# *Κεφάλαιο* 1

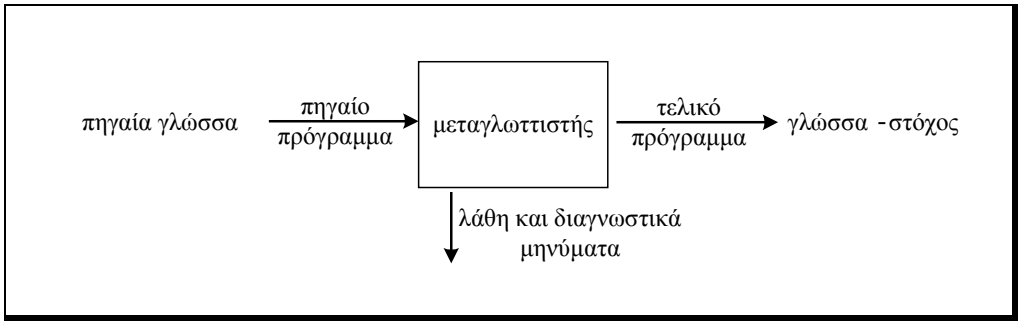
## Βασικές εισαγωγικές έννοιες

- Η δομή του μεταγλωττιστή
- Η διαδικασία της μεταγλώττισης
- Προγραμματιστικά εργαλεία
- Ανάπτυξη μεταγλωττιστών
- Μεταγλωττιστές μεταγλωττιστών

*Μεταγλωττιστής* είναι το λογισμικό, που ως σκοπό έχει τη μετάφραση ενός προγράμματος, από μία γλώσσα σε μία άλλη. Το πρόγραμμα στην αρχική του μορφή λέμε ότι είναι γραμμένο στην *πηγαία γλώσσα* και ο μεταγλωττιστής παράγει το ισοδύναμό του πρόγραμμα στη *γλώσσα-στόχο* (σχήμα 1.1). Η κατανόηση και η υλοποίηση των τεχνικών επεξεργασίας που χρησιμοποιούν οι μεταγλωττιστές δεν είναι μία απλή υπόθεση. Οι ίδιες τεχνικές βρίσκουν εφαρμογή όχι μόνο στην κατασκευή γλωσσών προγραμματισμού, αλλά και στην ανάπτυξη προγραμμάτων αλληλεπίδρασης, όπως π.χ. διερμηνευτές εντολών για την αυτοματοποίηση σεναρίων εργασιών και λογισμικό επισκόπησης κειμένου σε κάποια γλώσσα σήμανσης (π.χ. HTML). Κατά συνέπεια, η γνώση των τεχνικών αυτών και η δυνατότητα εφαρμογής τους, αποτελεί μία ζωτικής σημασίας ικανότητα για κάθε επιστήμονα της Πληροφορικής.

## 1.1 Η δομή του μεταγλωττιστή

Οι λειτουργίες ενός μεταγλωττιστή διακρίνονται λογικά σε αυτές της *ανάλυσης του πηγαίου προγράμματος* και της *σύνθεσής* του στην *τελική του μορφή* (γλώσσα-στόχο). Στην παράγραφο 1.2 περιγράφονται διεξοδικότερα όλες οι φάσεις ανάλυσης και σύνθεσης ενός προγράμματος. Οι τεχνικές επεξεργασίας, που χρησιμοποιούνται κατά την ανάλυση, είναι γενικά περισσότερο τυποποιημένες από αυτές, που επιστρατεύονται για τη σύνθεση του τελικού προγράμματος.

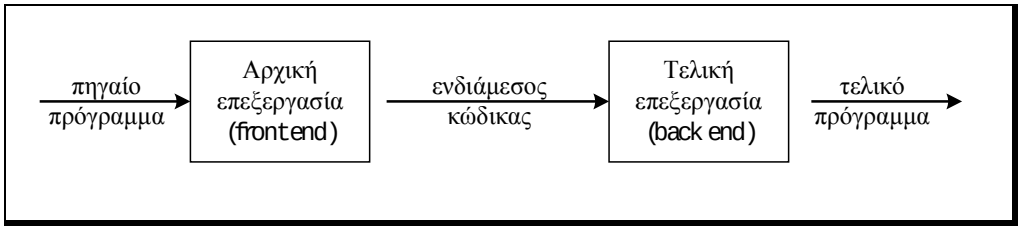



**Σχήμα 11** Η συνολική δομή του μεταγλωττιστή

Κάθε μεταγλωττιστής σε φυσικό επίπεδο, απαρτίζεται (σχήμα 12) από τα τμήματα:

- αρχικής επεξεργασίας (frontend), όπου ενσωματώνονται όλες εκείνες οι λειτουργίες, που εξαρτώνται από την πηγαία γλώσσα και
- τελικής επεξεργασίας (back end) με τις λειτουργίες που εξαρτώνται από τη γλώσσα -στόχο.

Ο φυσικός αυτός διαχωρισμός, κατά την υλοποίηση του μεταγλωττιστή, διευκολύνει τη δυνατότητα προσαρμογής του σε ενδεχόμενες αλλαγές είτε της πηγαίας γλώσσας, είτε της γλώσσας - στόχο. Στη δεύτερη περίπτωση μιλάμε για ένα *μεταφέρσιμο* μεταγλωττιστή.




**Σχήμα 12** Η φυσική δομή του μεταγλωττιστή

## 1.2 Η διαδικασία μεταγλώττισης

Η διαδικασία της μεταγλώττισης απαρτίζεται από έναν αριθμό φάσεων επεξεργασίας (σχήμα 13), που η κάθε μία δέχεται ως είσοδο ένα πρόγραμμα ισοδύναμο με το πηγαίο (σε κάποια ενδιάμεση μορφή αναπαράστασης) και παράγει ως αποτέλεσμα το ίδιο πρόγραμμα σε μία νέα μορφή. Οι φάσεις αυτές είναι η *λεξική ανάλυση*, η *συντακτική ανάλυση*, η *σημασιολογική ανάλυση*, η *βελτιστοποίηση του πηγαίου προγράμματος*, η *σύνθεση του τελικού προγράμματος* και η *βελτιστοποίησή* του. Κατά τη διάρκεια της επεξεργασίας κάθε φάσης υπάρχει αλληλεπίδραση με ένα ιδιαίτερα σημαντικό τμήμα του μεταγλωττιστή, που ονομάζεται *πίνακας συμβόλων*.

Ο πίνακας συμβόλων καταγράφει πληροφορίες, που σχετίζονται με *ονόματα*, είτε αυτά είναι ονόματα συναρτήσεων, είτε ονόματα τύπων δεδομένων, μεταβλητών ή και

σταθερών. Χρησιμοποιείται σχεδόν σε όλες τις φάσεις της μεταγλώττισης. Πιο συγκεκριμένα, κατά τη λεξική, τη συντακτική και τη σημασιολογική ανάλυση καταχωρούνται σε αυτόν ονόματα, η εκμετάλλευση των οποίων γίνεται κατά τη βελτιστοποίηση και τη σύνθεση του τελικού προγράμματος. Λειτουργίες όπως η εισαγωγή, η διαγραφή και η προσπέλαση δεδομένων στον πίνακα αυτό, χρειάζεται να εκτελούνται αποδοτικά, γι΄ αυτό και συνήθως γίνεται χρήση πίνακα Hash ή κάποιας δομής δένδρου.

Η επεξεργασία του πηγαίου προγράμματος γίνεται με *διαδοχικές σαρώσεις*. Μετά την αρχική σάρωση, που έχει ως αποτέλεσμα τη δημιουργία του συντακτικού δέντρου ή κάποιας άλλης ενδιάμεσης αναπαράστασης, οι περιπτώσεις σάρωσης που ακολουθούν έχουν ως στόχο είτε να προσθέσουν πληροφορίες στην επιλεγείσα ενδιάμεση αναπαράσταση, είτε να μεταβάλλουν τη δομή της ή ακόμη και να συνθέσουν μία νέα αναπαράσταση.

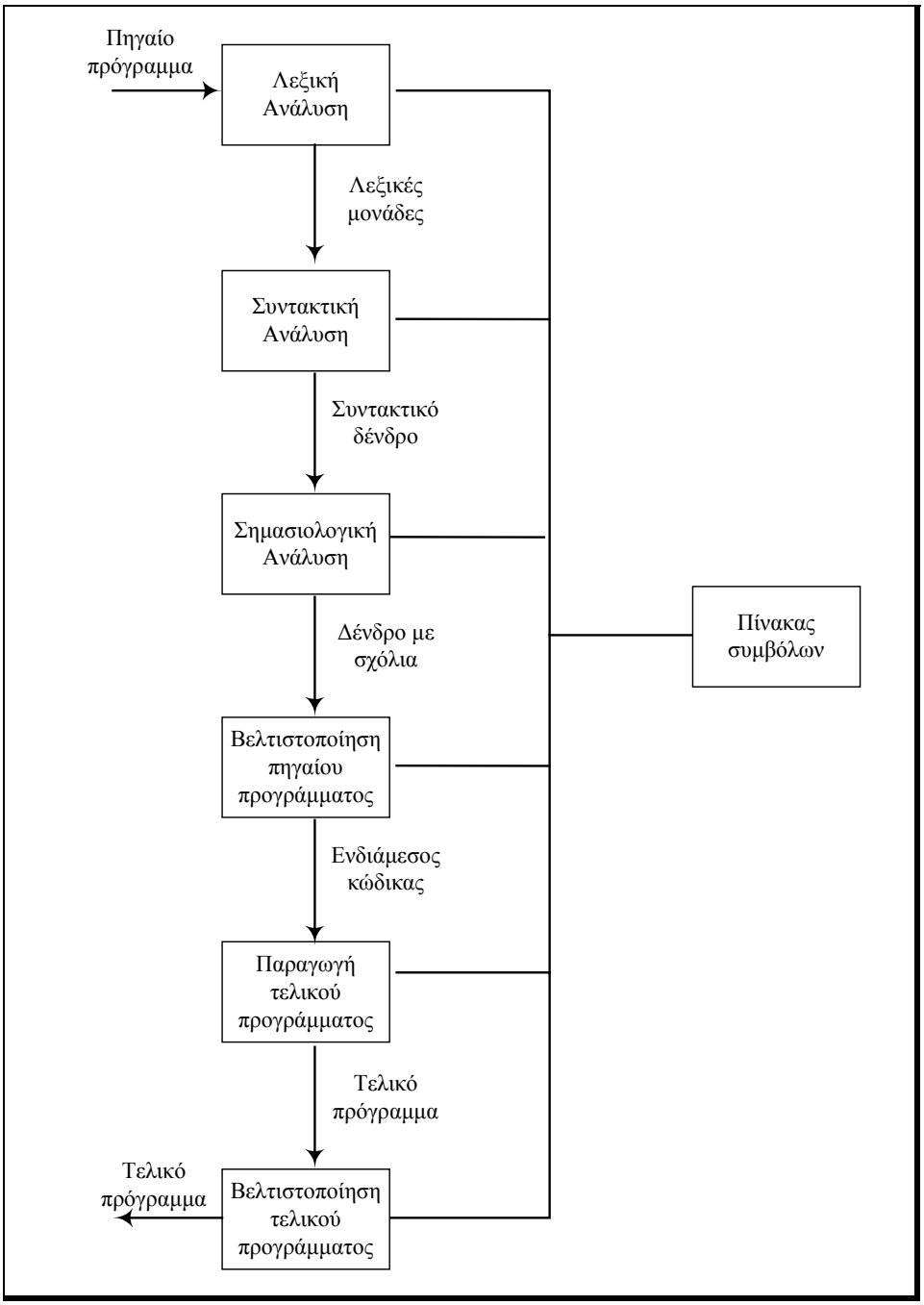
Σε κάθε σάρωση εκτελούνται πιθανώς περισσότερες της μιας φάσεις επεξεργασίας. Οι *μεταγλωττιστές μονής σάρωσης* (single pass compilers) εκτελούν όλες τις φάσεις επεξεργασίας σε μία σάρωση του προγράμματος και έχουν ως αποτέλεσμα μικρούς χρόνους μετάφρασης, αλλά όχι και τόσο γρήγορο τελικό πρόγραμμα. Γλώσσες όπως η Pascal και η C χαρακτηρίζονται από μία τέτοια δομή, που καθιστά εφικτή την ανάπτυξη μεταγλωττιστών αυτού του τύπου. Οι μεταγλωττιστές όμως με δυνατότητες βελτιστοποιήσεων συνήθως χρησιμοποιούν περισσότερες της μιας σαρώσεις, η πρώτη από τις οποίες περιλαμβάνει συνήθως τη λεξική και τη συντακτική ανάλυση του πηγαίου προγράμματος, η δεύτερη τη σημασιολογική ανάλυση και τη βελτιστοποίηση αυτού και η τελευταία τη δημιουργία του τελικού προγράμματος και τη βελτιστοποίησή του.

Σημαντικό ρόλο στην υλοποίηση κάθε μεταγλωττιστή παίζει επίσης η *διαχείριση των λαθών* του πηγαίου προγράμματος. Τα λάθη αυτά είναι δυνατό να ανιχνευθούν σε οποιαδήποτε από τις φάσεις του σχήματος 1.3 και ονομάζονται *λάθη μεταγλώττισης*.

