ΤΙΤΛΟΣ ΚΕΦΑΛΑΙΟΥ

6. Λογισμικό Συστήματος

Στην ενότητα αυτή διαπραγματευόμαστε τα εξής θέματα :

- Μειονεκτήματα προγραμματισμού και χρήσης του υπολογιστή του προηγούμενου κεφαλαίου και αναγκαιότητα ύπαρξης λογισμικού συστήματος και σύντομη περιγραφή των τμημάτων του:
  - Λειτουργικό Σύστημα
  - Μεταφραστές
  - Εκδότες-Συντάκτες
  - Φορτωτές
  - Λογισμικό Επικοινωνίας
- Επέκταση και εμβάθυνση στα πρώτα δύο τμήματα, το Λειτουργικό Σύστημα και τους Μεταφραστές

ΤΙΤΛΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

6.1 Εισαγωγή

ΠΡΟΑΠΑΙΤΟΥΜΕΝΗ ΓΝΩΣΗ

Σχεδίαση Αλγορίθμων (κεφάλαιο 2 του παρόντος μαθήματος)
Αρχιτεκτονική Υπολογιστών (κεφάλαιο 4 του παρόντος μαθήματος)

ΣΥΝΟΨΗ

Στην ενότητα αυτή θα εξετάσουμε τα μειονεκτήματα του υπολογιστή που αναπτύξαμε στο κεφάλαιο 4, όσο αφορά την χρήση του και τον προγραμματισμό του. Θα δούμε πως μπορούμε να αποφύγουμε τα μειονεκτήματα με το λογισμικό συστήματος. Το λογισμικό συστήματος αποτελείται κύρια από τα εξής τμήματα: το Λειτουργικό Σύστημα, τους Μεταφραστές, τους Εκδότες-Συντάκτες, τους Φορτωτές, και το Λογισμικό Επικοινωνίας. Θα περιγράψουμε σύντομα τα τμήματα του λογισμικού συστήματος και στις επόμενες ενότητες του κεφαλαίου θα επεκταθούμε στα δύο πρώτα τμήματα, το Λειτουργικό Σύστημα και τους Μεταφραστές.

ΓΝΩΣΤΙΚΟΙ ΣΤΟΧΟΙ

Στην ενότητα αυτή διαπραγματευόμαστε τα εξής θέματα :

- Μειονεκτήματα προγραμματισμού και χρήσης του υπολογιστή του προηγούμενου κεφαλαίου και αναγκαιότητα ύπαρξης λογισμικού συστήματος
- Σύντομη περιγραφή και χρησιμότητα των τμημάτων του λογισμικού συστήματος:
  - Λειτουργικό Σύστημα
  - Μεταφραστές Γλωσσών Προγραμματισμού
  - Εκδότες-Συντάκτες
  - Φορτωτές
  - Λογισμικό Επικοινωνίας

Λέξεις κλειδιά

Λογισμικό συστήματος, Λειτουργικό Σύστημα, Μεταφραστής, Εκδότες-Συντάκτες, Επεξεργαστής Κειμένου, Φορτωτές, Λογισμικό Επικοινωνίας.

ΚΥΡΙΟ ΜΕΡΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

Εισαγωγή

Στο προηγούμενο κεφάλαιο περιγράψαμε την δομή ενός τυπικού υπολογιστή και την εκτέλεση προγραμμάτων σε γλώσσα μηχανής. Αυτό το βασικό υπολογιστικό σύστημα έχει πολλά μειονεκτήματα**:**

1. **Δυσκολία Εισαγωγής και Ενεργοποίησης Εκτέλεσης Προγράμματος**. Δεν παρέχει εύκολο τρόπο για την εισαγωγή του προγράμματος στη μνήμη του υπολογιστή και την ενεργοποίηση της εκτέλεσης του.
2. **Απαιτεί την Διαχείριση μονάδων εισόδου/εξόδου**. Οι μονάδες εισόδου/εξόδου (I/O) είναι δύσκολο να χρησιμοποιηθούν καθώς ο προγραμματιστής θα πρέπει να γνωρίζει τις ιδιαιτερότητες του κάθε περιφερειακού. Τα προγράμματα πρέπει να ελέγχουν τους καταχωρητές κατάστασης περιφερειακών για να διαπιστώνει πότε η είσοδος ή η έξοδος έχει ολοκληρωθεί. Εναλλακτικά ο προγραμματιστής πρέπει να εφοδιάσει το σύστημα με στοιχεία διαχείρισης διακοπών. Η λεπτομερής οργάνωση των μονάδων εισόδου/εξόδου επιφορτίζει την προγραμματιστική προσπάθεια και παρεκτρέπει τον προγραμματιστή από την υλοποίηση του αλγόριθμου.
3. **Περιορισμός Μνήμης.** Δεν υπάρχει εύκολος τρόπος για το πρόγραμμα και τα δεδομένα του να υπερβούν το μέγεθος της κύριας μνήμης του υπολογιστή.
4. **Δέσμευση και Υποαπασχόληση Υπολογιστή**. Όλες οι μονάδες του υπολογιστή δεσμεύονται για την εκτέλεση ενός μόνο προγράμματος. Αυτό το πρόγραμμα ίσως καταλαμβάνει μικρό μέρος της μνήμης και να χρησιμοποιεί ένα ή δύο περιφερειακά αφήνοντας το υπόλοιπο της μνήμης και τα άλλα περιφερειακά αχρησιμοποίητα καθ' όλη την διάρκεια της εκτέλεσης του.
5. **Προγραμματισμός σε γλώσσα μηχανής**. Ο υπολογιστής πρέπει να προγραμματιστεί σε γλώσσα μηχανής, η οποία δεν είναι πολύ κατάλληλη για την υλοποίηση αλγορίθμων.

Το Λογισμικό Συστήματος

Το συμπέρασμα που βγαίνει από τα παραπάνω είναι ότι το υλικό μέρος ενός υπολογιστή, όπως εκείνου που περιγράψαμε στο κεφάλαιο 4, έχει πολλά μειονεκτήματα και περιορισμούς, και για αυτόν τον λόγο απαιτούνται επιπλέον διευκολύνσεις για την εύκολη και οικονομικότερη εκμετάλλευσή του. Αυτές οι διευκολύνσεις παρέχονται από μια ποικιλία προγραμμάτων που συνήθως τα προμηθεύει ο κατασκευαστής του υπολογιστή. Τα προγράμματα αυτά θεωρούνται μέρος του υπολογιστικού συστήματος και είναι γνωστά ως **λογισμικό συστήματος** (system software) ή **προγράμματα του συστήματος** (system programs). Σε αυτό το κεφάλαιο περιγράφουμε τα κυριότερα τμήματα του λογισμικού συστήματος, το **λειτουργικό σύστημα** (operating system), **μεταφραστές γλωσσών προγραμματισμού** (language translators), **εκδότες** ή **συντάκτες** (editors), **φορτωτές** (loaders) και **λογισμικό επικοινωνίας** (communications software).
Ακολουθεί σύντομη περιγραφή των τμημάτων του λογισμικού συστήματος. Το λειτουργικό σύστημα και οι μεταφραστές, αναλύονται διεξοδικότερα στις παραγράφους 6.2 έως 6.5 του παρόντος κεφαλαίου.

Το Λειτουργικό Σύστημα

Οι σύγχρονοι υπολογιστές είναι εφοδιασμένοι με ένα λειτουργικό σύστημα (αν και είναι δυνατό ο κατασκευαστής να προσφέρει περισσότερα του ενός) που παρέχει τις εξής λειτουργίες:

α) Παρέχει στον χρήστη μια εκτεταμένη (extended) ή ιδεατή (virtual) μηχανή πιο εύκολη στον προγραμματισμό και στην χρήση του υλικού. Απαλλάσσει τον χρήστη από τον προβληματισμό για ορισμένα χαρακτηριστικά του υπολογιστή, όπως η περιορισμένη μνήμη του ή οι ιδιομορφίες των μονάδων εισόδου/εξόδου. Παρέχει διευκολύνσεις στον χειρισμό προγραμμάτων ή δεδομένων, όπως για παράδειγμα να εκτελεστεί ένα πρόγραμμα με συγκεκριμένα δεδομένα και τα αποτελέσματα να εκτυπωθούν σε συγκεκριμένο εκτυπωτή.

β) Ρυθμίζει την αποδοτική χρήση των πόρων του συστήματος (resource management). Παρακολουθεί και κρατά στοιχεία για την χρήση πόρων και συντονίζει την απόδοση πόρων προς χρήση. Επιτρέπει την καταμερισμένη χρήση του υπολογιστή από πολλούς χρήστες ή/και από πολλά προγράμματα ενός χρήστη και συντονίζει την απόδοση πόρων του υπολογιστή σύμφωνα με τις απαιτήσεις τους.

Τα Λειτουργικά συστήματα περιγράφονται αναλυτικότερα στο εδάφιο 6.5.

Γνωστά λειτουργικά συστήματα είναι το Unix, που είναι εφοδιασμένοι συνήθως οι υπολογιστές για πολλούς χρήστες, το DOS, τα Windows95/98/NT, που είναι εφοδιασμένοι οι προσωπικοί υπολογιστές.

Μεταφραστές Γλωσσών Προγραμματισμού.

Είδαμε στο κεφάλαιο 2 ότι οι αλγόριθμοι εκφράζονται σε δομές ελέγχου όπως ακολουθία βημάτων, επιλογές, επαναλήψεις και αναδρομές. Ακόμη είδαμε ότι τα δεδομένα που επεξεργάζεται ο αλγόριθμος έχουν μια λογική δομή σε αρχεία, ακολουθίες ή δένδρα. Όμως λίγες δομές ελέγχου και δεδομένων είναι άμεσα διαθέσιμες σε γλώσσα μηχανής. Οι δομές ελέγχου πρέπει να αναλυθούν και να εκφραστούν από πρωτόγονες λειτουργίες όπως η JUMP, ενώ οι δομές δεδομένων από πρωτόγονες αναφορές σε θέσεις μνήμης που είναι σειριακά διατεταγμένες. Για να μπορούμε να εκφράζουμε πιο άμεσα τις δομές ελέγχου και δεδομένων που χρησιμοποιούν οι αλγόριθμοι αναπτύχθηκαν οι γλώσσες υψηλού επιπέδου (π.χ., Fortran, Cobol, ALGOL, Pascal και C).

Ένας αλγόριθμος για να εκφραστεί σε γλώσσα μηχανής χρειάζεται πολύ μεγαλύτερη ανάλυση απ' ότι σε μια γλώσσα υψηλού επιπέδου. Έτσι η απαιτούμενη προσπάθεια, αλλά και η πιθανότητα λαθών είναι πολύ μεγαλύτερη, ενώ ακόμα δυσκολότερη είναι η κατανόηση ενός προγράμματος σε γλώσσα μηχανής. Όμως οι γλώσσες υψηλού επιπέδου δεν είναι άμεσα κατανοητές από τον υπολογιστή. Οι μεταφραστές κάνουν ακριβώς αυτό που υπονοεί το όνομά τους, μεταφράζουν δηλαδή ένα πρόγραμμα γλώσσας υψηλού επιπέδου σε πρόγραμμα γλώσσας μηχανής. Κάθε γλώσσα υψηλού επιπέδου χρειάζεται διαφορετικό μεταφραστή προς μια γλώσσα μηχανής. Εκτός από τα μειονεκτήματα της γλώσσας μηχανής που ήδη έχουμε αναφέρει ένα ακόμη μειονέκτημα είναι ότι αναφέρεται σε συγκεκριμένο υπολογιστή. Η μεταφορά ενός προγράμματος από ένα υπολογιστή σε άλλον είναι εξαιρετικά δύσκολη καθώς πρέπει να ξαναγραφτεί το πρόγραμμα σε άλλη γλώσσα μηχανής. Οι όροι μεταφερσιμότητα ή φορητότητα (portability) χρησιμοποιούνται για να δηλώσουν την ιδιότητα μεταφοράς ενός προγράμματος σε διαφορετικούς υπολογιστές και είναι πολύ χαμηλή για προγράμματα σε γλώσσα μηχανής και πολύ υψηλή για προγράμματα σε γλώσσα υψηλού επιπέδου. Κάθε γλώσσα μηχανής χρειάζεται διαφορετικό μεταφραστή από μια γλώσσα υψηλού επιπέδου.

Η γλώσσα μηχανής έχει και μερικά πλεονεκτήματα και χρησιμοποιείται σε ειδικές περιπτώσεις. Επιτρέπει στον προγραμματιστή να παράγει προγράμματα μικρότερα σε μέγεθος και ταχύτερα σε εκτέλεση απ' ότι ένας μεταφραστής ενός προγράμματος σε γλώσσα υψηλού επιπέδου. Απαιτείται βεβαίως μεγάλη δεξιοτεχνία και κόστος σε ανθρώπινο δυναμικό, γενικά πολύ μεγαλύτερο της αποτελεσματικότερης εκτέλεσης του προγράμματος.
Σε λίγες περιπτώσεις όπου απαιτείται μεγάλη αποδοτικότητα (όπως οι χρονικά κρίσιμες εφαρμογές που αναφέρθηκαν στο εδάφιο 3.2.1), μπορεί να επιτευχθεί με τον συντονισμό (tuning) του προγράμματος γλώσσας μηχανής που δίνει ο μεταφραστής. Ο συντονισμός συνίσταται στην αναγνώριση τμημάτων που εκτελούνται συχνότερα (όπως οι εσωτερικοί βρόχοι) και στην αντικατάστασή τους με ρουτίνες σε γλώσσα μηχανής ειδικά γραμμένες ώστε να έχουν μεγαλύτερη αποδοτικότητα.
Ένα ακόμη πλεονέκτημα της γλώσσας μηχανής είναι ότι επιτρέπει στον προγραμματιστή την πρόσβαση σε στοιχεία του υλικού, όπως οι καταχωρητές CPU και οι καταχωρητές κατάστασης του περιφερειακού (βλέπε κεφ 4). Η πρόσβαση αυτή δεν είναι απαραίτητη στις συνήθεις εφαρμογές, γι' αυτό οι περισσότερες γλώσσες προγραμματισμού δεν την παρέχουν. Όπου είναι απαραίτητη, ιδίως σε ορισμένα τμήματα του λειτουργικού συστήματος, χρησιμοποιούνται γλώσσες μηχανής.
Συνοψίζοντας μπορούμε να πούμε ότι οι γλώσσες υψηλού επιπέδου χρησιμοποιούνται στις περισσότερες περιπτώσεις προγραμματισμού. Η γλώσσα μηχανής χρησιμοποιείται για συντονισμό ή για την πρόσβαση σε τμήματα του υπολογιστή που αλλιώς παραμένουν «κρυφά».

Εκδότες-Συντάκτες

Κατά την ανάπτυξη ή συντήρηση ενός προγράμματος απαιτούνται αλλαγές (διόρθωση λαθών, πρόσθεση στοιχείων κτλ). Για να αποφύγουμε την ολική επαναπληκτρολόγιση του διατηρούμε (αποθηκεύουμε) το ενημερωμένο πρόγραμμα σε αρχείο στην δευτερεύουσα μνήμη. Για την αρχική ανάπτυξή του και τις αλλαγές που χρειάζονται χρησιμοποιούμε το λογισμικό συστήματος που ονομάζεται εκδότης ή συντάκτης (editor). Ο συντάκτης μπορεί να χρησιμοποιηθεί όχι μόνο να αλλάξει προγράμματα αλλά οποιοδήποτε κείμενο που βρίσκεται στη δευτερεύουσα μνήμη. Η φύση των πληροφοριών ή του κειμένου εξαρτάται από την εφαρμογή. Μπορεί να είναι τεκμηρίωση προγράμματος, εγχειρίδια χρήστη, τυποποιημένα έγγραφα, κατάλογοι, βιβλιογραφίες, λογοτεχνικά κείμενα. Γνωστοί συντάκτες είναι το vi στο Unix και το Wordpad στα Windows.
Οι διευκολύνσεις που παρέχει ο συντάκτης επιτρέπουν στο χρήστη να εντοπίσει, να σβήσει ή να αντικαταστήσει συγκεκριμένα τμήματα του κειμένου ή να εισαγάγει νέα τμήματα στα σημεία που επιθυμεί. Πολλοί συντάκτες χρησιμοποιούνται διαλογικά (interactively), δηλαδή ο χρήστης δίνει στον συντάκτη εντολές που εκτελούνται αμέσως. Τα τερματικά οθόνης προσφέρονται για διόρθωση καθώς ο χρήστης μπορεί να παρακολουθεί τις αλλαγές καθώς γίνονται. Ένας τέτοιος συντάκτης ονομάζεται επεξεργαστής κειμένου (word processor). Γνωστός επεξεργαστής κειμένου είναι ο Word της Microsoft.

Φορτωτές

Ορισμένοι μεταφραστές γλωσσών αποθηκεύουν τα μεταφρασμένα προγράμματα στη δευτερεύουσα μνήμη. Το λογισμικό συστήματος που είναι υπεύθυνο να μεταφέρει το πρόγραμμα στην μνήμη και να ενεργοποιήσει την εκτέλεση του

ονομάζεται φορτωτής (loader). Οι περισσότεροι φορτωτές μπορούν να συνενώσουν μεταφρασμένα μέρη προγραμμάτων σε ένα και μόνο πρόγραμμα. Τα μεταφρασμένα μέρη μπορεί να είναι έτοιμες βιβλιοθήκες ή μέρη προγραμμάτων τα οποία έχουν μεταφραστεί σε διαφορετικούς χρόνους. Με αυτό τον τρόπο μέρη του προγράμματος που δεν χρειάζονται αλλαγές δεν ξαναμεταφράζονται.

Λογισμικό επικοινωνίας

Όπως είδαμε στο προηγούμενο κεφάλαιο οι υπολογιστές μπορούν να επικοινωνούν μεταξύ τους μέσω δικτύων. Το λογισμικό συστήματος που είναι υπεύθυνο για την μετάδοση και λήψη δεδομένων ονομάζεται λογισμικό επικοινωνίας. Στις λειτουργίες του περιλαμβάνονται η μετατροπή δεδομένων στις διαφορετικές απεικονίσεις που απαιτεί κάθε υπολογιστής, ο εντοπισμός και η διόρθωση λαθών που προκύπτουν κατά την μετάδοση, η σωστή δρομολόγηση δεδομένων διαμέσου του δικτύου και η προστασία του δικτύου από υπερφόρτωση.

Ολική Θεώρηση Συστήματος

Σχηματικά, μπορούμε να παραστήσουμε την σχέση των προγραμμάτων ως εξής

<table>
<tr><td colspan="5">ΕΦΑΡΜΟΓΕΣ (Τραπεζικές, Συστήματα Διαχείρισης ΒΔ, Βάσεις Παιχνίδια, …)</td></tr>
<tr><td>Λειτουργικό Σύστημα (Εντολές)</td><td>Φορτωτές</td><td>Εκδότες-Συντάκτες</td><td>Μεταγλωττιστές</td><td>Λογισμικό Επικοινωνίας</td></tr>
<tr><td colspan="5">Λειτουργικό Σύστημα (Πυρήνας)</td></tr>
<tr><td colspan="5">Υπολογιστής Κεφαλαίου 4
Φυσικοί πόροι (Κεντρική μονάδα, κύρια και δευτερεύουσα μνήμη, I/O, κλπ)
Γλώσσα Μηχανής
Μικροπρογραμματισμός</td></tr>
</table>

Στις επόμενες ενότητες θα επικεντρωθούμε στους Μεταφραστές και στα Λειτουργικά Συστήματα.

ΑΝΑΚΕΦΑΛΑΙΩΣΗ (απαντά στους αρχικούς στόχους)

1. Ο υπολογιστής του προηγούμενου κεφαλαίου έχει πολλά μειονεκτήματα χρήσης και προγραμματισμού:
   - Δυσκολία Εισαγωγής και Ενεργοποίησης Εκτέλεσης Προγράμματος
   - Απαιτεί την Διαχείριση μονάδων εισόδου/εξόδου
   - Περιορισμός Μνήμης
   - Δέσμευση και Υποαπασχόληση Υπολογιστή
   - Προγραμματισμός σε γλώσσα μηχανής
2. Το λογισμικό συστήματος παρέχει διευκολύνσεις χρήσης και προγραμματισμού και κύρια αποτελείται από τα εξής τμήματα: το λειτουργικό σύστημα, τους μεταφραστές γλωσσών προγραμματισμού, τους συντάκτες ή εκδότες, τους φορτωτές και το λογισμικό επικοινωνίας.
3. Το λειτουργικό σύστημα
   - παρέχει διευκολύνσεις στον χειρισμό προγραμμάτων ή δεδομένων και
   - ρυθμίζει την αποδοτική χρήση των πόρων του συστήματος.

4. Οι μεταφραστές μεταφράζουν ένα πρόγραμμα γλώσσας υψηλού επιπέδου σε πρόγραμμα γλώσσας μηχανής. Ο προγραμματισμός σε γλώσσα υψηλού επιπέδου έχει τα εξής πλεονεκτήματα σε σχέση με αυτόν σε γλώσσα μηχανής:
   - εκφράζουμε πιο άμεσα τις δομές ελέγχου και δεδομένων που χρησιμοποιούν οι αλγόριθμοι
   - η απαιτούμενη προσπάθεια και η πιθανότητα λαθών είναι πολύ μικρότερη
   - η κατανόηση προγραμμάτων υψηλού επιπέδου είναι ευκολότερη
   - έχουν υψηλή μεταφερσιμότητα

   Οι γλώσσες μηχανής έχουν και πλεονεκτήματα:
   - επιτρέπουν (με μεγάλη δεξιοτεχνία και κόστος) προγράμματα μικρότερα σε μέγεθος και ταχύτερα σε εκτέλεση απ' ότι ένα μεταφρασμένο
   - επιτρέπουν στον προγραμματιστή την πρόσβαση σε στοιχεία του υλικού
5. Οι συντάκτες-εκδότες χρησιμοποιούνται για την αρχική ανάπτυξή και τις αλλαγές προγραμμάτων, καθώς και οποιοδήποτε κειμένου.
6. Ο φορτωτής μεταφέρει εκτελέσιμα προγράμματα από την δευτερεύουσα μνήμη στην κύρια μνήμη και ενεργοποιεί την εκτέλεση του.
7. Το Λογισμικό Επικοινωνίας είναι υπεύθυνο για την μετάδοση και λήψη δεδομένων.

