ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ ΣΤΟΥΣ ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΠΡΟΛΟΓΟΣ ΒΙΒΛΙΟΥ

Η σχεδίαση και η κατασκευή ενός Μεταγλωττιστή ή ενός Διερμηνευτή για μια γλώσσα προγραμματισμού δεν είναι ούτε απλή ούτε εύκολη υπόθεση. Το υλικό αυτής της Θεματικής Υποενότητας προορίζεται για να σας διδάξει τις βασικές εισαγωγικές έννοιες και να σας μάθει πως μπορείτε να κατασκευάσετε όχι ένα πλήρη Μεταγλωττιστή ή Διερμηνευτή αλλά κάποια βασικά τμήματα αυτού. Δίνει έμφαση στα πρακτικά προβλήματα σχεδίασης και υλοποίησης και όχι στην πραγματικά πολύ εκτεταμένη θεωρία που υπάρχει στη διεθνή βιβλιογραφία.

Για να καταφέρετε να κατανοήσετε πλήρως το υλικό αυτής της Θεματικής Υποενότητας χρειάζεστε να έχετε αποκτήσει τουλάχιστον ενός χρόνου προγραμματιστική εμπειρία σε κάποια γλώσσα προγραμματισμού και να έχετε βασικές γνώσεις από Δομές Δεδομένων, γλώσσα Assembly και στοιχεία θεωρίας γλωσσών προγραμματισμού. Μην ανησυχείτε όμως διότι το μέρος των βασικών αυτών γνώσεων που είναι προαπαιτούμενο θα σας δοθεί μέσα από την ίδια την Θεματική Υποενότητα (εκτός βέβαια από την προγραμματιστική εμπειρία), ενώ πολύ περισσότερες θεωρητικές και πρακτικές γνώσεις θα τις αποκτήσετε από τις αντίστοιχες Θεματικές Υποενότητες “Δομές Δεδομένων”, “Αρχιτεκτονική Υπολογιστών”, “Μικροεπεξεργαστές” και “Αυτόματα και Τυπικές Γλώσσες”.

'Οσοι από εσάς θελήσετε να ασχοληθείτε σοβαρότερα με Μεταγλωττιστές, Διερμηνευτές και γενικότερα με Μεταφραστές θα έχετε την ευκαιρία να μάθετε περισσότερα πράγματα και κυρίως να κατασκευάσετε ένα μικρό αλλά πλήρη Μεταγλωττιστή ή Διερμηνευτή στα πλαίσια της Θεματικής Υποενότητας “Εργαστηριακές Ασκήσεις σε Μεταγλωττιστές”. Τέλος, όσον αφορά τη βιβλιογραφία, επέλεξα για λόγους οικονομίας χώρου να σας την παρουσιάσω συγκεντρωμένη στο τέλος του βιβλίου, αντί να την δίνω σε κάθε κεφάλαιο ξεχωριστά.

Στόχος

Στόχος του κεφαλαίου αυτού είναι να σας παρουσιάσει σύντομα και περιληπτικά τί είναι ο μεταγλωττιστής, που χρειάζεται, πώς είναι δομημένος από διάφορες λειτουργικές ενότητες, και πώς οι ενότητες αυτές επικοινωνούν και συνεργάζονται μεταξύ τους. Επίσης σας βοηθάει να έχετε μια πλήρη εικόνα για το περιεχόμενο των κεφαλαίων που θα ακολουθήσουν.

Προσδοκώμενα αποτελέσματα

'Οταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- εξηγήσετε τι είναι ένας μεταγλωττιστής,
- περιγράψετε τις διαφορές μεταξύ μεταγλωττιστών, διερμηνευτών, μεταφραστών και προ-επεξεργαστών,
- περιγράψετε εν συντομία τη διαδικασία μετάφρασης-φόρτωσης-εκτέλεσης ενός προγράμματος,
- περιγράψετε τις φάσεις (δομή) ενός μεταλωττιστή και ενός διερμηνευτή,
- εξηγήσετε τι είναι τα "περάσματα" (passes) και πώς τα μειώνουμε,
- περιγράψετε σε συντομία τις έξι φάσεις της μεταφραστικής διαδικασίας που επιτελεί ένας μεταγλωττιστής,
- εξηγήσετε τι είναι τα tokens,
- εξηγήσετε τι είναι και πού χρησιμοποιείται ο πίνακας συμβόλων,
- αναφέρετε εργαλεία που χρησιμοποιούνται για την κατασκευή των μεταγλωττιστών,
- περιγράψετε μια ειδική περίπτωση διαδικασίας κατασκευής ενός μεταγλωττιστή(bootstrapping)
- περιγράψετε τα δένδρα ανίχνευσης και πού χρησιμοποιούνται.