Για κάθε λάθος, ο μεταγλωττιστής παράγει τα κατάλληλα μηνύματα (σχήμα 1.1) ενώ συνεχίζει τη διαδικασία μεταγλώττισης. Οι λειτουργίες διαχείρισης λαθών διαφέρουν από φάση σε φάση, γι΄ αυτό και η περιγραφή τους γίνεται για κάθε περίπτωση, στο αντίστοιχο κεφάλαιο του βιβλίου.

*Λεξικά λάθη* εμφανίζονται όταν εκδηλώνεται αδυναμία ταύτισης μιας συγκεκριμένης ομάδας χαρακτήρων με κάποιο πιθανό δομικό στοιχείο του ορισμού της γλώσσας. Τέτοιο λάθος είναι για παράδειγμα ένα σύμβολο που δεν περιέχεται στον ορισμό της γλώσσας.

Τα *συντακτικά λάθη* εντοπίζονται κατά τη συντακτική ανάλυση του πηγαίου προγράμματος και παρουσιάζονται όταν κάνει την εμφάνισή του ένα μη αναμενόμενο δομικό στοιχείο της γλώσσας. Παράδειγμα τέτοιου λάθους είναι μία έκφραση με διαφορετικό αριθμό αριστερών και δεξιών παρενθέσεων.




**Σχήμα 13** Η διαδικασία της μεταγλώττισης

Τέλος, *σημασιολογικά λάθη* εμφανίζονται όταν, αν και η σύνταξη ενός προγράμματος είναι σωστή, γίνεται χρήση στοιχείων της γλώσσας, με σημασία μη συμβατή προς το περιεχόμενο του προγράμματος. Περίπτωση τέτοιου λάθους είναι η χρήση ενός μη ενδεδειγμένου τελεστή σε μία αριθμητική έκφραση.

## 1.2.1 Η Λεξική Ανάλυση

Στη φάση αυτή, ο μεταγλωττιστής δέχεται ως είσοδο μία ροή χαρακτήρων, που δεν είναι άλλη από το πηγαίο πρόγραμμα. Η *λεξική ανάλυση* έχει ως στόχο το διαχωρισμό των χαρακτήρων σε ομάδες, με συγκεκριμένη σημασία, σύμφωνη με τον ορισμό της πηγαίας γλώσσας. Οι διαχωρισμένες αυτές ομάδες χαρακτήρων αποκαλούνται *λεξικές μονάδες* (lexemes) και σε κάθε μία από αυτές αντιστοιχεί κάποιο *αναγνωριστικό* (token).

Έτσι, για παράδειγμα το ακόλουθο τμήμα κώδικα σε C,

```
ar1<=i+25*ar2
```

δίνει ως αποτέλεσμα τις παρακάτω λεξικές μονάδες και τα αντίστοιχα αναγνωριστικά:

| λεξική μονάδα: | αναγνωριστικό: |
|---|---|
| ar1 | όνομα |
| <= | τελεστής <= |
| i | όνομα |
| + | τελεστής + |
| 25 | αριθμός |
| * | τελεστής * |
| ar2 | όνομα |

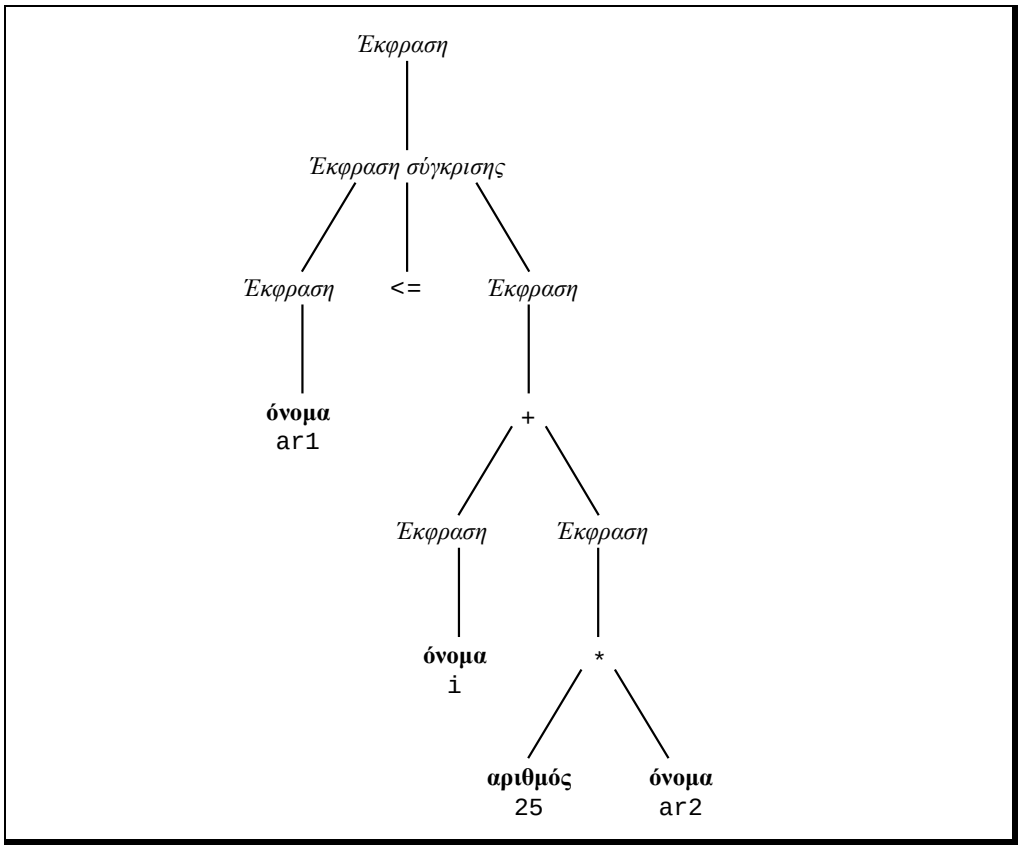
Είναι σαφές ότι κάθε λεξική μονάδα αποτελείται από έναν ή περισσότερους χαρακτήρες.

## 1.2.2 Η Συντακτική Ανάλυση

Στη φάση αυτή, ο μεταγλωττιστής δέχεται ως είσοδο τις λεξικές μονάδες, που παράγονται κατά τη λεξική ανάλυση και εκτελεί τη *συντακτική ανάλυση*, όπου ελέγχεται η δομή του προγράμματος. Η συγκεκριμένη επεξεργασία παρουσιάζει αναλογίες με τη διενέργεια γραμματικής ανάλυσης σε μία πρόταση διατυπωμένη σε φυσική γλώσσα. Ουσιαστικά εντοπίζονται τα δομικά στοιχεία του προγράμματος και καθορίζεται η μεταξύ τους σχέση. Το αποτέλεσμα της συντακτικής ανάλυσης είναι ένα δένδρο, το οποίο ανάλογα με τη μορφή, που έχει, αποκαλείται *παράγωγο* ή *συντακτικό δένδρο*. Ένα συντακτικό δένδρο είναι μία συμπτυγμένη μορφή του αντίστοιχου παράγωγου δένδρου.

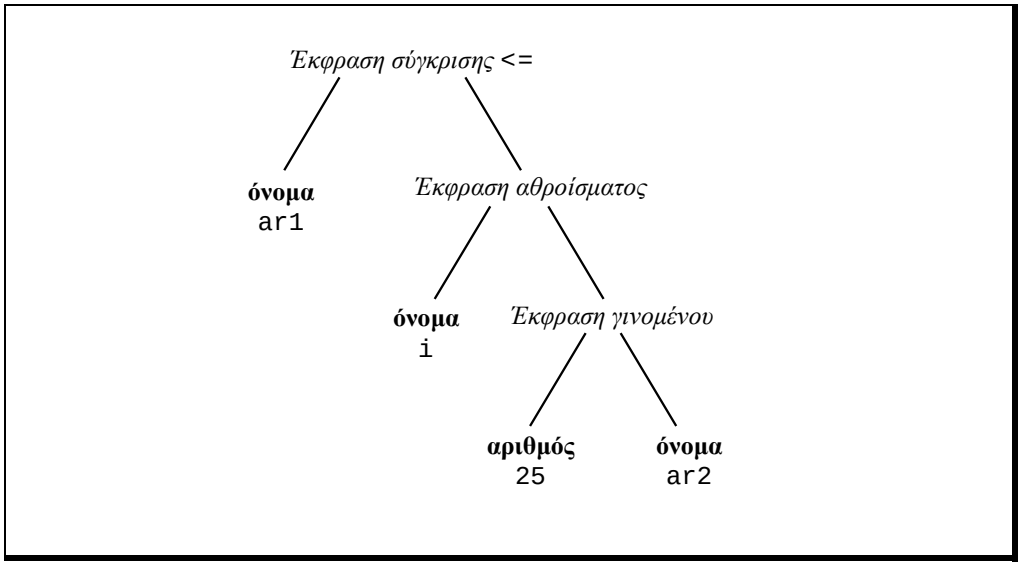
Στο παράδειγμα της παραγράφου 1.2.1 το αποτέλεσμα της συντακτικής ανάλυσης θα είχε αποδώσει τη δομή, που εικονίζεται στο Σχήμα 1.4 και εκφράζεται από ένα μόνο δομικό στοιχείο, που ονομάζεται *έκφραση*.

Στο παράγωγο δένδρο του σχήματος 1.4 ο κάθε εσωτερικός κόμβος αναγράφει το είδος του δομικού στοιχείου, που αναπαριστά το τμήμα του δένδρου που βρίσκεται κάτω από αυτόν. Στα φύλλα αναπαριστώνται οι λεξικές μονάδες που αναγνωρίστηκαν, με τη σειρά με την οποία αυτές εμφανίστηκαν στο μεταγλωττιστή.




**Σχήμα 1.4** Το παράγωγο δένδρο της έκφρασης `ar1<=i+25*ar2`

Κατ΄ επέκταση, η αναπαράσταση ενός ολόκληρου προγράμματος με παράγωγο δένδρο δεν είναι ιδιαίτερα αποδοτική εξαιτίας του μεγέθους, που μπορεί αυτό να προσλάβει. Έτσι, συχνά προτιμάται η ανάπτυξη συντακτικού δένδρου, δηλαδή μιας συμπτυγμένης μορφής του παράγωγου δένδρου. Στο δένδρο αυτό δεν εμφανίζεται ένας αριθμός κόμβων, στους οποίους μπορεί να περιλαμβάνονται ακόμη και οι κόμβοι κάποιων λεξικών μονάδων που αναγνωρίστηκαν. Στο σχήμα 1.5 απεικονίζεται το συντακτικό δένδρο της ίδιας έκφρασης (`ar1<=i + 25*ar2`).



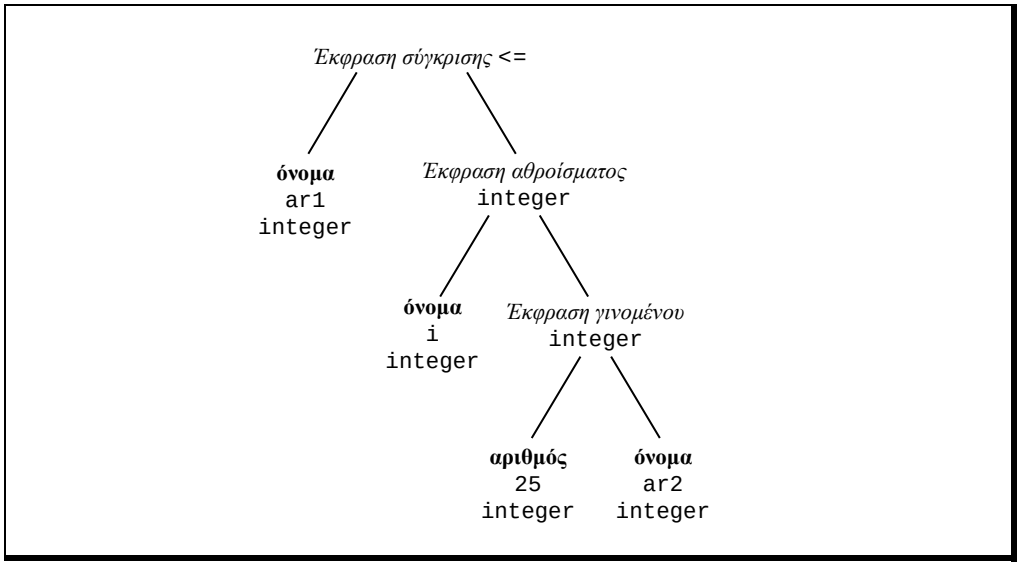

**Σχήμα 15** Το συντακτικό δένδρο της έκφρασης `ar1<=i+25*ar2`

## 12.3 Η Σημασιολογική Ανάλυση

Οι *ιδιότητες*, που σχετίζονται με το περιεχόμενο των δομικών στοιχείων που αναγνωρίζονται και δε μπορούν να καθορισθούν μόνο με τη συντακτική ανάλυση του προγράμματος, συγκροτούν τη *στατική σημασία* αυτού. Αντίθετα, η *δυναμική σημασία* ενός προγράμματος εκφράζεται από ιδιότητες, που καθορίζονται πλήρως μόνο κατά την εκτέλεσή του.

