραμματισμού. Μια τυπική γλώσσα είναι απλώς ένα σύνολο συμβολοσειρών, χωρίς συνοδευτική σημασιολογία.)

2.1.1 Λεκτικές μονάδες και κανονικές εκφράσεις

Οι λεκτικές μονάδες είναι τα βασικά δομικά υλικά των προγραμμάτων. Περιλαμβάνουν τις λέξεις-κλειδιά, τα αναγνωριστικά, τους αριθμούς, και πολλά είδη συμβόλων. Η Pascal, που είναι μια σχετικά απλή γλώσσα, έχει 64 είδη λεκτικών μονάδων: 21 σύμβολα (`+`, `-`, `;`, `:=`, `..`, κλπ.), 35 λέξεις κλειδιά (`begin`, `end`, `div`, `record`, `while`, κλπ.), ακέραιες σταθερές (π.χ., `137`), πραγματικές σταθερές κινητής υποδιαστολής (π.χ., `6.022e23`), σταθερούς χαρακτήρες και συμβολοσειρές σε εισαγωγικά (π.χ., `'snerk'`), αναγνωριστικά (`MyVariable`, `YourType`, `maxint`, `readln`, κλπ., 39 από τα οποία είναι προκαθορισμένα), και δύο διαφορετικά είδη σχολίων.

Τα πεζά και τα κεφαλαία γράμματα στα αναγνωριστικά και τις λέξεις-κλειδιά θεωρούνται διαφορετικά σε κάποιες γλώσσες (π.χ., στις Modula-2/3 και στη C και τους απογόνους της), και ταυτόσημα σε άλλες (π.χ., στην Ada, την Common Lisp, τη Fortran 90, και την Pascal). Έτσι, `foo`, `Foo` και `FOO` αναπαριστούν το ίδιο αναγνωριστικό στην Ada αλλά διαφορετικά αναγνωριστικά στη C. Η Modula-2 και η Modula-3 απαιτούν οι λέξεις-κλειδιά και τα προκαθορισμένα αναγνωριστικά να γράφονται με κεφαλαία γράμματα, ενώ η C και οι απόγονοί της με πεζά. Μερικές γλώσσες (με πιο αξιοσημείωτες τη Modula-3 και τη Standard Pascal) επιτρέπουν μόνο γράμματα και ψηφία στα αναγνωριστικά. Οι περισσότερες (μεταξύ των οποίων και πολλές υλοποιήσεις της Pascal) επιτρέπουν το χαρακτήρα υπογράμμισης (underscore). Μερικές (με πιο αξιοσημείωτη τη Lisp) επιτρέπουν μια ποικιλία πρόσθετων χαρακτήρων. Σε κάποιες γλώσσες (π.χ., στην Java, τη C, και τη Modula-3) υπάρχουν καθιερωμένες συμβάσεις για τη χρήση των πεζών και των κεφαλαίων γραμμάτων στα ονόματα.[3]

Με την παγκοσμιοποίηση των υπολογιστών, τα σύνολα χαρακτήρων που δε βασίζονται στο λατινικό αλφάβητο αποκτούν ολοένα αυξανόμενη σημασία. Πολλές σύγχρονες γλώσσες, όπως η C99, η C++, η Ada 95, η Java, η C#, και η Fortran 2003, διαθέτουν ρητή υποστήριξη για σύνολα χαρακτήρων πολλών byte (multibyte character sets), που γενικά βασίζονται στα διεθνή πρότυπα Unicode και ISO/IEC 10646. Οι περισσότερες σύγχρονες γλώσσες προγραμματισμού επιτρέπουν τη χρήση μη λατινικών χαρακτήρων μέσα σε σχόλια και συμβολοσειρές, ενώ ο αριθμός των γλωσσών που επιτρέπουν τέτοιους χαρακτήρες και στα αναγνωριστικά αυξάνεται συνεχώς. Οι συμβάσεις για τη φορητότητα μεταξύ διαφορετικών συνόλων χαρακτήρων και για την *τοπική προσαρμογή* (localization) σε ένα συγκεκριμένο σύνολο χαρακτήρων μπορούν να είναι εξαιρετικά πολύπλοκες, ιδιαίτερα όταν απαιτούνται διάφορες μορφές αναδρομικής συμβατότητας με προγενέστερο κώδικα (το συνοδευτικό έγγραφο C99 Rationale αφιερώνει πέντε σελίδες σε αυτό το θέμα [Int99, σελίδες 19–23]). Σε αυτό το βιβλίο γενικά αγνοούμε τέτοιου είδους ζητήματα.

Ορισμένες υλοποιήσεις γλωσσών θέτουν περιορισμούς στο μέγιστο μήκος των αναγνωριστικών, αλλά οι περισσότερες αποφεύγουν τέτοιους περιττούς περιορισμούς. Οι περισσότερες σύγχρονες γλώσσες έχουν επίσης λίγο-πολύ *ελεύθερη μορφή*, δηλαδή τα προγράμματα είναι απλώς ακολουθίες λεκτικών μονάδων: αυτό που έχει σημασία είναι η σειρά με την οποία εμφανίζονται οι λεκτικές μονάδες, όχι η φυσική τους θέση μέσα σε μια τυπωμένη γραμμή ή σελίδα. Τα "λευκά διαστήματα" (white space) — κενά διαστήματα, στηλοθέτες, χαρακτήρες επαναφοράς κεφαλής και αλλαγής γραμμής ή σελίδας — μεταξύ λεκτικών μονάδων συνήθως αγνοούνται, με εξαίρεση όταν διαχωρίζουν μια λεκτική μονάδα από την επόμενη. Σε αυτούς τους κανόνες υπάρχουν μερικές εξαιρέσεις. Μερικές υλοποιήσεις γλωσσών περιορίζουν το μέγιστο μήκος μιας γραμμής, ώστε ο μεταγλωττιστής να μπορεί να αποθηκεύσει την τρέχουσα γραμμή σε μια προσωρινή μνήμη (buffer)

3 Για λόγους συνέπειας, σε αυτό το βιβλίο δε χρησιμοποιούμε πάντα αυτές τις συμβάσεις. Τα περισσότερα παραδείγματα ακολουθούν τη συνηθισμένη πρακτική των προγραμματιστών της C, σύμφωνα με την οποία, αν ένα όνομα αποτελείται από περισσότερες "υπολέξεις", αυτές χωρίζονται με χαρακτήρες υπογράμμισης.

σταθερού μήκους. Οι διάλεκτοι της Fortran πριν από τη Fortran 90 χρησιμοποιούν *σταθερή μορφή* με 72 χαρακτήρες ανά γραμμή (το πλάτος μιας χάρτινης διάτρητης κάρτας στην οποία κάποτε αποθηκεύονταν τα προγράμματα), και διαφορετικές στήλες σε αυτή τη γραμμή δεσμευμένες για διαφορετικούς σκοπούς. Οι αλλαγές γραμμών χρησιμοποιούνται για το διαχωρισμό εντολών σε μερικές άλλες γλώσσες, όπως οι Haskell, Occam, SR, Tcl, και Python. Οι Haskell, Occam, και Python επίσης δίνουν ειδική σημασία στη στοίχιση (indentation). Για παράδειγμα, το σώμα ενός βρόχου αποτελείται ακριβώς από τις επόμενες εντολές που είναι στοιχημένες πιο μέσα από την επικεφαλίδα του βρόχου.