Ασκήσεις Αυτοαξιολόγησης

1. Τι μειονεκτήματα έχει η χρήση και ο προγραμματισμός του βασικού υπολογιστή του προηγούμενου κεφαλαίου;

Απάντηση

1. Δυσκολία εισαγωγής προγραμμάτων στην μνήμη και ενεργοποίησης της εκτέλεσης τους
2. Απαιτεί και επιφορτίζει τον προγραμματιστή με την διαχείριση μονάδων εισόδου/εξόδου
3. Πρόγραμμα και δεδομένα περιορίζονται στο μέγεθος της κύριας μνήμης
4. Δέσμευση και Υποαπασχόληση Υπολογιστή σε ένα πρόγραμμα
5. Απαιτεί προγραμματισμό σε γλώσσα μηχανής

2. Ποια είναι τα κύρια τμήματα του λογισμικού συστήματος;

Απάντηση

Το λειτουργικό σύστημα, οι μεταφραστές γλωσσών προγραμματισμού, οι συντάκτες, οι φορτωτές και λογισμικό επικοινωνίας.

3. Για ποιο λόγο χρειαζόμαστε το λογισμικό συστήματος;

Απάντηση

Το υλικό μέρος ενός υπολογιστή έχει πολλά μειονεκτήματα (δες ερώτηση 1)στην χρήση και τον προγραμματισμό του και απαιτούνται διευκολύνσεις. Τα προγράμματα του λογισμικού συστήματος παρέχουν διευκολύνσεις για την εύκολη και αποδοτικότερη χρήση και προγραμματισμό των υπολογιστών.

4. Τι διευκολύνσεις παρέχει το Λειτουργικό Σύστημα;

Απάντηση

1. Παρέχει τον χρήστη μια εκτεταμένη μηχανή πιο εύκολη στον προγραμματισμό και την χρήση της. Απαλλάσσει τον χρήστη από περιορισμούς μνήμης και την διαχείριση μονάδων εισόδου/εξόδου.

2. Ρυθμίζει την αποδοτική χρήση των πόρων του συστήματος. Παρακολουθεί και κρατά στοιχεία για την χρήση πόρων και συντονίζει την απόδοση πόρων προς χρήση

5. Τι διευκολύνσεις παρέχουν οι μεταφραστές;

Απάντηση

Μεταφράζουν προγράμματα σε γλώσσες υψηλού επιπέδου σε ισοδύναμα προγράμματα γλώσσας μηχανής.. Οι αλγόριθμοι και τα δεδομένα που επεξεργάζονται μπορούν να εκφραστούν πιο άμεσα σε γλώσσες υψηλού επιπέδου από γλώσσες μηχανής. Οι μεταφραστές επιτρέπουν και διευκολύνουν τον προγραμματισμό.

6. Τι διευκολύνσεις παρέχουν οι Συντάκτες ή Εκδότες;

Απάντηση

Επιτρέπει την εισαγωγή, αλλαγή (πρόσθεση, αφαίρεση, διόρθωση) και αποθήκευση προγραμμάτων, πληροφοριών και κειμένων γενικά.

7. Τι διευκολύνσεις παρέχουν οι Φορτωτές;

Απάντηση

Ο φορτωτής μεταφέρει εκτελέσιμα προγράμματα από την δευτερεύουσα μνήμη στην κύρια μνήμη και ενεργοποιεί την εκτέλεση του.

8. Τι διευκολύνσεις παρέχει το Λογισμικό Επικοινωνίας;

Απάντηση

Το λογισμικό επικοινωνίας διευκολύνει την μετάδοση και λήψη δεδομένων μέσω δικτύων υπολογιστών.

ΒΙΒΛΙΟΓΡΑΦΙΑ, ΠΗΓΕΣ

1. *Εισαγωγή στη Σύγχρονη Επιστήμη των Υπολογιστών, Les Goldschlager & Andew Lister, εκδόσεις Δίαυλος 1994, σελ. 221-224.*
2. *Σύγχρονα Λειτουργικά Συστήματα,*

ΤΙΤΛΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

6.2 Μεταφραστές γλωσσών προγραμματισμού

ΠΡΟΑΠΑΙΤΟΥΜΕΝΗ ΓΝΩΣΗ

Σχεδίαση Αλγορίθμων (κεφάλαιο 2 του παρόντος μαθήματος)
Αρχιτεκτονική Υπολογιστών (κεφάλαιο 4 του παρόντος μαθήματος)
Υποενότητα 6.1

ΣΥΝΟΨΗ

Παρουσιάζουμε και συγκρίνουμε τις μεθόδους μετάφρασης και εκτέλεσης ενός προγράμματος υψηλού επιπέδου. Ανάλογα με την μέθοδο που χρησιμοποιούν διαχωρίζονται σε διερμηνείς και μεταγλωττιστές. Οι διερμηνείς μεταφράζουν και αμέσως εκτελούν τις εντολές του προγράμματος, ενώ οι μεταγλωττιστές μεταφράζουν το πλήρες πρόγραμμα που κατόπιν εκτελείται. Εξετάζουμε πλεονεκτήματα και μειονεκτήματα των διερμηνέων και μεταγλωττιστών.

ΓΝΩΣΤΙΚΟΙ ΣΤΟΧΟΙ

- Η διαδικασία της μετάφρασης γλώσσας υψηλού επιπέδου σε γλώσσα μηχανής
- Οι μέθοδοι μετάφρασης προγραμμάτων υψηλού επιπέδου σε γλώσσα μηχανής και η διαφοροποίησή των μεταφραστών σε δύο κατηγορίες, τους διερμηνείς και τους μεταγλωττιστές, ανάλογα με τον μέθοδο μετάφρασης και εκτέλεσης που χρησιμοποιούν.
- Οι βασικοί αλγόριθμοι μετάφρασης που χρησιμοποιούν οι διερμηνείς και οι μεταγλωττιστές.
- Πλεονεκτήματα και μειονεκτήματα των διερμηνέων και των μεταγλωττιστών.

Λέξεις κλειδιά
Διερμηνέας, μεταγλωττιστής, πηγαίο πρόγραμμα, αντικείμενο πρόγραμμα.

ΚΥΡΙΟ ΜΕΡΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

Εισαγωγή

Η λειτουργία του μεταφραστή γλωσσών προγραμματισμού είναι να μεταφράζει προγράμματα υψηλού επιπέδου σε ισοδύναμα προγράμματα γραμμένα σε γλώσσα μηχανής. Σ' αυτήν την ενότητα θα εξετάσουμε πως πραγματοποιείται η μετάφραση προγραμμάτων υψηλού επιπέδου σε γλώσσα μηχανής.
Σε ένα πρόγραμμα υψηλού επιπέδου υπάρχουν δύο οντότητες: οι εντολές και οι δομές δεδομένων. Ο μεταφραστής πρέπει να μετασχηματίσει ή να μετατρέψει τις οντότητες υψηλού επιπέδου σε αντίστοιχες ή ισοδύναμες της γλώσσας μηχανής.
Οι εντολές γλώσσας υψηλού επιπέδου είναι γενικά πιο «δυνατές» από εκείνες της γλώσσας μηχανής, δηλαδή από την μια μεριά μπορούμε να υλοποιήσουμε πιο εύκολα τους αλγόριθμους, και από την άλλη χρειάζονται πολλές εντολές γλώσσας μηχανής για να περιγράψουμε μια εντολή γλώσσας υψηλού επιπέδου. Έτσι κάθε εντολή γλώσσας υψηλού επιπέδου συνήθως μεταφράζεται σε έναν αριθμό εντολών γλώσσας μηχανής.

Οι δομές δεδομένων που διαχειρίζεται μια γλώσσα υψηλού επιπέδου (π.χ. λίστες, δένδρα, συμβολοσειρές) δεν είναι άμεσα διαθέσιμες σε γλώσσα μηχανής. Πρέπει να αναπαρασταθούν ως bits, αριθμοί και διευθύνσεις, και ο μεταφραστής πρέπει να μετασχηματίσει τις δομές δεδομένων σε αναπαράσταση στο επίπεδο της γλώσσας μηχανής.
Ανάλογα με την μέθοδο που χρησιμοποιούν οι μεταφραστές για τον μετασχηματισμό των οντοτήτων υψηλού επιπέδου τους κατατάσσονται σε δύο μεγάλες κατηγορίες, τους διερμηνείς (interpreters) και τους μεταγλωττιστές (compilers). Στις δύο επόμενες παραγράφους περιγράφουμε τις μεθόδους που χρησιμοποιούν και εξετάζουμε τα πλεονεκτήματα και μειονεκτήματα τους.

6.2.1 Διερμηνείς

Ο βασικός Αλγόριθμος

Ένας διερμηνέας μεταφράζει και εκτελεί άμεσα τις εντολές του προγράμματος υψηλού επιπέδου μια-μια, δηλαδή η εκτέλεση κάθε εντολής προηγείται της μετάφρασης της επόμενης. Ένας διερμηνέας εκτελεί τον εξής αλγόριθμο:

Ξεκίνα από την αρχή του προγράμματος υψηλού επιπέδου
Repeat
    Μετάφρασε την επόμενη εντολή υψηλού επιπέδου
    Κανόνισε να εκτελεστεί η μετάφραση
Until *το τέλος του προγράμματος υψηλού επιπέδου* (6.1)

Στις περισσότερες περιπτώσεις ο διερμηνέας είναι εφοδιασμένος με σειρά από έτοιμα στοιχεία (ρουτίνες-procedures) που αντιστοιχούν σε συγκεκριμένους τύπους εντολών υψηλού επιπέδου και αποτελούνται από σειρές ισοδύναμων εντολών σε γλώσσα μηχανής. Για να μεταφράσει κάθε εντολή υψηλού επιπέδου, ο διερμηνέας εξετάζει την εντολή για διαπιστώσει τον τύπο της και καλεί το αντίστοιχο στοιχείο. Τα δεδομένα στα οποία ενεργεί η εντολή υψηλού επιπέδου περνάνε στο στοιχείο ως παράμετροι. Όταν το στοιχείο εκτελεστεί ο διερμηνέας ξαναπαίρνει τον έλεγχο και επαναλαμβάνει τον κύκλο για την επόμενη εντολή υψηλού επιπέδου.

Εξακρίβωση τύπου εντολής

Ο διερμηνέας εξακριβώνει τον τύπο κάθε εντολής χρησιμοποιώντας τους συντακτικούς κανόνες της γλώσσας (βλ. Εδάφιο 2.2). Η ταχύτητα της μετάφρασης εξαρτάται από το συντακτικό της γλώσσας που μεταφράζεται. Όσο πιο πολύπλοκο το συντακτικό τόσο πιο αργή η μετάφραση. Αφού ο διερμηνέας μεταφράζει κάθε εντολή υψηλού επιπέδου κάθε φορά που εκτελείται είναι πλεονέκτημα να έχουμε όσο το δυνατό απλούστερο συντακτικό. Αυτός είναι και ο κυριότερος στόχος στη σχεδίαση γλωσσών υψηλού επιπέδου που συνήθως μεταφράζονται με διερμηνέα. Στην Basic, για παράδειγμα, ο τύπος κάθε εντολής καθορίζεται από την πρώτη λέξη. Υπάρχουν όμως και πολύπλοκες ανώτερες γλώσσες προγραμματισμού οι οποίες μεταφράζονται με διερμηνέα, όπως η APL. Ο τύπος μιας εντολής γλώσσας μηχανής καθορίζεται από τον κώδικα λειτουργίας της και ο μοναδικός της συντελεστής είναι τα δεδομένα που είναι αποθηκευμένα στη θέση της μνήμης που καθορίζεται από το πεδίο διεύθυνσής της.

Μειονεκτήματα Διερμηνέα

Από τον αλγόριθμο του διερμηνέα 6.1 καθίσταται σαφές ότι μια εντολή προγράμματος μεταφράζεται όσες φορές εκτελείται. Αυτή η επαναλαμβανόμενη μετάφραση είναι εντελώς περιττή και σε αρκετές περιπτώσεις απαράδεκτη:

1. Ο απαιτούμενος χρόνος για να μεταφραστεί μια εντολή ξεπερνά τον χρόνο που χρειάζεται για να εκτελεστεί.
2. Το πρόγραμμα εκτελείται συχνά χωρίς αλλαγή (π.χ. πρόγραμμα μισθοδοσίας)
3. Η ταχύτητα εκτέλεσης έχει μεγάλη σημασία (π.χ. σύστημα κράτησης θέσεων ή ελέγχου μηχανών βιομηχανίας)

Ο τρόπος για να αποφύγουμε την επιβάρυνση είναι να χρησιμοποιήσουμε διαφορετική μέθοδο μετάφρασης, την μεταγλώττιση.

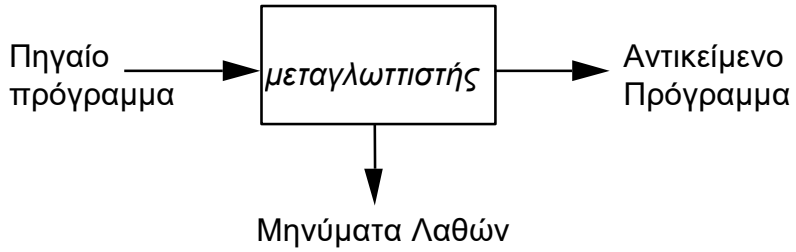
6.2.2 Μεταγλωττιστές (Compilers)

Εισαγωγή

Η μεταφραστική μέθοδος που χρησιμοποιούν οι μεταγλωττιστές είναι ο διαχωρισμός της μετάφρασης από την εκτέλεση του προγράμματος. Το πρόγραμμα μεταφράζεται ολόκληρο σε γλώσσα μηχανής και κατόπιν εκτελείται. Η μετάφραση και η εκτέλεση του προγράμματος με αυτή την μέθοδο περιγράφεται από τον αλγόριθμο 6.2

Ξεκίνα από την αρχή του προγράμματος υψηλού επιπέδου
Repeat
    Μετάφρασε την επόμενη εντολή υψηλού επιπέδου
Until *το τέλος του προγράμματος υψηλού επιπέδου*
Κανόνισε να εκτελεστεί ολόκληρο το μεταφρασμένο πρόγραμμα (6.2)

Η μετάφραση του προγράμματος υψηλού επιπέδου όπως περιγράφεται από τις πρώτες τέσσερις γραμμές του αλγόριθμου 6.2, καλείται μεταγλώττιση (compilation) και πραγματοποιείται από ένα πρόγραμμα που καλείται μεταγλωττιστής (compiler). Απλά ένας μεταγλωττιστής μεταφράζει ένα πρόγραμμα σε γλώσσα υψηλού επιπέδου, που καλείται πηγαίο πρόγραμμα (*source program*), σε ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής, που καλείται αντικείμενο πρόγραμμα (*object program*).



Σχήμα 6.2.1

Το αντικείμενο πρόγραμμα είναι αυτό που εκτελείται. Κατά την μετάφραση μπορούν να διαπιστωθούν λάθη τα οποία κοινοποιούνται στον προγραμματιστή μέσω μηνυμάτων.

Ανάλυση Συντακτικού Πηγαίου κώδικα

Όπως και ο διερμηνέας, ο μεταγλωττιστής αναλύει το συντακτικό του πηγαίου κώδικα χρησιμοποιώντας τους συντακτικούς κανόνες της γλώσσα υψηλού επιπέδου. Στην

επόμενη ενότητα 6.3 θα δούμε πως ορίζουμε το συντακτικό γλωσσών προγραμματισμού και αργότερα στην 6.4 πώς γίνεται η ανάλυση του συντακτικού.

Η Εκτέλεση του Αντικείμενου κώδικα

Αφού η μεταγλώττιση και η εκτέλεση είναι ξεχωριστές διαδικασίες, ένα αντικείμενο πρόγραμμα, που έχει παραχθεί από έναν μεταγλωττιστή, μπορεί να εκτελεστεί πολλές φορές. Το μόνο που απαιτείται είναι ένα μέσο αποθήκευσης του αντικείμενου προγράμματος. Γι' αυτό το σκοπό χρησιμοποιείται συνήθως η δευτερεύουσα μνήμη του υπολογιστή και μεταφέρεται στην κύρια μνήμη όταν απαιτείται η εκτέλεση του με την βοήθεια του φορτωτή.

Μία ακόμη διαφοροποίηση διερμηνέων και μεταγλωττιστών είναι στον τρόπο που αντιμετωπίζουν λάθη κατά την εκτέλεση των προγραμμάτων. Ο διερμηνέας έχει το πηγαίο πρόγραμμα στη διάθεσή του όταν εκτελείται μία μεταφρασμένη εντολή. Αν συμβεί λάθος ο διερμηνέας μπορεί εύκολα να δώσει πληροφορίες για την θέση λάθους και την τρέχουσα κατάσταση υπολογισμού. Επιπλέον οι πληροφορίες αυτές εκφράζονται κατευθείαν με το πηγαίο πρόγραμμα, δηλαδή ποια εντολή εκτελείται και ποιες οι τιμές των δεδομένων. Αντίθετα όταν εκτελείται ένα μεταφρασμένο πρόγραμμα το πηγαίο πρόγραμμα μπορεί να μην είναι διαθέσιμο. Αυτό σημαίνει ότι όταν συμβεί λάθος είναι πολύ δύσκολο να έχουμε σημαντικές πληροφορίες. Εξαιτίας της μεγαλύτερης ικανότητάς του να δίνει πληροφορίες για λάθη, ο διερμηνέας είναι κατάλληλος κυρίως για ανάπτυξη προγράμματος και εκσφαλμάτωση. Αφού το πρόγραμμα διορθωθεί θα πρέπει να μεταγλωττιστεί για να εκτελείται γρηγορότερα, ειδικά αν πρόκειται να εκτελείται συχνά. Έχει υπολογιστεί ότι η εκτέλεση ενός προγράμματος με διερμηνέα είναι 5-10 φορές πιο αργή από την εκτέλεση του αντίστοιχου μεταγλωττισμένου προγράμματος.
Οι διερμηνείς χρησιμοποιούνται συνήθως σε περιπτώσεις όπου το πρόγραμμα δεν εκτελείται συχνά, είναι συνήθως μικρά σε μέγεθος και η ταχύτητα εκτέλεσης δεν έχει ιδιαίτερη σημασία (π.χ. σε ένα εκπαιδευτικό περιβάλλον). Οι μεταγλωττιστές χρησιμοποιούνται όταν το πρόγραμμα εκτελείται συχνά ή η ταχύτητα εκτέλεσης είναι σημαντική.

ΑΝΑΚΕΦΑΛΑΙΩΣΗ (απαντά στους αρχικούς στόχους)

- Ο μεταφραστής μεταφράζει προγράμματα γλώσσας υψηλού επιπέδου σε ισοδύναμα προγράμματα εκφρασμένα σε γλώσσα μηχανής. Εντολές υψηλού επιπέδου μεταφράζονται σε αριθμό εντολών γλώσσας μηχανής. Οι δομές δεδομένων υψηλού επιπέδου μετασχηματίζονται σε αναπαράσταση γλώσσας μηχανής.

- Οι δύο μέθοδοι μετάφρασης προγραμμάτων υψηλού επιπέδου σε γλώσσα μηχανής (διερμηνεία και μεταγλώττιση) διαφοροποιούν τους μεταφραστές σε δύο κατηγορίες, τους διερμηνείς και τους μεταγλωττιστές.

- Ο διερμηνέας μεταφράζει εντολές μια-μια και τις εκτελεί αμέσως. Ο μεταγλωττιστής μεταφράζει όλο το πρόγραμμα σε γλώσσα μηχανής που κατόπιν εκτελείται.

- Πλεονεκτήματα και μειονεκτήματα των διερμηνέων και των μεταγλωττιστών:

	Διερμηνείς	Μεταγλωττιστές
Πλεονεκτήματα	- Πλήρεις πληροφορίες για λάθη εκτέλεσης	- Ταχύτητα εκτέλεσης - Κατάλληλοι όταν τα

	- Κατάλληλοι για ανάπτυξη και εκσφαλμάτωση και εκτέλεση όταν τα προγράμματα δεν τρέχουν συχνά ή η ταχύτητα δεν είναι σημαντική	προγράμματα τρέχουν συχνά ή η ταχύτητα είναι σημαντική
Μειονεκτήματα	- Χαμηλή ταχύτητα εκτέλεσης	- Περιορισμένες Πληροφορίες σε λάθη εκτέλεσης

Άσκηση 1. Ποια είναι η λειτουργία του μεταφραστή;

Απάντηση

Η λειτουργία του μεταφραστή γλωσσών προγραμματισμού είναι να μεταφράζει προγράμματα υψηλού επιπέδου σε ισοδύναμα προγράμματα γραμμένα σε γλώσσα μηχανής. Σε ένα πρόγραμμα υψηλού επιπέδου υπάρχουν δύο οντότητες: οι εντολές και οι δομές δεδομένων. Ο μεταφραστής πρέπει να μετασχηματίσει ή να μετατρέψει τις οντότητες υψηλού επιπέδου σε αντίστοιχες ή ισοδύναμες της γλώσσας μηχανής.

Άσκηση 2. Συμπληρώστε τις φράσεις με τις λέξεις «μηχανής» ή «υψηλού επιπέδου»

Οι αλγόριθμοι υλοποιούνται πιο εύκολα σε γλώσσα …….. . Για να περιγράψουμε μια εντολή γλώσσας …….. χρειάζονται πολλές εντολές γλώσσας ……. Κάθε εντολή γλώσσας …….. συνήθως μεταφράζεται σε έναν αριθμό εντολών γλώσσας ……..

Απάντηση

Οι αλγόριθμοι υλοποιούνται πιο εύκολα σε γλώσσα **υψηλού επιπέδου** . Για να περιγράψουμε μια εντολή γλώσσας **υψηλού επιπέδου** χρειάζονται πολλές εντολές γλώσσας **μηχανής**. Κάθε εντολή γλώσσας **υψηλού επιπέδου** συνήθως μεταφράζεται σε έναν αριθμό εντολών γλώσσας **μηχανής.**

Ερώτηση 3. Ποια η διαφορά διερμηνέα και μεταγλωττιστή;

Απάντηση

Ο διερμηνέας μεταφράζει εντολές μια-μια και τις εκτελεί αμέσως. Ο μεταγλωττιστής μεταφράζει όλο το πρόγραμμα σε γλώσσα μηχανής που κατόπιν εκτελείται.