Οι πιο συνηθισμένες περιπτώσεις στατικής σημασίας, που αποτελούν το αντικείμενο της επονομαζόμενης *σημασιολογικής ανάλυσης*, είναι οι δηλώσεις και οι έλεγχοι τύπων δεδομένων. Οι πληροφορίες που προκύπτουν από μία τέτοια ανάλυση μπορεί να ενσωματώνονται στο *συντακτικό δένδρο με τη μορφή σχολίων*.

Ας θεωρήσουμε πάλι την έκφραση `ar1<=i+25*ar2` των παραγράφων 12.1 και 12.2 και ας υποθέσουμε ότι οι μεταβλητές `ar1`, `i` και `ar2` έχουν δηλωθεί ως ακέραιες μεταβλητές (`int ar1,i,ar2`). Τότε, κατά τη σημασιολογική ανάλυση, οι πληροφορίες αυτές συνδέονται με τα αντίστοιχα ονόματα και ακολούθως χρησιμοποιούνται στον έλεγχο της σύγκρισης ως προς τους συγκεκριμένους τύπους. Το συντακτικό δένδρο που προκύπτει είναι αυτό του σχήματος 16. Βέβαια, σε περίπτωση *ασυμφωνίας τύπων* (type mismatch) θα πρέπει να γίνουν οι κατάλληλες ενέργειες διαχείρισης λάθους.




**Σχήμα 1.6** Σημασιολογική ανάλυση της έκφρασης `ar1<=i+25*ar2`

## 1.2.4 Η βελτιστοποίηση του πηγαίου προγράμματος

Κάθε μεταγλωττιστής διαφοροποιείται όχι μόνο ως προς τις δυνατότητες βελτιστοποιήσεων που διαθέτει, αλλά και ως προς τις φάσεις μεταγλώττισης, που αυτές εκτελούνται. Είναι σαφές ότι οι πρώτες βελτιστοποιήσεις έπονται της σημασιολογικής ανάλυσης και αφορούν βελτιώσεις στο επίπεδο του πηγαίου προγράμματος. Μία τέτοια περίπτωση είναι η *σύμπτυξη σταθερών*, κατά την οποία αντικαθιστώνται οι πράξεις με το τελικό αποτέλεσμα (π.χ. η `x=2*4+ab2` γίνεται `x=8+ab2`).

Βελτιώσεις αυτού του τύπου μπορούν να ενσωματωθούν απευθείας στο συντακτικό δένδρο, με τη σύμπτυξη τμήματος αυτού σε ένα μόνο κόμβο. Παρόλα αυτά, συχνά προτιμάται η βελτιστοποίηση μιας γραμμικής αναπαράστασης του συντακτικού δένδρου, που είναι ευρέως γνωστή ως *κώδικας τριών διευθύνσεων*.

Είναι χαρακτηριστικό ότι στο Σχήμα 1.3 χρησιμοποιείται ο όρος *ενδιάμεσος κώδικας* ακριβώς για να συμπεριλάβει την πιθανότητα χρήσης διαφορετικής αναπαράστασης του πηγαίου προγράμματος, είτε αυτή είναι συντακτικό δένδρο, είτε κώδικας τριών διευθύνσεων, είτε *κώδικας P* (ιδιαίτερα διαδεδομένος στους μεταγλωττιστές της Pascal), είτε κάτι άλλο.

Άλλες βελτιστοποιήσεις που μπορούν να γίνουν στο επίπεδο του ενδιάμεσου κώδικα είναι η διάδοση των εντολών εκχώρησης στο υπόλοιπο πρόγραμμα, η απομάκρυνση των μη προσιτών τμημάτων του κώδικα (που ουδέποτε εκτελείται), η αντικατάσταση υπολογιστικά ακριβών εκφράσεων από άλλες, η μετονομασία μεταβλητών, η απαλοιφή κοινών εκφράσεων, η μετακίνηση κώδικα έξω από βρόχους, όπου αυτό

είναι εφικτό και η απλοποίηση μεταβλητών, που οι τιμές τους μεταβάλλονται μέσα σε βρόχους επανάληψης (*επαγωγικών μεταβλητών*).

## 1.2.5 Η σύνθεση του τελικού προγράμματος

Στη φάση αυτή, χρησιμοποιείται η ενδιάμεση αναπαράσταση του πηγαίου προγράμματος, όπως αυτή έχει προκύψει από την επεξεργασία των προηγούμενων φάσεων, για τη δημιουργία κώδικα στη γλώσσα-στόχο.

Ο κώδικας που δημιουργείται μπορεί να προορίζεται για εκτέλεση σε συγκεκριμένο επεξεργαστή ή σε κάποια υπερβατική μηχανή (π.χ. Java Virtual Machine). Γενικά, η δημιουργία κώδικα για συγκεκριμένο επεξεργαστή απλουστεύει τη σχεδίαση του μεταγλωττιστή. Σε ότι αφορά την περίπτωση της υπερβατικής μηχανής ο κώδικας που δημιουργείται χρησιμοποιείται ως ενδιάμεση αναπαράσταση, η οποία, σε μεταγενέστερη φάση, θα μετατραπεί σε κώδικα για συγκεκριμένο επεξεργαστή. Έτσι επιτυγχάνεται μεγαλύτερη μεταφερσιμότητα.

Η απευθείας σύνθεση κώδικα μηχανής είναι διαδικασία χρονοβόρα και ιδιαίτερα επιρρεπής σε λάθη, αφού εξαρτάται από τον ακριβή καθορισμό μεγάλου πλήθους ψηφιακών μορφών. Έτσι, ως γλώσσα-στόχος επιλέγεται συνήθως η συμβολική γλώσσα του επεξεργαστή όπου θα εκτελείται το τελικό πρόγραμμα. Όταν ένα πρόγραμμα στην τελική του μορφή είναι διατυπωμένο σε συμβολική γλώσσα, τότε για τη μετατροπή του σε εκτελέσιμο, αρκεί η χρήση του κατάλληλου συμβολομεταφραστή. Τελευταία χρησιμοποιείται η C ή η C++ ως γλώσσα-στόχος, λόγω της ευρείας διάδοσης μεταγλωττιστών σύγχρονων προδιαγραφών, για τις συγκεκριμένες γλώσσες.

## 1.2.6 Η βελτιστοποίηση του τελικού προγράμματος

Αν και η βελτιστοποίηση του τελικού προγράμματος μπορεί να αντιμετωπίζεται ως μία ανεξάρτητη φάση επεξεργασίας, συχνά επιλέγεται η ενσωμάτωση τεχνικών βελτίωσης κατά τη σύνθεση του τελικού προγράμματος. Φυσικά, σε κάθε περίπτωση και για οποιαδήποτε βελτίωση, πρέπει πάντα να διασφαλίζεται η μη αλλοίωση της σημασίας του κώδικα.

Οι εναλλακτικές δυνατότητες βελτίωσης εξαρτώνται πάντα από τον επεξεργαστή για τον οποίο δημιουργήθηκε ένα πρόγραμμα στην τελική του μορφή. Πιο συγκεκριμένα, εξαρτώνται από χαρακτηριστικά του, όπως το πλήθος και η εξειδίκευση των καταχωρητών του, οι μορφές προσπέλασης στη μνήμη και οι εντολές του. Είναι πάντα επιθυμητή η καλύτερη δυνατή διαχείριση των καταχωρητών, η χρήση του πιο κατάλληλου τρόπου προσπέλασης των δεδομένων και η χρήση εντολών - ιδιώματα του επεξεργαστή, που βελτιώνουν αισθητά την ταχύτητα εκτέλεσης του τελικού προγράμματος.

### 1.2.7 Δομές δεδομένων

Από τα εισαγωγικά στοιχεία, που παρουσιάσθηκαν σχετικά με τη διαδικασία μεταγλώττισης, είναι σαφές ότι υπάρχει μία διαρκής αλληλεπίδραση των αλγορίθμων που χρησιμοποιούνται κατά τις διάφορες φάσεις επεξεργασίας και των δομών δεδομένων που επιστρατεύονται σε κάθε μία από αυτές. Έτσι, η υλοποίηση των αλγορίθμων πρέπει να γίνεται με όσο το δυνατό πιο αποδοτικό τρόπο, χωρίς να υφίσταται σημαντικά μεγάλος βαθμός πολυπλοκότητας. Έχουμε ήδη αναφερθεί στη δομή του πίνακα συμβόλων με την οποία υπάρχει αλληλεπίδραση σε όλες τις φάσεις επεξεργασίας του μεταγλωττιστή. Στη συνέχεια γίνεται σύντομη αναφορά και στις υπόλοιπες δομές, αρχής γενομένης από την αναπαράσταση των λεξικών μονάδων.

Οι λεξικές μονάδες αναπαριστώνται μέσω ενός απαριθμητού τύπου δεδομένων, οι τιμές του οποίου αντιστοιχούν στο σύνολο των αναγνωριστικών της πηγαίας γλώσσας. Συχνά όμως είναι αναγκαία και η αποθήκευση στη μνήμη ολόκληρης της ομάδας των χαρακτήρων, που συγκροτούν τη λεξική μονάδα ή πληροφοριών που προκύπτουν από αυτήν, όπως για παράδειγμα το όνομα ή η τιμή της, αν βέβαια πρόκειται για αριθμό.

Γενικά, αρκεί η *ανάγνωση των λεξικών μονάδων μία προς μία* και η προσωρινή αποθήκευση της τρέχουσας, σε κάποια μεταβλητή καθολικής εμβέλειας. Όταν όμως γίνεται ταυτόχρονη ανάγνωση περισσοτέρων λεξικών μονάδων (όπως συνήθως στη γλώσσα FORTRAN), τότε χρησιμοποιείται μία κατάλληλη δομή πίνακα.

Στις περιπτώσεις που το αποτέλεσμα της συντακτικής ανάλυσης του προγράμματος έχει τη μορφή συντακτικού δένδρου, αυτό αναπαριστάται με μία τυπική δενδροειδή δομή, που αναπτύσσεται δυναμικά με τη χρήση δεικτών. Η προσπέλαση στη δομή αυτή γίνεται από μία μόνο μεταβλητή, που δείχνει τη διεύθυνση της ρίζας. Κάθε κόμβος της δομής περιλαμβάνει μία εγγραφή, τα πεδία της οποίας περιέχουν τις πληροφορίες, που συλλέγονται κατά τη συντακτική, αλλά και κατά τη σημασιολογική ανάλυση του προγράμματος. Πολλές φορές για την εξοικονόμηση χώρου προτιμάται η δυναμική ανάπτυξη των πεδίων της κάθε εγγραφής, ενώ σε κάποιες άλλες περιπτώσεις προτιμάται η αποθήκευση πληροφοριών στον πίνακα συμβόλων. Τέλος, το γεγονός ότι οι κόμβοι του συντακτικού δένδρου εκφράζουν διαφορετικά είδη δομικών στοιχείων της γλώσσας (εκφράσεις, εντολές, δηλώσεις κλπ.), με διαφορετικές απαιτήσεις αποθήκευσης πληροφοριών, συχνά παραπέμπει στη χρήση εγγραφών μεταβλητού μήκους (unions).

Αν όμως ως ενδιάμεση αναπαράσταση επιλέγεται να χρησιμοποιηθεί ο κώδικας τριών διευθύνσεων ή κάποια παραλλαγή του, τότε αυτός διατηρείται στη μνήμη είτε ως πίνακας συμβολοσειρών, είτε ως ένα προσωρινό αρχείο κειμένου, είτε ως μία συνδετική λίστα. Ιδιαίτερα για μεταγλωττιστές που εκτελούν πολύπλοκες βελτιστοποιήσεις, προτιμούνται δομές, που αφήνουν ανοικτή την πιθανότητα αναδιοργάνωσης των δεδομένων.

## 1.3 Θέματα που σχετίζονται με τη λειτουργία των μεταγλωττιστών

Στην παράγραφο αυτή γίνεται μία σύντομη περιγραφή των προγραμματιστικών εργαλείων που η χρήση τους σχετίζεται, συνδυάζεται ή υποκαθιστά τη χρήση μεταγλωττιστών και που συχνά όλα μαζί συγκροτούν ένα πλήρες περιβάλλον ανάπτυξης. Τέλος, γίνεται αναφορά και στις προαιρετικές επιλογές χρήσης και τους μηχανισμούς διεπαφής, που διαθέτουν οι περισσότεροι μεταγλωττιστές.

### 1.3.1 Ο Διερμηνευτής

Ένας *διερμηνευτής* διαφέρει από ένα μεταγλωττιστή στο γεγονός ότι στην περίπτωσή του γίνεται απευθείας εκτέλεση του πηγαίου προγράμματος και όχι επεξεργασία του για τη σύνθεση του τελικού προς εκτέλεση προγράμματος.

Ο διερμηνευτής μεταφράζει και εκτελεί κάθε εντολή του προγράμματος ξεχωριστά και στις περιπτώσεις εντοπισμού συντακτικών λαθών, παράγει τα κατάλληλα μηνύματα. Αν μία εντολή είναι μέρος ενός επαναληπτικού βρόχου, τότε η εντολή αυτή μεταφράζεται εκ νέου κάθε φορά που εκτελείται με αποτέλεσμα την επιμήκυνση του χρόνου εκτέλεσης.