Για να καθορίσουμε τις λεκτικές μονάδες, χρησιμοποιούμε σημειογραφία κανονικών εκφράσεων. Μια κανονική έκφραση είναι ένα από τα εξής:

1. Ένας χαρακτήρας
2. Η κενή συμβολοσειρά, που συμβολίζεται με ϵ
3. Δύο κανονικές εκφράσεις η μία δίπλα στην άλλη, που αναπαριστούν οποιαδήποτε συμβολοσειρά μπορεί να παραχθεί από την πρώτη ακολουθούμενη από (συνενωμένη με) οποιαδήποτε συμβολοσειρά μπορεί να παραχθεί από τη δεύτερη
4. Δύο κανονικές εκφράσεις διαχωρισμένες με μια κατακόρυφη γραμμή (|), που σημαίνουν οποιαδήποτε συμβολοσειρά μπορεί να παραχθεί από την πρώτη *ή* οποιαδήποτε συμβολοσειρά μπορεί να παραχθεί από τη δεύτερη
5. Μια κανονική έκφραση ακολουθούμενη από το άστρο του Kleene, που σημαίνει τη συνένωση καμίας ή περισσότερων συμβολοσειρών που έχουν παραχθεί από την έκφραση πριν από το άστρο

Οι παρενθέσεις χρησιμοποιούνται για να αποφεύγονται οι αμφισημίες, όταν δεν είναι σαφές πού αρχίζουν και πού τελειώνουν οι διάφορες υποεκφράσεις.[4]

ΠΑΡΑΔΕΙΓΜΑ 2.2

Σύνταξη των αριθμών στην Pascal

Επιστρέφοντας στο παράδειγμα της Pascal, οι αριθμητικές σταθερές μπορούν να παραχθούν από τις εξής κανονικές εκφράσεις.[5]

digit ⟶ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`
unsigned_integer ⟶ *digit digit* *
unsigned_number ⟶ *unsigned_integer* ((`.` *unsigned_integer*) | ϵ)
(((`e` | `E`) (`+` | `-` | ϵ) *unsigned_integer*) | ϵ)

Για να παραγάγουμε μια έγκυρη συμβολοσειρά, διατρέχουμε την κανονική έκφραση από αριστερά προς τα δεξιά, επιλέγοντας μεταξύ εναλλακτικών σε κάθε κατακόρυφη γραμμή, και επιλέγοντας πλήθος επαναλήψεων σε κάθε άστρο του Kleene. Σε κάθε επανάληψη μπορούμε να κάνουμε διαφορετικές επιλογές στις κατακόρυφες γραμμές, παράγοντας έτσι διαφορετικές υποσυμβολοσειρές. Σημειώστε ότι, παρόλο που επιτρέπουμε σε

ΣΧΕΔΙΑΣΗ & ΥΛΟΠΟΙΗΣΗ

Περιορισμοί μορφοποίησης

Οι περιορισμοί της μορφοποίησης που οφείλονται σε θέματα υλοποίησης — όπως στην περίπτωση των κανόνων της Fortran 77 και των προγόνων της, που γράφηκαν με βάση τη μορφή των διάτρητων καρτών — τείνουν να γίνουν ανεπιθύμητοι και αναχρονιστικοί καθώς βελτιώνονται οι τεχνικές υλοποίησης. Με δεδομένη την τάση ορισμένων επεξεργαστών κειμένου να "συμπληρώνουν" ή να μορφοποιούν αυτόματα το κείμενο, οι κανόνες στοίχισης και αλλαγής γραμμής γλωσσών όπως οι Haskell, Occam, και Python είναι κάπως αμφιλεγόμενοι.

4 Κάποιοι συγγραφείς αναπαριστούν την κενή συμβολοσειρά με λ. Κάποιοι άλλοι χρησιμοποιούν μια τελεία (.), αντί για την παράθεση, για να συμβολίσουν τη συνένωση. Άλλοι χρησιμοποιούν το σύμβολο συν (+), αντί για την κατακόρυφη γραμμή, για να συμβολίσουν την εναλλαγή.

5 Οι αριθμητικές σταθερές σε πολλές γλώσσες είναι σημαντικά πιο πολύπλοκες. Η Java, για παράδειγμα, υποστηρίζει ακέραιες σταθερές των 32 και των 64 bit στο δεκαδικό, το οκταδικό, και το δεκαεξαδικό σύστημα αρίθμησης.

μεταγενέστερους ορισμούς να χρησιμοποιούν άλλους προγενέστερους, κανένας ορισμός δε χρησιμοποιεί τον εαυτό του. Τέτοιοι αναδρομικοί ορισμοί είναι το διακριτικό γνώρισμα των γραμματικών χωρίς συμφραζόμενα, που περιγράφονται στην Ενότητα 2.1.2. ■

Πολλοί αναγνώστες θα είναι εξοικειωμένοι με τις κανονικές εκφράσεις από την οικογένεια εργαλείων `grep` του Unix, από τις λειτουργίες αναζήτησης διαφόρων διορθωτών κειμένου (π.χ., `emacs`), ή από γλώσσες σεναρίων και εργαλεία όπως η Perl, η Python, η Ruby, η `awk`, και η `sed`. Τα περισσότερα από αυτά τα εργαλεία παρέχουν ένα πλούσιο σύνολο επεκτάσεων στη σημειογραφία των κανονικών εκφράσεων. Κάποιες επεκτάσεις, όπως η συντομογραφία για "καμία ή μία επαναλήψεις" ή "οτιδήποτε εκτός από κενό διάστημα", δεν αλλάζουν την εκφραστική ισχύ της σημειογραφίας. Άλλες, όπως η δυνατότητα απαίτησης μιας δεύτερης εμφάνισης αργότερα στη συμβολοσειρά εισόδου, της ίδιας ακολουθίας χαρακτήρων που εμφανίζεται σε προηγούμενο σημείο, αυξάνουν την εκφραστική ισχύ της σημειογραφίας ώστε να μην περιορίζεται στην παραγωγή κανονικών συνόλων. Άλλες επεκτάσεις έχουν σχεδιαστεί όχι για να αυξήσουν την εκφραστικότητα της σημειογραφίας αλλά μάλλον για να τη συνδέσουν με άλλες ευκολίες των γλωσσών. Για παράδειγμα, σε πολλά εργαλεία μπορεί κάποιος να σημειώσει μια κανονική έκφραση με τέτοιο τρόπο ώστε, όταν μια συμβολοσειρά ταυτίζεται με αυτή, τα περιεχόμενα αυτής της συμβολοσειράς να ανατίθενται σε κάποια επώνυμη τοπική μεταβλητή. Θα επιστρέψουμε σε αυτά τα ζητήματα στην Ενότητα 13.4.2, στο πλαίσιο των γλωσσών σεναρίων.