Ερώτηση 4. Ποια πλεονεκτήματα έχει ένας διερμηνέας και ποια μειονεκτήματα;

Απάντηση

Πλεονεκτήματα: Δίνει πλήρεις πληροφορίες για λάθη εκτέλεσης και κατά συνέπεια είναι κατάλληλοι για ανάπτυξη και εκσφαλμάτωση.

Μειονεκτήματα: Η χαμηλή ταχύτητα εκτέλεσης των προγραμμάτων

Ερώτηση 5. Ποια πλεονεκτήματα έχει ένας μεταγλωττιστής και ποια μειονεκτήματα;

Απάντηση

Πλεονεκτήματα: Η υψηλή ταχύτητα εκτέλεσης των προγραμμάτων. Κατάλληλοι όταν τα προγράμματα τρέχουν συχνά ή η ταχύτητα είναι σημαντική.

Μειονεκτήματα: Δεν δίνει πλήρεις πληροφορίες για λάθη εκτέλεσης.

Ερώτηση 6. Συμπληρώστε τα κενά με τις λέξεις: εκσφαλμάτωσης, μεταγλωττιστή, διερμηνέα, ανάπτυξης, υψηλού επιπέδου, χαμηλή, γλώσσα μηχανής

Για την ανάπτυξη ενός αλγόριθμου σε γλώσσα …… έχω να επιλέξω ανάμεσα σε έναν διερμηνέα και έναν μεταγλωττιστή. Αποφάσισα να συνδυάσω τα πλεονεκτήματα και των δύο. Έτσι κατά το αρχικό στάδιο της …… και …….. χρησιμοποιώ τον ……., ώστε να με βοηθάει να εντοπίζω πιο εύκολα τα λάθη. Σε αυτό το στάδιο δεν με ενδιαφέρει η ……. ταχύτητα εκτέλεσης. Όταν είμαι σίγουρος ότι το πρόγραμμα δεν έχει λάθη το μεταφράζω σε …….. χρησιμοποιώντας τον ……..

Απάντηση

Για την ανάπτυξη ενός αλγόριθμου σε γλώσσα **υψηλού επιπέδου** έχω να επιλέξω ανάμεσα σε έναν διερμηνέα και έναν μεταγλωττιστή. Αποφάσισα να συνδυάσω τα πλεονεκτήματα και των δύο. Έτσι κατά το αρχικό στάδιο της **ανάπτυξης** και **εκσφαλμάτωσης** χρησιμοποιώ τον διερμηνέα ώστε να με βοηθάει να εντοπίζω πιο εύκολα τα λάθη. Σε αυτό το στάδιο δεν με ενδιαφέρει η **χαμηλή** ταχύτητα εκτέλεσης. Όταν είμαι σίγουρος ότι το πρόγραμμα δεν έχει λάθη το μεταφράζω σε **γλώσσα μηχανής** χρησιμοποιώντας τον **μεταγλωττιστή**.

ΒΙΒΛΙΟΓΡΑΦΙΑ, ΠΗΓΕΣ

1. *Εισαγωγή στη Σύγχρονη Επιστήμη των Υπολογιστών, Les Goldschlager & Andew Lister, εκδόσεις Δίαυλος 1994, σελ. 224-229.*
2. Compilers: Principles, Techniques and Tools, A. Aho, R. Sethi, J.D.Ullman, Addison Wesley, 1986, σελ. 1-4.

ΤΙΤΛΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

6.3 Ορισμός Συντακτικού

ΠΡΟΑΠΑΙΤΟΥΜΕΝΗ ΓΝΩΣΗ

Σχεδίαση Αλγορίθμων (κεφάλαιο 2 του παρόντος μαθήματος)
Ενότητα 6.2 του παρόντος κεφαλαίου

ΣΥΝΟΨΗ

Στην ενότητα αυτή εισάγουμε τον τυπικό ορισμό συντακτικού γλωσσών προγραμματισμού. Παρουσιάζεται ο συμβολισμός BNF, με τον οποίο ορίζεται τυπικά το συντακτικό γλωσσών προγραμματισμού. Δίνονται παραδείγματα BNF.

ΓΝΩΣΤΙΚΟΙ ΣΤΟΧΟΙ

- Ορισμός συντακτικού γλωσσών προγραμματισμού
- Εισαγωγή στο BNF συμβολισμό
  - Τερματικά και μη τερματικά σύμβολα
  - Κανόνες παραγωγής
  - Αναδρομή στους κανόνες παραγωγής

Λέξεις κλειδιά

Ορισμός Συντακτικού, BNF Συμβολισμός, Τερματικά σύμβολα, Μη τερματικά σύμβολα, Συντακτικές κατηγορίες, Κανόνες Παραγωγής, Συντακτικό γλώσσας

ΚΥΡΙΟ ΜΕΡΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

Εισαγωγή

Για να γίνει η μετάφραση προγράμματος γλώσσας υψηλού επιπέδου πρέπει η σύνταξη του να είναι σωστή. Επιπλέον, η μετάφραση σωστών προγραμμάτων εξαρτάται άμεσα από το συντακτικό της γλώσσας υψηλού επιπέδου. Σε αυτήν την ενότητα περιγράφουμε πως μπορεί να καθοριστεί το συντακτικό γλωσσών προγραμματισμού.

Σύμβολα και Συντακτικές Κατηγορίες

Θυμίζουμε (από τα κεφ. 2 και 3) ότι ένα πρόγραμμα αποτελείται από μια σειρά βασικών συμβόλων όπως τα ονόματα μεταβλητών, κάποια σημεία στίξης, τελεστές και ειδικές λέξεις, όπως **if** και **repeat**. Τα επιτρεπόμενα σύμβολα μιας γλώσσας ονομάζονται τερματικά σύμβολα (terminal symbols). Όλα τα σωστά προγράμματα σε αυτή την γλώσσα περιέχουν μόνο τα τερματικά αυτά σύμβολα. Τα τερματικά σύμβολα μπορούν να συνδυαστούν με διάφορους τρόπους για το σχηματισμό προγραμματιστικών δομών, όπως οι συνθήκες (π.χ. X<=5) και οι παραστάσεις (π.χ. a+b*c). Κάθε είδος δομής που κατασκευάζεται με αυτό τον τρόπο ονομάζεται συντακτική κατηγορία (syntactic category) ή μη τερματικό σύμβολο (non-terminal symbol).
Συντακτικές κατηγορίες μπορούν να συνδυαστούν με άλλες συντακτικές κατηγορίες ή και με τερματικά σύμβολα για να κατασκευαστούν άλλες ανώτερες συντακτικές κατηγορίες. Π.χ. Οι εντολές ανάθεσης τιμής αποτελούνται από ένα όνομα (το αριστερό μέρος), το σύμβολο απόδοσης (έστω :=) και μια παράσταση (δεξιό μέρος). Η μεγαλύτερη συντακτική κατηγορία, που συνδυάζει έμμεσα ή άμεσα όλες τις υπόλοιπες είναι ολόκληρο το πρόγραμμα. (Υπάρχει μια αναλογία με φυσικές γλώσσες, της οποίας τα τερματικά σύμβολα είναι οι αποδεκτές λέξεις και τα σημεία στίξης –γραμματολογικά στοιχεία-, και οι συντακτικές κατηγορίες περιλαμβάνουν φράσεις, όρους και προτάσεις –συντακτικά στοιχεία-).

Ο συμβολισμός BNF

Χρειαζόμαστε τον συντακτικό ορισμό ως την μέθοδο με την οποία κατασκευάζονται επακριβώς οι συντακτικές κατηγορίες μιας γλώσσας. Αυτό γίνεται διατυπώνοντας κανόνες ή παραγωγές (rules ή productions) καθένας από τους οποίους προσδιορίζει πως μπορεί να κτιστεί μια συντακτική κατηγορία από

1. μέλη άλλων συντακτικών κατηγοριών, και/ή
2. τερματικά σύμβολα της γλώσσας.

Το σύνολο των τερματικών συμβόλων, των συντακτικών κατηγοριών και των κανόνων προσδιορίζουν το συντακτικό της γλώσσας. Η πρωτοποριακή δουλειά στο συντακτικό των γλωσσών προγραμματισμού έγινε στο τέλος της δεκαετίας του 50 και κορυφώθηκε με την έκδοση της έκθεσης της γλώσσας ALGOL60 το 1960 και 1962. Η ALGOL60 ήταν η πρώτη διαδεδομένη γλώσσα προγραμματισμού που το συντακτικό της ορίστηκε με ολοκληρωμένο και τυπικό ορισμό συντακτικού. Η σημειογραφία της έκθεσης ήταν η **Backus-Naur Form** (BNF) και φέρει τα ονόματα των επινοητών της. Η ίδια σημειογραφία, με μικρές διαφοροποιήσεις, χρησιμοποιήθηκε στη συνέχεια για να οριστεί

το συντακτικό πολλών άλλων γλωσσών προγραμματισμού. Η σημειογραφία που θα χρησιμοποιηθεί εδώ είναι μια παραλλαγή της.

Παράδειγμα BNF

Ως παράδειγμα θα θεωρήσουμε το συντακτικό ορισμό των καταχωρήσεων στον τηλεφωνικό κατάλογο που περιέχει τα ονόματα, διευθύνσεις και τηλέφωνα των συνδρομητών με μια συγκεκριμένη διάταξη. Έστω ότι μια τυπική καταχώρηση στον κατάλογο είναι

Σιδηρόπουλος Ιωάννης, Πανεπιστημίου 20, 3636224

Το συντακτικό των καταχωρήσεων μπορεί να προσδιοριστεί από μια BNF παραγωγή

καταχώρηση -> όνομα_προσώπου , *διεύθυνση* , αριθμός

όπου οι συντακτικές κατηγορίες («καταχώρηση», «όνομα_προσώπου» και «διεύθυνση») είναι με πλάγια μορφή, ενώ τα τερματικά σύμβολα («αριθμός» και το «,») σε κανονική. Το σύμβολο «->» ανήκει στην σημειογραφία του συμβολισμού BNF και σημαίνει ότι η συντακτική κατηγορία από τα αριστερά του συνδυάζει συντακτικές κατηγορίες και τερματικά στα δεξιά του.

Η παραγωγή μπορεί να διαβαστεί ως εξής: μια καταχώρηση καθορίζεται ως ένα όνομα προσώπου, ακολουθούμενου από κόμμα, ακολουθούμενου από μια διεύθυνση, μετά από κόμμα και τέλος έναν αριθμό. Ξέρουμε ότι ένας αριθμός αποτελείται από συνεχόμενα ψηφία και γι' αυτό θεωρούμε ότι είναι τερματικό σύμβολο. Χρειάζεται όμως να καθοριστούν οι κατηγορίες *όνομα_προσώπου* και *διεύθυνση*. Για το πρώτο μπορούμε να γράψουμε

όνομα_προσώπου -> επίθετο μικρό_όνομα { *μικρό_όνομα* }

Τα άγκιστρα ανήκουν στον συμβολισμό BNF και σημαίνουν ότι η περιεχόμενη κατηγορία ή σύμβολο μπορεί να εμφανιστεί σε εκείνο το σημείο μηδέν ή περισσότερες φορές. Έτσι η παραπάνω παραγωγή μπορεί να διαβαστεί ως: ένα όνομα_προσώπου ορίζεται ως ένα επίθετο ακολουθούμενο τουλάχιστον από ένα μικρό_όνομα. Αν δεχθούμε ότι ένα μικρό όνομα μπορεί να συντμηθεί σε αρχικό γράμμα, ορίζουμε

μικρό_όνομα -> κύριο_όνομα | *αρχικό*

Η κάθετη γραμμή επίσης ανήκει στον συμβολισμό BNF και σημαίνει διάζευξη. Η παραγωγή διαβάζεται: ένα *μικρό_όνομα* ορίζεται ως ένα *κύριο_όνομα* ή ένα *αρχικό*. Ο ορισμός του αρχικού είναι

αρχικό -> κεφαλαίο_γράμμα.

Που σημαίνει ότι ένα αρχικό είναι ένα κεφαλαίο γράμμα ακολουθούμενο από μια τελεία (η οποία είναι τερματικό σύμβολο). Ο ορισμός του κεφαλαίου γράμματος είναι φυσικά

κεφαλαίο_γράμμα -> A | B | … | Ω

Ο ορισμός των υπόλοιπων συντακτικών κατηγοριών (επίθετο, κύριο όνομα, διεύθυνση) γίνεται με τον ίδιο τρόπο και φαίνεται στο σχήμα 6.3.1

καταχώρηση -> όνομα_προσώπου , *διεύθυνση* , αριθμός
όνομα_προσώπου -> επίθετο μικρό_όνομα { *μικρό_όνομα* }
επίθετο -> όνομα
μικρό_όνομα -> κύριο_όνομα | *αρχικό*
κύριο όνομα -> όνομα
αρχικό -> κεφαλαίο_γράμμα.
διεύθυνση -> όνομα_οδού αριθμός
όνομα_οδού -> όνομα
κεφαλαίο_γράμμα -> A | B | … | Ω

Σχήμα 6.3.1. Το συντακτικό των καταχωρήσεων σ' έναν απλό τηλεφωνικό κατάλογο

Ασκήσεις Αυτοαξιολόγησης

1. Ποιοι είναι οι συντακτικοί κανόνες του σχήματος 6.3.1 που ορίζουν τις συντακτικές κατηγορίες

α. *Επίθετο*
β. *Διεύθυνση*
γ. όνομα

Απάντηση
α. Ένας κανόνας *επίθετο* -> όνομα
β. Δύο κανόνες
διεύθυνση -> όνομα_οδού αριθμός
όνομα_οδού -> όνομα
γ. Το όνομα δεν είναι συντακτική κατηγορία, αλλά τερματικό. Επομένως δεν αναλύεται με άλλον κανόνα.

2. Ορίστε ένα όνομα όχι ως τερματικό, αλλά ως συντακτική κατηγορία που ξεκινάει με κεφαλαίο γράμμα και συνεχίζει με ένα ή περισσότερα μικρά γράμματα.

Απάντηση
- αλλάζουμε όπου όνομα σε *όνομα* (πλάγια γράμματα), και
- προσθέτουμε τους κανόνες
όνομα -> *κεφαλαίο_γράμμα μικρό_γράμμα* { *μικρό_γράμμα* }
μικρό_γράμμα -> α | β | … |ω
(Ο κανόνας για το *κεφαλαίο γράμμα* ήδη υπάρχει)

3. Γράψτε άλλες καταχωρήσεις σύμφωνα με τους συντακτικούς κανόνες του 6.3.1

Απάντηση
Οι δυνατότητες είναι άπειρες αφού μπορούμε να γράψουμε οποιοδήποτε όνομα ως επίθετο, μικρό όνομα, όνομα οδού και οποιοδήποτε αριθμό για αριθμό οδού ή τηλεφώνου. Δύο («παράξενες» αλλά συντακτικά σωστές) καταχωρήσεις
α) Πανεπιστημίου Ι. Ιωάννης Κ., Σιδηροπούλου 3636224, 524
β) Ιωάννης Σιδηρόπουλου Ι. Κ., Ιωάννης 20, 3636224
Παρατηρούμε ότι με τους συντακτικούς κανόνες του 6.3.1 δεν ελέγχουμε τα τηλέφωνα, που πρέπει να έχουν συγκεκριμένα ψηφία, ούτε τα κύρια, μικρά και ονόματα δρόμων, που μπορούν να μπουν σε οποιαδήποτε θέση ονόματος.

Το BNF μιας απλής γλώσσας προγραμματισμού

Το συντακτικό μιας γλώσσας προγραμματισμού είναι φυσικά πολυπλοκότερο από αυτό του παραδείγματος του τηλεφωνικού καταλόγου, αλλά μπορεί να οριστεί με τον ίδιο τρόπο. Η Pascal για παράδειγμα χρειάζεται εκατό περίπου παραγωγές. Το σχήμα 6.3.2 δίνει μια μικρή γεύση ενός συντακτικού ορισμού μιας απλής υποθετικής γλώσσας προγραμματισμού, βασισμένου στην δομή που παρουσιάστηκε στο κεφάλαιο 2.

εντολή -> *υπό_συνθήκη* | *βρόχος* | *ανάθεση*
υπό_συνθήκη - > **if** *συνθήκη* **then** *εντολή*
βρόχος -> **while** *συνθήκη* **do** *εντολή*
ανάθεση -> **θέσε το** όνομα **ίσο με** *έκφραση*
έκφραση -> όνομα *τελεστής* όνομα

τελεστής -> +|-
συνθήκη -> όνομα *σχέση* αριθμός
σχέση -> = | ^=

Σχήμα 6.3.2 Τμήμα συντακτικού απλής γλώσσας προγραμματισμού

Η γλώσσα περιέχει τριών ειδών εντολές:

1. Μια ανάθεση (assignment) που εκφράζει τον υπολογισμό μιας τιμής και την ανάθεση του αποτελέσματος σε ένα δεδομένο. Παραδείγματα ανάθεσης είναι
   θέσε το X ίσο με Y + Z
   θέσε το NET ίσο με GROSS-TAX
2. Ένας *βρόχος* (loop) που εκφράζει επανάληψη
3. Μια *υπό_συνθήκη* (conditional) που εκφράζει επιλογή

Οι συντακτικές κατηγορίες *υπό_συνθήκη* και *βρόχος* περιέχουν την συντακτική κατηγορία *συνθήκη*, η οποία αποτελεί έλεγχο της τιμής κάποιου δεδομένου.
Παραδείγματα συνθήκης είναι

X ^= 5 και GROSS=0

Κατά συνέπεια ένα παράδειγμα *υπό_συνθήκη* εντολής είναι

if X^=5 then θέσε το X ίσο με Y + Z

και παράδειγμα *βρόχου* το

while X^=0 do θέσε X ίσο με X – Y

Ασκήσεις Αυτοαξιολόγησης

Βάσει των κανόνων του 6.3.2

1. δώστε δύο ακόμη εντολές αναθέσεων
   Απάντηση
   Άπειρες εντολές μπορούν να δημιουργηθούν. Δίνουμε δύο
   θέσε το X ίσο με X – 1
   θέσε το Y ίσο με X + Y

2. δώστε δύο ακόμη συνθήκες
   Απάντηση
   Επίσης άπειρες *συνθήκες* μπορούν να δημιουργηθούν. Δίνουμε δύο
   X=3 και Y^=0
3. Συνδυάστε τις αναθέσεις και τις συνθήκες και δώστε δύο εντολές *υπό_συνθήκη* και δύο *βρόχους*
   Απάντηση
   Επίσης άπειρες *υπό_συνθήκη* και *βρόχοι* μπορούν να δημιουργηθούν. Δίνουμε δύο παραδείγματα από την κάθε συντακτική κατηγορία συνδυάζοντας τις απαντήσεις στις προηγούμενες δύο ασκήσεις

   if X=3 then θέσε το Y ίσο με X + Y
   if Y^=0 then θέσε το X ίσο με X – 1

   while Y^=0 do θέσε το Y ίσο με X + Y
   while X=3 do θέσε το X ίσο με X – 1

Αναδρομικοί Ορισμοί

Παρατηρούμε ότι ο ορισμός της *εντολής* στο συντακτικό 6.3.2 είναι αναδρομικός, αφού περιέχει συντακτικές κατηγορίες οι οποίες και οι ίδιες ορίζονται με εντολές. Έτσι για

παράδειγμα μια εντολή μπορεί να είναι μέρος βρόχου, που (ως εντολή και ο ίδιος) να περιέχεται σε άλλο *βρόχο*. Αυτό σημαίνει ότι μια εντολή αποτελείται από οποιοδήποτε αριθμό εμφωλιασμένων *βρόχων*. Όμοια μια εντολή μπορεί να περιέχει οποιοδήποτε αριθμό εμφωλιασμένων *υπο_επιλογών*, ή και οποιοδήποτε συνδυασμό εμφωλιασμένων *βρόχων* και *υπο_επιλογών*. Για παράδειγμα μια έγκυρη εντολή είναι

if X^=0 then while Y^=0 do θέσε το Y ίσο με Y-X

Ο αναδρομικός ορισμός είναι ένα πολύ ισχυρό εργαλείο του BNF, αφού επιτρέπει την δημιουργία οποιαδήποτε αριθμό εμφωλιασμένων δομών με σχετικά λίγες παραγωγές. Κατά συνέπεια μπορούμε με αυτές τις λίγες (και φυσικά πεπερασμένες) παραγωγές να περιγράφουμε το συντακτικό άπειρων προγραμμάτων. Σημειώνουμε, ότι οι αναδρομικές παραγωγές πρέπει να περιέχουν τουλάχιστον μία μη αναδρομική εναλλακτική λύση (συγκρίνετε το δρόμο διαφυγής –escape route- από τους αναδρομικούς αλγόριθμους στο εδάφιο 2.9). Στο συντακτικό 6.3.2, η *ανάθεση* είναι ένας μη αναδρομικός ορισμός για την *εντολή*.

Δεν θα ασχοληθούμε λεπτομερέστερα με τον ορισμό του συντακτικού. Ο ενδιαφερόμενος αναγνώστης παραπέμπεται στη βιβλιογραφία ή σε οποιοδήποτε σύγχρονο εγχειρίδιο γλώσσας προγραμματισμού το οποίο περιέχει τον τυπικό ορισμό του συντακτικού της γλώσσας. Η χρήση του ορισμού του συντακτικού στην ανάλυση του συντακτικού θα συζητηθεί στην υπο-ενότητα 6.4.2.

Ασκήσεις Αυτοαξιολόγησης

1. Συνδυάστε εμφωλιασμένες εντολές *υπό_συνθήκη* και *βρόχους* σε εντολές σύμφωνα με τους κανόνες του 6.3.2

   Απάντηση

   Άπειρες *υπό_συνθήκη* και *βρόχοι* μπορούν να εμφωλιαστούν σε εντολές. Δίνουμε τρία παραδείγματα

   if X=3 then if Y^=0 then θέσε το Y ίσο με X + Y
   while Y^=0 do if X=3 θέσε το Y ίσο με X + Y
   if X=3 then if Y^=0 then while X=3 do θέσε το X ίσο με X – 1

2. Αναπτύξτε συντακτικό ορισμό της γλώσσας δηλώσεων μεταβλητών της μορφής var x, y, z real

   Απάντηση

   Α τρόπος (με αναδρομή)
   Declaration -> var List real
   List -> name , List | name

   Β τρόπος (με επανάληψη)
   Declaration -> var name {, name} real
   Παρατηρούμε ότι οι δύο συντακτικοί ορισμοί περιγράφουν με διαφορετικό τρόπο, με διαφορετικές συντακτικές κατηγορίες, την ίδια γλώσσα.