ΕΝΝΟΙΕΣ-ΚΛΕΙΔΙΑ

Μεταφραστής, Μεταγλωττιστής, Διερμηνευτής, Προεπεξεργαστής, Πηγαίος Κώδικας, Τελικός Κώδικας, Περάσματα, Tokens, Bootstrapping, Λεκτική Ανάλυση, Συντακτική Ανάλυση, Σημασιολογική Ανάλυση, Δημιουργία Ενδιάμεσου Κώδικα, Βελτιστοποίηση Κώδικα και Δημιουργία Τελικού Κώδικα.

Το κεφάλαιο αυτό θα σας βοηθήσει να αποκτήσετε μια συνοπτική μεν αλλά πλήρη εικόνα του τι είναι ένας Μεταγλωττιστής και ένας Διερμηνευτής, ποιά είναι η δουλειά

τους πώς είναι δομημένοι και πώς διαφοροποιείται η λειτουργία του ενός από τον άλλο. Είναι το βασικότερο κεφάλαιο της Θεματικής Υποενότητας που έχετε στα χέρια σας μια και σας δίνει συνολική και σφαιρική εικόνα της οργάνωσης αλλά κυρίως των λειτουργιών που επιτελούνται μέσα σε ένα μεταφραστικό σύστημα (πρόγραμμα) όπως είναι ο Μεταγλωττιστής και ο Διερμηνευτής.

Δεν θα πρέπει να έχετε ιδιαίτερη δυσκολία με το κεφάλαιο αυτό μια και σας παρουσιάζει τις λειτουργίες των μεταφραστικών αυτών συστημάτων από ένα υψηλό επίπεδο και με αρκετά απλό τρόπο. Επειδή όμως στη συνέχεια των κεφαλαίων θα εξετάσουμε μία-μία τις λειτουργίες αυτές και θα δούμε αρκετές από τις λεπτομέρειές τους θα σας διευκολύνει πολύ να έχετε στο μυαλό σας την συνολική εικόνα της μεταφραστικής διαδικασίας.

'Ετσι, ξεκινάμε με το να ορίσουμε τι είναι οι Μεταφραστές και ειδικότερα οι Μεταγλωττιστές και οι Διερμηνευτές, στη συνέχεια κάνουμε μια σύντομη αναφορά στις γλώσσες προγραμματισμού και την διαδικασία μετάφρασης-φόρτωσης-εκτέλεσης ενός προγράμματος για να δούμε κατόπιν τη δομή ενός Μεταγλωττιστή και τη διαφοροποίηση του από ένα Διερμηνευτή. Κατόπιν βλέπουμε τις λειτουργίες τους και στο τέλος κάποια εργαλεία χρήσιμα στην κατασκευή μεταφραστικών συστημάτων.

ΕΝΟΤΗΤΑ 1.1 ΕΙΣΑΓΩΓΗ

'Ενας **Μεταφραστής** (Translator) είναι ένα πρόγραμμα το οποίο παίρνει σαν είσοδο ένα πρόγραμμα γραμμένο σε κάποια γλώσσα προγραμματισμού (αρχική γλώσσα - source language) και δίνει στην έξοδο ένα πρόγραμμα σε κάποια άλλη γλώσσα (τελική γλώσσα -target or object language). Αν η αρχική γλώσσα είναι υψηλού επιπέδου όπως C, PL/I, FORTRAN, PASCAL κλπ και η τελική γλώσσα είναι χαμηλού επιπέδου όπως Assembly ή γλώσσα μηχανής, τότε ο μεταφραστής λέγεται **Συμβολομεταφραστής ή Μεταγλωττιστής** (Compiler).

Παλαιότερα οι Μεταγλωττιστές εθεωρούντο εξαιρετικά χρονοβόρα προγράμματα για να τα γράψει κανείς. Σήμερα γράφονται με αρκετά λιγότερη προσπάθεια χάρις στην κατανόηση της οργάνωσης της διαδικασίας συμβολομετάφρασης (Compilation) και την ανάπτυξη προγραμματιστικών εργαλείων για την υλοποίηση των διαφόρων τμημάτων των Μεταγλωττιστών.