Η χρήση διερμηνευτών αποτελεί μία αποδοτική επιλογή στις περιπτώσεις που:

- ο προγραμματιστής εργάζεται σε διαδραστικό περιβάλλον και είναι επιθυμητό να βλέπει τα αποτελέσματα της κάθε εντολής πριν την εισαγωγή της επόμενης,
- το πρόγραμμα χρησιμοποιείται μόνο μία φορά και μετά εγκαταλείπεται,
- η ταχύτητα εκτέλεσης δεν είναι σημαντική,
- οι περισσότερες εντολές του προγράμματος εκτελούνται μόνο μία φορά ή τουλάχιστο όχι συχνά και
- οι εντολές έχουν απλή μορφή και έτσι μπορούν να αναλυθούν εύκολα και γρήγορα.

Αντίθετα, όταν ένα πρόγραμμα προορίζεται για λειτουργική χρήση και η ταχύτητα εκτέλεσής του έχει καθοριστική σημασία, τότε σίγουρα η χρήση διερμηνευτή δεν αποτελεί την πιο αποδοτική επιλογή. Είναι χαρακτηριστικό ότι η εκτέλεση ενός προγράμματος μέσα σε ένα περιβάλλον διερμήνευσης μπορεί να είναι μέχρι και εκατό φορές πιο αργή από ότι η εκτέλεσή του μετά από μεταγλώττιση.

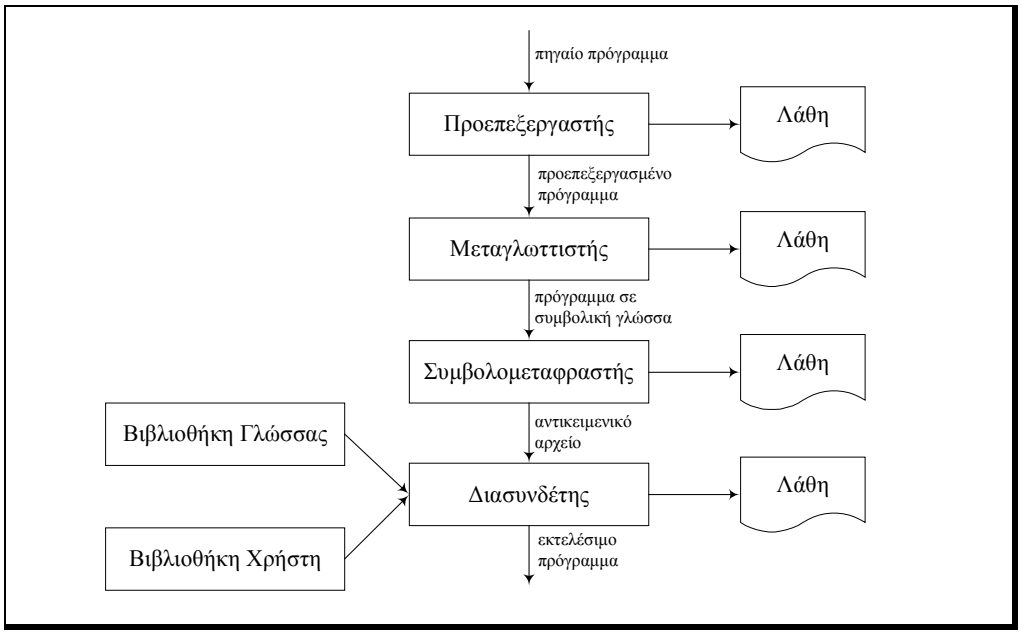
Κάποιες γλώσσες προγραμματισμού, όπως για παράδειγμα η BASIC και οι περισσότερες συναρτησιακές γλώσσες (π.χ. η LISP), χρησιμοποιούν συνήθως διερμηνευτή. Άλλες άξιες λόγου περιπτώσεις διερμήνευσης είναι το κέλυφος του λειτουργικού συστήματος UNIX και η ευρέως διαδεδομένη διαδραστική γλώσσα ερωτημάτων SQL.

## 1.3.2 Ο Συμβολομεταφραστής

Οι *συμβολομεταφραστές* χρησιμοποιούνται για τη μεταγλώττιση προγραμμάτων *συμβολικής γλώσσας μηχανής*. Ακριβώς επειδή η συγκεκριμένη γλώσσα είναι μία συμβολική μορφή της ίδιας της γλώσσας μηχανής του υπολογιστή, η μεταγλώττιση αυτή δεν παρουσιάζει καμία δυσκολία. Συνήθως οι μεταγλωττιστές χρησιμοποιούν ως γλώσσα-στόχο τη συμβολική γλώσσα και ακολούθως τον κατάλληλο συμβολομετφραστή, για την παραγωγή του *αντικειμενικού* αλλά όχι πάντα εκτελέσιμου προγράμματος.

## 1.3.3 Ο Διασυνδέτης

Ο *διασυνδέτης* συλλέγει μονάδες προγραμμάτων (π.χ. συναρτήσεις, units κ.α.) που έχουν ήδη μεταγλωττιστεί και υπάρχουν σε *αντικειμενικά αρχεία* και παράγει το τελικό εκτελέσιμο πρόγραμμα. Ο όρος *αντικειμενικό αρχείο* αναφέρεται ουσιαστικά σε πρόγραμμα γλώσσας μηχανής, το οποίο όμως δεν είναι εκτελέσιμο. Ο διασυνδέτης ενσωματώνει τον απαραίτητο κώδικα για να καταστεί το μεταφρασμένο πρόγραμμα εκτελέσιμο. Ο κώδικας αυτός παρέχει τη διασύνδεση με πόρους του συστήματος, όπως οι εκχωρητές μνήμης και οι συσκευές εισόδου-εξόδου και τη διασύνδεση με βιβλιοθήκες της γλώσσας.




**Σχήμα 1.7** Αλληλεπιδράσεις εργαλείων σε σχέση με τη χρήση μεταγλωττιστή

Η όλη διαδικασία εξαρτάται σε μεγάλο βαθμό από τον τύπο του λειτουργικού συστήματος και τον επεξεργαστή που χρησιμοποιείται και για το λόγο αυτό δε δίνουμε ιδιαίτερη έμφαση στην περιγραφή της. Στη συνέχεια όταν αναφερόμαστε στο

τελικό πρόγραμμα μιας μεταγλώττισης δε γίνεται διάκριση μεταξύ αντικειμενικού και εκτελέσιμου, καθώς αυτό δεν είναι σημαντικό για την περιγραφή των τεχνικών επεξεργασίας στις οποίες θα αναφερθούμε.

Οι διασυνδέτες τελευταίας γενιάς έχουν τη δυνατότητα διασύνδεσης προγραμμάτων που προέρχονται από περισσότερες της μιας πηγαίες γλώσσες. Τέλος, οι επονομαζόμενοι *διασυνδέτες επικάλυψης* (overlay linkers) δημιουργούν το εκτελέσιμο πρόγραμμα σε αρχείο από το οποίο ένα μέρος μόνο φορτώνεται στη μνήμη τη στιγμή της εκτέλεσης, αφήνοντας το υπόλοιπο στο δίσκο μέχρι τη στιγμή που θα χρησιμοποιηθεί.

## 1.3.4 Ο Προεπεξεργαστής

Ο *προεπεξεργαστής* είναι το πρόγραμμα που καλείται από το μεταγλωττιστή, πριν από την έναρξη της μετάφρασης. Η αποστολή του είναι η διαγραφή των σχολίων, η συμπερίληψη άλλων αρχείων, που περιέχουν πηγαίο κώδικα και η αντικατάσταση κειμένου που ορίζεται σε *μακροεντολές* (macros).

Οι μακροεντολές επιτρέπουν τη δημιουργία ονομάτων σταθερών, που δεν επιβαρύνουν το χρόνο εκτέλεσης του προγράμματος. Σε κάθε περίπτωση χρήσης μιας σταθεράς, που η τιμή της ορίζεται σε μακροεντολή, η προεπεξεργασία εκτελεί αντικατάσταση με τη δηλωθείσα τιμή ακριβώς πριν από την εκκίνηση της μετάφρασης. Συχνά παρέχεται και η δυνατότητα αντικατάστασης προτύπων συναρτήσεων από τις αντίστοιχες συναρτησιακές εκφράσεις ([ΛΑ03]).

Η χρήση μακροεντολών καθιστά ένα πρόγραμμα εύκολα αναγνώσιμο και τροποποιήσιμο, αφού συνοψίζει τη δήλωση συναρτήσεων και σταθερών που χρησιμοποιούνται συχνά στο πρόγραμμα, σε μία μόνο γραμμή που προηγείται αυτού.

Σταθερές και συναρτήσεις που ορίζονται σε επίπεδο μακροεντολών χρησιμοποιούνται και στην εκσφαλμάτωση του προγράμματος αποφεύγοντας έτσι την ενσωμάτωση του κώδικα εκσφαλμάτωσης στο εκτελέσιμο που τελικά παράγεται.

### *Παράδειγμα* 1.3.1

Στη γλώσσα προγραμματισμού C η ενσωμάτωση κώδικα εκσφαλμάτωσης μπορεί εύκολα να γίνει με τον ορισμό της μακροεντολής

```
#define DEBUG_FLAG 0
```

και την εισαγωγή κώδικα εκσφαλμάτωσης (π.χ. εκτύπωση για έλεγχο των τιμών συγκεκριμένων μεταβλητών) ως εξής:

```
#if DEBUG_FLAG
      ... κώδικας εκσφαλμάτωσης ...
#endif
```

Όταν η τιμή της σταθεράς DEBUG_FLAG τίθεται σε 1 τότε εκτελούνται οι εντολές εκσφαλμάτωσης που έχουν εισαχθεί. Επανεφέροντας την τιμή της DEBUG_FLAG σε 0 αποφεύγουμε την ενσωμάτωση του κώδικα εκσφαλμάτωσης στο εκτελέσιμο που παράγεται. ■

## 1.3.5 Ο Συντάκτης/Διορθωτής

Τα πηγαία προγράμματα συγγράφονται μέσα στο περιβάλλον ενός συντάκτη/διορθωτή, που αποθηκεύει τελικά το κείμενο σε κάποια τυποποιημένη μορφή (π.χ. κώδικα ASCII). Σε πολλές περιπτώσεις ο συντάκτης/διορθωτής αποτελεί μέρος ενός ολοκληρωμένου διαδραστικού περιβάλλοντος ανάπτυξης, με δυνατότητες κλήσης επιπλέον βοηθητικών λειτουργιών, κατά τη συγγραφή του προγράμματος.

Πέρα από τις στοιχειώδεις λειτουργίες επεξεργασίας κειμένου ένας συντάκτης/διορθωτής κατάλληλος για προγραμματισμό πρέπει να διαθέτει δυνατότητες:

- μεταγλώττισης και εντοπισμού λαθών μέσα από το περιβάλλον του συντάκτη,
- συνοπτικής εμφάνισης προγραμμάτων (π.χ. εμφάνισης μόνο των ονομάτων των συναρτήσεων),
- κατατοπιστικής βοήθειας σχετικά με τη γλώσσα που υποστηρίζει,
- αυτόματης στοίχισης αγκύλων ή εντολών begin-end,
- πρότυπα ή χαρακτηριστικά αυτόματης συμπλήρωσης των πιο κοινών δομών σύνταξης (π.χ. συμπλήρωση της δομής ενός βρόχου for ευθύς μόλις ο προγραμματιστής εισάγει τη δεσμευμένη λέξη),
- «έξυπνης» στοίχισης και εύκολης αλλαγής της στοίχισης μιας ενότητας εντολών μετά από αλλαγή στη λογική του προγράμματος,
- μνήμης για συμβολοσειρές που χρησιμοποιούνται συχνά, ώστε να μην επαναλαμβάνεται η εξολοκλήρου δακτυλογράφησή τους κάθε φορά που αυτό χρειάζεται,
- αναζήτησης και αντικατάστασης συμβολοσειρών σε ομάδα αρχείων,
- ταυτόχρονης σύνταξης κώδικα σε περισσότερα του ενός αρχεία και
- αναίρεσης προηγούμενων ενεργειών.

Το περιβάλλον ενός συντάκτη/διορθωτή είναι επίσης επιθυμητό να περιλαμβάνει χρηστικές δυνατότητες επισκόπησης κώδικα (browsing). Οι σημαντικότερες από αυτές είναι οι δυνατότητες:

- παραγωγής διασταυρωτικών αναφορών, που επιτρέπουν π.χ. τον άμεσο εντοπισμό της δήλωσης μιας μεταβλητής ή συνάρτησης, μαζί με όλα τα σημεία του προγράμματος όπου χρησιμοποιείται και
- επισκόπησης της δομής των κλήσεων των συναρτήσεων, όταν αυτές καλούν η μία την άλλη.

### 1.3.6 Το εργαλείο make

Ο σκοπός του εργαλείου `make` είναι το γρήγορο «κτίσιμο» του εκτελέσιμου προγράμματος από τα επί μέρους τμήματά του, με μία μόνο κλήση εντολής. Το συγκεκριμένο εργαλείο υιοθετήθηκε από όλους τους προγραμματιστές της `C` που χρησιμοποιούσαν μεταγλωττιστή από τη γραμμή εντολών, αλλά η χρήση του διαδόθηκε και στον προγραμματισμό με άλλες γλώσσες, ακόμη και όταν αυτός γίνεται μέσα από ένα ολοκληρωμένο περιβάλλον ανάπτυξης.