2.1.2 Γραμματικές χωρίς συμφραζόμενα

ΠΑΡΑΔΕΙΓΜΑ 2.3
Συντακτική ένθεση στις εκφράσεις

Οι κανονικές εκφράσεις επαρκούν για τον ορισμό λεκτικών μονάδων. Είναι όμως ανίσχυρες να καθορίσουν *ένθετες* (nested) δομές, που έχουν κεντρική σημασία για τις γλώσσες προγραμματισμού. Θεωρήστε για παράδειγμα τη δομή μιας αριθμητικής έκφρασης:

$$expr \longrightarrow \texttt{id} \mid \texttt{number} \mid \texttt{-}\ expr \mid \texttt{(}\ expr\ \texttt{)} \mid expr\ op\ expr$$
$$op \longrightarrow \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/}$$

Εδώ, η δυνατότητα να οριστεί μια δομή συναρτήσει του εαυτού της έχει μεγάλη σημασία. Μεταξύ άλλων, μας επιτρέπει να εξασφαλίζουμε ότι οι αριστερές και οι δεξιές παρενθέσεις ταιριάζουν, κάτι που δεν μπορεί να εξασφαλιστεί με κανονικές εκφράσεις (δείτε την Ενότητα 2.4.3 για περισσότερες λεπτομέρειες). ■

Οι κανόνες σε μια γραμματική χωρίς συμφραζόμενα ονομάζονται *συντακτικοί κανόνες*, ή απλώς κανόνες.[6] Τα σύμβολα στο αριστερό μέλος των κανόνων ονομάζονται *μεταβλητές* ή *μη τερματικά σύμβολα* (nonterminals). Μπορούν να υπάρχουν περισσότεροι κανόνες με το ίδιο σύμβολο στο αριστερό μέλος. Τα σύμβολα από τα οποία αποτελούνται οι συμβολοσειρές που παράγονται από τη γραμματική ονομάζονται *τερματικά σύμβολα* (terminals) και εμφανίζονται εδώ με γραμματοσειρά γραφομηχανής. Δεν επιτρέπεται να υπάρχουν στο αριστερό μέλος κανενός κανόνα. Σε μια γλώσσα προγραμματισμού, τα τερματικά σύμβολα της γραμματικής χωρίς συμφραζόμενα είναι οι λεκτικές μονάδες της γλώσσας. Ένα από τα μη τερματικά σύμβολα, συνήθως αυτό που βρίσκεται στο αριστερό μέλος του πρώτου κανόνα, ονομάζεται *αρχικό σύμβολο* (start symbol). Αυτό ονομάζει τη συντακτική δομή που ορίζεται από ολόκληρη τη γραμματική.

Η σημειογραφία των γραμματικών χωρίς συμφραζόμενα ονομάζεται συχνά μορφή Backus-Naur (Backus-Naur Form, BNF), προς τιμή του John Backus και του Peter Naur, που την επινόησαν για τον ορισμό της γλώσσας προγραμματισμού Algol 60 [NBB+63].[7] Αυστηρά, το άστρο του Kleene και η χρήση παρενθέσεων στις κανονικές εκφράσεις δεν

6 *Σ.τ.Μ.:* Στο πρωτότυπο κείμενο χρησιμοποιείται η λέξη "production" για τους κανόνες μιας γραμματικής, που μεταφράζεται στα ελληνικά ως "παραγωγή". Στην ελληνική μετάφραση προτιμήθηκε ο όρος "συντακτικός κανόνας" για την αποφυγή σύγχυσης, καθώς η λέξη "παραγωγή" χρησιμοποιήθηκε για τη μετάφραση της αγγλικής "derivation".

7 Ο John Backus (1924–), ήταν επίσης ο εφευρέτης της Fortran. Το μεγαλύτερο μέρος της επαγγελματικής του σταδιοδρομίας συνδέεται με την εταιρεία IBM, και ονομάστηκε IBM Fellow το 1987. Έλαβε το βραβείο Turing της ACM το 1977.

επιτρέπονται στην BNF, αλλά δεν αλλάζουν την εκφραστική ισχύ της σημειογραφίας και συχνά χρησιμοποιούνται για λόγους ευκολίας. Μερικές φορές, χρησιμοποιείται επίσης το "συν του Kleene" ($^+$), που αναπαριστά μία ή περισσότερες εμφανίσεις του συμβόλου ή της ομάδας συμβόλων που βρίσκεται πριν από αυτό.[8] Όταν επεκτείνεται με αυτούς τους επιπλέον τελεστές, η σημειογραφία ονομάζεται συχνά επεκτεταμένη BNF (EBNF). Η δομή

ΠΑΡΑΔΕΙΓΜΑ 2.4
Επεκτεταμένη BNF (EBNF)

```
id_list ⟶ id ( , id)*
```

είναι συντομογραφία του

```
id_list ⟶ id
id_list ⟶ id_list , id
```

Κάτι ανάλογο μπορεί να γραφεί για το "συν του Kleene". Η κατακόρυφη γραμμή είναι επίσης περιττή κατά μία έννοια, παρόλο που υποστηρίζεται στην αρχική BNF. Η δομή

```
op ⟶ + | - | * | /
```

μπορεί να θεωρηθεί ως συντομογραφία για την

```
op ⟶ +
op ⟶ -
op ⟶ *
op ⟶ /
```

που γράφεται και ως εξής:

```
op ⟶ +
   ⟶ -
   ⟶ *
   ⟶ /
```

Πολλές λεκτικές μονάδες, όπως οι `id` και `number` παραπάνω, έχουν πολλές διαφορετικές γραφές (δηλαδή μπορούν να αναπαρασταθούν από πολλές δυνατές ακολουθίες χαρακτήρων). Αυτό είναι αδιάφορο για το συντακτικό αναλυτή, ο οποίος δε διακρίνει ένα αναγνωριστικό από ένα άλλο. Ο σημασιολογικός αναλυτής όμως τα διακρίνει, οπότε ο λεκτικός αναλυτής πρέπει να αποθηκεύει την ακριβή γραφή κάθε "ενδιαφέρουσας" λεκτικής μονάδας για μεταγενέστερη χρήση.

2.1.3 Παραγωγές και δένδρα συντακτικής ανάλυσης

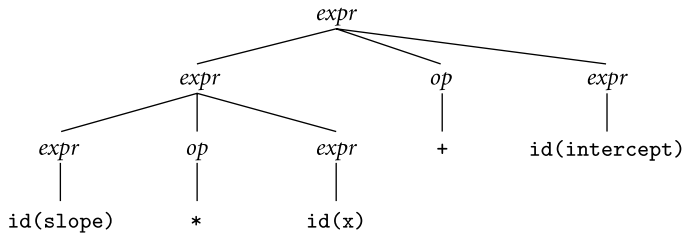
Μια γραμματική χωρίς συμφραζόμενα μάς δείχνει πώς να δημιουργήσουμε μια συντακτικά έγκυρη ακολουθία τερματικών συμβόλων: Ξεκινούμε από το αρχικό σύμβολο. Επιλέγουμε έναν κανόνα με το αρχικό σύμβολο στο αριστερό μέλος. Αντικαθιστούμε το αρχικό σύμβολο με το δεξιό μέλος αυτού του κανόνα. Τώρα επιλέγουμε ένα οποιοδήποτε μη τερματικό σύμβολο A στη συμβολοσειρά που προέκυψε, επιλέγουμε έναν κανόνα P με το A στο αριστερό του μέλος, και αντικαθιστούμε το A με το δεξιό μέλος του P. Επαναλαμβάνουμε τη διαδικασία μέχρι να μην απομένει κανένα μη τερματικό σύμβολο.