Στις περισσότερες περιπτώσεις συμβολομετάφρασης, οι μεταφραστές μετασχηματίζουν το πρόγραμμα από την αρχική γλώσσα σε μια πιο απλοποιημένη που λέγεται ενδιάμεσος κώδικας ή γλώσσα (Intermediate Code or Language). Μπορούμε να φαντασθούμε τον ενδιάμεσο κώδικα σαν μια γλώσσα μηχανής ενός υποθετικού υπολογιστή ο οποίος μπορεί να εκτελέσει τον κώδικα αυτόν.

Ο μεταφραστής ο οποίος μετασχημάτησε το αρχικό πρόγραμμα στην ενδιάμεση γλώσσα και στη συνέχεια εκτελεί τη γλώσσα αυτή καλείται **Διερμηνευτής** (Interpreter). Αντίθετα, ο μεταφραστής με βάση την ενδιάμεση γλώσσα θα συνεχίσει για να κατασκευάσει τελικό κώδικα. Μερικοί Διερμηνευτές εκτελούν τις εντολές της αρχικής γλώσσας χωρίς να κατασκευάζουν ενδιάμεσο κώδικα (π.χ. Basic Interpreters ή και JCL Interpreters). Οι Interpreters είναι συνήθως μικρότεροι σε μέγεθος από τους Μεταγλωττιστές αλλά και αρκετές φορές πιο αργοί στην εκτέλεση των αρχικών προγραμμάτων.

Υπάρχουν και διάφοροι άλλοι τύποι μεταφραστών. Ο **Assembler** μεταφράζει προγράμματα από assembly γλώσσα σε γλώσσα μηχανής ενώ ένας **Προεπεξεργαστής** (Preprocessor) μετασχηματίζει προγράμματα από μια υψηλή γλώσσα σε ισοδύναμα προγράμματα μιας άλλης υψηλής γλώσσας. Π.χ. Preprocessor για δομημένη (structured) Fortran, preprocessor για γραφήματα (graphs) σε Fortran, Pascal κ.λ.π.

ΕΝΟΤΗΤΑ 1.2 ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Συνέπεια των δυσκολιών του προγραμματισμού σε γλώσσα μηχανής υπήρξε το πρώτο βήμα ανάπτυξης που είναι οι συμβολικές γλώσσες Assembly. Σ' αυτές τις γλώσσες αντικαταστάθηκαν οι λειτουργικοί κώδικες των εντολών και οι διευθύνσεις δεδομένων και εντολών, με συμβολικά ονόματα για ευκολία του προγραμματιστή.

Το συμβολικό Assembly πρόγραμμα πρέπει πρώτα να μεταφραστεί σε γλώσσα μηχανής από κάποιον Assembler και κατόπιν να εκτελεστεί. Τα macros (macro-εντολές) είναι ψευδοεντολές Assembly οι οποίες μετασχηματίζονται από τον Macro-Assembler σε μια ακολουθία εντολών Assembly και ίσως και άλλων macros. Οι περισσότεροι σύγχρονοι Assemblers είναι στην πραγματικότητα Macro-Assemblers.

Οι γλώσσες υψηλού επιπέδου αποτελούν την φυσική συνέχεια στην εξέλιξη των γλωσσών προγραμματισμού και ήλθαν για να ανακουφίσουν τον προγραμματιστή από λεπτομέρειες που δεν είχαν καμιά σχέση με το προγραμματιζόμενο πρόβλημα, αλλά και να διευκολύνουν χρήστες οι οποίοι δεν είχαν άμεση επαφή με Assemblers και ούτε υπήρχε λόγος να έχουν.