Το εργαλείο `make` ουσιαστικά το μόνο που κάνει είναι να καλεί άλλα προγράμματα, ακολουθώντας ένα σαφώς ορισμένο σχέδιο «κτισίματος» του εκτελέσιμου. Το σχέδιο δημιουργείται από τον προγραμματιστή ή το περιβάλλον ανάπτυξης σε ένα απλό αρχείο κειμένου με το όνομα `makefile`. Το `makefile` δεν περιγράφει τη σειρά των εντολών που πρέπει να εκτελεστούν, αλλά ποια τμήματα του προγράμματος χρησιμοποιούνται στη δημιουργία άλλων τμημάτων και τελικά στη δημιουργία του εκτελέσιμου. Η περιγραφή αυτή γίνεται με τη διατύπωση συγκεκριμένων κανόνων που ονομάζονται *εξαρτήσεις*.

Σε ένα πρόγραμμα που απαρτίζεται από υπομονάδες, κάθε τμήμα κώδικα δημιουργείται με την κατάλληλη κλήση ενός μεταγλωττιστή ή ενός συμβολομεταφραστή και ενός διασυνδέτη. Σε κάθε κλήση τα προαναφερόμενα εργαλεία δέχονται ως είσοδο ένα ή περισσότερα αρχεία και δημιουργούν από αυτά νέα αρχεία.

Ένας συμβολομεταφραστής δέχεται ως είσοδο το πηγαίο πρόγραμμα σε συμβολική γλώσσα και δημιουργεί ένα αντικειμενικό αρχείο ή σπανιότερα το ίδιο το εκτελέσιμο. Κατά συνέπεια η ύπαρξη κάθε αντικειμενικού αρχείου *εξαρτάται* από το αντίστοιχο πηγαίο. Όταν ο διασυνδέτης χρησιμοποιεί έναν αριθμό αντικειμενικών αρχείων για τη δημιουργία του εκτελέσιμου, είναι σαφές ότι η διαδικασία *εξαρτάται* τελικά από την προηγούμενη ύπαρξη όλων των αντικειμενικών αρχείων. Το περιεχόμενο ενός `makefile` προσδιορίζει ποια αρχεία είναι απαραίτητα στη δημιουργία ποιων άλλων αρχείων και πως αυτά δημιουργούνται.

Με την κλήση του `make` το εργαλείο διαβάζει τις εξαρτήσεις που περιέχονται στο `makefile` και καλεί τα κατάλληλα εργαλεία με τον προκαθορισμένο τρόπο και την κατάλληλη σειρά για τη δημιουργία του εκτελέσιμου. Για την καλύτερη κατανόηση της δομής ενός αρχείου `makefile` σχολιάζουμε το απλό παράδειγμα που ακολουθεί:

αρχείο `makefile`

```
myProg: myProg.o
        gcc myProg.o –o myProg
myProg.o: myProg.asm
        nasm –f elf myProg.asm
```

Η πρώτη γραμμή ορίζει μία εξάρτηση και πιο συγκεκριμένα αυτή της δημιουργίας του εκτελέσιμου: η δημιουργία του εξαρτάται από την υπάρξη του αντικειμενικού αρχείου και αυτή γίνεται με την κλήση του διασυνδέτη του `gcc`, που βρίσκεται στη δεύτερη γραμμή. Ομοίως, η δημιουργία του αντικειμενικού αρχείου εξαρτάται από την ύπαρξη του πηγαίου αρχείου συμβολικής γλώσσας (`.asm`) και γίνεται με την κλήση του συμβολομεταφραστή `nasm` που βρίσκεται στην τέταρτη γραμμή.

Όταν λοιπόν καλείται το `make` πρώτα εξετάζει τη χρονική στιγμή που για τελευταία φορά ενημερώθηκαν όλα τα πηγαία και τα αντικειμενικά αρχεία που αναφέρονται στο `makefile`. Αν το (από προηγούμενη κλήση) εκτελέσιμο είναι νεότερο από όλα τα αντικειμενικά αρχεία από τα οποία εξαρτάται, τότε δεν εκτελείται καμία εντολή του `makefile`. Αν αντίθετα ένα ή περισσότερα από τα πηγαία αρχεία είναι νεότερα από το υπάρχον εκτελέσιμο ή από κάποιο αντικειμενικό αρχείο που εξαρτά τη δημιουργία του από αυτά, τότε πριν από τη κλήση του διασυνδέτη εκτελείται νέα μεταγλώττιση.

Έτσι το εργαλείο `make` εκμεταλλεύεται τις δηλωμένες εξαρτήσεις επαναλαμβάνοντας τη δημιουργία μόνο των αρχείων που ο προγραμματιστής έχει αλλάξει τον πηγαίο τους κώδικα και αποφεύγοντας τη δημιουργία όσων ο κώδικας δεν έχει αλλάξει.

## 1.3.7 Ο Ανιχνευτής Λαθών

Ο *ανιχνευτής λαθών* είναι το πρόγραμμα που χρησιμοποιείται στον εντοπισμό *λαθών εκτέλεσης*. Τα λάθη αυτά δεν ανήκουν σε καμία από τις περιπτώσεις λαθών που αναφέρθηκαν στην παράγραφο 1.2, καθώς δεν είναι δυνατός ο εντοπισμός τους κατά τη μεταγλώττιση. Είναι πιθανό να οφείλονται είτε στην εξάντληση πόρων του συστήματος, είτε σε *λογικό λάθος* λόγω εσφαλμένης υλοποίησης ενός αλγορίθμου.

Ένας ανιχνευτής λαθών επιτρέπει τον ορισμό σημείων διακοπής της εκτέλεσης σε συγκρεκριμένες γραμμές του προγράμματος ή μετά την n-οστή φορά που το πρόγραμμα επανέρχεται σε συγκεκριμένη γραμμή ή όταν αλλάζει η τιμή μιας καθολικής μεταβλητής ή όταν αποδίδεται σε αυτή μία συγκεκριμένη τιμή. Στη συνέχεια ο προγραμματιστής μπορεί να προχωρήσει την εκτέλεση του προγράμματος γραμμή – γραμμή και να προσπεράσει γρήγορα την εκτέλεση συναρτήσεων ή αυτή να γίνει επίσης γραμμή - γραμμή. Επιτρέπεται επίσης η προς τα πίσω εκτέλεση του προγράμματος ή η επιστροφή στο σημείο όπου προκλήθηκε λάθος. Συχνά υπάρχει δυνατότητα καταγραφής της εκτέλεσης συγκεκριμένων εντολών, όπως αν ενσωματώναμε στο πρόγραμμα εντολές εκτύπωσης μηνυμάτων π.χ. «ΕΙΜΑΙ ΕΔΩ!».

Ο ανιχνευτής λαθών επιτρέπει την επισκόπηση των δεδομένων, είτε αυτά είναι απλά, είτε δομημένα, αλλά ακόμη και δυναμικά μεταβαλλόμενα δεδομένα. Για την τελευταία περίπτωση παρέχονται λειτουργίες που καθιστούν εύκολη την επισκόπηση των περιεχομένων π.χ. μιας συνδετικής λίστας (κόμβο - κόμβο) ή άλλων δομών

οριζόμενων από τον προγραμματιστή. Επιτρέπεται ακόμη η αλλαγή τιμής στα δεδομένα και η συνέχιση της εκτέλεσης όπως περιγράψαμε προηγούμενα.

Συνήθως υπάρχει η δυνατότητα επισκόπησης του κώδικα είτε στην πηγαία του γλώσσα, είτε σε συμβολική γλώσσα μηχανής. Τέλος, ένας καλός ανιχνευτής λαθών διατηρεί το περιβάλλον εκσφαλμάτωσης (σημεία διακοπής εκτέλεσης, μεταβλητές που ελέγχονται κ.α.) και για την επόμενη χρήση.

Η δημιουργία των πληροφοριών εκσφαλμάτωσης γίνεται με τη χρήση της κατάλληλης επιλογής (option) κατά την κλήση του μεταγλωττιστή και όταν αυτό γίνεται, ο παραγόμενος κώδικας δεν είναι ο βέλτιστος δυνατός. Πάντως, η περαιτέρω ανάπτυξη θεμάτων ανίχνευσης λαθών εκτέλεσης είναι πέρα από τους στόχους αυτού του βιβλίου.

## 1.3.8 Ο Καταγραφέας (profiler)

Οι περισσότεροι μεταγλωττιστές διαθέτουν τη δυνατότητα εισαγωγής κλήσεων χρονομέτρησης στην είσοδο και έξοδο όλων των συναρτήσεων, ενεργοποιώντας την κατάλληλη επιλογή κατά την κλήση τους. Στη συνέχεια, καθώς το πρόγραμμα εκτελείται, καταγράφονται σε ένα αρχείο οι χρόνοι εισόδου και εξόδου όλων των συναρτήσεων που καλούνται. Τα αποτελέσματα αξιοποιούνται από τον *καταγραφέα*, ένα εργαλείο που τα συνοψίζει και παράγει μία τελική αναφορά με το ποσοστό του χρόνου εκτέλεσης που αντιστοιχεί σε κάθε κλήση συνάρτησης του κώδικα ή συνάρτησης βιβλιοθήκης. Δύο χαρακτηριστικές περιπτώσεις εργαλείων καταγραφής στο λειτουργικό σύστημα UNIX είναι τα `prof` και `gprof`.

## *Παράδειγμα* 1.3.2

Για τη χρονομέτρηση προγράμματος C στο UNIX με το μεταγλωττιστή `cc`, χρησιμοποιούμε την επιλογή `-p`. Έστω το πρόγραμμα `myProg`:

```
main() {
       int l;
       for (l=0;l<1000;l++) {
              if (l<500) aaa();
              bbb();
              ccc();
       }
}
aaa(){
       int j;
       for (j=0;j<300;j++);
}
bbb(){
       int i;
       for (i=0;i<300;i++);
}
ccc(){
```

```
        int k;
        for (k=0;k<500;k++);
}
```

Μετά από εκτέλεση του myProg καλούμε το εργαλείο prof με την εντολή

```
myProg > myProg.prof
```

και στο αρχείο myProg.prof παράγεται μία αναφορά όπως αυτή που ακολουθεί:

```
%Time  Seconds      Cumsecs      #Calls       msec/call    Name
 56.8     0.50         0.50        1000         0.500      _ccc
 27.3     0.24         0.74        1000         0.240      _bbb
 15.9     0.14         0.88         500         0.28       _aaa
  0.0     0.00         0.88           1         0.         _creat
  0.0     0.00         0.88           2         0.         _profil
  0.0     0.00         0.88           1         0.         _main
  0.0     0.00         0.88           3         0.         _getenv
  0.0     0.00         0.88           1         0.         _strcpy
  0.0     0.00         0.88           1         0.         _write
```

Τα στατιστικά που περιγράφει η κάθε στήλη είναι τα:

| | |
|---|---|
| %Time | ποσοστό χρόνου CPU για την εκτέλεση της συνάρτησης |
| Seconds | ο χρόνος CPU για την εκτέλεση της συνάρτησης |
| Cumsecs | το άθροισμα των χρόνων CPU όλων των προαναφερόμενων συναρτήσεων μαζί με αυτή της συγκεκριμένης γραμμής |
| Calls | ο συνολικός αριθμός κλήσεων της συνάρτησης |
| msec/call | ο μέσος χρόνος CPU για κάθε κλήση της συνάρτησης (το πηλίκο του χρόνου σε seconds προς τον αριθμό κλήσεων της συνάρτησης) |
| Name | το όνομα της συνάρτησης |

Τα ονόματα των συναρτήσεων που μπορούμε να αναγνωρίσουμε είναι αυτά που περιλαμβάνονται στο πρόγραμμα και φυσικά και η main. Τα ονόματα που δεν αναγνωρίζονται είναι συναρτήσεις βιβλιοθήκης της C. Εξαίρεση αποτελεί η _profil που αντιστοιχεί στον επιπλέον κώδικα που έχει ενσωματώσει ο διασυνδέτης για την καταγραφή δεδομένων εκτέλεσης. ■

Σε μία αναφορά καταγραφής, όπως αυτή του παραδείγματος 1.3.2, το μεγαλύτερο μέρος του χρόνου CPU αντιστοιχεί σε μία ή δύο συναρτήσεις. Έτσι, οποιαδήποτε αλλαγή προς την κατεύθυνση ενός πιο αποδοτικού κώδικα επικεντρώνεται στις συναρτήσεις αυτές, αφού ακόμη και μια μικρή βελτίωση μπορεί να αποφέρει μία κατά πολύ ταχύτερη εκτέλεση του προγράμματος. Παρόμοιες αναφορές καταγραφής αποδίδουν όλα τα προγράμματα επιστημονικών υπολογισμών που κάνουν χρήση πινάκων. Αντίθετα, σε κάποια άλλα προγράμματα εμφανίζεται μία σχετικά ομοιόμορφη κατανομή του χρόνου CPU σε όλες τις συναρτήσεις. Τέλος, πολλά προγράμματα εμφανίζουν κατανομή χρόνου που κινείται μεταξύ των δύο ακραίων περιπτώσεων που αναφέραμε.