ΠΑΡΑΔΕΙΓΜΑ 2.5
Παραγωγή του `slope * x + intercept`

Ως παράδειγμα, μπορούμε να χρησιμοποιήσουμε τη γραμματική μας των εκφράσεων για να παραγάγουμε τη συμβολοσειρά "`slope * x + intercept`":

```
expr ⟹ expr op expr
     ⟹ expr op id
```

8 Ορισμένοι συγγραφείς χρησιμοποιούν άγκιστρα ({ }) για να αναπαραστήσουν καμία ή περισσότερες επαναλήψεις των συμβόλων ανάμεσά τους. Κάποιοι χρησιμοποιούν αγκύλες ([ ]) για να αναπαραστήσουν καμία ή μία εμφάνιση των συμβόλων ανάμεσά τους — δηλαδή για να δείξουν ότι αυτά τα σύμβολα είναι προαιρετικά.



Εικόνα 2.1 Δένδρο συντακτικής ανάλυσης για την έκφραση `slope * x + intercept` (γραμματική του Παραδείγματος 2.3).

$$
\begin{aligned}
&\Longrightarrow \quad \underline{expr}\ \texttt{+}\ \texttt{id} \\
&\Longrightarrow \quad expr\ op\ \underline{expr}\ \texttt{+}\ \texttt{id} \\
&\Longrightarrow \quad expr\ \underline{op}\ \texttt{id}\ \texttt{+}\ \texttt{id} \\
&\Longrightarrow \quad \underline{expr}\ \texttt{*}\ \texttt{id}\ \texttt{+}\ \texttt{id} \\
&\Longrightarrow \quad \texttt{id}\ \ \texttt{*}\ \texttt{id}\ \texttt{+}\ \ \texttt{id} \\
&\qquad\ \ \texttt{(slope)}\ \ \texttt{(x)}\ \ \texttt{(intercept)}
\end{aligned}
$$

Το μετασύμβολο $\Longrightarrow$ σημαίνει ότι το δεξιό μέλος προέκυψε με τη χρήση ενός κανόνα για την αντικατάσταση κάποιου μη τερματικού συμβόλου στο αριστερό μέλος. Σε κάθε γραμμή έχουμε υπογραμμίσει το σύμβολο *A* που αντικαθίσταται στην επόμενη γραμμή. ■

Μια σειρά αντικαταστάσεων που δείχνει πώς μπορεί να παραχθεί μια ακολουθία τερματικών συμβόλων ξεκινώντας από το αρχικό σύμβολο ονομάζεται *παραγωγή* (derivation). Κάθε ενδιάμεση συμβολοσειρά ονομάζεται *προτασιακή μορφή* (sentential form). Η τελική προτασιακή μορφή, που αποτελείται μόνο από τερματικά σύμβολα, ονομάζεται *αποτέλεσμα* (yield) της παραγωγής. Μερικές φορές παραλείπουμε τα ενδιάμεσα βήματα και γράφουμε *expr* $\Longrightarrow^*$ `slope * x + intercept`, όπου το μετασύμβολο $\Longrightarrow^*$ σημαίνει "δίνει ως αποτέλεσμα μετά από καμία ή περισσότερες αντικαταστάσεις". Σε αυτήν τη συγκεκριμένη παραγωγή, επιλέξαμε σε κάθε βήμα να αντικαταστήσουμε το δεξιότερο μη τερματικό σύμβολο με το δεξιό μέλος κάποιου συντακτικού κανόνα. Αυτή η στρατηγική αντικατάστασης οδηγεί στη *δεξιότερη* παραγωγή, που συχνά ονομάζεται και *κανονική* παραγωγή. Υπάρχουν και πολλές άλλες δυνατές παραγωγές, όπως η *αριστερότερη* και άλλες ενδιάμεσες επιλογές. Οι περισσότεροι συντακτικοί αναλυτές έχουν κατασκευαστεί έτσι ώστε να βρίσκουν μια συγκεκριμένη παραγωγή (συνήθως την αριστερότερη ή τη δεξιότερη).

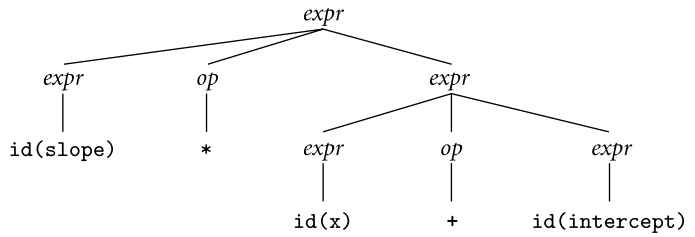
Είδαμε στο Κεφάλαιο 1 ότι μπορούμε να αναπαραστήσουμε σχηματικά μια παραγωγή ως ένα *δένδρο συντακτικής ανάλυσης*. Η ρίζα του δένδρου είναι το αρχικό σύμβολο της γραμματικής. Τα φύλλα του δένδρου είναι το αποτέλεσμα της παραγωγής. Κάθε εσωτερικός κόμβος, μαζί με τα παιδιά του, αντιπροσωπεύει τη χρήση ενός συντακτικού κανόνα.

ΠΑΡΑΔΕΙΓΜΑ 2.6

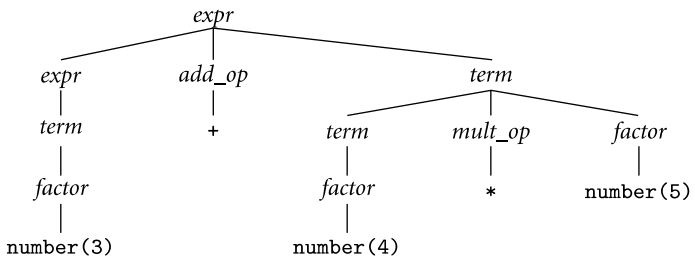
Δένδρα συντακτικής ανάλυσης για την έκφραση `slope * x + intercept`