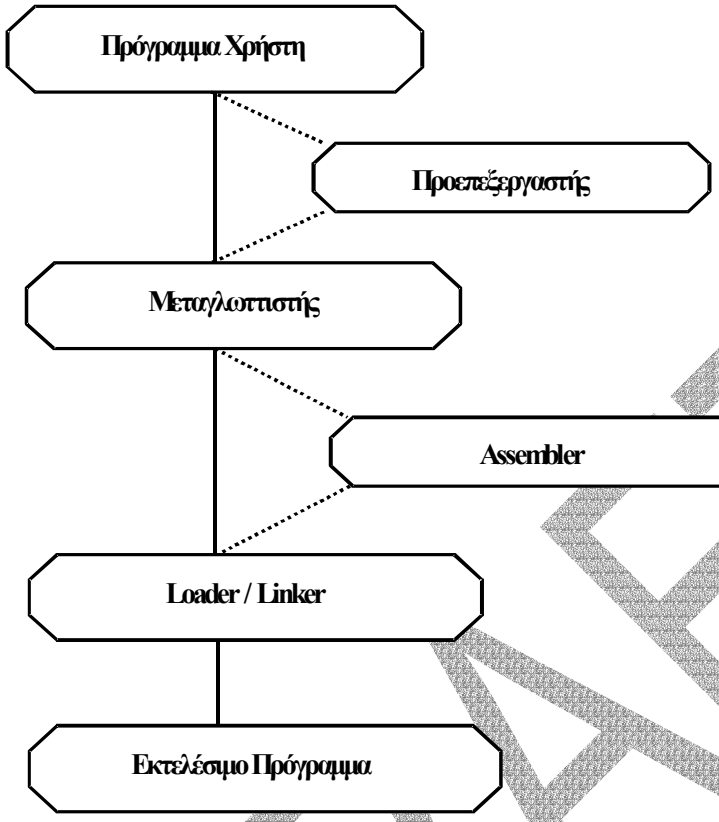
Το βασικό πλεονέκτημα των γλωσσών προγραμματισμού υψηλού επιπέδου είναι ότι επιτρέπουν στον χρήστη να εκφράζει (προγραμματίζει) αλγόριθμους σε μορφές είτε κοντά στα μαθηματικά είτε κοντά στη φυσική του γλώσσα, είτε ακόμα και κοντά στη λύση του προβλήματος που εκφράζει ο αλγόριθμος. Το πρόβλημα που εισάγεται με τις υψηλές γλώσσες είναι ότι το μεταφραστικό πρόγραμμα (Μεταγλωττιστής) είναι πολύ πιο πολύπλοκο από οποιονδήποτε Macro Assembler.

1.2.1 Η ΔΙΑΔΙΚΑΣΙΑ ΜΕΤΑΦΡΑΣΗΣ-ΦΟΡΤΩΣΗΣ-ΕΚΤΕΛΕΣΗΣ

Στο σχήμα 1.1 δίνεται διαγραμματικά η διαδικασία Μετάφρασης-Φόρτωσης-Εκτέλεσης ενός προγράμματος. Ο προγραμματιστής γράφει σε κάποια γλώσσα υψηλού επιπέδου το πρόγραμμά του, το οποίο ουσιαστικά είναι, η κωδικοποιημένη λύση ενός προβλήματος. Το πρόγραμμα αυτό στη συνέχεια μπορεί να χρειάζεται προεπεξεργασία ή όχι. 'Αν χρειάζεται, τότε θα τροφοδοτηθεί στον προεπεξεργαστή του Μεταγλωττιστή, αλλιώς τροφοδοτείται κατ' ευθείαν στον Μεταγλωττιστή. Ο Μεταγλωττιστής εν γένει βγάζει έξοδο γλώσσα μηχανής υπό μορφή μεταθετού κώδικα (relocatable code). Ο μεταθετός κώδικας πιθανότατα περιέχει και αναφορές σε βιβλιοθήκες προκατασκευασμένων ρουτινών (π.χ. μαθηματικές συναρτήσεις κ.λ.π).

Οι αναφορές αυτές (procedure ή function calls) πρέπει να συνδεθούν με τον αντίστοιχο κώδικα από τις βιβλιοθήκες (μέσω του Linker), ώστε να καταλήξουμε στον εκτελέσιμο κώδικα. Ο κώδικας σε εκτελέσιμη και μεταθετή μορφή μπορεί, μετά από αίτημα του χρήστη πρός το λειτουργικό σύστημα (και τελικά πρός τον φορτωτή Loader), να εκτελεσθεί από τον υπολογιστή. Σε κάποιες λίγες περιπτώσεις ο μεταγλωττιστής παράγει κώδικα Assembly, οπότε στην περίπτωση αυτή χρειάζεται να κληθεί ο Assembler και η διαδικασία συνεχίζεται όπως παραπάνω.

ΕΝΟΤΗΤΑ 1.3 Η ΔΟΜΗ ΕΝΟΣ ΜΕΤΑΓΛΩΤΤΙΣΤΗ



Σχήμα 1.1. Διαδικασία Μετάφρασης-Φόρτωσης-Εκτέλεσης

Ο μεταγλωττιστής (compiler) παίρνει είσοδο ένα πρόγραμμα και παράγει στην έξοδο μια ισοδύναμη ακολουθία εντολών μηχανής (σε μερικές περιπτώσεις παράγει ισοδύναμο πρόγραμμα σε Assembly). Η διαδικασία της μεταφραστικής αυτής δουλειάς είναι τόσο πολύπλοκη που συνήθως χωρίζεται σε διάφορες φάσεις (phases), (δείτε το σχήμα 1.2). Κάθε φάση είναι μια διαδικασία (μέρος της συνολικής μετάφρασης) η οποία παίρνει είσοδο κάποια μορφή του αρχικού προγράμματος και παράγει στην έξοδο κάποια άλλη αναπαράσταση.