3. Πως αλλάζει ο ανωτέρω συντακτικός ορισμός για να δηλώσουμε επίσης τύπους μεταβλητών integer, boolean και char;

   Απάντηση

   Επέκταση Α τρόπου
   Declaration -> var List type
   List -> name , List | name
   Type -> real | integer | boolean | char

Επέκταση Β τρόπου
Declaration -> var name {, name} type
Type -> real | integer | boolean | char

ΑΝΑΚΕΦΑΛΑΙΩΣΗ (απαντά στους αρχικούς στόχους)

Ο συντακτικός ορισμός είναι η τυπική μέθοδος με την οποία κατασκευάζονται επακριβώς οι συντακτικές κατηγορίας μιας γλώσσας. Στον συμβολισμό BNF διατυπώνονται κανόνες ή παραγωγές καθένας από τους οποίους προσδιορίζει πως μπορεί να κτιστεί μια συντακτική κατηγορία συνδυάζοντας μέλη άλλων συντακτικών κατηγοριών, και/ή τερματικά σύμβολα της γλώσσας. Στους συνδυασμούς μπορεί α) το ένα σύμβολο να ακολουθεί το άλλο, β) σύμβολα να εμφανίζονται εναλλακτικά και γ) επαναληπτικά. Επίσης μπορεί να εφαρμοστεί αναδρομή στους κανόνες, που είναι ισχυρό στοιχείο του BNF, γιατί με λίγους κανόνες μπορούμε να περιγράψουμε το συντακτικό όλων των προγραμμάτων κάποιας γλώσσας προγραμματισμού. Το σύνολο των τερματικών συμβόλων, των συντακτικών κατηγοριών και των κανόνων προσδιορίζουν το συντακτικό της γλώσσας με ακριβή και αυστηρό τρόπο.

ΒΙΒΛΙΟΓΡΑΦΙΑ, ΠΗΓΕΣ

1. *Les Goldschlager & Andew Lister, Εισαγωγή στη Σύγχρονη Επιστήμη των Υπολογιστών, εκδόσεις Δίαυλος 1994, σελ. 229-232*
2. A. Aho, R. Sethi, J.D.Ullman, Compilers: Principles, Techniques and Tools, Addison Wesley, 1986, σελ. 25-32.
3. R.L. Backhouse, The syntax of programming languages: theory and practice. Hemel Hempstead: Prentice Hall. 1979.

ΤΙΤΛΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

6.4 Μεταγλωττιστές

ΠΡΟΑΠΑΙΤΟΥΜΕΝΗ ΓΝΩΣΗ

Σχεδίαση Αλγορίθμων (κεφάλαιο 2 του παρόντος μαθήματος)
Αρχιτεκτονική Υπολογιστών (κεφάλαιο 4 του παρόντος μαθήματος)
Διδακτικές ενότητες 6.1-6.3

ΣΥΝΟΨΗ

Εξετάζεται η δομή και η λειτουργία των μεταγλωττιστών σε διαδοχικές φάσεις. Παρουσιάζονται οι τρεις φάσεις της ανάλυσης του πηγαίου κώδικα (λεξική, συντακτική, σημασιολογική), οι τρεις φάσεις της σύνθεσης του αντικείμενου κώδικα (παραγωγή ενδιάμεσου κώδικα, βελτιστοποίηση ενδιάμεσου κώδικα, παραγωγή γλώσσας μηχανής), καθώς επίσης και δύο βοηθητικές φάσεις, η διαχείριση πίνακα συμβόλων και διαχείριση λαθών. Εξετάζουμε την ομαδοποίηση των φάσεων.
Επίσης, παρουσιάζουμε βοηθητικά εργαλεία για την ανάπτυξη μεταγλωττιστών και μια ειδική κατηγορία μεταγλωττιστών, τους συμβολομεταφραστές.

ΓΝΩΣΤΙΚΟΙ ΣΤΟΧΟΙ

Στην ενότητα αυτή θα γνωρίσετε την εσωτερική λειτουργία των μεταγλωττιστών. Θα γνωρίσετε τις φάσεις της μεταγλώττισης, δηλαδή την διαδικασία μετατροπής του πηγαίου κώδικα σε κώδικα μηχανής. Θα γνωρίσετε τις τρεις φάσεις της ανάλυσης ενός πηγαίου προγράμματος

- Την λεξική ανάλυση, όπου αναγνωρίζονται οι λέξεις του
- Την συντακτική ανάλυση, όπου προσδιορίζεται η συντακτική δομή του
- Την σημασιολογική ανάλυση, όπου ελέγχονται μη συντακτικά χαρακτηριστικά του

Επίσης θα γνωρίσετε τις τρεις φάσεις της σύνθεσης του κώδικα μηχανής:

- Την παραγωγή ενός ενδιάμεσου κώδικα για μια αφηρημένη μηχανή
- Την βελτιστοποίηση του ενδιάμεσου κώδικα, που αφορά το μέγεθος και την ταχύτητα εκτέλεσης του, και
- Την παραγωγή του τελικού βελτιστοποιημένου κώδικα

Επίσης θα γνωρίσετε δύο βοηθητικές φάσεις, που συνεργάζονται με όλες τις άλλες:

- Την διαχείριση πίνακα συμβόλων, και
- Την διαχείριση μηνυμάτων λαθών

Θα γνωρίσετε διάφορους τρόπους και τους λόγους ομαδοποίησης των φάσεων, εργαλεία κατασκευής μεταγλωττιστών και τους συμβολομεταφραστές, μια ειδική κατηγορία μεταφραστών.

Λέξεις κλειδιά
Λεξική Ανάλυση, Κουπόνι Συντακτική Ανάλυση, Συντακτικό δένδρο, Σημασιολογική Ανάλυση, Δημιουργία Ενδιάμεσου Κωδικα, Βελτιστοποίηση Ενδιάμεσου Κώδικα, Παραγωγή_Τελικού Κωδικα, Διαχείριση Πίνακα Συμβόλων, Διαχείριση λαθών, Γεννήτορας συντακτικών αναλυτών, Μεταγλωττιστής –μεταγλωττιστών, Συμβολο-μεταφραστές

ΚΥΡΙΟ ΜΕΡΟΣ ΔΙΔΑΚΤΙΚΗΣ ΕΝΟΤΗΤΑΣ

Εισαγωγή

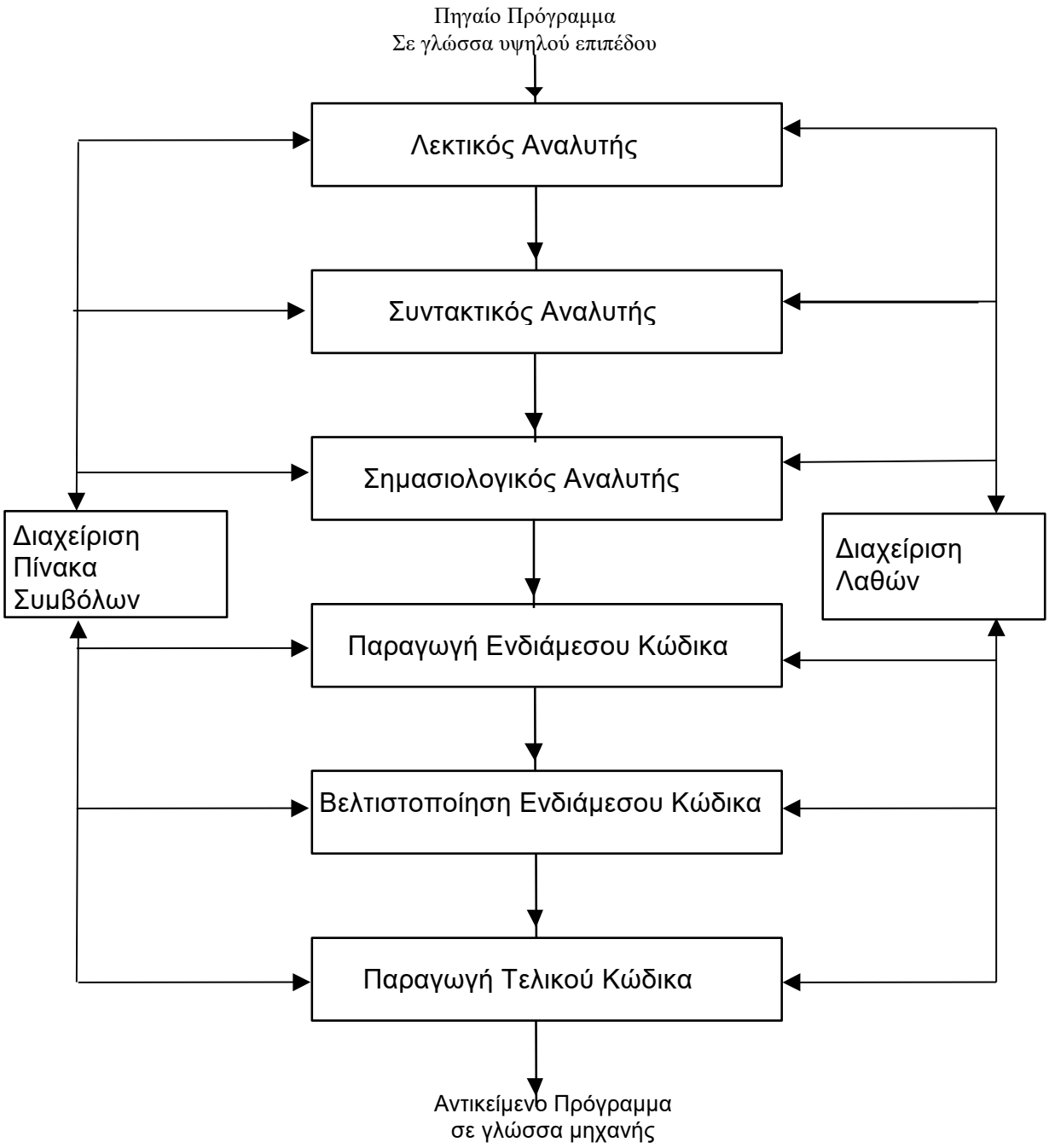
Η κατασκευή ενός μεταγλωττιστή είναι μια πολύπλοκη διαδικασία. Ειδικά για τους πρώτους μεταγλωττιστές (δεκαετία του 50) απαιτήθηκε τεράστια προσπάθεια. Αναφέρεται ότι ο πρώτος FORTRAN μεταγλωττιστής χρειάστηκε 18 ανθρωποέτη. Από τότε αφενός μεν έχουν συστηματικοποιηθεί οι τεχνικές για τον σχεδιασμό των διαφόρων διαδικασιών που εκτελούνται από ένα μεταγλωττιστή και αφετέρου έχουν αναπτυχθεί πολλά εργαλεία λογισμικού.
Στην ενότητα 6.2 πήραμε μια γενική εικόνα της διαδικασίας της μετάφρασης ενός προγράμματος από διερμηνείς και μεταφραστές. Το περιεχόμενο αυτής της ενότητας μπορεί να το δει κανείς ως ανάλυση του βήματος:

Μετάφρασε την επόμενη εντολή υψηλού επιπέδου

που είναι η τρίτη γραμμή των αλγορίθμων για διερμηνείς και μεταγλωττιστές. Η παρουσίαση θα διεξαχθεί στο πλαίσιο των μεταγλωττιστών, αλλά πολλά ισχύουν και για τους διερμηνείς.

Οι φάσεις της μεταγλώττισης

Τα στάδια της μετάφρασης ενός προγράμματος θα μπορούσαν να χωριστούν σε δύο στην ανάλυση και στη σύνθεση. Κατά την ανάλυση το πρόγραμμα εισόδου διαβάζεται, ελέγχεται η ορθότητά του, αναγνωρίζονται τα συστατικά μέρη του και το πηγαίο πρόγραμμα απεικονίζεται σε εσωτερικές δομές δεδομένων. Κατά την σύνθεση δημιουργείται ένας ενδιάμεσος κώδικας (ένα πρόγραμμα ισοδύναμο με αυτό της εισόδου), ο οποίος βετιστοποιείται, και από αυτόν παράγεται το τελικό πρόγραμμα σε γλώσσα assembler ή γλώσσα μηχανής.
Αυτός όμως ο διαχωρισμός είναι πολύ γενικός. Αναλυτικότερα, ένας μεταγλωττιστής μεταφράζει σε διαδοχικές φάσεις. Το στάδιο της ανάλυσης περιλαμβάνει 3 φάσεις: την Λεξική Ανάλυση, την Συντακτική Ανάλυση και την Σημασιολογική Ανάλυση. Το στάδιο της σύνθεσης περιλαμβάνει επίσης 3 φάσεις: Δημιουργία Ενδιάμεσου Κωδικα, Βελτιστοποίηση Ενδιάμεσου Κώδικα και Παραγωγή Τελικου Κώδικα. Κάθε φάση μετατρέπει το αρχικό πρόγραμμα από μια απεικόνιση σε μια άλλη ισοδύναμη. Το σχήμα 6.4.1 απεικονίζει τις φάσεις της μετάφρασης.



Σχήμα 6.4.1 Φάσεις της Μεταγλώττισης

Στο σχήμα 6.4.1 απεικονίζονται δυο επιπλέον διαδικασίες η Διαχείριση Πίνακα Συμβόλων και η Διαχείριση Λαθών που επικοινωνούν με τις έξι φάσεις που προαναφέραμε. Θα ονομάζουμε και αυτές φάσεις. Στην συνέχεια αναλύουμε την λειτουργία κάθε φάσης.

6.4.1 Διαχείριση Πίνακα Συμβόλων

Μια σημαντική λειτουργία ενός μεταγλωττιστή είναι να καταγράφει τις μεταβλητές που χρησιμοποιούνται στο πηγαίο πρόγραμμα και να συλλέγει πληροφορίες για τα χαρακτηριστικά των μεταβλητών αυτών. Παραδείγματα χαρακτηριστικών είναι

- Η θέση στον πηγαίο κώδικα που δηλώθηκε
- Ο τύπος της μεταβλητής (ακέραιος, πραγματικός, πίνακας, εγγραφή, κλπ)
- Το πεδίο ισχύος της μεταβλητής (το block που είναι δηλωμένο)
- Σε περίπτωση procedure ή function τον αριθμό παραμέτρων της

Τα χαρακτηριστικά αυτά δεν είναι τα ίδια για όλες τις γλώσσες. Κάθε γλώσσα μπορεί να απαιτεί ιδιαίτερα χαρακτηριστικά.
Οι πληροφορίες αποθηκεύονται σε μια δομή, στον *πίνακα συμβόλων* που επιτρέπει την γρήγορη αναζήτηση κάποιας μεταβλητής καθώς και την αποθήκευση και ανάκτηση πληροφοριών για την μεταβλητή. Κάθε φάση συνεργάζεται με την διαχείριση πίνακα συμβόλων και αποθηκεύει και συμπληρώνει ορισμένα χαρακτηριστικά της μεταβλητής στον πίνακα. Καθώς εξετάζουμε τις φάσεις θα αναφερθούμε στα χαρακτηριστικά που αποθηκεύουν ή ανακτούν από τον πίνακα συμβόλων.

6.4.2 Αναγνώριση Λαθών και μηνύματα

Κάθε φάση μπορεί να διαπιστώσει λάθη. Όταν ένα λάθος αναγνωριστεί δίδεται ένα αντίστοιχο μήνυμα λάθους και πρέπει να αντιμετωπιστεί ώστε να συνεχιστεί η μετάφραση του προγράμματος. Η λειτουργία αυτή ονομάζεται ανάκαμψη από λάθη (error recovery). Κάθε φάση έχει τις δικές της απαιτήσεις για ανάκαμψη. Ένας μεταγλωττιστής που σταματάει στο πρώτο λάθος δεν είναι ιδιαίτερα χρήσιμος, αν και υπάρχουν περιπτώσεις που κάποιος χρήστης θέλει να διορθώνει τα λάθη ένα-ένα καθώς αναγνωρίζονται. Στις περισσότερες περιπτώσεις είναι πιο χρήσιμο να μας δίνονται όλα τα λάθη ώστε να διορθώνονται όλα μαζί.
Η συντακτική και σημασιολογική ανάλυση αντιμετωπίζουν το μεγαλύτερο μέρος των λαθών.

6.4.3 Λεξική Ανάλυση (lexical analysis)

Στην πρώτη φάση γίνεται η αναγνώριση των συμβόλων ή κουπονιών (tokens) του προγράμματος. Κουπόνι είναι κάθε μια στοιχειώδης νοηματική μονάδα (ένα σύμβολο) που χρησιμοποιεί μια γλώσσα προγραμματισμού π.χ δεσμευμένες λέξεις (reserved words), όπως **while**, **if,** ονόματα μεταβλητών, τελεστές (<, <=, :=) κλπ. Ο λεξικός αναλυτής διαβάζει τους χαρακτήρες του πηγαίου προγράμματος και από αυτούς αναγνωρίζει λεκτικά (lexemes) που ανταποκρίνονται σε σύμβολα (tokens) της γλώσσας.
Ο λεξικός αναλυτής απλά φτιάχνει λέξεις από τους χαρακτήρες που πληκτρολογήσαμε. Ο λεξικός αναλυτής είναι όπως ένας ελεγκτής ορθογραφίας, που ελέγχει μόνο την ορθή γραμματική μορφή των λέξεων του κειμένου.

Λειτουργίες Λεξικού Αναλυτή

Πιο συγκεκριμένα ο λεξικός αναλυτής:

1. Προσδιορίζει αν ένας χαρακτήρας είναι άμεσα τερματικό σύμβολο (π.χ. “+”, “-“, ".", ",") ή αν πρέπει να ομαδοποιηθεί με επόμενους του για να σχηματίσουν τερματικό σύμβολο, π.χ. όνομα, αριθμό ή μια δεσμευμένη λέξη. Έτσι αναγνωρίζει σύμβολα όπως το ':=', που αποτελείται από δύο χαρακτήρες και το σύμβολο "**begin**" που αποτελείται από πέντε. Επίσης αναγνωρίζει ως όνομα την λέξη GROSS,

ομαδοποιώντας τούς πέντε χαρακτήρες (γράμματα) της και ως αριθμό την λέξη 60, ομαδοποιώντας τους δύο χαρακτήρες (ψηφία) της.

2. Εισάγει ονόματα μεταβλητών στον πίνακα συμβόλων. Π.χ. όταν αναγνωρίσει ένα όνομα θα τοποθετήσει στον πίνακα συμβόλων το λεκτικό του αν δεν είναι ήδη στον πίνακα.
3. Απομακρύνει τους συντακτικά πλεονάζοντες χαρακτήρες, όπως πολλαπλά κενά και σχόλια προοριζόμενα για τον αναγνώστη του πηγαίου προγράμματος.
4. Απορρίπτει άσχετα σύμβολα, δηλαδή χαρακτήρες που δεν σχηματίζουν σύμβολα της γλώσσας, ως λάθη στο πηγαίο πρόγραμμα.

Ο σχηματισμός κουπονιών

Τα τερματικά σύμβολα περνάνε από τον λεξικό στον συντακτικό αναλυτή υπό μορφή κουπονιών (tokens), που μπορούν να έχουν δύο συστατικά:
1. το είδος του τερματικού συμβόλου που παριστάνει το κουπόνι που αναγνωρίστηκε, π.χ. την δεσμευμένη λέξη if, το σύμβολο «+», αριθμό ή όνομα.
2. Σε μερικά σύμβολα θα περιλαμβάνει και κάποια "λεκτική τιμή". Για παράδειγμα, η τιμή ενός αριθμητικού κουπονιού είναι απλά ο αριθμός που παριστάνει. Η τιμή ενός ονόματος είναι το όνομα το ίδιο ή ο δείκτης στον πίνακα συμβόλων όπου έχει εισαχθεί.

Ο λεξικός αναλυτής, κατ' αναλογία με την γραμματολογική αναγνώριση λέξεων κειμένου, είναι σαν να χαρακτηρίζει με το πρώτο συστατικό μια λέξη ως ρήμα, ουσιαστικό ή συγκεκριμένο σημείο στίξης, κλπ. και με το δεύτερο να δίδει είτε την ίδια την λέξη είτε μια αναφορά της σε λεξικό όπου μπορούμε να βρούμε περισσότερες πληροφορίες.

Παράδειγμα Δημιουργίας Κουπονιών

Θα εφαρμόσουμε τη λεξική ανάλυση στην εντολή προγράμματος:

```
if X ^= 1 then
        θέσε το Y ίσο με X + Y
```

στη γλώσσα που ορίζεται στο σχήμα 6.3.2. Αν υποθέσουμε ότι το X και το Y έχουν αποθηκευτεί στις θέσεις 1 και 2 αντίστοιχα του πίνακα συμβόλων, όπως στο σχήμα 6.4.2, ο λεξικός αναλυτής μετασχηματίζει την εντολή στη σειρά των έντεκα κουπονιών
<if> <όνομα θέση1> <^=> <αριθμός 1> <then> <θέσε το> <όνομα θέση2> <ίσο με> <όνομα θέση1> <+> <όνομα θέση2>

Πίνακας Συμβόλων

Θέση	Όνομα	χαρακτηριστικά
1	X	
2	Y	
3		

Σχήμα 6.4.2 Ο Πίνακας Συμβόλων κατά τη λεξική ανάλυση

Ας δούμε πως εισάγονται τα ονόματα στον πίνακα συμβόλων. Ο λεξικός αναλυτής αναγνωρίζει σύμβολα όπως **if**, **then**, **θέσε το**, **ίσο με**, ^= και +, από την ακολουθία των χαρακτήρων τους. Αναγνωρίζει επίσης ότι οι χαρακτήρες X και Y είναι ονόματα, τα οποία αναζητά στον πίνακα συμβόλων. Αν βρει ένα όνομα στον πίνακα συμβόλων, πχ το X στη θέση 1, διαμορφώνει το αντίστοιχο κουπόνι, πχ <όνομα θέση1>. Αν δεν το βρει τότε πρώτα εισάγει το όνομα στον πίνακα συμβόλων και μετά διαμορφώνει το κουπόνι.