Ένα δένδρο συντακτικής ανάλυσης για την έκφραση του παραδείγματός μας φαίνεται στην Εικόνα 2.1. Το δένδρο αυτό δεν είναι μοναδικό. Στο δεύτερο επίπεδο του δένδρου, θα μπορούσαμε να είχαμε επιλέξει να αντικαταστήσουμε τον τελεστή με `*`, αντί για `+`, και να αναπτύξουμε στη συνέχεια την έκφραση στα δεξιά, αντί γι' αυτή στα αριστερά (δείτε την Εικόνα 2.2). Το γεγονός ότι κάποιες συμβολοσειρές είναι αποτέλεσμα περισσότερων από ένα δένδρων συντακτικής ανάλυσης μάς λέει ότι η γραμματική μας είναι *διφορούμενη* ή *αμφίσημη* (ambiguous). Οι αμφισημίες δημιουργούν προβλήματα όταν προσπαθούμε να κατασκευάσουμε συντακτικούς αναλυτές: χρειάζεται κάποιος πρόσθετος μηχανισμός που να καθοδηγεί την επιλογή μεταξύ εξίσου αποδεκτών εναλλακτικών. ■

Μετά από λίγη σκέψη, καταλαβαίνουμε ότι υπάρχουν άπειρες γραμματικές χωρίς συμφραζόμενα για κάθε γλώσσα χωρίς συμφραζόμενα. Κάποιες από αυτές τις γραμματικές είναι πολύ πιο χρήσιμες από άλλες. Σε αυτό το βιβλίο, θα αποφύγουμε τη χρήση διφορούμενων γραμματικών, παρότι οι περισσότερες γεννήτριες συντακτικών αναλυτών τις επιτρέπουν, με τη χρήση *κανόνων αποφυγής αμφισημίας* (disambiguating rules). Θα αποφύγουμε επίσης τη χρήση των αποκαλούμενων *άχρηστων* συμβόλων: μη τερματικών συμβόλων που δεν μπορούν να παραγάγουν καμία ακολουθία τερματικών συμβόλων, ή τερματικών συμβόλων που δεν εμφανίζονται στο αποτέλεσμα καμίας παραγωγής.



Εικόνα 2.2 Εναλλακτικό (λιγότερο επιθυμητό) δένδρο συντακτικής ανάλυσης για την έκφραση `slope * x + intercept` (γραμματική του Παραδείγματος 2.3). Το γεγονός ότι υπάρχουν περισσότερα από ένα δένδρα σημαίνει ότι η γραμματική μας είναι διφορούμενη.



Εικόνα 2.3 Δένδρο συντακτικής ανάλυσης για την έκφραση `3 + 4 * 5`, με προτεραιότητα (γραμματική του Παραδείγματος 2.7).

Όταν σχεδιάζουμε τη γραμματική μιας γλώσσας προγραμματισμού, γενικά προσπαθούμε να βρούμε μια γραμματική που αντανακλά την εσωτερική δομή των προγραμμάτων με τρόπο χρήσιμο για το υπόλοιπο του μεταγλωττιστή. (Στην Ενότητα 2.3.2 θα δούμε ότι προσπαθούμε επίσης να βρούμε μια γραμματική που μπορεί να αναλυθεί αποδοτικά, κάτι το οποίο μπορεί να αποδειχθεί πρόκληση.) Η δομή είναι ιδιαίτερα σημαντική στις αριθμητικές εκφράσεις, όπου με συντακτικούς κανόνες μπορούμε να αποτυπώσουμε την *προσεταιριστικότητα* και την *προτεραιότητα* των διαφόρων τελεστών. Η προσεταιριστικότητα μας λέει ότι οι τελεστές στις περισσότερες γλώσσες ομαδοποιούν από αριστερά προς τα δεξιά, άρα `10 - 4 - 3` σημαίνει `(10 - 4) - 3` και όχι `10 - (4 - 3)`. Η προτεραιότητα μας λέει ότι ο πολλαπλασιασμός και η διαίρεση στις περισσότερες γλώσσες ομαδοποιούν ισχυρότερα από την πρόσθεση και την αφαίρεση, άρα `3 + 4 * 5` σημαίνει `3 + (4 * 5)` και όχι `(3 + 4) * 5`. (Οι κανόνες αυτοί δεν ισχύουν παντού· θα τους εξετάσουμε ξανά στην Ενότητα 6.1.1.)

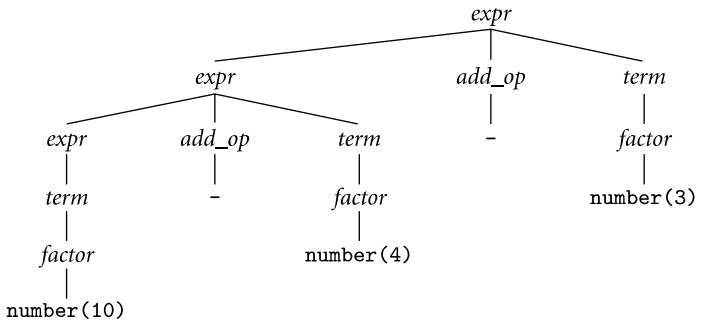
ΠΑΡΑΔΕΙΓΜΑ 2.7

Γραμματική εκφράσεων με προτεραιότητα και προσεταιριστικότητα

Ακολουθεί μια καλύτερη έκδοση της γραμματικής μας για τις εκφράσεις.

1. *expr* ⟶ *term* | *expr add_op term*
2. *term* ⟶ *factor* | *term mult_op factor*
3. *factor* ⟶ `id` | `number` | `-` *factor* | `(` *expr* `)`
4. *add_op* ⟶ `+` | `-`
5. *mult_op* ⟶ `*` | `/`

Η γραμματική αυτή δεν είναι διφορούμενη. Αποδίδει την προτεραιότητα των τελεστών με τον τρόπο που ορίζονται τα σύμβολα *factor*, *term*, και *expr*, με διαφορετικούς τελεστές να εμφανίζονται σε κάθε επίπεδο. Αποδίδει την προσεταιριστικότητα στο δεύτερο μισό των γραμμών 1 και 2, οι οποίες δημιουργούν τις εκφράσεις (*expr*) και τους όρους (*term*) στα αριστερά των τελεστών και όχι στα δεξιά. Στην Εικόνα 2.3, μπορούμε να δούμε πώς η ενσωμάτωση της έννοιας της προτεραιότητας στη γραμματική αποσαφηνίζει ότι ο πολλαπλασιασμός ομαδοποιεί ισχυρότερα από την πρόσθεση στην έκφραση `3 + 4 * 5`, ακόμη και χωρίς παρενθέσεις. Στην Εικόνα 2.4, βλέπουμε ότι η αφαίρεση ομαδοποιεί ισχυρότερα προς τα αριστερά, οπότε η έκφραση `10 - 4 - 3` θα αποτιμηθεί σε 3 και όχι σε 9. ■



Εικόνα 2.4 Δένδρο συντακτικής ανάλυσης για την έκφραση `10 - 4 - 3`, με αριστερή προσεταιριστικότητα (γραμματική του Παραδείγματος 2.7).