Η πρώτη φάση -Λεκτικός Αναλυτής (lexical analyzer ή scanner) χωρίζει χαρακτήρες της αρχικής γλώσσας (προγράμματος) σε λογικές ομάδες-tokens. Τα συνηθέστερα tokens είναι λέξεις-κλειδιά (keywords) όπως DO ή IF, ονόματα μεταβλητών (identifiers) όπως Y ή SPEED, σύμβολα τελεστών (operator symbols) όπως = ή *, και σύμβολα στίξης όπως παρένθεση ή κόμμα. Η έξοδος του Λεκτικού Αναλυτή είναι μια ακολουθία από tokens που δίνεται στην επόμενη φάση, του

Συντακτικού Αναλυτή (Syntax Analyzer ή Parser). Τα tokens μπορεί να παριστάνονται με ακέραιους κώδικες, π.χ. το DO μπορεί να παριστάνεται με 1, το * με 2 και ο "identifier" με 3. Στην περίπτωση του token "identifier" χρειάζεται και μια δεύτερη ποσότητα που καθορίζει ποιός από τους identifiers του προγράμματος αναφέρεται από το συγκεκριμένο token.

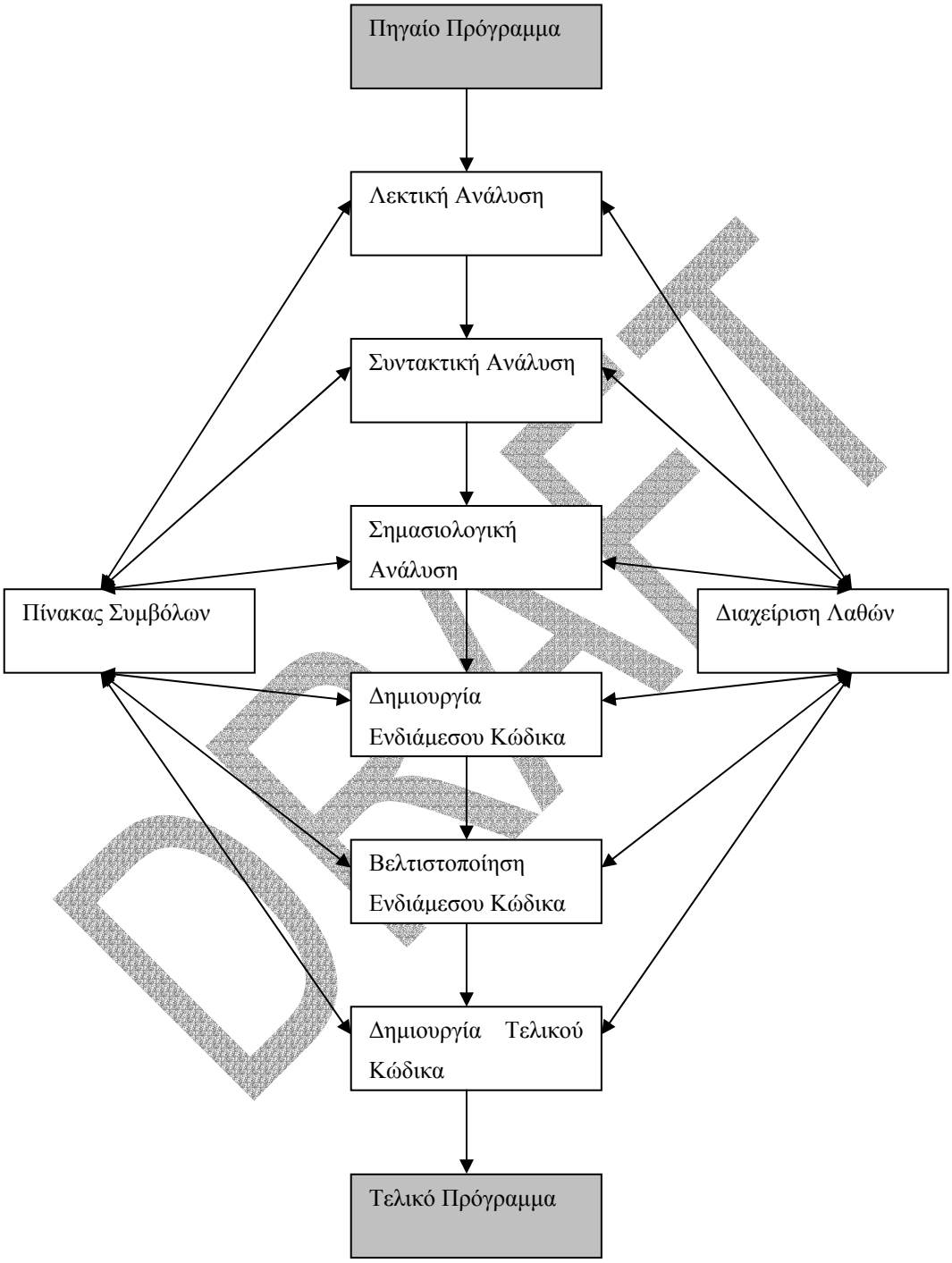
Ο Συντακτικός Αναλυτής (Syntax Analyzer) ομαδοποιεί τα tokens σε συντακτικές δομές. Π.χ. τα τρία tokens που παριστούν το A+B μπορεί να ομαδοποιηθούν σε μια συντακτική δομή που καλείται έκφραση (expression). Οι εκφράσεις μπορεί να ομαδοποιηθούν σε εντολές (statements). Συχνά η συντακτική δομή θεωρείται σαν ένα δένδρο του οποίου τα φύλλα είναι τα tokens που λογικά ανήκουν μαζί.