Παράδειγμα γλώσσας δηλώσεων μεταβλητών

Η λειτουργία του λεξικού αναλυτή είναι περιορισμένη. Αναγνωρίζει μόνο λέξεις και όχι συνδυασμούς τους. Π.χ. δεν αναγνωρίζει ότι η πιο πάνω εντολή είναι σωστή. Ας δώσουμε ένα ακόμη παράδειγμα. Έστω ότι προσθέτουμε στη γλώσσα μας δηλώσεις μεταβλητών της μορφής:

var X **real**;
var Y **integer**;

(δες ασκήσεις 2 και 3 του 6.3.2). Ο λεξικός αναλυτής

1. απλά αναγνωρίζει τα οκτώ σύμβολα με την σειρά που εμφανίζονται
   1) var 2) όνομα 3) real 4) ";" 5) var 6) όνομα 7) integer 8) ";"
2. αναγνωρίζει τα X και Y ως ονόματα και τα εισάγει στον πίνακα συμβόλων στις θέσεις 1 και 2 αντίστοιχα και
3. στέλνει στον συντακτικό αναλυτή τα κουπόνια:
   <var> <όνομα θέση1> <real> <;> <var> <όνομα θέση2> <integer> <;>

Ο λεξικός αναλυτής δεν αναγνωρίζει ότι πρόκειται για δήλωση μεταβλητών και πολύ περισσότερο δεν αναγνωρίζει τον τύπο των μεταβλητών. Ο συντακτικός αναλυτής θα αναγνωρίσει ότι πρόκειται για συντακτικά σωστή δήλωση και ο σημασιολογικός αναλυτής θα αναγνωρίσει τους τύπους των μεταβλητών.

Οι περισσότεροι λεξικοί αναλυτές εκτυπώνουν ή εμφανίζουν στη οθόνη, για διευκόλυνση του προγραμματιστή, το πηγαίο πρόγραμμα όπως αυτό δίδεται. Όποιο λάθος βρεθεί κατά την λεξική ανάλυση ή σε άλλες φάσεις αναφέρεται στην εκτύπωση του προγράμματος.

Ασκήσεις Αυτοαξιολόγησης

Ερώτηση 1.

Στην δήλωση μεταβλητών:

var position, initial, rate **real**;

Ποια σύμβολα αναγνωρίζει ο λεξικός αναλυτής, ποια ονόματα εισάγει στον πίνακα συμβόλων και τι κουπόνια στέλνει στον συντακτικό αναλυτή;

Απάντηση

ο λεξικός αναλυτής αναγνωρίζει επτά σύμβολα

1) var 2) όνομα 3) "," 4) όνομα 5) "," 6) όνομα 7) real

εισάγοντας τα τρία ονόματα των συμβόλων 2,4,6 στον πίνακα συμβόλων.

Πίνακας Συμβόλων

Θέση	Όνομα	χαρακτηριστικά
1	position	
2	initial	
3	rate	
4		

ο λεξικός αναλυτής θα στείλει στον συντακτικό αναλυτής τα κουπόνια
<var> <όνομα θέση1> <,> <όνομα θέση2> <,> <όνομα θέση3> <real>

όπου θέση1, θέση2 και θέση3 είναι οι δείκτες στη θέση του πίνακα συμβόλων όπου έχουν αποθηκευτεί αντίστοιχα οι μεταβλητές position, initial, rate.

Ερώτηση 2
Στην εντολή του προγράμματος όπου ήδη έχουν οριστεί οι μεταβλητές position, initial και rate

Θέσε position ίσο με initial + 60

ποιά κουπόνια στέλνει ο λεξικός αναλυτής;

Απάντηση

Διαμορφώνονται επτά κουπόνια:

<θέσε> <όνομα θέση1> < ίσο με > <όνομα θέση2> < + > <αριθμός 60>

Ο λεξικός αναλυτής αναγνωρίζει το θέσε ως ειδική λέξη της γλώσσας, τα position και initial ως ονόματα, τους τελεστές ":=" και "+" καθώς και τον αριθμό 60.

6.4.4 Συντακτική ανάλυση

Εισαγωγή

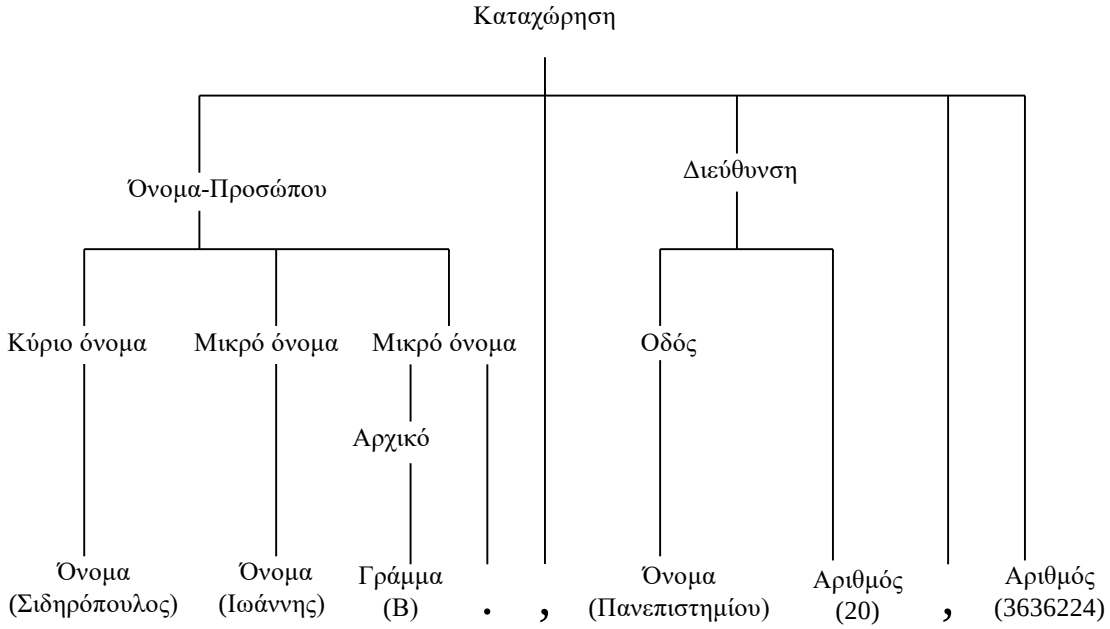
Ακολουθεί η δεύτερη φάση της μεταγλώττισης, η συντακτική ανάλυση (syntax analysis ή parsing). Σκοπός της είναι να αναλύσει το πηγαίο πρόγραμμα, δηλαδή να προσδιορίσει την συντακτική δομή του. Στη φάση αυτή από τη γραμμική έξοδο του λεξικού αναλυτή, την ακολουθία δηλαδή των κουπονιών, δημιουργούνται γραμματικές φράσεις σύμφωνα με τους συντακτικούς κανόνες της γλώσσας (δες 6.3).

Συντακτικά δένδρα- Παράδειγμα

Συνήθως οι γραμματικές αυτές φράσεις παριστάνονται με ένα δένδρο το οποίο λέγεται συντακτικό δένδρο (syntax tree ή parse tree) και έχει τα τερματικά σύμβολα στα φύλλα του και στους εσωτερικούς κόμβους συντακτικές κατηγορίες. Για παράδειγμα, το συντακτικό δένδρο που αντιστοιχεί στην καταχώρηση του τηλεφωνικού καταλόγου:

Σιδηρόπουλος Ιωάννης Β., Πανεπιστημίου 20, 3636224

Φαίνεται στο σχήμα 6.4.3. Το συντακτικό δένδρο κατασκευάστηκε σύμφωνα με τους συντακτικούς κανόνες του σχήματος 6.2.1.



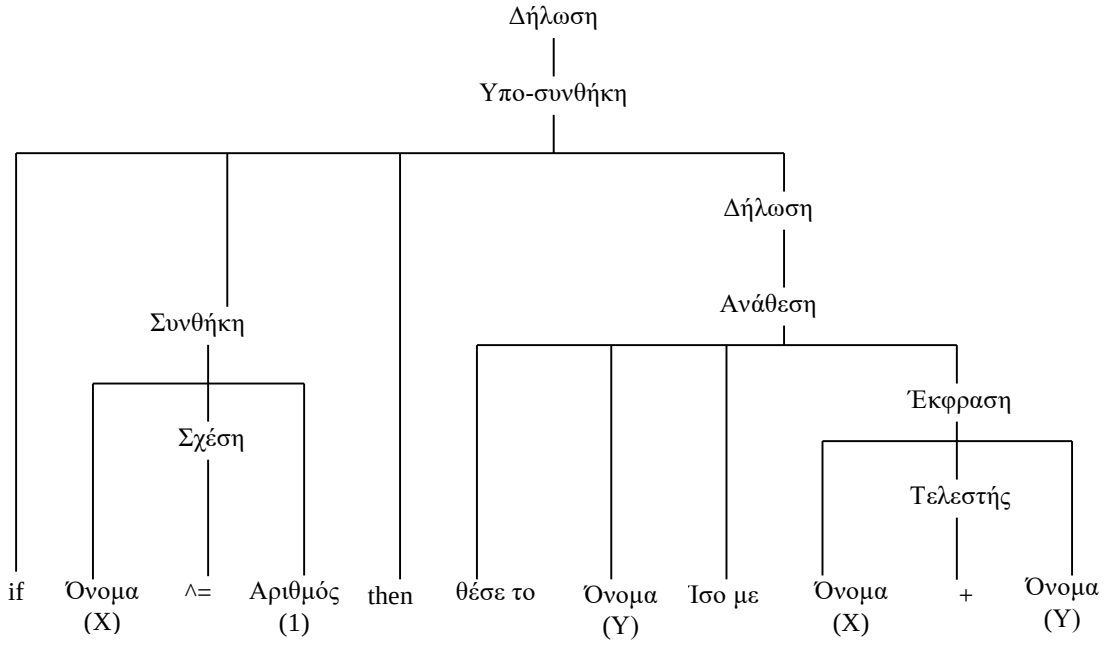
Σχήμα 6.4.3 Το συντακτικό δένδρο του παραδείγματος καταχώρησης

Άλλο Παράδειγμα Συντακτικού Δένδρου

Το Σχήμα 6.4.4 δείχνει ένα άλλο συντακτικό δένδρο αυτή την φορά της εντολής

if X^=1 then put Y equal to X + Y

με τους συντακτικούς κανόνες του 6.3.2.



Σχήμα 6.4.4 Συντακτικό δένδρο εντολής απλής γλώσσας προγραμματισμού

Τα δύο συντακτικά δένδρα στα σχήματα 6.4.3 και 6.4.4 δείχνουν ότι οι συγκεκριμένες ακολουθίες από σύμβολα είναι σωστές για τις αντίστοιχες γλώσσες. Τα συντακτικά δένδρα περιέχουν επίσης πληροφορίες για την δομή της ακολουθίας, δηλαδή όχι μόνο ότι είναι αποδεκτές στις συγκεκριμένες γλώσσες, αλλά δείχνουν και τους κανόνες που εφαρμόζονται.

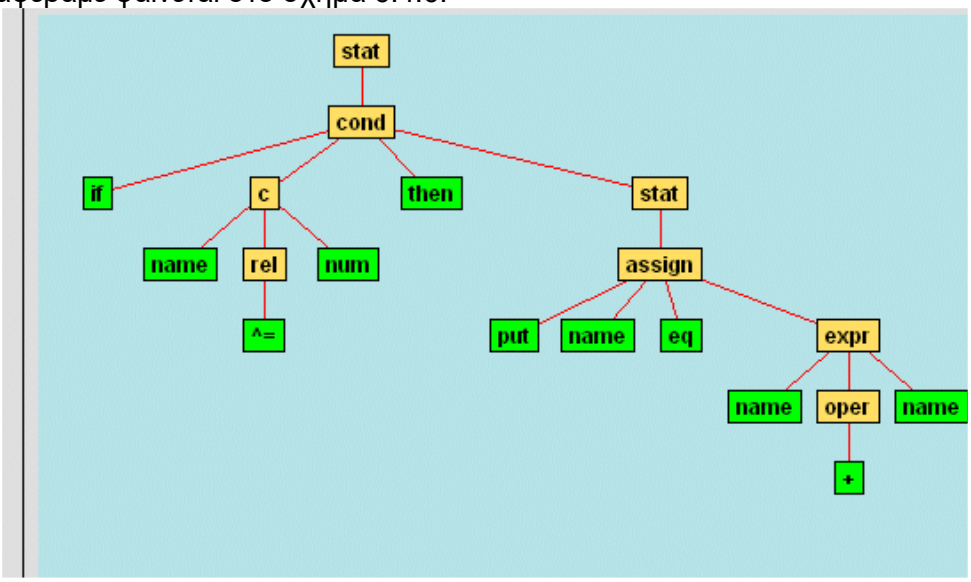
Εναλλακτικός Συμβολισμός BNF

Ας θεωρήσουμε τη γλώσσα που προσδιορίστηκε στο Σχήμα 6.3.2, αλλάζοντας όμως τα ελληνικά τερματικά (όνομα, αριθμός) σε αγγλικά (name, num). Τα μη τερματικά σύμβολα έχουν και αυτά μετατραπεί σε αγγλικές συντμήσεις. Τέλος το σύμβολο «->» μετατρέπεται σε «:». Οι αλλαγές γίνονται για να μπορέσετε αργότερα να χρησιμοποιήσετε ένα εργαλείο για την διερεύνηση γραμματικών το οποίο έχει μερικούς περιορισμούς, όπως π.χ. ότι δεν χειρίζεται ελληνικούς χαρακτήρες. Η νέα γραμματική μαζί με την αρχική μορφή για αντιπαράθεση είναι:

stat : cond | iter | assign cond : **if** c **then** stat iter : **while** c **do** stat assign : **put** name **eq** expr expr : name oper name oper : + | - c : name rel num rel : = | ^=	*εντολή* -> *υπό_συνθήκη* | *βρόχος* | *ανάθεση* *υπό_συνθήκη* - > **if** *συνθήκη* **then** *εντολή* *βρόχος* -> **while** *συνθήκη* **do** *εντολή* *ανάθεση* -> **θέσε το** όνομα **ίσο με** *έκφραση* *έκφραση* -> όνομα *τελεστής* όνομα *τελεστής* -> +|- *συνθήκη* -> όνομα *σχέση* αριθμός *σχέση* -> = | ^=
Νέα μορφή γραμματικής	Τμήμα συντακτικού απλής γλώσσας προγραμματισμού

Σχήμα 6.4.5 Νέα μορφή Γραμματικής 6.3.2

Το αντίστοιχο συντακτικό δένδρο του σχήματος 6.4.4 που παράγει το εργαλείο που προαναφέραμε φαίνεται στο σχήμα 6.4.6.



Σχήμα 6.4.6 Συντακτικό δένδρο εντολής απλής γλώσσας προγραμματισμού που παράγεται από το εκπαιδευτικό εργαλείο

Κατασκευή Συντακτικών δένδρων

Το συντακτικό δένδρο ενός έγκυρου προγράμματος δεν είναι άλλο από τα τερματικά σύμβολα που συνιστούν το πρόγραμμα και μια ρίζα που είναι η ανώτερη συντακτική κατηγορία ***πρόγραμμα***. Η συντακτική αυτή κατηγορία περιλαμβάνει οποιαδήποτε άλλη και όλα τα συντακτικά δένδρα ξεκινάνε από αυτήν ως ρίζα. Η βασική εργασία του συντακτικού αναλυτή είναι να κατασκευάσει το συντακτικό δένδρο από το πηγαίο πρόγραμμα ή πιο σωστά από την σειρά των λεξικών κουπονιών που έχει παράγει ο λεξικός αναλυτής. Υπάρχουν πολλοί αλγόριθμοι γι' αυτό τον σκοπό, καθένας σχεδιασμένος με διαφορετικούς στόχους. Μερικοί είναι κατάλληλοι για γλώσσες με συγκεκριμένο συντακτικό του οποίου οι παραγωγές ικανοποιούν ορισμένους περιορισμούς. Άλλοι σχεδιάζονται για να μειώσουν τον χρόνο της ανάλυσης, άλλοι για να μειώσουν τις απαιτήσεις μνήμης, ενώ άλλοι για να παρέχουν εξελιγμένες δυνατότητες διαχείρισης λαθών. Ο αλγόριθμος που επιλέγεται για έναν μεταφραστή στηρίζεται στο συντακτικό της εκάστοτε γλώσσας και στους στόχους του κατασκευαστή.
Παρά την πληθώρα των επιλογών, υπάρχουν δύο βασικές προσεγγίσεις για την συντακτική ανάλυση ενός προγράμματος, την δημιουργία δηλαδή του συντακτικού δένδρου που αντιστοιχεί στο πηγαίο πρόγραμμα.

Κατασκευή από-κάτω -προς-τα-πάνω

Η πρώτη, που ονομάζεται **από-κάτω-προς-τα-πάνω** (bottom-up), ξεκινά από τα φύλλα του δένδρου και δουλεύει προς την ρίζα (υπενθυμίζουμε ότι στη επιστήμη των υπολογιστών τα δένδρα μεγαλώνουν ανάποδα, δηλαδή η ρίζα είναι πάνω από τα φύλλα). Ο αναλυτής προσπαθεί να συνδυάσει γειτονικά κουπόνια του πηγαίου προγράμματος ώστε να σχηματίσει στοιχεία μιας συντακτικής κατηγορίας. Στη συνέχεια άλλα γειτονικά κουπόνια ή και συντακτικές κατηγορίες συνδυάζονται σε στοιχεία πιο

πολύπλοκων κατηγοριών, και ούτω καθεξής. Η ανάλυση ολοκληρώνεται όταν διαδοχικοί συνδυασμοί οδηγούν στην αναγνώριση όλου του πηγαίου προγράμματος ως μέλος της κατηγορίας πρόγραμμα. Βέβαια σε κάθε στάδιο μπορεί να είναι δυνατοί διάφοροι συνδυασμοί, ωστόσο αν δεν υπάρχουν ασάφειες μόνο ένας τελικά θα οδηγήσει σε πλήρες συντακτικό δένδρο. Ο συντακτικός αναλυτής είναι δυνατόν να δοκιμάζει διάφορους ανεπιτυχείς συνδυασμούς πριν καταλήξει σ' αυτόν που του επιτρέπει να συνεχίσει. Η διαδικασία της ελεγχόμενης δοκιμής και λάθους είναι μια περίπτωση οπισθοδρόμησης (back-tracking) όπου όταν η διερεύνηση ενός συνδυασμού αποδειχθεί ανεπιτυχής ο αναλυτής επιστρέφει για την διερεύνηση άλλων συνδυασμών.

Παράδειγμα ανάλυσης από-κάτω-προς-τα-πάνω

Ας δώσουμε ένα παράδειγμα ανάλυσης από-κάτω-προς-τα-πάνω βασισμένο στην καταχώρηση του τηλεφωνικού καταλόγου του σχήματος 6.4.3. Ο αναλυτής ίσως αρχικά αναγνωρίσει ότι το Σιδηρόπουλος μπορεί να είναι επώνυμο και το Ιωάννης μπορεί να είναι κύριο όνομα, και γι' αυτό να προσπαθήσει να τα συνδυάσει σε ένα όνομα_προσώπου. Ο συνδυασμός είναι μεν σωστός, αλλά οδηγεί σε αδιέξοδο, αφού δεν υπάρχει κανόνας που να συνδυάζει το Β. που ακολουθεί να συνδυαστεί με το όνομα_προσώπου. Ο αναλυτής κατά συνέπεια οπισθοδρομεί και δοκιμάζει έναν διαφορετικό συνδυασμό, έως ότου συνδυάσει τα τέσσερα σύμβολα Σιδηρόπουλος Ιωάννης Β. στην συντακτική κατηγορία όνομα_προσώπου. Συνεχίζει με παρόμοια διαδικασία συνδυάζοντας τα υπόλοιπα σύμβολα.

Κατασκευή από-πάνω -προς-τα-κάτω

Η άλλη βασική μέθοδος ανάλυσης είναι η ***από-πάνω-προς-τα-κάτω*** (top-down). Όπως υποδηλώνει και το όνομα, η μέθοδος αρχίζει με την ρίζα του δένδρου και προχωράει προς τα φύλλα. Ο αναλυτής υποθέτει ότι το πηγαίο πρόγραμμα είναι σωστό και προσπαθεί να επιβεβαιώσει την υπόθεση του αναζητώντας τμήματα του προγράμματος που ανήκουν σε συντακτικές κατηγορίες οι οποίες παρουσιάζονται στο πρώτο επίπεδο του δένδρου. Η αναγνώριση τέτοιων τμημάτων στο πρώτο επίπεδο, απαιτεί αναζήτηση για τμήματα στο δεύτερο επίπεδο, και αυτό με την σειρά του αναζήτηση στο τρίτο επίπεδο, κ.ο.κ.

Παράδειγμα ανάλυσης από-πάνω -προς-τα-κάτω

Για παράδειγμα η ανάλυση του προγράμματος στο Σχήμα 6.4.2 ξεκινάει με την υπόθεση ότι η καταχώρηση είναι σωστή. Η επαλήθευσή της απαιτεί αναζήτηση ενός ονόματος προσώπου, μιας διεύθυνσης και ενός αριθμού με ενδιάμεσα κόμματα, τα συντακτικά στοιχεία στο πρώτο επίπεδο του δένδρου. Στη συνέχεια, η αναγνώριση ονόματος προσώπου απαιτεί αναζήτηση ενός κύριου ονόματος και μικρών ονομάτων. Εάν μπορούν να βρεθούν όλες οι απαραίτητες κατηγορίες και τα σύμβολα, τότε η καταχώρηση είναι πράγματι σωστή και το συντακτικό δένδρο έχει κατασκευαστεί κατά την διάρκεια της αναζήτησης. Σημειώστε ότι ίσως χρειαστεί κάποια οπισθοδρόμηση: για παράδειγμα κατά την αναζήτηση ενός ονόματος ο αναλυτής ίσως αναζητήσει ένα κύριο όνομα αλλά πιθανότατα να γυρίσει πίσω και να ψάξει για ένα αρχικό.