ΕΛΕΓΧΟΣ ΚΑΤΑΝΟΗΣΗΣ

1. Ποια είναι η διαφορά μεταξύ σύνταξης και σημασιολογίας;
2. Ποιες είναι οι τρεις βασικές πράξεις που μπορεί κανείς να χρησιμοποιήσει για να δημιουργήσει πολύπλοκες κανονικές εκφράσεις από άλλες απλούστερες;
3. Ποια επιπλέον πράξη (εκτός των τριών των κανονικών εκφράσεων) υποστηρίζουν οι γραμματικές χωρίς συμφραζόμενα;
4. Τι είναι η *μορφή Backus-Naur*; Πότε και γιατί επινοήθηκε;
5. Κατονομάστε μια γλώσσα στην οποία η στοίχιση επηρεάζει τη σύνταξη των προγραμμάτων.
6. Όταν μιλούμε για γλώσσες χωρίς συμφραζόμενα, τι είναι η *παραγωγή*; Τι είναι *προτασιακή μορφή*;
7. Ποια είναι η διαφορά ανάμεσα σε μια *δεξιότερη* παραγωγή και σε μια *αριστερότερη* παραγωγή; Ποια από τις δύο λέγεται και *κανονική*;
8. Τι σημαίνει ότι μια γραμματική χωρίς συμφραζόμενα είναι *διφορούμενη*;
9. Τι είναι η *προσεταιριστικότητα* και η *προτεραιότητα*; Γιατί είναι σημαντικές στα δένδρα συντακτικής ανάλυσης;

2.2 Λεκτική ανάλυση

Ο λεκτικός και ο συντακτικός αναλυτής μιας γλώσσας προγραμματισμού αναλαμβάνουν από κοινού να ανακαλύψουν τη συντακτική δομή του προγράμματος. Αυτή η διαδικασία *ανάλυσης της σύνταξης* είναι ένα αναγκαίο πρώτο βήμα για τη μετάφραση του αρχικού προγράμματος σε ένα ισοδύναμο τελικό πρόγραμμα. (Είναι επίσης το πρώτο βήμα για την άμεση διερμηνεία του προγράμματος. Γενικά, στη συνέχεια του βιβλίου θα εστιάσουμε στη μεταγλώττιση και όχι στη διερμηνεία. Τα περισσότερα από αυτά που θα εξετάσουμε είτε έχουν προφανή εφαρμογή και στη διερμηνεία, είτε είναι προφανώς άσχετα με αυτή.)

Ομαδοποιώντας τους χαρακτήρες εισόδου σε λεκτικές μονάδες, ο λεκτικός αναλυτής μειώνει δραστικά τον αριθμό των ανεξάρτητων τερματικών συμβόλων που πρέπει να εξετάσει ο συντακτικός αναλυτής, ο οποίος είναι υπολογιστικά πιο πολύπλοκος. Επιπλέον, ο λεκτικός αναλυτής συνήθως αφαιρεί τα σχόλια (ώστε ο συντακτικός αναλυτής να μη χρειάζεται να ασχοληθεί με το ότι αυτά μπορούν να εμφανίζονται οπουδήποτε στη γραμματική χωρίς συμφραζόμενα), αποθηκεύει το κείμενο των "ενδιαφερουσών" λεκτικών μονάδων όπως τα αναγνωριστικά, οι συμβολοσειρές και οι αριθμητικές σταθερές, και προσθέτει στις λεκτικές μονάδες μια ετικέτα με τον αριθμό γραμμής και στήλης, ώστε να διευκολύνεται η παραγωγή μηνυμάτων σφάλματος υψηλής ποιότητας σε μεταγενέστερες φάσεις.

```
προσπερνούμε τυχόν αρχικούς χαρακτήρες λευκών διαστημάτων (κενά διαστήματα, στηλοθέτες
    και αλλαγές γραμμής)
διαβάζουμε τον επόμενο χαρακτήρα
αν είναι ( κοιτάζουμε τον επόμενο χαρακτήρα
    αν είναι * έχουμε ένα σχόλιο
        διαβάζουμε μέχρι το τελικό *)
    διαφορετικά επιστρέφουμε αριστερή παρένθεση και ξαναχρησιμοποιούμε την
        προανάγνωση
αν είναι μια από τις λεκτικές μονάδες του ενός χαρακτήρα ([ ] , ; = + - κλπ.)
    την επιστρέφουμε
αν είναι . κοιτάζουμε τον επόμενο χαρακτήρα
    αν είναι . επιστρέφουμε ..  †
    διαφορετικά επιστρέφουμε . και ξαναχρησιμοποιούμε την προανάγνωση
αν είναι < κοιτάζουμε τον επόμενο χαρακτήρα
    αν είναι = επιστρέφουμε <=
    διαφορετικά επιστρέφουμε < και ξαναχρησιμοποιούμε την προανάγνωση
κ.ο.κ.
αν είναι γράμμα διαβάζουμε οσαδήποτε επόμενα γράμματα και ψηφία
    και πιθανόν χαρακτήρες υπογράμμισης, μέχρι που να μην υπάρχει επόμενο
    στη συνέχεια ελέγχουμε αν πρόκειται για λέξη κλειδί
        αν είναι, την επιστρέφουμε
        διαφορετικά επιστρέφουμε ένα αναγνωριστικό
    και στις δύο περιπτώσεις ξαναχρησιμοποιούμε την προανάγνωση
αν είναι ψηφίο διαβάζουμε οσαδήποτε ψηφία μέχρι να μην υπάρχει επόμενο
    αν ο επόμενος χαρακτήρας δεν είναι . επιστρέφουμε ακέραια σταθερά
    και ξαναχρησιμοποιούμε την προανάγνωση
    διαφορετικά συνεχίζουμε να διαβάζουμε μια πραγματική σταθερά
        αν ο χαρακτήρας μετά την . δεν είναι ψηφίο επιστρέφουμε ακέραια σταθερά
        και ξαναχρησιμοποιούμε την . και την προανάγνωση
κ.ο.κ.
```

Εικόνα 2.5 Σκελετός ενός αυτοσχέδιου λεκτικού αναλυτή για την Pascal. Μόνο ένα μέρος του κώδικα φαίνεται εδώ.
†Η διπλή τελεία `..` χρησιμοποιείται στην Pascal για να καθορίσει τύπους περιοχής (π.χ., `type day = 1..31`).