Στα πλαίσια της Σημασιολογικής Ανάλυσης θα προσδιορισθεί ο τύπος του αποτελέσματος A+B (π.χ. integer ή real) και θα εισαχθούν εντολές που κάνουν μετατροπές τύπων στα A και B εάν και όπου χρειάζεται.

Στη Δημιουργία του Ενδιάμεσου Κώδικα (Intermediate Code Generation) χρησιμοποιείται η δομή που παράγεται από τον Συντακτικό Αναλυτή για να κατασκευαστεί μια σειρά από απλές εντολές. Οι εντολές αυτές συχνά αποτελούνται από ένα τελεστή (operator) και μικρό αριθμό από έντελα (operands) και μπορούν να θεωρηθούν σαν ένα είδος macro εντολών. Η βασική διαφορά μεταξύ ενδιάμεσου κώδικα και κώδικα Assembly είναι ότι στον ενδιάμεσο κώδικα δεν χρειάζεται να γίνει αναφορά στους Καταχωρητές (registers) που θα χρησιμοποιηθούν στις διάφορες πράξεις.

Η Βελτιστοποίηση του Κώδικα (Code Optimization) είναι μια προαιρετική φάση που χρησιμοποιείται για να βελτιώσει τον ενδιάμεσο κώδικα ώστε το τελικό (object) πρόγραμμα να λειτουργεί-τρέχει πιο γρήγορα και/ή να καταλαμβάνει λιγότερο χώρο στη μνήμη.

Η τελική φάση-Δημιουργία Κώδικα (Code Generation)-παράγει τον τελικό κώδικα (object code), αποφασίζοντας για τις θέσεις μνήμης των δεδομένων, διαλέγοντας κώδικα για την προσπέλαση κάθε δεδομένου και επιλέγοντας τους καταχωρητές με τους οποίους θα γίνει κάθε υπολογιστική πράξη. Αποτελεί τη δυσκολότερη φάση πρακτικά και θεωρητικά.