Επιλογές Σχεδιασμού Γλωσσών και Συντακτικών Κατηγοριών

Όπως αναφέρθηκε, τόσο η από-κάτω-προς-τα-πάνω όσο και η από πάνω-προς-τα-κάτω ανάλυση μπορεί να απαιτήσουν οπισθοδρόμηση, που είναι σημαντικός

παράγοντας καθυστέρησης του προγράμματος συντακτικής ανάλυσης. Εάν η γρήγορη ανάλυση είναι ένας από τους χαρακτηριστικούς στόχους στο σχεδιασμό της γλώσσας τότε η οπισθοδρόμηση πρέπει και είναι δυνατόν να αποφεύγεται. Μπορεί να επιτευχθεί με την σχεδίαση της γλώσσας αλλά και των συντακτικών κανόνων με τέτοιο τρόπο ώστε σε κάθε επίπεδο του δένδρου η σωστή επιλογή συντακτικών κατηγοριών να φαίνεται από τα αμέσως επόμενα σύμβολα του προγράμματος. Ο αναλυτής τότε χρειάζεται να κοιτάξει μόνο λίγα σύμβολα μπροστά για να αποφασίσει ποια επιλογή θα ακολουθήσει. Υπάρχουν γλώσσες, όπως η Pascal, που έχουν σχεδιαστεί έτσι ώστε ένα σύμβολο να είναι αρκετό.
Για παράδειγμα, το συντακτικό της υποθετικής μας γλώσσας 6.3.2 και 6.4.5 έχει οριστεί έτσι ώστε ο τύπος κάθε εντολής να φαίνεται από το πρώτο του σύμβολο (if, while, θέσε). Κατά συνέπεια κατά την προσπάθεια για την ανάλυση εντολής ο αναλυτής αρκεί να διαπιστώσει πιο είναι το πρώτο σύμβολο για να αποφανθεί πια από τις τρεις επιλογές πρέπει να επιλέξει.

Άλλοι σχεδιαστικοί περιορισμοί

Το κόστος από την αποφυγή της οπισθοδρόμησης είναι οι πρόσθετοι περιορισμοί στη γλώσσα και στο τρόπο που περιγράφονται οι συντακτικές κατηγορίες. Οι στόχοι της σχεδίασης θα καθορίσουν και τις τελικές επιλογές. Το εργαλείο διερεύνησης συντακτικών ορισμών είναι ένας συντακτικός αναλυτής από το πάνω-προς-τα-κάτω χωρίς οπισθοδρόμιση, γι' αυτό και απαιτεί ορισμένο τρόπο περιγραφής συντακτικών κατηγοριών. Αν διαπιστώσει ότι οι συντακτικές κατηγορίες δεν είναι αποδεκτού τύπου μπορεί να τις μετασχηματίσει σε αποδεκτό τύπο. Οι κανόνες μετατροπής είναι πέραν των στόχων του μαθήματος, όπως επίσης και ο τρόπος κατασκευής του συντακτικού αναλυτή. Εκτός από την λειτουργία εμφάνισης των συμβόλων και την κατασκευή συντακτικών δένδρων, μπορείτε να δοκιμάσετε τον έλεγχο ορθότητας προγραμμάτων (κουμπί parse), όπου μπορείτε να δείτε και το αντίστοιχο συντακτικό δένδρο που έχει κατασκευάσει ο αναλυτής.

Συντακτικά λάθη

Μέχρι τώρα θεωρήσαμε ότι το πηγαίο πρόγραμμα είναι συντακτικά σωστό. Όπως όμως κάθε προγραμματιστής γνωρίζει, τέτοιες υποθέσεις είναι αναληθείς, γι' αυτό ο μεταγλωττιστής πρέπει να είναι ικανός να αντιμετωπίσει οποιαδήποτε συντακτικά λάθη. Η ανίχνευση ενός λάθους είναι σχετικά απλή, αφού η κατασκευή του συντακτικού δένδρου θα αποτύχει. Παρ' όλα αυτά, μπορεί να μην είναι φανερό το γιατί προέκυψε η αποτυχία, αφού το λάθος μπορεί να οφείλεται σε μέρος του δένδρου που έχει ήδη κατασκευαστεί.

Παράδειγμα Συντακτικού λάθους

Αν θεωρήσουμε ότι η καταχώρηση του τηλεφωνικού καταλόγου:

Τζώνου Μαρία, Χέϋδεν 42, 7223456

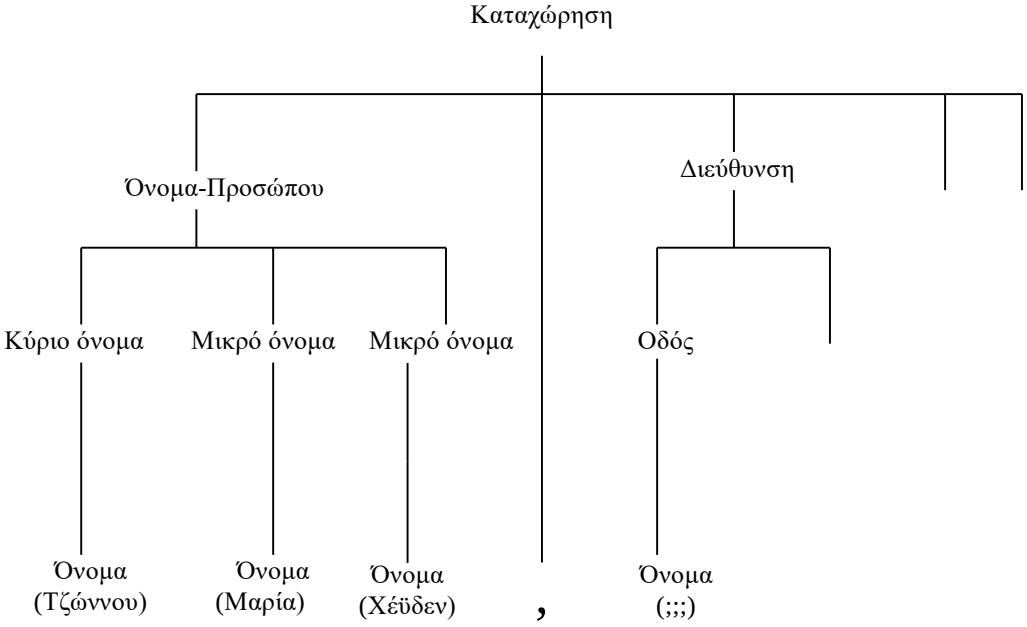
είναι λανθασμένα γραμμένη ώς

Τζώνου Μαρία Χέϋδεν, 42, 7223456

όπου δηλαδή το κόμμα αντί να είναι μεταξύ των συμβόλων Μαρία και Χέϋδεν είναι μεταξύ Χέϋδεν και 42. Έτσι τα σύμβολα Τζώνου, Μαρία και Χέϋδεν θα σχηματίσουν την συντακτική κατηγορία όνομα_προσώπου. Επίσης αναγνωρίζεται το κόμμα και το λάθος θα βρεθεί όταν ο αναλυτής αναγνωρίσει το 42 ως αριθμό, ενώ για να συνεχίσει θα έπρεπε να βρει όνομα_οδού. Θα έχει δηλαδή σχηματιστεί το μερικό δένδρο (θεωρώντας

ανάλυση από πάνω προς τα κάτω) όπως στο σχήμα 6.4.7. Για ευκολία επαναλαμβάνουμε εδώ τους συντακτικούς κανόνες της καταχώρησης του 6.3.1.

καταχώρηση -> *όνομα_προσώπου* , *διεύθυνση* , αριθμός
όνομα_προσώπου -> *επίθετο μικρό_όνομα* { *μικρό_όνομα* }
επίθετο -> όνομα
μικρό_όνομα -> *κύριο_όνομα* | *αρχικό*
κύριο όνομα -> όνομα
αρχικό -> *κεφαλαίο_γράμμα*.
διεύθυνση -> όνομα_οδού αριθμός
όνομα_οδού -> όνομα
κεφαλαίο_γράμμα -> Α | Β | … | Ω



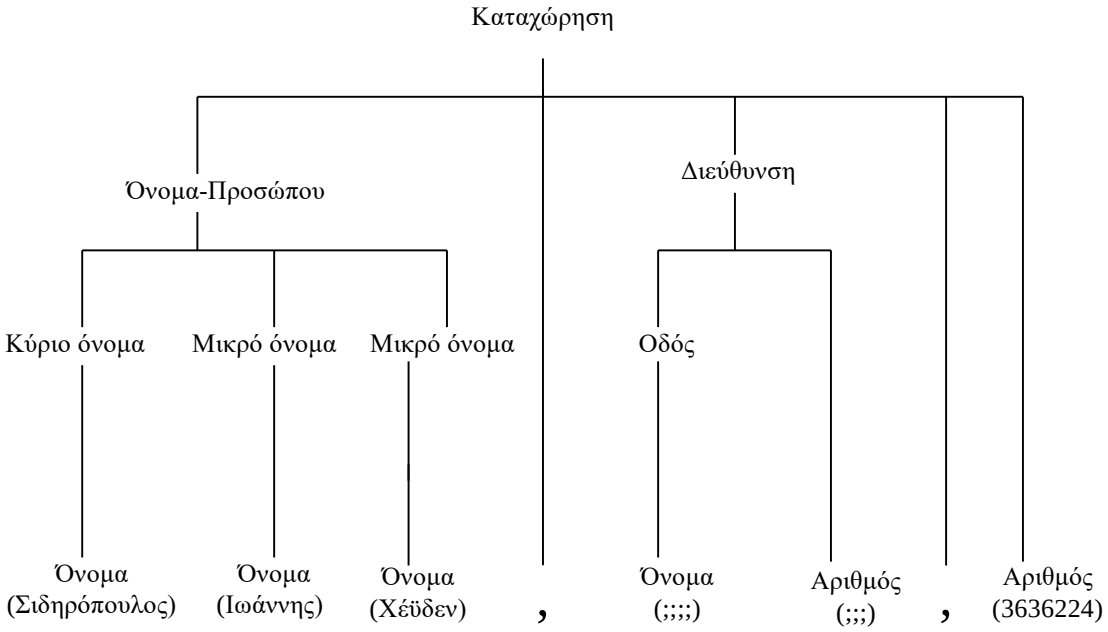
Σχήμα 6.4.7 Το ημιτελές συντακτικό δένδρο της λανθασμένης καταχώρησης

Αν και το λάθος έχει συμβεί νωρίτερα δεν μπορεί να ανιχνευτεί.

Ανάκαμψη από λάθη

Όπως και στη λεξική ανάλυση υπάρχουν περιπτώσεις που η ανίχνευση ενός λάθους δεν πρέπει να εμποδίζει την παραπέρα ανάλυση. Σ' αυτές τις περιπτώσεις ο συντακτικός αναλυτής μέσω της διαχείρισης λαθών πρέπει να αναφέρει το λάθος. Επίσης η διαχείριση λαθών πρέπει να παραλείψει όσο το δυνατόν λιγότερα σύμβολα και να επιτρέψει στον συντακτικό αναλυτή να συνεχίσει με την επόμενη συντακτική κατηγορία που μπορεί να αναγνωρίσει. Αυτή η διαδικασία επιτρέπει τον εντοπισμό και άλλων λαθών, ανεξάρτητων από τα ήδη ανιχευμένα, αλλά το συντακτικό δένδρο που παράγεται είναι ημιτελές και δεν έχει νόημα η παραγωγή ενδιάμεσου κώδικα. Στο παράδειγμα λάθους καταχώρησης θα μπορούσε ο συντακτικός αναλυτής να αγνοήσει την διεύθυνση

και να συνεχίσει μετά το κόμμα και τον αριθμό τηλεφώνου, σχηματίζοντας το ημιτελές δένδρο του 6.4.8.



Σχήμα 6.4.8 Το ημιτελές συντακτικό δένδρο της λανθασμένης καταχώρησης με ανάκαμψη από το λάθος

Άσκηση Αυτοαξιολόγησης

1. Χρησιμοποιήστε το εργαλείο ανάλυσης γραμματικών (πατήστε την URL που ακολουθεί http://www.di.uoa.gr/~compiler/javatool/English/Sun/tutor.htm) για να αναγνωρίσετε τις εντολές των οποίων τα συντακτικά δένδρα κατασκευάσατε στις προηγούμενες ασκήσεις αυτοαξιολόγησης. Δείτε τα συντακτικά δένδρα που κατασκευάζονται.
   Απάντηση
   Στο εργαλείο πατήστε το πλήκτρο Parse String και δώστε στο παράθυρο τις εντολές (προσοχή όλα τα σύμβολα της γλώσσας πρέπει να χωρίζονται με ένα κενό τουλάχιστον). Πατήστε το πλήκτρο Parse και θα πάρετε μήνυμα επιτυχούς συντακτικής αναγνώρισης. Αν πάρετε λάθος ελέγξτε την συμβολοσειρά εισόδου. Πατήστε το πλήκτρο Tree για να δείτε το συντακτικό δένδρο που αναγνωρίστηκε. Σε περίπτωση λάθους συμβολοσειράς μπορείτε να δείτε το ημιτελές αναγνωρισμένο δένδρο ώστε να προσδιορίσετε το λάθος.

6.4.5 Σημασιολογική Ανάλυση

Εισαγωγή

Κατά τη σημασιολογική ανάλυση το πρόγραμμα ελέγχεται για σημασιολογικά λάθη και συλλέγει πληροφορίες για την επόμενη φάση της παραγωγής κώδικα. Η συντακτική ανάλυση έχει αναγνωρίσει την σωστή σειρά των συμβόλων και έχει κατασκευάσει το συντακτικό δένδρο ενός προγράμματος, χωρίς όμως να έχει προσδώσει στα σύμβολα

κάποια σημασία. Για παράδειγμα κατά την συντακτική ανάλυση το «+» είναι απλά ένα σύμβολο. Η σημασία του ως τελεστής της πρόσθεσης ή ως τελεστής ένωσης συνόλων (όπως ισχύει στην Pascal) προσδίδεται κατά την σημασιολογική ανάλυση.

Έλεγχος Τύπων

Σημαντικό μέρος της σημασιολογικής ανάλυσης είναι ο έλεγχος των τύπων των μεταβλητών και δεδομένων (type checking). Ο τύπος μιας μεταβλητής (integer, real, boolean, char, κλπ) είναι μέρος της σημασίας της. Ο τύπος των μεταβλητών που συμμετέχουν σε μια πράξη καθορίζει αν η πράξη είναι σημασιολογικά σωστή ή λάθος. Π.χ. στη Pascal δεν επιτρέπεται η πρόσθεση integer και character μεταβλητής. Ο σημασιολογικός αναλυτής ελέγχει ότι οι μεταβλητές ενός τελεστή είναι τύπου που επιτρέπεται από τις προδιαγραφές της γλώσσας του πηγαίου κώδικα.

Παράδειγμα ελέγχου τύπων

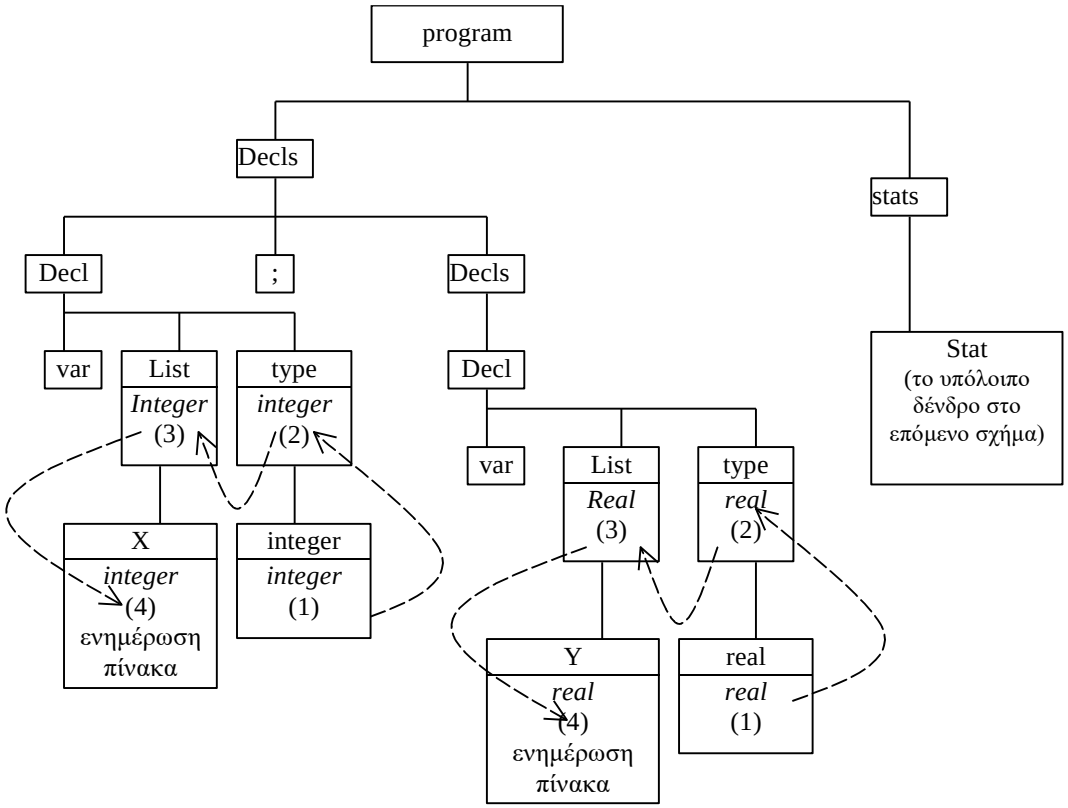
Ας εξετάσουμε το πρόγραμμα που ανήκει στη γλώσσα που περιγράφεται στη άσκηση 1 της συντακτικής ανάλυσης της υποενότητας 6.4.4.

```
var X integer; var Y real;
if X^=1 then put Y equal to X + Y * 60
```

Το συντακτικό του δένδρο του ανωτέρω προγράμματος εμφανίζεται σε δύο υπο-δένδρα για ευκολία παρουσίασης. Το πρώτο υποδένδρο περιλαμβάνει όλες τις συντακτικές κατηγορίες των ορισμών των μεταβλητών και την γενική κατηγορία Stat, η οποία όμως αναλύεται σε άλλες συντακτικές κατηγορίες στο δεύτερο υπο-δένδρο.

Παράδειγμα (συνέχεια)

Στο πρώτο υπο-δένδρο οι κόμβοι που αναφέρονται στις συντακτικές κατηγορίες list, type και στα τερματικά X, Y, integer και real είναι σημειωμένοι με τα σημασιολογικά χαρακτηριστικά integer ή real και έχουμε προσθέσει κατευθυνόμενα βέλη από και προς αυτούς τους κόμβους.
Η σημασιολογική ανάλυση προσημειώνει τους κόμβους των συντακτικών δένδρων με σημασιολογικά χαρακτηριστικά που ορίζει η γλώσσα. Εξηγούμε αμέσως την σημείωση των κόμβων και τα κατευθυνόμενα βέλη παίρνοντας ως παράδειγμα τον ορισμό του X. Έχει ήδη αναγνωριστεί από τον συντακτικό αναλυτή ότι ο ορισμός «var X integer» είναι σωστός και έχει ήδη κατασκευαστεί το συντακτικό δένδρο. Το πρόβλημα που αντιμετωπίζουμε είναι ότι πρέπει να χαρακτηρίσουμε τη μεταβλητή X με τύπο integer. Η πληροφορία όμως υπάρχει στον τερματικό κόμβο integer και από εκεί πρέπει ο σημασιολογικός αναλυτής να την μεταβιβάσει στον τερματικό κόμβο X με την δέσμευση ότι η πληροφορία μπορεί μόνο να μεταβιβαστεί μέσω κόμβων γονέων ή παιδιών ή αδελφών. Έτσι από τον κόμβο integer, όπου δημιουργείται ο χαρακτηρισμός “integer” και γι’ αυτό υπάρχει η ένδειξη (1), μεταβιβάζεται στον γονέα του, κόμβος «type», με την ένδειξη (2), μετά στον αδελφό κόμβο «list» με ένδειξη (3) και τέλος στο παιδί του «X» με την ένδειξη (4). Τώρα μπορεί να ενημερωθεί ο πίνακας συμβόλων με τον τύπο της μεταβλητής X. Παρόμοια διαδικασία γίνεται για να χαρακτηρίσουμε το Y με τύπο real.