ΠΑΡΑΔΕΙΓΜΑ 2.8

Σκελετός ενός λεκτικού αναλυτή για την Pascal

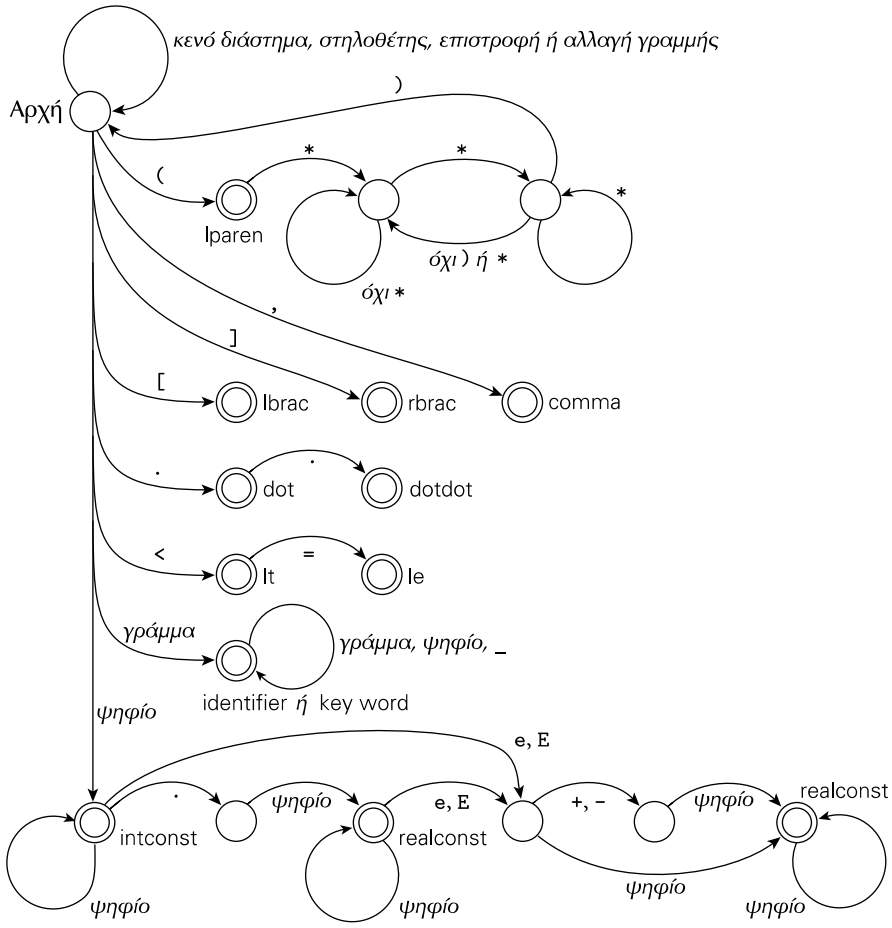
Ας υποθέσουμε προς στιγμή ότι γράφουμε ένα λεκτικό αναλυτή για την Pascal.[9] Η Εικόνα 2.5 δείχνει αυτή τη διαδικασία. Η δομή του κώδικα εξαρτάται αποκλειστικά από τον προγραμματιστή, αλλά είναι λογικό να ελέγχουμε τις απλούστερες και πιο συνηθισμένες περιπτώσεις πρώτα, να διαβάζουμε μπροστά (τον επόμενο χαρακτήρα της εισόδου) όταν χρειάζεται, και να ενσωματώσουμε βρόχους για σχόλια και για λεκτικές μονάδες μεγάλου μήκους, όπως τα αναγνωριστικά, οι αριθμοί, και οι συμβολοσειρές.

Όταν βρει μια λεκτική μονάδα, ο λεκτικός αναλυτής επιστρέφει στο συντακτικό αναλυτή. Όταν αργότερα κληθεί πάλι, επαναλαμβάνει τον αλγόριθμο από την αρχή, χρησιμοποιώντας τους επόμενους χαρακτήρες της εισόδου — μεταξύ των οποίων και τυχόν χαρακτήρες *προανάγνωσης* (look-ahead), που διαβάστηκαν αλλά δεν καταναλώθηκαν την προηγούμενη φορά. ■

Κατά κανόνα, ο λεκτικός αναλυτής αποδέχεται τη λεκτική μονάδα με το μεγαλύτερο μήκος σε κάθε κλήση του. Επομένως, η συμβολοσειρά εισόδου `foobar` δίνει πάντα το αναγνωριστικό `foobar` και ποτέ `f` ή `foo` ή `foob`. Επίσης, το `3.14159` δίνει πάντα έναν πραγματικό αριθμό και ποτέ τις τρεις ξεχωριστές λεκτικές μονάδες `3`, `.` και `14159`. Τα λευκά διαστήματα (κενά διαστήματα, στηλοθέτες, αλλαγές γραμμών, σχόλια) γενικά αγνοούνται από το λεκτικό αναλυτή, εκτός από όταν διαχωρίζουν λεκτικές μονάδες (π.χ., το `foo bar` είναι διαφορετικό από το `foobar`).

Δεν είναι δύσκολο να βασιστεί κανείς στην Εικόνα 2.5 και να γράψει με το χέρι κώδικα σε κάποια γλώσσα προγραμματισμού. Τέτοιοι αυτοσχέδιοι λεκτικοί αναλυτές χρη-

9 Όπως και στο Παράδειγμα 1.17, χρησιμοποιούμε την Pascal γιατί η λεκτική της δομή είναι σημαντικά απλούστερη από αυτή των περισσότερων σύγχρονων προστακτικών γλωσσών προγραμματισμού.



Εικόνα 2.6 Σχηματική αναπαράσταση ενός (μέρους) λεκτικού αναλυτή για την Pascal ως πεπερασμένου αυτομάτου. Η ανάλυση για κάθε λεκτική μονάδα ξεκινάει στην κατάσταση που ονομάζεται "Αρχή". Οι *τελικές* καταστάσεις, στις οποίες αναγνωρίζεται μια λεκτική μονάδα, σημειώνονται με διπλούς κύκλους.

ΠΑΡΑΔΕΙΓΜΑ 2.9

Πεπερασμένο αυτόματο για ένα μέρος του λεκτικού αναλυτή της Pascal

σιμοποιούνται πολλές φορές στην πράξη στους εμπορικούς μεταγλωττιστές, επειδή ο κώδικάς τους είναι γρήγορος και μικρός. Σε μερικές περιπτώσεις όμως, είναι λογικό να κατασκευάσει κανείς ένα λεκτικό αναλυτή με πιο δομημένο τρόπο, ως ρητή αναπαράσταση ενός *πεπερασμένου αυτομάτου*. Ένα παράδειγμα τέτοιου αυτομάτου, για ένα μέρος του λεκτικού αναλυτή της Pascal, φαίνεται στην Εικόνα 2.6. Το αυτόματο ξεκινάει σε μια διακεκριμένη αρχική κατάσταση. Στη συνέχεια μετακινείται από κατάσταση σε κα-

ΣΧΕΔΙΑΣΗ & ΥΛΟΠΟΙΗΣΗ

Ένθετα σχόλια

Τα ένθετα σχόλια είναι βολικά για τον προγραμματιστή (π.χ., για να "μετατρέπει σε σχόλια" μεγάλα τμήματα κώδικα, δηλαδή να τα απενεργοποιεί προσωρινά στο πρόγραμμα). Οι λεκτικοί αναλυτές κανονικά χειρίζονται μόνο μη αναδρομικές δομές, οπότε τα ένθετα σχόλια απαιτούν ειδική μεταχείριση. Κάποιες γλώσσες τα απαγορεύουν. Άλλες απαιτούν από τους κατασκευαστές των υλοποιήσεων να προσθέτουν στο λεκτικό αναλυτή ειδικό κώδικα για το χειρισμό των σχολίων. Οι C++ και C99 ακολουθούν μια μέση λύση: τα σχόλια `/* ... */` δεν επιτρέπεται να είναι ένθετα, αλλά σχόλια `/* ... */` μπορούν να υπάρχουν μέσα σε σχόλια `//...` και αντίστροφα. Οι προγραμματιστές επομένως μπορούν να χρησιμοποιούν τη μια μορφή σχολίων για "κανονικά" σχόλια και την άλλη για να μετατρέπουν τμήματα κώδικα σε σχόλια. Οι σχεδιαστές της C99 σημειώνουν όμως ότι η μεταγλώττιση υπό συνθήκη (`#if`) είναι προτιμητέα [Int03a, σελίδα 58].