Σχήμα 1.2 Οι φάσεις ενός μεταγλωττιστή

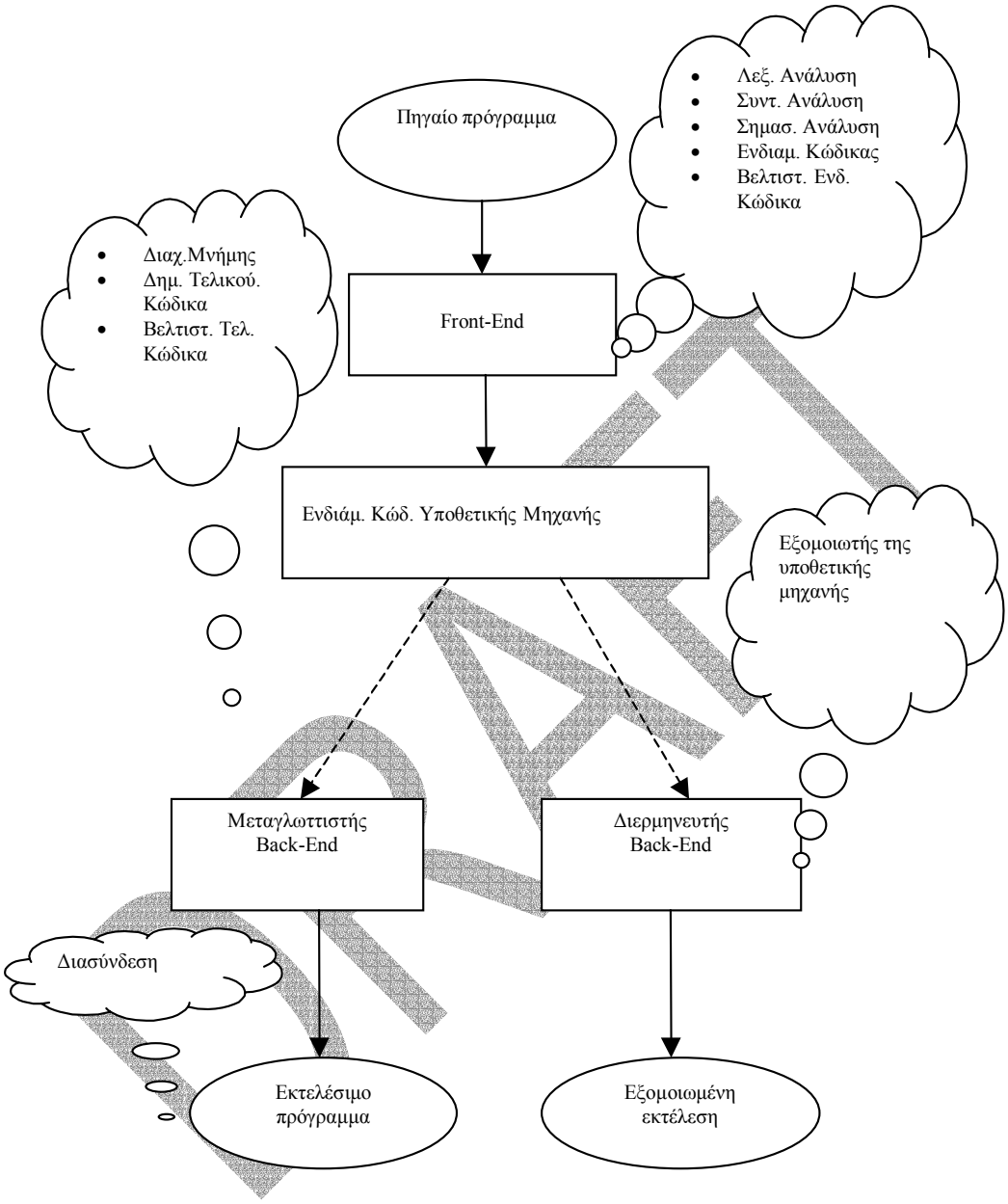
Η διαχείριση Πινάκων (Table-Management, Bookkeeping), διατηρεί πληροφορίες για τα ονόματα που χρησιμοποιούνται στο πρόγραμμα, όπως τύπος (real, character κ.λ.π.), μέγεθος και άλλες. Η φυσική δομή που χρησιμοποιείται για τις πληροφορίες αυτές καλείται Πίνακας Συμβόλων (Symbol Table).

Ο Χειριστής Λαθών (Error Handler) καλείται κάθε φορά που αναγνωρίζεται κάποια ανωμαλία στο πρόγραμμα. Εκδίδει διαγνωστικά μηνύματα και ρυθμίζει τις πληροφορίες που τροφοδοτούνται από μια φάση σε άλλη ώστε κάθε φάση να μπορεί να συνεχίσει την δουλειά της. Είναι λογική απαίτηση από κάθε μεταγλωττιστή η μεταφραστική διαδικασία να φτάνει μέχρι το τέλος της ώστε στα εσφαλμένα προγράμματα να αποκαλύπτονται από τον Μεταγλωττιστή όσον το δυνατόν περισσότερα λάθη.