### *Παράδειγμα* 1.3.3

Για τη χρονομέτρηση προγράμματος C στο UNIX με το μεταγλωττιστή cc, μπορούμε ακόμη να χρησιμοποιούμε την επιλογή -pg. Στη συνέχεια με χρήση του εργαλείου gprof εκτός από αναφορά χρονομέτρησης, το αποτέλεσμα είναι ταυτόχρονα και μία αναφορά του πως οι συναρτήσεις καλούν η μία την άλλη και πόσες φορές. ■

Πολλοί μεταγλωττιστές διαθέτουν επίσης δυνατότητες καταγραφής όχι μόνο σε επίπεδο συναρτήσεων αλλά και σε επίπεδο ενότητας εντολών (block). Στη συνέχεια ένα κατάλληλο εργαλείο μπορεί να συνοψίζει των αριθμό των εισόδων της ροής εκτέλεσης για όλες τις ενότητες εντολών του προγράμματος (π.χ. για όλους τους βρόχους for και τις εντολές if του κώδικα).

### *Παράδειγμα* 1.3.4

Για την καταγραφή σε επίπεδο ενότητας εντολών προγραμμάτων C στο UNIX ο μεταγλωττιστής cc, διαθέτει την επιλογή -a. Για τη συνέχεια, το εργαλείο tcov που παρέχει η Sun δημιουργεί την επιθυμητή αναφορά καταγραφής για όλες τις ενότητες εντολών του προγράμματος. ■

## 1.3.9 Τα προγράμματα διαχείρισης πηγαίου κώδικα

Το σύγχρονο λογισμικό έχει συχνά μέγεθος, που παραπέμπει σε ανάπτυξη στο πλαίσιο ομάδας. Αυτό καθιστά απαραίτητη την αξιοποίηση *προγραμμάτων διαχείρισης πηγαίου κώδικα*, που έχουν στόχο το συγχρονισμό των πηγαίων αρχείων και την επιλογή κάθε φορά των κατάλληλων αρχείων, που η χρήση τους οδηγεί σε μία νέα αναθεωρημένη έκδοση ή σε μία παλαιότερη έκδοση του υπό ανάπτυξη λογισμικού.

Ένα πρόγραμμα διαχείρισης πηγαίου κώδικα είναι ανεξάρτητο από το μεταγλωττιστή που χρησιμοποιείται, αλλά οι εντολές μπορούν επίσης να ενσωματώνονται στο αρχείο makefile που χρησιμοποιείται στο «κτίσιμο» του εκτελέσιμου. Δύο από τα πιο γνωστά προγράμματα διαχείρισης κώδικα στο λειτουργικό σύστημα UNIX είναι τα RCS (Revision Control System) και SCCS (Source Code Control System).

### *Παράδειγμα* 1.3.5

Το σύστημα RCS κυκλοφορεί και για το MS-DOS, διαθέτει ένα σύνολο εντολών διαχείρισης και βασίζει τη λειτουργία του στην καταγραφή των αλλαγών στα πηγαία αρχεία, με αρκετή λεπτομέρεια για την επανασύνθεση οποιασδήποτε προηγούμενης έκδοσης. Επίσης για κάθε αλλαγή επιτρέπει την αποθήκευση σχολίων που είναι χρήσιμα κατά την επισκόπηση της ιστορίας των αλλαγών του αρχείου. Το RCS

αποθηκεύει μόνο τις αλλαγές και όχι ολόκληρο το πηγαίο αρχείο, καθώς αυτό αλλάζει από μία παλαιότερη σε μία νεότερη έκδοση. Έτσι, είναι οικονομικό στη χρήση χώρου στο δίσκο και ταυτόχρονα καθιστά εφικτή την επανάκτηση προηγούμενης απωλεσθείσας έκδοσης λόγω π.χ. κατά λάθος διαγραφής αρχείου.

Έστω το στοιχειώδες πρόγραμμα `HelloWorld.c` με τον κώδικα:

```
#include <stdio.h>
main() {
        printf("Hello World\n");
}
```

Το σύστημα `RCS` αρχικοποιείται με την εντολή που ακολουθεί μαζί με τη συνακόλουθη είσοδο ενός σχολίου:

```
$rcs –i HelloWorld.c
RCS file: HelloWorld.c,v
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> This is the file created for managing the HelloWorld project
>> .
done
```

Στη συνέχεια, με την παρακάτω εντολή διαπιστώνουμε τη δημιουργία του αρχείου με κατάληξη `v`:

```
$ls -l
-rw-r--r--     1 user  users   226 Feb 11 16:25 HelloWorld.c
-r--r--r--     1 user users   105 Feb 11 16:26 HelloWorld.c,v
$
```

Η τρέχουσα έκδοση του πηγαίου κώδικα δηλώνεται με

```
$ci HelloWorld.c
HelloWorld.c,v <- HelloWorld.c
initial revision: 1.1
done
$
```

και προκαλεί διαγραφή του `HelloWorld.c`, οπότε τα περιεχόμενα του πηγαίου μαζί με τις πληροφορίες ελέγχου θα διατηρούνται στο `HelloWorld.c,v`.

Για την εμφάνιση του κώδικα της τρέχουσας έκδοσης ή την επεξεργασία του χρησιμοποιείται η εντολή `co` (check out). Στην πρώτη περίπτωση δημιουργείται πάλι το αρχείο `HelloWorld.c` με δυνατότητα μόνο ανάγνωσης. Αν πρόκειται για επεξεργασία, τότε το πηγαίο αρχείο που δημιουργεί η εντολή `co –l` «κλειδώνει», ώστε να μην είναι δυνατή η ταυτόχρονη επεξεργασία από άλλο μέλος της προγραμματιστικής ομάδας:

```
$co -l HelloWorld.c
HelloWorld.c,v -> HelloWorld.c
revision 1.1 (locked)
done
$
```

Το αποτέλεσμα είναι:

```
$ls -l
-rw-r--r--     1 user  users   226 Feb 11 16:35 HelloWorld.c
```

```
-r--r--r--     1 user users   452 Feb 11 16:35 HelloWorld.c,v
$
```

Έστω ότι προστίθεται στο πηγαίο πρόγραμμα μία νέα γραμμή με τη `main()` πλέον να περιλαμβάνει τις

```
printf("Hello World\n");
printf("This is the new line added in the source file\n");
```

Η νέα έκδοση πρέπει να δηλωθεί όπως και η προηγούμενη

```
$ci HelloWorld.c
HelloWorld.c,v <- HelloWorld.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Added a single new line of code
>> .
done
```

Μία από τις λειτουργίες του `RCS` είναι η επισκόπηση της ιστορίας των αλλαγών στο πηγαίο αρχείο:

```
$rlog HelloWorld.c

RCS file: HelloWorld.c,v
Working file: HelloWorld.c
head: 1.2
branch:
locks: strict
access list:
symbolic names:
comment leader: "*"
keyword substitution: kv
total revisions: 2;          selected revisions: 2
description:
This is the file created for managing the HelloWorld project
-------------
revision 1.2
date: 2004/02/08 16:37:35; author:user; state:Exp; lines: +1 -0
Added a single new line of code
-------------
revision 1.1
date: 2004/02/08 16:30:19; author:user; state:Exp;
Initial revision
=============================================================
$
```

Αν είναι επιθυμητή η επανασύνθεση της πρώτης έκδοσης, αυτό επιτυγχάνεται ως εξής

```
$co -r1.1 HelloWorld.c
HelloWorld.c,v -> HelloWorld.c
revision 1.1
done
$
```

Οι διαφορές μεταξύ δύο εκδόσεων του ίδιου πηγαίου αρχείου δίνονται με την εντολή,

```
$rcsdiff -r1.1 -r1.2 HelloWorld.c
=============================================================
RCS file: HelloWorld.c,v
retrieving revision 1.1
```

```
retrieving revision 1.1
diff –r1.1 –r1.2
l1al2
>   printf(“This is the new line added in the source file\n”);
$
```

που στο παράδειγμά μας αποτυπώνει την εισαγωγή μιας νέας γραμμής (l2) αμέσως μετά από τη γραμμή l1 της πρώτης αναφερόμενης έκδοσης.

Ένα ιδιαίτερα ενδιαφέρον χαρακτηριστικό είναι η λειτουργία των μακροεντολών $RCSfile$ και $Id$, που είναι δυνατό να αξιοποιηθούν μέσα στο πηγαίο πρόγραμμα. Η $RCSfile$ αντικαθιστάται από το όνομα του πηγαίου αρχείου, ενώ η $Id$ από μία συμβολοσειρά που αναφέρεται στην τρέχουσα έκδοσή του. Η αντικατάσταση αυτή γίνεται κάθε φορά που έχουμε εκτέλεση της co και η ενημέρωση γίνεται αυτόματα, όταν δηλώνεται μία νέα έκδοση με την εντολή ci. Έστω

```
$co -l HelloWorld.c
HelloWorld.c,v -> HelloWorld.c
revision 1.2 (locked)
done
$
```

και έστω ότι μετά από επεξεργασία το πρόγραμμα γίνεται

```
/*
    This is a test HelloWorld program
    Filename: $RCSfile$
*/
#include <stdio.h>
static char *RCSinfo=“$Id$”;
main() {
  printf(“Hello World\n”);
  printf(“This is the new line added in the source file\n”);
  printf(“This file is under RCS control. It’s ID is        \n%s\n”,
       RCSinfo);
}
```

και η νέα έκδοση δηλώνεται με την εντολή

```
$ci HelloWorld.c
HelloWorld.c,v <- HelloWorld.c
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single ‘.’ or end of file:
>> Added $RCSfile$ and $Id$ strings
>> .
done
$
```

Μετά από εκτέλεση της co το πηγαίο πρόγραμμα παίρνει τη μορφή

```
/*
    This is a test HelloWorld program
    Filename: $RCSfile$: HelloWorld.c,v $
*/
#include <stdio.h>
static char *RCSinfo=“$Id: HelloWorld.c,v 1.3 2004/02/08 16:55:04
                      user Exp $”;
main() {
  printf(“Hello World\n”);
```

```
    printf("This is the new line added in the source file\n");
    printf("This file is under RCS control. It's ID is        \n%s\n",
         RCSinfo);
}
```

Το εργαλείο make της GNU, που διατίθεται μέσα στο διαδίκτυο χωρίς χρέωση, έχει ενσωματωμένες δυνατότητες διαχείρισης αρχείων RCS. Έτσι, αν το πηγαίο αρχείο δεν υπάρχει και εφόσον το πηγαίο πρόγραμμα βρίσκεται κάτω από τον έλεγχο του RCS, το make από μόνο του συνθέτει την τρέχουσα έκδοση εκτελώντας τις εντολές:

```
$make HelloWorld
co HelloWorld.c,v
HelloWorld.c,v -> HelloWorld.c
revision 1.3
done
cc -c HelloWorld.c -o HelloWorld.o
cc HelloWorld.o -o HelloWorld
rm HelloWorld.o HelloWorld.c
$
```

Τέλος, σημαντική είναι και η δυνατότητα χρήσης της εντολής ident, που εμφανίζει την έκδοση του προγράμματος από την οποία προέρχεται ένα εκτελέσιμο, χωρίς να είναι απαραίτητη η εκτύπωση της συμβολοσειράς με printf, όπως στον κώδικα του παραδείγματος. Προϋπόθεση είναι φυσικά να περιλαμβάνεται στο πηγαίο πρόγραμμα η μακροεντολή $Id$.

```
$ident HelloWorld
HelloWorld:
  $Id: HelloWorld.c,v 1.3  2004/02/08 16:55:04        user Exp $
$
```

Το χαρακτηριστικό αυτό είναι ιδιαίτερα χρήσιμο στην αναγνώριση της έκδοσης του προγράμματος, που ο χρήστης του αναφέρει την ύπαρξη λάθους. Στη συνέχεια μπορούν να χρησιμοποιηθούν εντολές όπως οι rlog και rcsdiff, για την επισκόπηση των αλλαγών που έχουν γίνει στον κώδικα μεταξύ της έκδοσης με το λάθος και της τρέχουσας. ■

## 1.3.10 Προαιρετικές επιλογές και μηχανισμοί διεπαφής

Σημαντικό για ένα μεταγλωττιστή είναι η διεπαφή του με το λειτουργικό σύστημα και η παροχή επιλογών στο χρήστη, για την αξιοποίηση των προαιρετικών χαρακτηριστικών του. Παραδείγματα μηχανισμών διεπαφής είναι οι όποιες ευκολίες εισόδου-εξόδου και προσπέλασης στο σύστημα αρχείων της πλατφόρμας εκτέλεσης. Προαιρετικά χαρακτηριστικά ενός μεταγλωττιστή είναι οι επιλογές αναφοράς (μηνύματα λάθους, πίνακες διασταύρωσης κ.λ.π.) και οι επιλογές βελτιστοποίησης (στοιχειώδεις βελτιστοποιήσεις, βελτιστοποιήσεις με βάση καταγραφείσες πληροφορίες εκτέλεσης, βελτιστοποιήσεις κινητής υποδιαστολής κ.α.). Ιδιαίτερα σημαντική είναι η ρύθμιση της ευαισθησίας του μεταγλωττιστή στην παραγωγή μηνυμάτων ειδοποίησης (warnings). Αν και τα μηνύματα ειδοποίησης δεν είναι λάθη, μπορεί να είναι προάγγελοι λαθών και συνιστάται η επιλογή μη περικοπής κάποιων

απ΄ αυτά. Οι καλοί μεταγλωττιστές διαθέτουν επιπλέον και τη δυνατότητα αντιμετώπισης των μηνυμάτων ειδοποίησης ως λάθη.