ΠΡΑΓΜΑΤΟΛΟΓΙΑ ΤΩΝ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Michael L. Scott | ΔΕΥΤΕΡΗ ΑΜΕΡΙΚΑΝΙΚΗ ΕΚΔΟΣΗ

Στο βιβλίο *Πραγματολογία των γλωσσών προγραμματισμού*, ο Michael Scott εξηγεί με αξιοθαύμαστη σαφήνεια τις βασικές έννοιες των γλωσσών προγραμματισμού και τις λεπτομέρειες που αφορούν την υλοποίηση. Ταυτόχρονα, όμως, δείχνει στον αναγνώστη με ποιον τρόπο η αρχιτεκτονική των υπολογιστών και οι μεταγλωττιστές επηρεάζουν τη σχεδίαση και την υλοποίηση των γλωσσών προγραμματισμού. Αυτό το βιβλίο δείχνει ότι οι γλώσσες προγραμματισμού αποτελούν το πραγματικό κέντρο της επιστήμης των υπολογιστών — είναι αυτές που γεφυρώνουν το χάσμα ανάμεσα στον προγραμματιστή και τον υπολογιστή.

—JIM LARUS, Microsoft Research

Αυτή η νέα έκδοση του βιβλίου *Πραγματολογία των γλωσσών προγραμματισμού* πετυχαίνει με εξαιρετικό τρόπο να εξισορροπήσει τις τρεις βασικές προδιαγραφές που πρέπει να καλύπτει ένα βιβλίο εκμάθησης: το εύρος των γνώσεων, το βάθος της ανάλυσης, και τη σαφήνεια. Είναι βέβαιο ότι θα αποτελέσει πρότυπο για όλα τα βιβλία του συγκεκριμένου επιστημονικού τομέα.

—CHRISTOPHER VICKERY, Queens College of CUNY

Στις μέρες μας, με δεδομένη την εξέλιξη των εικονικών μηχανών (virtual machines), των γλωσσών συγγραφής σεναρίων (scripting languages), του κώδικα προγραμματισμού κινητών τηλεφώνων και συσκευών, και των διασυνδέσεων χρήστη με τη βοήθεια γραφικών, η ανάγκη για μια ολοκληρωμένη αντιμετώπιση της σχεδίασης και της υλοποίησης των γλωσσών προγραμματισμού έχει γίνει επιτακτικότερη από ποτέ. Στο βιβλίο *Πραγματολογία των γλωσσών προγραμματισμού*, ο Michael Scott δίνει έμφαση σε αυτή τη συνολική θεώρηση, ενώ ταυτόχρονα παραμένει προσηλωμένος στη σχεδίαση των γλωσσών προγραμματισμού. Ο τρόπος με τον οποίο εξετάζει αναλυτικά τις βασικές έννοιες στις οποίες βασίζονται οι σημαντικότερες σύγχρονες γλώσσες προγραμματισμού θα φανεί ανεκτίμητος τόσο στους σπουδαστές της επιστήμης των υπολογιστών όσο και στους πεπειραμένους προγραμματιστές. Αυτή η δεύτερη έκδοση του βιβλίου είναι πλήρως ενημερωμένη, περιλαμβάνει ένα νέο κεφάλαιο για τις γλώσσες συγγραφής σεναρίων, και καλύπτει τις C99, C# 2.0, και Java 5.

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ 2ης ΑΜΕΡΙΚΑΝΙΚΗΣ ΕΚΔΟΣΗΣ

- Καλύπτονται οι πιο πρόσφατες εξελίξεις στη σχεδίαση των γλωσσών προγραμματισμού, συμπεριλαμβανομένων των C99, C# 2.0, και Java 5.
- Εξετάζονται οι γλώσσες συγγραφής σεναρίων, συγκεντρωτικά σε ένα νέο κεφάλαιο αλλά και σε πολλά άλλα σημεία του βιβλίου, όπου καλύπτονται οι γλώσσες Perl, Python, Ruby, Tcl, PHP, JavaScript, XSLT, και άλλες.
- Περιλαμβάνεται ένα περιεκτικό κεφάλαιο με θέμα τον ταυτοχρονισμό (concurrency), στο οποίο καλύπτονται η C# και το νέο πακέτο ταυτοχρονισμού της Java (JSR 166).
- Περιλαμβάνονται πολλές νέες ενότητες και θέματα, όπως οι επαναλήπτες (iterators), οι εξαιρέσεις (exceptions), ο πολυμορφισμός, τα πρότυπα (templates) και η γενικότητα (generics), οι κανόνες εμβέλειας (scope) και η σειρά των δηλώσεων, η ξεχωριστή μεταγλώττιση, η συλλογή απορριμμάτων (garbage collection), τα νήματα εκτέλεσης (threads) και ο συγχρονισμός.
- Παρέχονται επιπλέον πόροι (στην Αγγλική γλώσσα) στο συνοδευτικό CD του βιβλίου, με προχωρημένο/προαιρετικό υλικό, εκατοντάδες παραδείγματα, μια λειτουργία αναζήτησης, και πολλούς συνδέσμους προς εγχειρίδια, οδηγούς εκμάθησης, μεταγλωττιστές, και διερμηνείς (interpreters) στον Παγκόσμιο Ιστό.

Ο ΣΥΓΓΡΑΦΕΑΣ

Ο **Michael L. Scott** είναι Καθηγητής και πρώην Πρόεδρος του Τμήματος Επιστήμης των Υπολογιστών στο Πανεπιστήμιο του Rochester. Είναι κάτοχος διδακτορικού από το Πανεπιστήμιο Wisconsin-Madison (1985). Έχει συμμετάσχει στη σχεδίαση της γλώσσας κατανεμημένου προγραμματισμού Lynx, των παράλληλων λειτουργικών συστημάτων Charlotte και Psyche, του παράλληλου συστήματος αρχείων Bridge, των συστημάτων κατανεμημένης κοινής μνήμης Cashmere και InterWeave, καθώς και μιας πληθώρας ευρέως χρησιμοποιούμενων αλγορίθμων συγχρονισμού και ταυτοχρονισμένων δομών δεδομένων. Το 2001 του απονεμήθηκε από το Πανεπιστήμιό του το βραβείο Robert and Pamela Goergen Award «για τα διακεκριμένα επιτεύγματα και την τεχνική του» ως καθηγητής προπτυχιακών σπουδών.

εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ
Δορυκτού 4, Στεφάνου Λαμπίρη, 10440 ΑΘΗΝΑ, Τηλ. 210-5237035

Επισκεφθείτε μας στο Internet
www.klidarithmos.gr

ISBN 978-960-461-230-7