Σχήμα 6.4.9 Συντακτικό δένδρο με σημασιολογικές σημειώσεις

Όταν σημασιολογικός αναλυτής διαπιστώνει τον τύπο μιας μεταβλητής ενημερώνει το αντίστοιχο πεδίο τύπου του πίνακα συμβόλων. Μετά την αναγνώριση των τύπων των X και Y ο πίνακας συμβόλων έχει διαμορφωθεί ως εξής:

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Άλλα χαρακτηριστικά
1	X	integer	
2	Y	real	
3			

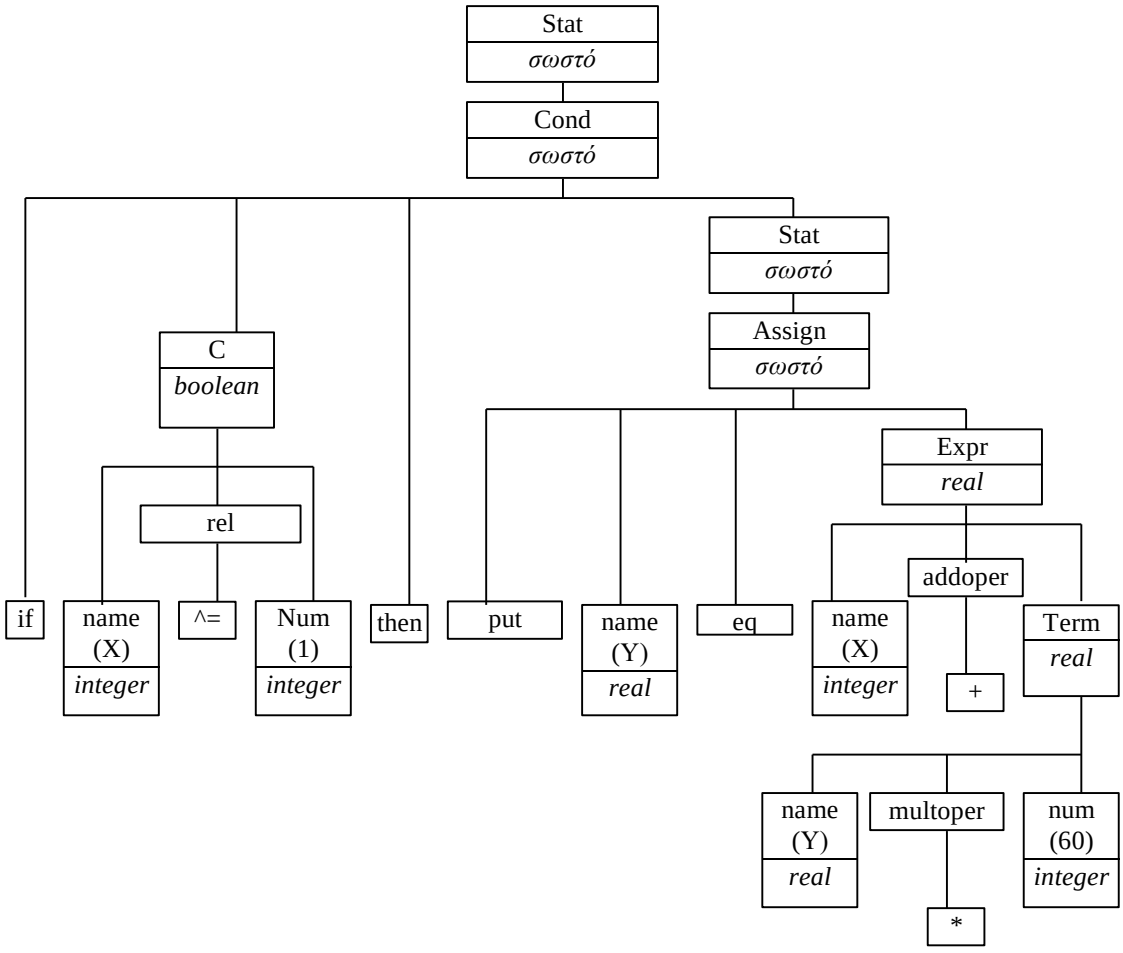
Παράδειγμα (συνέχεια)

Έχοντας διαπιστώσει τον τύπο των X και Y, ο σημασιολογικός αναλυτής συνεχίζει με τον έλεγχο της συμβατότητας των πράξεων και των αποδόσεων τιμών, όπως ορίζει η γλώσσα. Π.χ. Ελέγχει σημασιολογικούς κανόνες της μορφής: Το αποτέλεσμα της άθροισης δύο ακεραίων είναι ακέραιος, ενώ ενός ακεραίου και ενός πραγματικού είναι πραγματικός. Επίσης το “αποτέλεσμα” χαρακτηρίζεται ως λάθος, αν η πράξη δεν είναι ορισμένη. Π.χ η αν άθροιση ενός ακεραίου και ενός χαρακτήρα δεν ορίζεται.
Με τον ίδιο τρόπο που υπολογίστηκε ο τύπος των μεταβλητών, υπολογίζεται ο τύπος του αποτελέσματος μιας πράξης. Στο παράδειγμα μας ξεκινάμε χαρακτηρίζοντας τον τύπο των μεταβλητών και αριθμών των τερματικών. Ο τύπος των κόμβων X (integer) και Y (real) αρχικά ανακτάται από τον πίνακα συμβόλων. Ξεκινώντας λοιπόν από τους κόμβους που έχουν τις αρχικές πληροφορίες τύπων ελέγχουμε την εγκυρότητα των πράξεων και υπολογίζουμε τον τύπο του αποτελέσματος, που χαρακτηρίζει τώρα τον

κόμβο της συντακτικής κατηγορίας. Π.χ. έχοντας χαρακτηρίσει το Y ως real και το 60 ως integer, ο τύπος του αποτελέσματος Y*60 είναι real και χαρακτηρίζει τον αντίστοιχο κόμβο Term. Στο σχήμα 6.4. δεν έχουμε σχεδιάσει κατευθυνόμενα βέλη που να δείχνουν την σειρά υπολογισμού των τύπων, για δύο λόγους:

- για να μην επιβαρύνουμε το σχήμα και
- γιατί στην περίπτωση αυτή ο τύπος μιας συντακτικής κατηγορίας εξαρτάται μόνο από τους τύπους των συντακτικών κατηγοριών στα οποία αναλύεται και επομένως τα βέλη θα ταυτιζόντουσαν με τις ακμές του δένδρου.

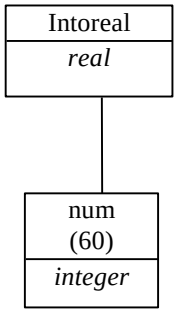
Στην συνέχεια ελέγχεται και υπολογίζεται η αμέσως πιο γενική συντακτική κατηγορία, που είναι η Expr και υπολογίζεται ο τύπος του με παρόμοιο τρόπο. Σε επίπεδο της assign εντολής ελέγχουμε την συμβατότητα αριστερού και δεξιού μέρους. Αν ισχύει δίνουμε τον ενδεικτικό τύπο «σωστό», αλλιώς «λάθος». Αν ο τύπος μιας εντολής είναι λάθος τότε όλες οι εντολές στις οποίες πιθανά να είναι εμφωλιασμένη παίρνουν τον τύπο «λάθος». Ο σημασιολογικός αναλυτής συνεχίζει μέχρι να χαρακτηρίσει την ανώτερη συντακτική κατηγορία program. Αν έχει χαρακτηριστεί «σωστή» τότε όλο το πρόγραμμα είναι σημασιολογικά σωστό.



Σχήμα 6.4.10 Το δεύτερο μέρος του συντακτικού προσημειωμένου δένδρου

Άλλες λειτουργίες του Σημασιολογικού Αναλυτή

Τέλος, η σημασιολογική ανάλυση φροντίζει για τις απαραίτητες μετατροπές τύπων όπως αυτές ορίζονται από τη γλώσσα. Π.χ. αν η πρόσθεση αριθμών και μεταβλητών real και integer επιτρέπεται, ο ακέραιος μετατρέπεται σε πραγματικό, με χρήση του τελεστή intoreal. Στο παράδειγμα μας ο σημασιολογικός αναλυτής θα μετατρέψει τον απλό κόμβο num (60) στο υποδένδρο



Σχήμα 6.4.11 Μετατροπή τύπου

Με την σημασιολογική ανάλυση ολοκληρώνεται το front-end μέρος του μεταγλωττιστή και συνεχίζουμε με την παραγωγή ενδιάμεσου κώδικα.

Ασκήσεις Αυτοαξιολόγησης

1. Αναπτύξτε την διαδικασία χαρακτηρισμού του Y με τύπο real. Βασιστείτε στο 6.4.9.
   Απάντηση
   Έχει ήδη αναγνωριστεί από τον συντακτικό αναλυτή ότι ο ορισμός «var Y real» είναι σωστός και έχει ήδη κατασκευαστεί το συντακτικό δένδρο. Το πρόβλημα που αντιμετωπίζουμε είναι ότι πρέπει να χαρακτηρίσουμε τη μεταβλητή Y με τύπο real. Η πληροφορία όμως υπάρχει στον τερματικό κόμβο real και από εκεί πρέπει ο σημασιολογικός αναλυτής να την μεταβιβάσει στον τερματικό κόμβο Y με την δέσμευση ότι η πληροφορία μπορεί μόνο να μεταβιβαστεί μέσω κόμβων γονέων ή παιδιών ή αδελφών. Έτσι από τον κόμβο real, όπου δημιουργείται ο χαρακτηρισμός “real” και γι’ αυτό υπάρχει η ένδειξη (1), μεταβιβάζεται στον γονέα του, κόμβος «type», με την ένδειξη (2), μετά στον αδελφό κόμβο «list» με ένδειξη (3) και τέλος στο παιδί του «X» με την ένδειξη (4). Τώρα μπορεί να ενημερωθεί ο πίνακας συμβόλων με τον τύπο της μεταβλητής Y.

2. Συμπληρώστε την σημασιολογική ανάλυση του δένδρου 6.4.10.
   Απάντηση
   Στο κείμενο αναπτύξαμε την σημασιολογική ανάλυση του υποδένδρου με ρίζα assign. Θα συμπληρώσουμε την ανάλυση στο υπόλοιπο δένδρο. Ο τύπος του X ανακτάται από τον πίνακα συμβόλων (integer). Ο τύπος του 1 είναι ήδη διαθέσιμος από τον λεξικό αναλυτή. Ελέγχεται η συμβατότητα και ο τύπος της σύγκρισης X^=1. Η συμβατότητα ισχύει και ο τύπος του αποτελέσματος της σύγκρισης είναι boolean που φυλάσσεται στον κόμβο c. Κατόπιν ελέγχεται η εντολή cond (if c then stat), η οποία είναι στην σωστή μορφή και ο τύπος της παίρνει την τιμή σωστό. Η ίδια τιμή τέλος μεταφέρεται και στον κόμβο stat.

6.4.6 Παραγωγή Ενδιάμεσου Κώδικα

Εισαγωγή

Στην εισαγωγή του 6.4 αναφέραμε ότι το *front end* ενός μεταγλωττιστή μεταφράζει ένα πρόγραμμα σε ενδιάμεσο κώδικα από τον οποίο το *back end* δημιουργεί κώδικα μηχανής. Αν και η κατασκευή ενδιάμεσου κώδικα δεν είναι απαραίτητη και ένα πρόγραμμα μπορεί να μεταφρασθεί άμεσα σε γλώσσα μηχανής υπάρχουν ορισμένα πλεονεκτήματα όταν δημιουργείται ενδιάμεσος κώδικας. Τετοια πλεονεκτήματα είναι:
α) Η δυνατότητα ανάπτυξης βελτιστοποιητή (optimizer) ανεξάρτητου από το back end.
β) Ενας μεταγλωττιστής μπορεί να γραφεί για διαφορετικές μηχανές με την προϋπόθεση ότι υπάρχει ένας back end για κάθε μηχανή.

Τα σημασιολογικά σημειωμένα συντακτικά δένδρα είναι μία μορφή ενδιάμεσου κώδικα. Στα επόμενα θα παρουσιάσουμε και μια άλλη μορφή ενδιάμεσου κώδικα, τον κώδικα τριών διευθύνσεων.Από ένα συντακτικά και σημασιολογικά σωστό πρόγραμμα θα δημιουργηθεί ένας ενδιάμεσος κώδικας, που μπορεί να έχει πολλές μορφές. Η ενδιάμεση αυτή απεικόνιση μπορεί να θεωρηθεί σαν ένα πρόγραμμα για μια αφηρημένη μηχανή. Πρέπει να έχει δύο χαρακτηριστικά: να παράγεται εύκολα από το σημασιολογικά σημειωμένο συντακτικό δένδρο και να μετατρέπεται εύκολα στη τελική γλώσσα μηχανής.

Εντολές τριών διευθύνσεων

Μια διαδεδομένη μορφή ενδιάμεσου κώδικα είναι αυτή των "τριών διευθύνσεων", όπου όλες οι εντολές χρησιμοποιούν το πολύ τρεις τελεστέους, δηλαδή μια πράξη μεταξύ των δύο εξ αυτών και μια απόδοση τιμής στον τρίτο. Όλες οι παραστάσεις και εντολές του πηγαίου προγράμματος πρέπει να μετασχηματιστούν σε μια σειρά από απλές τέτοιες εντολές. π.χ. Η εντολή απόδοσης τιμής

Put Y eq X+Y*intoreal(60)

Δεν είναι του τύπου κώδικα τριών διευθύνσεων αφού γίνονται τρεις πράξεις (intoreal, *, +) μεταξύ τριών «διευθύνσεων» (X, Y, 60) και μια απόδοση. Η ανωτέρω εντολή μπορεί όμως να μετατραπεί σε ισοδύναμες εντολές κώδικα τριών διευθύνσεων:

t_1 := intoreal(60)
t_2 := Y * t_1
t_3 := X + t_3
Y := t_3

όπου τα t1, t2, t3 είναι προσωρινές ενδιάμεσες βοηθητικές μεταβλητές.
Το πηγαίο πρόγραμμα μπορεί να μετατραπεί σε κώδικα τριών διευθύνσεων από το σημασιολογικά σημειωμένο συντακτικό δένδρο. Συνήθως αρκεί να δώσουμε προσωρινά ονόματα στους εσωτερικούς κόμβους του συντακτικού δένδρου και να αναλύσουμε τις παραστάσεις και εντολές σε κώδικα τριών διευθύνσεων χρησιμοποιώντας αυτά τα προσωρινά ονόματα, τα οποία πρέπει και αυτά να εισαχθούν στον πίνακα συμβόλων. Π.χ Οι προσωρινές μεταβλητές t1, t2, t3 αντιστοιχούν στους κόμβους «intoreal», «term» και «expr» των δένδρων των σχημάτων 6.4.10 και 6.4.11.

Μετατροπή εντολών σε κώδικα τριών διευθύνσεων

Όλες οι εντολές μπορούν να μετασχηματιστούν σε κώδικα τριών διευθύνσεων. Η γλώσσα που χρησιμοποιούμε ως παράδειγμα περιέχει εκτός από απόδοση τιμής, που ήδη έχουμε μετατρέψει σε κώδικα τριών διευθύνσεων, τις εντολές: **If** συνθήκη **then**

εντολή και **while** συνθήκη **do** εντολή που μετασχηματίζονται σε κώδικα τριών διευθύνσεων ως εξής:

If συνθήκη then εντολή	If συνθήκη=false goto ετικέτα Εντολή Ετικέτα:
while συνθήκη do εντολή	Ετικέττα1: If συνθήκη=false goto ετικέτα 2 Εντολή Goto ετικέτα1 Ετικέτα 2:

Παράδειγμα
Οι εντολές κώδικα τριών διευθύνσεων της εντολής

if X^=1 then put Y equal to X + Y * 60

με βάση τα ανωτέρω είναι

t4 := X ^=1
If t4= false goto ετικέττα
t_1 := intoreal(60)
t_2 := Y * t_1
t_3 := X + t_2
Y := t_3
Ετικέτα

Και ο πίνακας συμβόλων με τις τρεις νέες προσωρινές μεταβλητές έχει διαμορφωθεί ως εξής (σχήμα 6.4.12):
Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Άλλα χαρακτηριστικά
1	X	integer	
2	Y	real	
3	T1	Real	
4	T2	Real	
5	T3	Real	
6	T4	boolean	

Σχήμα 6.4.12 Ο Πίνακας Συμβόλων με πεδίο Τύπου μεταβλητών

Θέσεις μνήμης μεταβλητών

Όταν εξετάζονται οι ορισμοί των μεταβλητών μπορεί να υπολογιστεί και ο χώρος αποθήκευσης τους. Για κάθε τοπικό όνομα δημιουργείται μια είσοδος στο πίνακα συμβόλων με την σχετική διεύθυνση που αποθηκεύεται η τιμή του. Όταν δημιουργούνται οι διευθύνσεις πρέπει να λαμβάνεται υπόψη η τελική μηχανή για την οργάνωση της αποθήκευσης, όπως ο απαιτούμενος χώρος για τιμές τύπου *integer* (π.χ 4 bytes), *real* (πχ 8 bytes), boolean (π.χ.1 byte), κλπ.
Για τον υπολογισμό του χώρου μνήμης χρησιμοποιούμε μια μεταβλητή *offset* για να μας δείξει την επόμενη διαθέσιμη σχετική διεύθυνση μνήμης. Αρχικά η τιμή της *offset* είναι 0, που δηλώνει ότι η πρώτη μεταβλητή βρίσκεται στην αρχή του σχετικού χώρου μνήμης. Για κάθε όνομα μεταβλητής στο πίνακα συμβόλων εισάγεται η τρέχουσα τιμή της *offset* και η τιμή της αυξάνει όσο είναι το μέγεθος που απαιτείται για το όνομα αυτό. Στον

κατωτέρω πίνακα συμβόλων 6.4.13 για κάθε μεταβλητή που χρησιμοποιείται ενημερώνεται το αντίστοιχο πεδίο Θέση μνήμης.

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Θέση μνήμης	Άλλα χαρακτηριστικά
1	X	integer	0	
2	Y	real	4 (= 0+4)	
3	T1	Real	12 (=4+8)	
4	T2	Real	20 (=12+8)	
5	T3	Real	28 (=20+8)	
6	T4	Boolean	36 (=28+8)	

Σχήμα 6.4.13 Ο Πίνακας Συμβόλων με πεδίο σχετικής Θέσης μνήμης μεταβλητών

Έτσι η X βρίσκεται στην θέση 0 και καταλαμβάνει 4 bytes. Η Y βρίσκεται στη θέση 4 (=0+4 bytes του ακεραίου X). Η t1 στη θέση 12 (=4+8 bytes του πραγματικού Y), κ.ο.κ.

Άσκηση Αυτοαξιολόγησης

Μετατρέψτε την εντολή

If X ^= 1 then put Y eq X + Y

Σε κώδικα τριών διευθύνσεων. Δώστε επίσης τον πίνακα συμβόλων και υπολογίστε τις θέσεις μνήμης. Θεωρήστε ότι X και Y ακέραιοι.

Απάντηση

Οι εντολές κώδικα τριών διευθύνσεων της εντολής

if X^=1 then put Y eq X + Y

είναι

t2 := X ^=1
If t2= false goto ετικέττα
T_1 := X + Y
Y := t_1
Ετικέτα

Και ο πίνακας συμβόλων με το αντίστοιχο πεδίο Θέση μνήμης συμπληρωμένο

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Θέση μνήμης	Άλλα χαρακτηριστικά
1	X	integer	0	
2	Y	integer	4 (= 0+4)	
3	T1	integer	8 (=4+4)	
4	T2	Boolean	12 (=8+4)	

6.4.7 Βελτιστοποίηση Ενδιάμεσου Κώδικα

Εισαγωγή

Η φάση της βελτιστοποίησης έχει στόχο τη δημιουργία κώδικα που θα έχει σαν αποτέλεσμα κώδικα γλώσσας μηχανής που θα απαιτεί λιγότερο χώρο και θα τρέχει γρηγορότερα. Αυτό επιτυγχάνεται με μετασχηματισμούς του ενδιάμεσου κώδικα που

υπερβολικά ονομάζονται “βελτιστοποιήσεις” (optimisations), αφού συνήθως δεν υπάρχουν εγγυήσεις ότι ο νέος κώδικας είναι ο βέλτιστος.
Στη φάση αυτή μας ενδιαφέρουν βελτιστοποιήσεις ανεξάρτητες από αρχιτεκτονικές, που βελτιώνουν τον τελικό κώδικα χωρίς να λαμβάνουν υπ’ όψη τους την τελική αρχιτεκτονική.

Παράδειγμα

Μερικές βελτιώσεις είναι απλές. Για παράδειγμα ένας βελτιωμένος κώδικας τριών διευθύνσεων από αυτόν που είχε παραχθεί αυτόματα είναι ο επόμενος:

```
t4 := X ^=1
If t4= false goto ετικέτα
t2 := Y * 60.0
Y := X + t2
Ετικέτα
```

όπου έχουν γίνει τρεις βελτιώσεις:

1. Έχει γίνει κατευθείαν η μετατροπή του ακεραίου 60 σε πραγματικό 60.0, επομένως βελτιώσαμε το χρόνο μετατροπής της κατά την εκτέλεση. Η εντολή t1:= intoreal(60) έγινε κατ’ αρχάς t1:=60.0
2. Ενοποιήθηκαν οι εντολές t1:=60.0 και t_2 := Y * t1 στην t2:=Y*60.0, καταργώντας την μεταβλητή t1
3. Ενοποιήθηκαν οι εντολές t_3 := X + t2 και Y:=t3 στην Y:=X+ t2, καταργώντας την μεταβλητή t3

Ως συνέπεια της κατάργησης των μεταβλητών t1 και t2 είναι και η απάλειψη τους από τον πίνακα συμβόλων και η μικρότερη απαίτηση σε μνήμη.

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Θέση μνήμης	Άλλα χαρακτηριστικά
1	X	integer	0	
2	Y	real	4 (= 0+4)	
3	T2	Real	12 (=4+8)	
4	T4	Boolean	20 (=12+8)	

Σχήμα 6.4.14 Ο Πίνακας Συμβόλων μετά την απάλειψη μεταβλητών

Λίγα λόγια για τις βελτιστοποιήσεις

Υπάρχει μεγάλη διαφοροποίηση στη έκταση βελτιστοποίησης διαφόρων μεταγλωττιστών. Σ’ αυτούς που κάνουν τις περισσότερες βελτιστοποιήσεις, και λέγονται “μεταγλωττιστές βελτιστοποίησης”, σημαντικό ποσοστό του χρόνου τους αφιερώνεται στην βελτιστοποίηση. Υπάρχουν όμως απλές βελτιστοποιήσεις, σαν αυτές που είδαμε πιο πάνω, που βελτιώνουν σημαντικά το χρόνο εκτέλεσης χωρίς να καθυστερούν την μεταγλώττιση υπερβολικά.
Εκτός από τις βελτιώσεις ανεξάρτητα από μηχανή υπάρχουν και βελτιώσεις που στηρίζονται σε ειδικά χαρακτηριστικά των τελικών μηχανών, στις οποίες δεν αναφερθούμε περισσότερο.

Άσκηση Αυτοαξιολόγησης

Βελτιστοποιήστε τις εντολές κώδικα τριών διευθύνσεων της εντολής

if X^=1 then put Y eq X + Y

της προηγούμενης υποενότητας. Υπενθυμίζουμε τον κώδικα

t2 := X ^=1
If t2= false goto ετικέττα
T_1 := X + Y
Y := t_1
Ετικέτα

Και τονο πίνακα συμβόλων με το αντίστοιχο πεδίο Θέση μνήμης συμπληρωμένο

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Θέση μνήμης	Άλλα χαρακτηριστικά
1	X	integer	0	
2	Y	integer	4 (= 0+4)	
3	T1	integer	8 (=4+4)	
4	T2	Boolean	12 (=8+4)	

Απάντηση

Ο βελτιστοποιημένος κώδικας τριών διευθύνσεων είναι

t2 := X ^=1
If t2= false goto ετικέττα
Y := X + Y
Ετικέτα

Όπου καταργήσαμε την μεταβλητή t1, και ο πίνακας συμβόλων με το αντίστοιχο πεδίο Θέση μνήμης συμπληρωμένο είναι

Πίνακας Συμβόλων

Θέση	Όνομα	Τύπος	Θέση μνήμης	Άλλα χαρακτηριστικά
1	X	integer	0	
2	Y	integer	4 (= 0+4)	
3	T2	Boolean	8 (=4+4)	

6.4.8 Παραγωγή Τελικού Κώδικα

Εισαγωγή

Η τελευταία φάση του μεταγλωττιστή είναι η παραγωγή τελικού κώδικα είτε σε γλώσσα μηχανής είτε σε Assembly. Επιλέγονται θέσεις μνήμης για κάθε μεταβλητή που χρησιμοποιεί το πρόγραμμα.