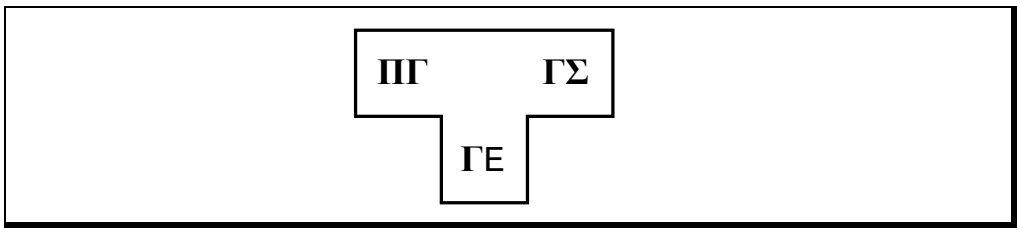
Οι μηχανισμοί διεπαφής και οι προαιρετικές επιλογές, που διαθέτει ένας μεταγλωττιστής, είναι συχνά εφικτό να ενεργοποιηθούν μέσα από το κείμενο του πηγαίου προγράμματος με τη χρήση κατάλληλων οδηγιών, των επονομαζόμενων pragmas.

## 1.4 Ανάπτυξη μεταγλωττιστών

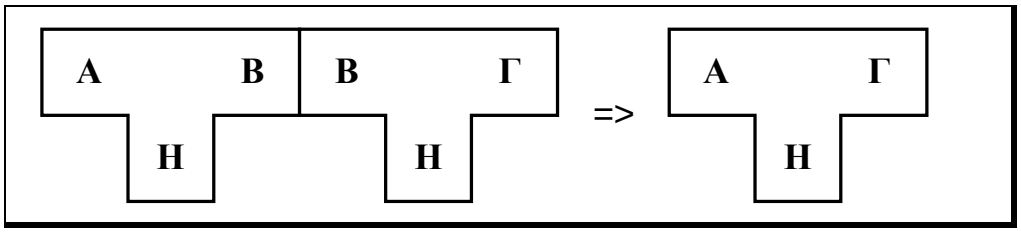
Οι βασικές απαιτήσεις για κάθε μεταγλωττιστή είναι να συμμορφώνεται απόλυτα με τις προδιαγραφές ορισμού της πηγαίας γλώσσας και της γλώσσας-στόχο και να διαθέτει δυνατότητα μετάφρασης προγραμμάτων μεγάλου μεγέθους. Τα στοιχειώδη λειτουργικά χαρακτηριστικά, που πρέπει να διαθέτει ένας μεταγλωττιστής, είναι η δυνατότητα παραγωγής αποδοτικού κώδικα σε αποδεκτό *χρόνο μεταγλώττισης* και η σαφήνεια των μηνυμάτων λάθους που παράγει. Όπως σε κάθε λογισμικό έτσι και στην περίπτωση των μεταγλωττιστών είναι πάντα επιθυμητή η διασφάλιση όσο το δυνατό μεγαλύτερων δυνατοτήτων μεταφερσιμότητας. Δυνατοτήτων, που έχουν να κάνουν τόσο με τον ίδιο το μεταγλωττιστή, όσο και με το τελικό πρόγραμμα που αυτός παράγει.

Στις προηγούμενες παραγράφους δεν έγινε καμία αναφορά στη γλώσσα του ίδιου του μεταγλωττιστή, που δεν είναι άλλη από τη *γλώσσα μηχανής* της πλατφόρμας στην οποία εκτελείται. Η συγγραφή ενός μεταγλωττιστή δε γίνεται απευθείας σε γλώσσα μηχανής, αλλά σε κάποια γλώσσα για την οποία ήδη υπάρχει μεταγλωττιστής στη συγκεκριμένη πλατφόρμα εκτέλεσης. Είναι ακόμη πιθανό ένας μεταγλωττιστής να μεταφράζει σε μία γλώσσα-στόχο διαφορετική από αυτήν στην οποία εκτελείται.

Κάθε μεταγλωττιστής σε ένα υψηλό επίπεδο περιγράφεται με τη χρήση *διαγραμμάτων σχήματος Τ*. Κάθε τέτοιο διάγραμμα απεικονίζει τη γλώσσα εκτέλεσης ΓΕ του μεταγλωττιστή, την πηγαία γλώσσα ΠΓ και τη γλώσσα -στόχο ΓΣ, όπως στο σχήμα 1.8. Αν η γλώσσα εκτέλεσης του μεταγλωττιστή είναι διαφορετική από τη γλώσσα-στόχο ΓΣ, τότε κάνουμε λόγο για ένα *διαμεταγλωττιστή*. Η χρήση των διαγραμμάτων σχήματος Τ θεμελιώθηκε συστηματικά στην εργασία [ES70], όπου και προτείνεται ένας αλγόριθμος για την παραγωγή νέων διαγραμμάτων από ένα δοθέν αρχικό σύνολο. Η σύνθεση νέων διαγραμμάτων γίνεται με δύο διαφορετικούς τρόπους.



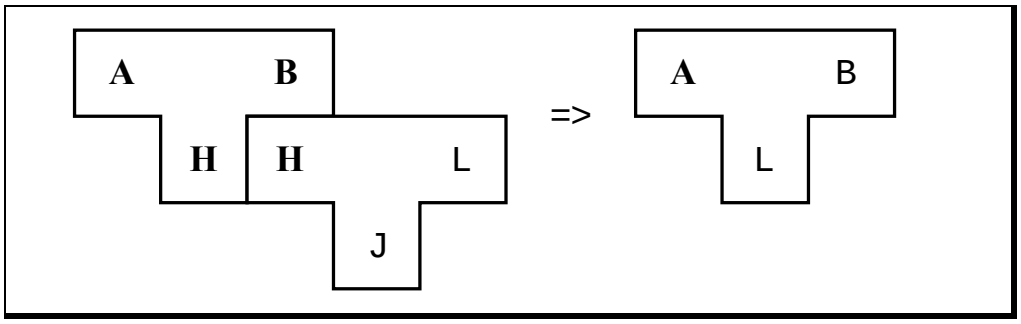

**Σχήμα 1.8** Ένας διαμεταγλωττιστής σε διάγραμμα σχήματος Τ




**Σχήμα 1.9** Σύνθεση διαγραμμάτων σχήματος Τ με την ίδια γλώσσα εκτέλεσης

Στην περίπτωση σύνθεσης δύο μεταγλωττιστών στη γλώσσα εκτέλεσης Η, αν ο πρώτος μεταφράζει από την Α στη Β και ο δεύτερος από τη Β στη Γ, τότε το αποτέλεσμα είναι μία διαδικασία μετάφρασης από την Α στη Γ, με γλώσσα εκτέλεσης τη γλώσσα Η (σχήμα 1.9).

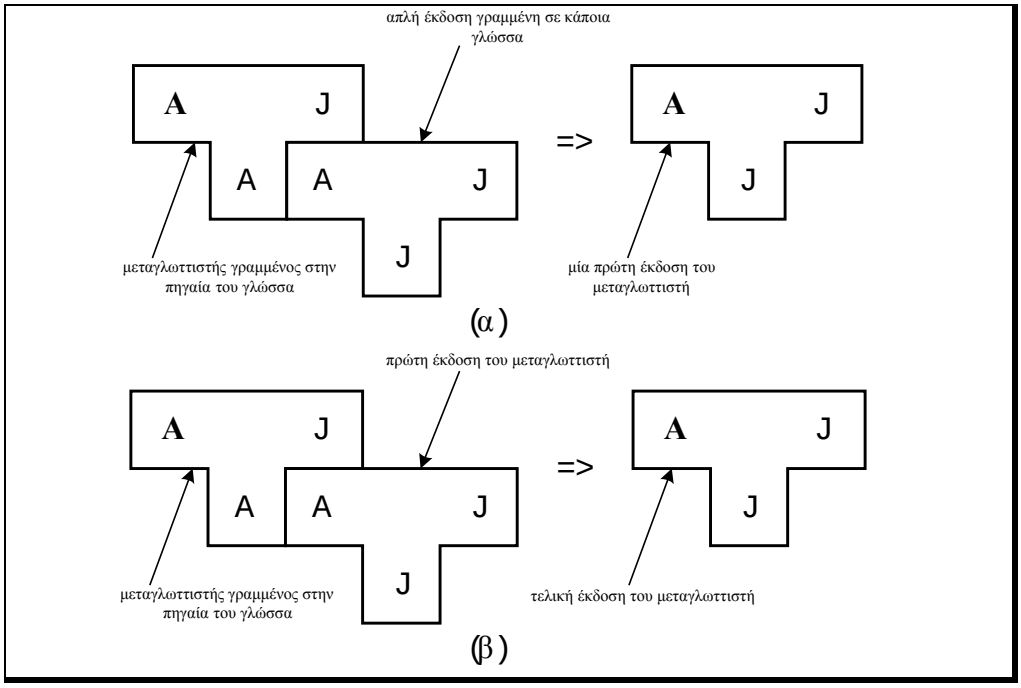
Στη δεύτερη περίπτωση σύνθεσης διαγραμμάτων σχήματος Τ έχουμε μετάφραση της γλώσσας εκτέλεσης του μεταγλωττιστή σε άλλη γλώσσα. Μία σύνθεση αυτού του τύπου απεικονίζεται στο σχήμα 1.10.



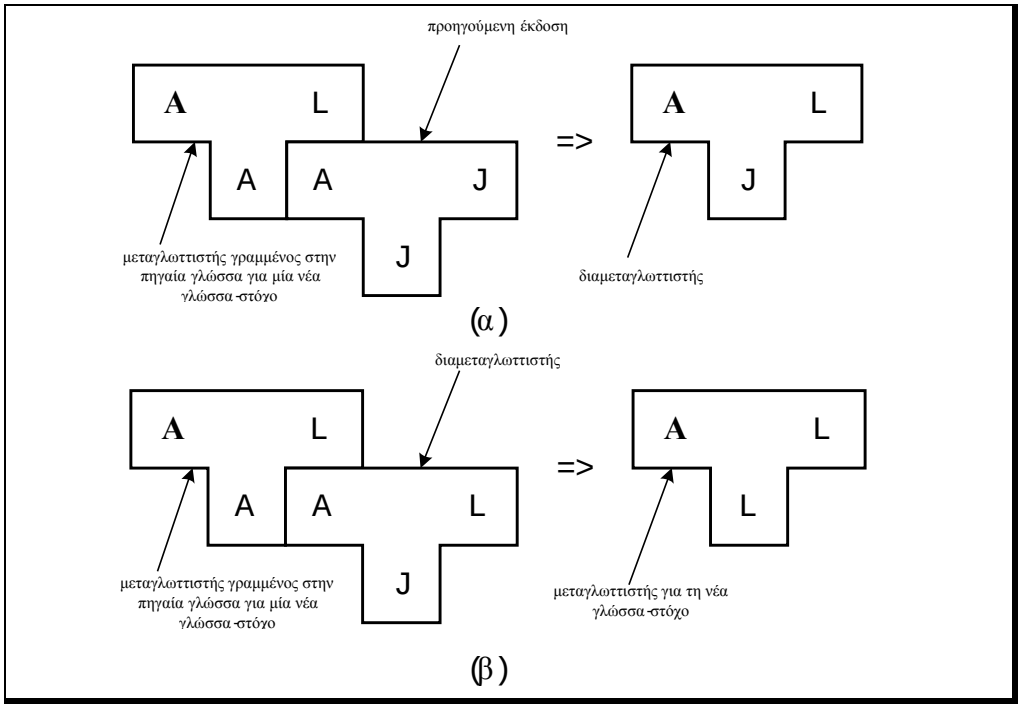

**Σχήμα 1.10** Μετάφραση της γλώσσας ενός μεταγλωττιστή

Μία ενδιαφέρουσα περίπτωση υλοποίησης μεταγλωττιστή είναι η επονομαζόμενη *αυτοδύναμη ανάπτυξη* (bootstrapping) κατά την οποία ο μεταγλωττιστής γράφεται στην ίδια γλώσσα, που χρησιμοποιείται ως πηγαία. Το πώς αυτό είναι εφικτό φαίνεται στο σχήμα 1.11. Αρχικά υλοποιείται σε συμβολική γλώσσα, μία απλοποιημένη και πιθανώς μη αποδοτική έκδοση του μεταγλωττιστή, η οποία περιλαμβάνει αυτά ακριβώς τα χαρακτηριστικά, που χρησιμοποιούνται για τη συγγραφή του στην πηγαία του γλώσσα. Μετά την πρώτη μετάφραση (σχήμα 1.11α) προκύπτει μία πλήρης, αλλά μη αποδοτική έκδοση του μεταγλωττιστή, η οποία χρησιμοποιείται σε ένα δεύτερο στάδιο, για την παραγωγή της τελικής του έκδοσης (σχήμα 1.11β).

Με την εφαρμογή μιας διαδικασίας αυτοδύναμης ανάπτυξης κάθε βήμα της αποδίδει ως αποτέλεσμα ένα μεταγλωττιστή γραμμένο στη γλώσσα που αυτός μεταφράζει (πηγαία γλώσσα). Έτσι, κάθε βελτίωση στο πηγαίο πρόγραμμά του μεταφέρεται, με μία απλή εφαρμογή της διαδικασίας του σχήματος 1.11β, σε μία νέα λειτουργική έκδοση αυτού.




**Σχήμα 1.11** Αυτοδύναμη ανάπτυξη (bootstrapping)




**Σχήμα 1.12** Μεταφορά μεταγλωττιστή αυτοδύναμης ανάπτυξης σε νέα γλώσσα

Ένα άλλο πλεονέκτημα της αυτοδύναμης ανάπτυξης είναι το γεγονός ότι για τη μεταφορά του μεταγλωττιστή σε μία νέα πλατφόρμα εκτέλεσης, αρκεί η αντικατάσταση μόνο του τμήματος τελικής επεξεργασίας αυτού. Ακολουθεί μετάφραση με τη χρήση της προηγούμενης έκδοσης του μεταγλωττιστή, που έχει ως αποτέλεσμα τη δημιουργία ενός διαμεταγλωττιστή (σχήμα 1.12α). Τέλος, ο μεταγλωττιστής μεταφράζεται για μία ακόμη φορά, με τη χρήση όμως του διαμεταγλωττιστή, για να ληφθεί το επιδιωκόμενο αποτέλεσμα (σχήμα 1.12β).

## 1.5 Μεταγλωττιστές μεταγλωττιστών

Για την κατασκευή μεταγλωττιστών χρησιμοποιούνται σήμερα εξειδικευμένα εργαλεία αυτόματης παραγωγής κώδικα σε κάποιες από τις φάσεις επεξεργασίας, που περιγράφηκαν στην παράγραφο 1.2. Τα εργαλεία αυτά ονομάζονται *μεταγλωττιστές μεταγλωττιστών* ή *γεννήτριες μεταγλωττιστών* και το χαρακτηριστικό τους είναι ότι το κάθε ένα είναι προσανατολισμένο σε συγκεκριμένο μοντέλο γλώσσας.

Έτσι, μία γεννήτρια λεξικού αναλυτή βασίζεται στην παραδοχή ότι για κάθε γλώσσα η λεξική ανάλυση γίνεται λίγο-πολύ με τον ίδιο τρόπο, αλλά με διαφορές στις λέξεις-κλειδιά και τα σύμβολα που αναγνωρίζονται. Η γεννήτρια παράγει τυποποιημένες ρουτίνες λεξικής ανάλυσης μετά τον ορισμό από τον κατασκευαστή της λίστας των λεξικών μονάδων που είναι δυνατό να αναγνωριστούν.

Τα πιο επιτυχημένα από τα εργαλεία αυτά παράγουν κώδικα που εύκολα ενσωματώνεται στο υπόλοιπο πρόγραμμα, ενώ οι λεπτομέρειες των αλγορίθμων παραγωγής κώδικα που χρησιμοποιούνται δε χρειάζεται να απασχολούν τον κατασκευαστή του μεταγλωττιστή.

Για τη δημιουργία του μεταγλωττιστή της εκπαιδευτικής γλώσσας YAPL, που περιγράφεται επιγραμματικά στην παράγραφο 1.6, χρησιμοποιήθηκε η γεννήτρια `flex` στην παραγωγή του κώδικα της λεξικής ανάλυσης και η γεννήτρια `byacc` στην παραγωγή του κώδικα της συντακτικής ανάλυσης. Μία λεπτομερέστερη αναφορά στα εργαλεία αυτά γίνεται στα αντίστοιχα κεφάλαια του βιβλίου.

## 1.6 Η εκπαιδευτική γλώσσα YAPL

Στο βιβλίο αυτό περιγράφουμε την ανάπτυξη ενός εκπαιδευτικού μεταγλωττιστή, ενός μεταγλωττιστή δηλαδή, που χωρίς να διαθέτει όλα τα χαρακτηριστικά μιας γλώσσας προγραμματισμού, είναι επαρκής για την κατανόηση των λειτουργιών και των αλληλεπιδράσεων των τμημάτων επεξεργασίας από τα οποία αποτελείται. Η πηγαία γλώσσα του συγκεκριμένου μεταγλωττιστή ονομάστηκε YAPL (Yet Another Simple Programming Language) και η παρουσίασή του γίνεται τμηματικά, έτσι ώστε για κάθε φάση μεταγλώττισης (από αυτές της παραγράφου 1.2), να έχει προηγηθεί η περιγραφή των τεχνικών επεξεργασίας, που είναι δυνατό να εφαρμοσθούν. Η

γλώσσα-στόχος, που επιλέχθηκε είναι η συμβολική γλώσσα μηχανής των επεξεργαστών αρχιτεκτονικής Pentium.

Στην παράγραφο αυτή γίνεται μία συνοπτική παρουσίαση της YAPL, ενώ στο δεύτερο κεφάλαιο δίνεται ο πλήρης ορισμός της γλώσσας και στο διαδίκτυο διατίθεται ολόκληρο το πηγαίο πρόγραμμα του μεταγλωττιστή.

Γενικά ένα πρόγραμμα YAPL ανοίγει και κλείνει με '{' και '}' αντίστοιχα και αποτελείται από δηλώσεις και από εντολές, κάθε μία από τις οποίες διαχωρίζεται από τις υπόλοιπες με το χαρακτήρα ';'. Δεν υπάρχει η δυνατότητα χρήσης διαδικασιών ή συναρτήσεων, όπως στις περισσότερες γλώσσες προγραμματισμού, ενώ όλες οι μεταβλητές είναι ακέραιες, καθώς δεν υποστηρίζεται κάποιος άλλος τύπος δεδομένων. Παρόλα αυτά, έχει προβλεφθεί η δυνατότητα χρήσης πινάκων ακεραίων. Σε ότι αφορά τις εντολές ελέγχου, αυτές περιλαμβάνουν τις απλές και σύνθετες `if` και την εντολή `while`. Για είσοδο-έξοδο παρέχονται οι εντολές `read` και `print`.

Οι εκφράσεις διακρίνονται σε λογικές και σε αριθμητικές εκφράσεις. Μία λογική έκφραση μπορεί να περιλαμβάνει τη σύγκριση δύο αριθμητικών εκφράσεων, με τη χρήση οποιουδήποτε από τους παρακάτω τελεστές:

'<', '>', '>=', '<=', '==', '<>'

ενώ είναι δυνατή και η χρήση των λογικών τελεστών,

AND, OR και NOT

Οι αριθμητικές εκφράσεις αποτελούνται από σταθερές και μεταβλητές ακεραίων, με ή χωρίς παρενθέσεις και δυνατότητα χρήσης οποιωνδήποτε από τους τελεστές '+', '-', '*', '/' και '%', για το υπόλοιπο της διαίρεσης δύο αριθμών.

Στο σχήμα 1.13 δίνεται ένα παράδειγμα προγράμματος γραμμένου στη YAPL.

```
{  int i, a[6];
   i=0;
   while(i<6)
   {
       a[i]=2*i+1;
       i=i+1;
   }
   i=0;
   while(i<6)
   {
       print(a[i]);
       i=i+1;
   }
}
```

**Σχήμα 1.13** Πρόγραμμα YAPL

Η μεταγλώττιση κάθε προγράμματος της YAPL επιτυγχάνεται με διπλή σάρωση. Στην πρώτη σάρωση γίνεται η λεξική, η συντακτική και η σημασιολογική ανάλυση για τη δημιουργία του κατάλληλου συντακτικού δένδρου. Στη δεύτερη γίνεται η σύνθεση του τελικού προγράμματος σε συμβολική γλώσσα μηχανής.

Ο μεταγλωττιστής έχει υλοποιηθεί σε γλώσσα ANSIC, έτσι ώστε να εξασφαλίζεται η δυνατότητα συνεργασίας με τα εργαλεία `flex` και `byacc`, που αντίστοιχα χρησιμοποιούνται στην κατασκευή του λεξικού και του συντακτικού αναλυτή.

Η μεταγλώττιση ενός προγράμματος όπως αυτό του σχήματος 1.13, γίνεται σε δύο στάδια: στο πρώτο στάδιο ο μεταγλωττιστής της YAPL παράγει το ισοδύναμο τελικό πρόγραμμα στη συμβολική γλώσσα του επεξεργαστή Pentium και στη συνέχεια ο συμβολομεταφραστής `ml` του επεξεργαστή Pentium δέχεται ως είσοδο τον παραγόμενο συμβολικό κώδικα (αρχείο `.asm`) και ένα αντικειμενικό αρχείο βιβλιοθήκης εισόδου -εξόδου και παράγει το εκτελέσιμο πρόγραμμα.

Είναι σημαντικό τέλος να τονιστεί ότι επειδή ακριβώς ο μεταγλωττιστής της γλώσσας YAPL έχει κατασκευαστεί πρωτίστως για εκπαιδευτική χρήση, διαθέτει λειτουργίες που καθιστούν εύκολη την κατανόηση του τρόπου επεξεργασίας του πηγαίου προγράμματος. Μία από αυτές είναι η δυνατότητα εκτύπωσης του παραγόμενου συντακτικού δένδρου. Έτσι, στην περίπτωση του προγράμματος του σχήματος 1.13, το συντακτικό δένδρο που δημιουργείται είναι αυτό που απεικονίζεται στο σχήμα 1.14.

Τα ονόματα astProgram, astDeclSeq, κ.α., ονομάζουν κόμβους του δένδρου, που αντιστοιχούν σε σύμβολα της γραμματικής της YAPL, που περιγράφονται στην παράγραφο 7 του κεφαλαίου 3.

```
AstProgram
  astDeclSeq
    astEmptyDeclSeq
    astDecl
      INT
      astDclrSeq
        astVarDclr
        astArrayDclr
          astVarDclr
          astDecimConst=6
  astStmtSeq
    astStmtSeq
      astStmtSeq
        astStmtSeq
          astEmptyStmtSeq
          astExprStmt
            astAssign
              astId=i
              astDecimConst=0
        astWhileStmt
          astLess
            astId=i
            astDecimConst=6
          astStmtSeq
            astStmtSeq
              astEmptyStmtSeq
              astExprStmt
                astAssign
                  astArrayElem
                    astId=a
                    astId=i
                  astAdd
                    astMult
                      astDecimConst=2
                      astId=i
                    astDecimConst=1
            astExprStmt
              astAssign
                astId=i
                astAdd
                  astId=i
                  astDecimConst=1
      astExprStmt
        astAssign
          astId=i
          astDecimConst=0
    astWhileStmt
      astLess
        astId=i
        astDecimConst=6
      astStmtSeq
        astStmtSeq
          astEmptyStmtSeq
          astPrintStmt
            astArrayElem
              astId=a
              astId=i
        astExprStmt
          astAssign
            astId=i
            astAdd
              astId=i
              astDecimConst=1
```

**Σχήμα 1.14** Το συντακτικό δένδρο του προγράμματος YAPL του σχ.1.13

## Ασκήσεις

1.1 Αναπτύξτε το παράγωγο και το συντακτικό δένδρο της εντολής εκχώρησης

```
a[i]=(5*i)+a[i]
```

1.2 Δώστε από δύο παραδείγματα λεξικών, συντακτικών και σημασιολογικών λαθών για μία γλώσσα της επιλογής σας.

1.3 Για ένα μεταγλωττιστή της αρεσκείας σας καταγράψτε τις επιλογές αναφοράς και τις επιλογές βελτιστοποίησης που αυτός διαθέτει.

1.4 Υποθέστε ότι διαθέτετε

- ένα μεταγλωττιστή Pascal που παράγει τελικό πρόγραμμα σε C και ο ίδιος είναι γραμμένος σε C και
- ένα μεταγλωττιστή της γλώσσας C.

Περιγράψτε, με τη χρήση διαγραμμάτων σχήματος Τ, τη δημιουργία ενός λειτουργικού μεταγλωττιστή Pascal.

## Σχόλια και αναφορές

Η κλασσική και πληρέστερη αναφορά για ό,τι έχει σχέση με μεταγλωττιστές, αλλά ιδιαίτερα για θεωρητικά θέματα και αλγορίθμους, είναι η [ASU86]. Από την ελληνική βιβλιογραφία ξεχωρίσαμε τα [ΛΑ99] και [ΠΣ02].

Περιπτώσεις αυτοδύναμης ανάπτυξης μεταγλωττιστών Pascal περιγράφονται στις εργασίες [WIR71] και [WQ72]. Ένα άλλο χαρακτηριστικό παράδειγμα αυτοδύναμης ανάπτυξης είναι αυτό της συναρτησιακής γλώσσας ML [MIL93]. Επιπλέον, στην [AMM81] επιστρατεύεται μια προσέγγιση αυτοδύναμης ανάπτυξης, για τη διαρκή βελτίωση του μεταγλωττιστή της εργασίας.