Παράδειγμα

Θα παράγουμε από τον κώδικα τριών διευθύνσεων του 6.4.7 κώδικα μηχανής DLX του κεφαλαίου 4. Θα μπορέσετε έτσι να «Τρέξετε» τον τελικό κώδικα στο προσομοιωτή του DLX που χρησιμοποιήσατε στο κεφάλαιο 4. Κατ' αρχήν από τον πίνακα συμβόλων και

τις σχετικές θέσεις μνήμες των μεταβλητών ορίζουμε τα μνημονικά ονόματα των θέσεων μνήμης και τον χώρο διευθύνσεων. Για παράδειγμα από τον πίνακα συμβόλων της προηγούμενης υποενότητας 6.4.7 δημιουργούμε τον εξής χώρο διευθύνσεων:

Χώρος Διευθύνσεων

0-offset of X	4- offset of Y	12- offset of t2	20- offset of t4

Κατόπιν ο ενδιάμεσος κώδικας μετατρέπεται σε ισοδύναμη ακολουθία εντολών μηχανής DLX. Για παράδειγμα από τον κώδικα τριών διευθύνσεων της προηγούμενης υποενότητας 6.4.7 δημιουργούμε τον πιο κάτω κώδικα (μεσαία στήλη του πίνακα). Για διευκόλυνσή σας στην πρώτη στήλη φαίνεται ο κώδικας τριών διευθύνσεων και στην τρίτη οι επεξηγήσεις, σε αντιστοίχηση με τον κώδικα σε DLX.

Κώδικας τριών διευθύνσεων	**Κώδικας DLX**		**Επεξηγήσεις**
			Αρχικοποίηση R1-R5, R0=0
	100	ADD R1, R0, #0	R1 holds address of X (0)
	101	ADD R2, R1, #4	R2 holds address of Y (0+ 4)
	102	ADD R3, R2, #8	R3 holds address of t2 (4+8)
	103	ADD R4, R3, #8	R4 holds address of t4 (12+8)
	104	ADD R5, R0, #60.0	R5 holds constant 60.0
t4 := X ^=1	105	LD R10, 0(R1)	Load R10 with value of X
	106	SUB R11, R10, #1	R11 holds result X-1
If t4= false goto ετικέτα	107	BEQZ R11, 112	Branch to end if R11(=X-1)=0
t_2 := Y * 60.0	108	LD R12, 0(R2)	Load R12 with value of Y
	109	MULT R13, R12, R5	R13 holds result t2= Y * 60.0
Y := X + t_2	110	ADD R14, R10, R13	R14 hold result X+t2
	111	SD 0(R2), R14	Store R13 to Y
Ετικέτα	112		End of program

Για να μπορέσετε να «τρέξετε» τον ανωτέρω κώδικα στον προσομοιωτή του κεφ. 4 πρέπει να αντικαταστήσετε την εντολή MULT στην θέση 109 με εντολές ADD και επανάληψη, γιατί η MULT δεν υποστηρίζεται από το εργαλείο. Μπορούμε να θεωρήσουμε ότι η εντολή t_2 := Y * 60.0 μεταφράζεται σε κώδικα τριών διευθύνσεων και μετά σε κώδικα μηχανής που δεν υποστηρίζουν MULT. Οι εντολές από την 107 και μετά διαμορφώνονται ως εξής.

		107	BEQZ R11, 115	
	T2 := 0	108	ADD R12, R0, #0	Load R12 with 0
E	T2 := t2 + Y	109	ADD R12, R12, R10	R12 holds result + Y
	Count := Count +1	110	SUB R5, R5, #1	Subtract 1 from counter
	If count^=0 goto E	111	BNEZ R5, 110	If not equal to 0 repeat
		112	ADD R14, R10, R12	R14 hold result X+t2
	Y := X + t_2	114	SD 0(R2), R14	Store R13 to Y
		115		

Πριν το τρέξετε στον προσομοιωτή συνιστούμε επίσης να αλλάξετε την τιμή 60 με μια μικρότερη (έστω έξι) για να μειώσετε τις επαναλήψεις.

Άσκηση Αυτοαξιολόγησης

Παράγετε κώδικα DLX από τον βελτιστοποιημένο κώδικα τριών διευθύνσεων της άσκησης της προηγούμενης υποενότητας. Υπενθυμίζουμε τον κώδικα

t2 := X ^=1
If t2= false goto ετικέττα
Y := X + Y
Ετικέτα

Χρησιμοποιήστε τον προσομοιωτή του 4ου κεφαλαίου για να επιβεβαιώσετε το αποτέλεσμα.

Απάντηση

Κώδικας τριών διευθύνσεων	**Κώδικας DLX**		**Επεξηγήσεις**

	100
	101
	102
		ADD R1, R0, #0
		ADD R2, R1, #4
		ADD R3, R2, #4
			Αρχικοποίηση R1-R5, R0=0
			R1 holds address of X (0)
			R2 holds address of Y (0+ 4)
			R3 holds address of t2 (4+4)

T2 := X ^=1	103 104	LD R10, 0(R1) SUB R11, R10, #1	Load R10 with value of X R11 holds result X-1
If t2= false goto ετικέτα	105	BEQZ R11, 109	Branch to end if R11(=X-1)=0

Y := X + Y	106
	107
	108
		LD R12, 0(R2)
		ADD R12, R10, R12
		SD 0(R2), R12
			Load R12 with value of Y
			R12 hold result X+Y
			Store R12 to Y

Ετικέτα	109		End of program

6.4.9 Η ομαδοποίηση των φάσεων

Front και back-end

Συχνά οι φάσεις διαχωρίζονται σε front end και σε back end. Οι πρώτες έχουν σχέση με οτιδήποτε εξαρτάται από τη πηγαία γλώσσα και είναι ανεξάρτητες από την μηχανή. Στην front end περιλαμβάνονται η λεξική ανάλυση, η συντακτική ανάλυση, η δημιουργία του πίνακα συμβόλων, η σημασιολογική ανάλυση και ο ενδιάμεσος κώδικάς. Επίσης μπορεί να περιλαμβάνει μερική βελτιστοποίηση κώδικα. Επίσης περιλαμβάνει και την διαχείριση λαθών που αφορούν αυτές τις φάσεις.
Η back end περιλαμβάνει τα μέρη του μεταγλωττιστή που εξαρτώνται από την μηχανή και γενικά αυτά που είναι ανεξάρτητα από την πηγαία γλώσσα. Το back end περιλαμβάνει ορισμένες πλευρές βελτιστοποίησης του κώδικα και τη δημιουργία κώδικα μηχανής και την διαχείριση λαθών και πράξεις στον πίνακα συμβόλων που αφορούν αυτές τις φάσεις.

Είναι πολύ συνηθισμένο να παίρνουμε τον front end ενός μεταγλωττιστή, να γράφουμε μόνο τον back end και να παράγουμε έτσι έναν μεταγλωττιστή από την ίδια γλώσσα

αλλά για άλλη μηχανή. Αν ο back end έχει σχεδιαστεί προσεκτικά είναι δυνατό να μην είναι απαραίτητο να επανασχεδιάσουμε πλήρως τον νέο back end. Είναι επίσης δελεαστικό να μεταγλωττίσουμε πολλές γλώσσες υψηλού επιπέδου στον ίδιο ενδιάμεσο κώδικα, ώστε να χρησιμοποιήσουμε τον ίδιο back end για όλους τους μεταγλωττιστές της ίδιας μηχανής. Επειδή όμως υπάρχουν λεπτές διαφορές σε διάφορες γλώσσες αυτή η κατεύθυνση δεν έχει μεγάλη εφαρμογή και επιτυχία.

Περάσματα και φάσεις

Διάφορες φάσεις μιας μετάφρασης συνήθως γίνονται σε ένα πέρασμα που περιλαμβάνει την ανάγνωση ενός αρχείου εισόδου και την εγγραφή ενός αρχείου εξόδου). Πρακτικά υπάρχουν πολλές παραλλαγές που οι φάσεις οργανώνονται σε περάσματα, για αυτό προτιμήσαμε να οργανώσουμε την παρουσίαση των μεταγλωττιστών με τις φάσεις και όχι με περάσματα.

Η σειρά εκτέλεσης των έξι φάσεων δεν σημαίνει ότι κάθε φάση ξεκινά όταν τελειώσει η επόμενη της. Μπορεί να υπάρχουν επικαλύψεις εκτέλεσης των φάσεων. Να ξεκινήσει μια φάση και πριν τελειώσει να αρχίσει η επόμενη της, που θα επεξεργάζεται όμως στοιχεία τα οποία έχει παράγει η πρώτη. Για παράδειγμα, η λεξική ανάλυση, η συντακτική ανάλυση, η σημασιολογική ανάλυση και ο ενδιάμεσος κώδικας μπορούν να ομαδοποιηθούν στο ίδιο πέρασμα. Στην περίπτωση αυτή η ροή των συμβόλων από τη λεξική ανάλυση μεταφράζεται άμεσα σε ενδιάμεσο κώδικα. Πιο συγκεκριμένα, ο συντακτικός αναλυτής έχει τον «έλεγχο» και προσπαθεί να ανακαλύψει την γραμματική δομή των συμβόλων. Λαμβάνει σύμβολα όταν τα θέλει καλώντας τον λεξικό αναλυτή να του δώσει το επόμενο σύμβολο. Οταν βρει την γραμματική δομή καλεί τη ρουτίνα που κάνει την σημασιολογική ανάλυση και δημιουργεί τον ενδιάμεσο κώδικα.

Ανεξάρτητα από την οργάνωση κάθε μεταγλωττιστή πρέπει να περιλαμβάνει λειτουργικά τις φάσεις που έχουμε περιγράψει.

Ασκήσεις Αυτοαξιολόγησης
1. Ποιοι οι λόγοι που χωρίζουμε τις φάσεις σε front και back-end;

Απάντηση

Με αυτό τον τρόπο μπορούμε να πάρουμε τον front end ενός μεταγλωττιστή και γράφοντας μόνο τον νέο back end να παράγουμε έναν μεταγλωττιστή από την ίδια γλώσσα αλλά για άλλη μηχανή. Αν ο back end έχει σχεδιαστεί προσεκτικά είναι δυνατό να μην είναι απαραίτητο να επανασχεδιάσουμε πλήρως τον νέο back end.
Η άλλη κατεύθυνση, να μεταγλωττίσουμε δηλαδή πολλές γλώσσες υψηλού επιπέδου στον ίδιο ενδιάμεσο κώδικα, ώστε να χρησιμοποιήσουμε τον ίδιο back end για όλους τους μεταγλωττιστές της ίδιας μηχανής δεν έχει μεγάλη εφαρμογή και επιτυχία.

2. Ποια η σειρά εκτέλεσης των φάσεων;

Απάντηση
Η σειρά εκτέλεσης των έξι φάσεων δεν σημαίνει ότι κάθε φάση ξεκινά όταν τελειώσει η επόμενη της. Μπορεί να υπάρχουν επικαλύψεις εκτέλεσης των φάσεων. Να ξεκινήσει μια φάση και πριν τελειώσει να αρχίσει η επόμενη της, που θα επεξεργάζεται όμως στοιχεία τα οποία έχει παράγει η πρώτη.

6.4.10 Εργαλεία για τη συγγραφή μεταγλωττιστών (Compiler-writing tools)

Εισαγωγή

Η κατασκευή του αλγόριθμου ενός από-πάνω-προς-τα-κάτω συντακτικού αναλυτή μπορεί να είναι αρκετά απλή, με την προϋπόθεση ότι το συντακτικό της γλώσσας έχει οριστεί από ένα σύνολο BNF παραγωγών. Μάλιστα η κατασκευή του αλγόριθμου του συντακτικού αναλυτή για ένα οποιοδήποτε συντακτικό γλώσσας μπορεί να περιγραφεί από έναν αλγόριθμο και μπορεί να υλοποιηθεί σε πρόγραμμα. Ένα τέτοιο πρόγραμμα λέγεται *γεννήτορας συντακτικών αναλυτών* (parse generator), αφού παράγει συντακτικούς αναλυτές από συντακτικούς ορισμούς. Υπάρχουν επίσης γεννήτορες συντακτικών αναλυτών από-κάτω-προς-τα-πάνω. Ο γεννήτορας συντακτικού αναλυτή είναι πολύ χρήσιμο εργαλείο, αφού μπορεί να αυτοματοποιήσει σημαντικά την κατασκευή της φάσης της συντακτικής ανάλυσης ενός μεταγλωττιστή. Το εργαλείο συντακτικής ανάλυσης της υποενότητας 6.4.3 είναι ένας τέτοιος γεννήτορας.

Μεταγλωττιστές-μεταγλωττιστών

Με μερικές επεκτάσεις ένας γεννήτορας συντακτικού αναλυτή μπορεί να κατασκευάσει όχι μόνο την φάση της συντακτικής ανάλυσης, αλλά και ολόκληρο τον μεταγλωττιστή. Ένας τέτοιος γεννήτορας, για ιστορικούς λόγους, ονομάζεται μεταγλωττιστής-μεταγλωττιστών (compiler-compiler), παρ' όλο που μια πιο ακριβής όρος θα ήταν γεννήτορας μεταγλωττιστών.
Ένας μεταγλωττιστής-μεταγλωττιστών παράγει μεταγλωττιστή για μια γλώσσα L σε υπολογιστή C από την προδιαγραφή:
1. του συντακτικού της L και
2. των εντολών της γλώσσας μηχανής του C, που αντιστοιχούν στις εντολές στην L.

Η πρώτη χρησιμοποιείται για να παραγάγει συντακτικό αναλυτή και η δεύτερη για να παραγάγει κατάλληλα στοιχεία γέννησης κώδικα.

Πλεονεκτήματα-Μειονεκτήματα αυτόματης παραγωγής μεταγλωττιστών

Ένας μεταγλωττιστής που παράγεται αυτόματα από έναν μεταγλωττιστή-μεταγλωττιστών δεν είναι τόσο καλός όσο κάποιος που έχει παραχθεί με «το χέρι» για κάποια συγκεκριμένη γλώσσα σε κάποια συγκεκριμένη μηχανή. Όμως ο μεταγλωττιστής-μεταγλωττιστών μπορεί να είναι αποτελεσματικό εργαλείο για την γρήγορη υλοποίηση μιας νέας γλώσσας ή μιας ήδη υπάρχουσας σε νέο υπολογιστή. Ένας πρόχειρος μεταγλωττιστής μπορεί να παραχθεί με μικρή προγραμματιστική προσπάθεια και αργότερα, αν κριθεί αναγκαίο, μπορεί να παραχθεί ένας πιο αποδοτικός μεταγλωττιστής.
Τονίζουμε ότι ένας μεταγλωττιστής-μεταγλωττιστών δεν είναι μέρος του λογισμικού συστήματος, αλλά ένα εργαλείο για να γράφουμε λογισμικό συστήματος. Έτσι σπάνια φαίνεται στα προϊόντα της κατασκευάστριας εταιρίας αφού οι περισσότεροι χρήστες υπολογιστών δεν γράφουν τους δικούς του μεταγλωττιστές. Οι πιο γνωστοί μεταγλωττιστές-μεταγλωττιστών είναι για το Unix το Lex-Yacc, για το Linux το Bison και για το DOS το PCLex-PCYacc.

Ασκήσεις Αυτοαξιολόγησης
1. Τι είναι ο γεννήτορας συντακτικών αναλυτών;

Απάντηση
Ένα πρόγραμμα που παράγει συντακτικούς αναλυτές από συντακτικούς ορισμούς. Η κατασκευή του αλγόριθμου του συντακτικού αναλυτή για ένα οποιοδήποτε συντακτικό γλώσσας μπορεί να περιγραφεί από έναν αλγόριθμο και μπορεί να υλοποιηθεί σε πρόγραμμα.

2. Τι είναι ο μεταγλωττιστής-μεταγλωττιστών;

Απάντηση
Με μερικές επεκτάσεις ένας γεννήτορας συντακτικού αναλυτή μπορεί να κατασκευάσει όχι μόνο την φάση της συντακτικής ανάλυσης, αλλά και ολόκληρο τον μεταγλωττιστή. Ένας τέτοιος γεννήτορας, για ιστορικούς λόγους, ονομάζεται μεταγλωττιστής-μεταγλωττιστών (compiler-compiler), παρ' όλο που μια πιο ακριβής όρος θα ήταν γεννήτορας μεταγλωττιστών.

3. Ποια είναι τα πλεονεκτήματα και μειονεκτήματα της αυτόματης παραγωγής μεταγλωττιστών;

Απάντηση
Ένας μεταγλωττιστής που παράγεται αυτόματα από έναν μεταγλωττιστή-μεταγλωττιστών δεν είναι τόσο καλός όσο κάποιος που έχει παραχθεί με «το χέρι» για κάποια συγκεκριμένη γλώσσα σε κάποια συγκεκριμένη μηχανή. Όμως ο μεταγλωττιστής-μεταγλωττιστών μπορεί να είναι αποτελεσματικό εργαλείο για την γρήγορη υλοποίηση μιας νέας γλώσσας ή μιας ήδη υπάρχουσας σε νέο υπολογιστή. Ένας πρόχειρος μεταγλωττιστής μπορεί να παραχθεί με μικρή προγραμματιστική προσπάθεια και αργότερα, αν κριθεί αναγκαίο, μπορεί να παραχθεί ένας πιο αποδοτικός μεταγλωττιστής.

6.4.11 Συμβολομεταφραστές (Assemblers)

Εισαγωγή

Ένα απλό παράδειγμα μεταφραστή γλώσσας είναι ο συμβολομεταφραστής (assembler), ο οποίος μεταφράζει προγράμματα γραμμένα σε συμβολική γλώσσα (assembly). Η συμβολική γλώσσα μπορεί να θεωρηθεί ως ελαφρά υψηλότερη μορφή γλώσσας από την γλώσσα μηχανής. Κάθε εντολή σε συμβολική γλώσσα συνήθως αντιστοιχεί σε μια μόνη εντολή γλώσσας μηχανής. Όπως και η γλώσσα μηχανής, μια συμβολική γλώσσα προορίζεται μόνο για το οποίο έχει σχεδιαστεί. Προγράμματα γραμμένα σε μια συμβολική γλώσσα δεν μπορούν να μεταφερθούν σε άλλους υπολογιστές.

Διαφορά συμβολικής γλώσσας και γλώσσας μηχανής

Η βασική διαφορά ανάμεσα στην συμβολική γλώσσα και την γλώσσα μηχανής βρίσκεται στο τρόπο με τον οποίο αναφερόμαστε σε θέσεις μνήμης. Σ' ένα πρόγραμμα γλώσσας μηχανής πρέπει να αναφερθούμε στις θέσεις μνήμης με την διεύθυνσή τους, ενώ σ' ένα πρόγραμμα σε συμβολική γλώσσα μια θέση μπορεί να αναφέρεται με όνομα επιλεγμένο από τον προγραμματιστή. Όμως η αντιστοιχία ονομάτων και διευθύνσεων πρέπει να καθοριστεί από στο πρόγραμμα (ο τρόπος αντιστοιχίας είναι έξω από τους διδακτικούς στόχους και δεν έχει ιδιαίτερη σημασία εδώ).

Παράδειγμα

Το συντακτικό μιας τυπικής εντολής διαφορετικής από την DLX του κεφαλαίου 4 σε συμβολική γλώσσα είναι:

Εντολή -> *Λειτουργία Διεύθυνση*
Λειτουργία -> LOAD | STORE | ADD | SUB | Branch
Διεύθυνση -> όνομα | αριθμός | *Διεύθυνση τελεστής* αριθμός
Τελεστής -> + | -

Παραδείγματα έγκυρων εντολών είναι οι:

ADD A LOAD A+1 STORE A-3

Όπου Α είναι το όνομα που δίνεται από τον προγραμματιστή σε μια συγκεκριμένη θέση μνήμης. Η διεύθυνση μνήμης Α+1 είναι αυτή που έπεται του Α, ενώ η Α-3 βρίσκεται τρεις θέσεις πριν το Α.

Ανάλυση συμβολικών γλωσσών και γέννηση κώδικα μηχανής

Επειδή οι περισσότερες συμβολικές γλώσσες έχουν απλό συντακτικό (όπως το παραπάνω) οι φάσεις της λεξικής και συντακτικής ανάλυσης είναι πολύ απλές. Κατά τη διάρκεια της συντακτικής ανάλυσης συγκεντρώνονται πληροφορίες σχετικά με την αντιστοιχία ανάμεσα στα ονόματα και στις διευθύνσεις των θέσεων μνήμης και αυτές οι πληροφορίες εισάγονται σε πίνακα συμβόλων.
Επίσης και η γέννηση κώδικα είναι απλή. Ο συμβολομεταφραστής παράγει μια εντολή σε γλώσσα μηχανής για κάθε εντολή σε συμβολική χώρα. Η λειτουργία (operation) για κάθε εντολή μεταφράζεται στον αντίστοιχο κώδικα λειτουργίας σε γλώσσα μηχανής. Για την γλώσσα μηχανής του εδαφίου 4.? Το ADD μεταφράζεται σε 0011. Χρησιμοποιώντας πληροφορίες του πίνακα συμβόλων, η διεύθυνση σε κάθε εντολή μεταφράζεται στην αριθμητική διεύθυνση της αντίστοιχης θέσης μνήμης. Έτσι αν Α είναι το όνομα που αντιστοιχεί στη θέση 42, η εντολή

ADD A+1

Μεταφράζεται σε

0011000000101011
(0011 για το ADD και 000000101011 για τον δυαδικό αριθμό 43=42+1)

Χρήση Συμβολικών Γλωσσών

Οι συμβολικές γλώσσες είναι κάτι περισσότερο από μια βολική μορφή γλώσσας μηχανής. Αρκετές έχουν και άλλα χαρακτηριστικά που θεωρούνται πρόγονοι γλωσσών προγραμματισμού. Παρόλα αυτά οι συμβολικές γλώσσες υποφέρουν από τα ίδια ελαττώματα που υποφέρουν και οι γλώσσες μηχανής και είναι ακατάλληλες για την έκφραση πολύπλοκων αλγορίθμων ή για την ανάπτυξη προγραμμάτων που πρέπει να χρησιμοποιηθούν σε διαφορετικούς υπολογιστές. Παρ' όλο που χρησιμοποιήθηκαν ευρέως στο παρελθόν, η έλευση των γλωσσών υψηλού επιπέδου έχει μειώσει σημαντικά τον ρόλο τους. Η χρήση τους τώρα έχει περιοριστεί στο συντονισμό ή σε λίγα

προγράμματα που εξαρτώνται πάρα πολύ από την μηχανή, όπως ο χειρισμός των διακοπών και ο έλεγχος περιφερειακών εισόδου και εξόδου.