'Ενας Μεταγλωττιστής και ένας Διερμηνευτής μπορούν να θεωρηθούν ότι αποτελούνται (δομικά) από δύο τμήματα όπως φαίνεται στο Σχήμα 1.3. Το εμπρόσθιο τμήμα (front-end) είναι ίδιο και στις δύο περιπτώσεις. Το οπίσθιο τμήμα, στην περίπτωση του μεταγλωττιστή (back-end) περιλαμβάνει τις λειτουργίες που αναφέρονται στο Σχήμα 1.3 και περιγράφτηκαν προηγούμενα, ενώ στην περίπτωση του διερμηνευτή περιλαμβάνει τον εξομειωτή που εκτελεί τις εντολές του ενδιάμεσου κώδικα.

'Ασκηση Αυτοαξιολόγησης 1 / Κεφ.1

Εξηγείστε τη δομική διαφορά μεταξύ ενός μεταγλωττιστή και ενός διερμηνευτή.



Σχήμα 1.3. Οι λειτουργίες μεταγλώττισης και διερμήνευσης

1.3.1 ΠΕΡΑΣΜΑΤΑ(PASSES)

Κατά την υλοποίηση ενός Μεταγλωττιστή κομμάτια από μια ή περισσότερες φάσεις συνδυάζονται σε ένα πέρασμα (pass). Το πέρασμα διαβάζει το πηγαίο πρόγραμμα ή την έξοδο του προηγούμενου περάσματος, κάνει τους μετασχηματισμούς που ορίζονται από τις φάσεις του, και γράφει το αποτέλεσμα σε κάποιο προσωρινό αρχείο, το οποίο ενδεχομένως διαβάζεται από κάποιο επόμενο πέρασμα. Ο αριθμός των περασμάτων και η ομαδοποίηση των φάσεων σε περάσματα, υπαγορεύονται από τη συγκεκριμένη γλώσσα και μηχανή στην οποία θα λειτουργήσει ο Μεταγλωττιστής. Μερικές γλώσσες όπως Algol 68 και PL/I χρειάζονται δύο τουλάχιστον περάσματα. Μεταγλωττιστές που τρέχουν σε υπολογιστές με μικρή μνήμη συνήθως χρειάζονται περισσότερα περάσματα από εκείνους που τρέχουν σε υπολογιστές μεγάλης μνήμης. Ακόμη, τα περισσότερα περάσματα ενός Μεταγλωττιστή υπαγορεύουν και μεγαλύτερο χρόνο για την μετάφραση των προγραμμάτων.

1.3.2 ΜΕΙΩΣΗ ΤΟΥ ΑΡΙΘΜΟΥ ΤΩΝ ΠΕΡΑΣΜΑΤΩΝ

Δημιουργείται το ερώτημα πώς διάφορες φάσεις μπορούν να συνδυαστούν σε ένα πέρασμα χωρίς γράψιμο και διάβασμα σε ενδιάμεσο (προσωρινό) αρχείο. Τυπική απάντηση δίνει ο Λεκτικός Αναλυτής όπου ένας μικρός βοηθητικός χώρος (buffer) π.χ. ένα διάνυσμα στη μνήμη χρησιμοποιείται σαν διάμεσο (Interface) μεταξύ των περασμάτων. Σε άλλες περιπτώσεις ενσωματώνονται φάσεις σε ένα πέρασμα με την τεχνική του πισωμπαλώματος (" backpatching ").

Παράδειγμα 1/Κεφ. 1 'Ενας Assembler μπορεί να έχει κάποια εντολή της μορφής " GOTO L " η οποία προηγείται κάποιας εντολής που έχει ετικέτα (Label) L, π.χ. L:ADD X.

'Ενας Assembler 2-περασμάτων χρησιμοποιεί το πρώτο πέρασμα για να εισάγει στον Πίνακα Συμβόλων μια λίστα από όλους τους identifiers (ετικέτες (Labels) εντολών και ονόματα δεδομένων-μεταβλητών) μαζί με τις διευθύνσεις μηχανής (σχετικές ως προς την αρχή του προγράμματος) που τους αντιστοιχούν. Κατόπιν το δεύτερο πέρασμα αντικαθιστά τους μνημονικούς κώδικες λειτουργίας, όπως GOTO,

με τα ισοδύναμά τους σε γλώσσα μηχανής και τους identifiers με τις διευθύνσεις μηχανής.

Από την άλλη μεριά ένας Assembler ενός-περάσματος μπορεί να δημιουργήσει ένα "σκελετό" της GOTO L και να προσθέσει την διεύθυνση μηχανής της GOTO L σε μια λίστα εντολών που πρόκειται να πισωμπαλωθούν (backpatched) μόλις η διεύθυνση μηχανής της επιγραφής L γίνει γνωστή στον Assembler. Η λίστα αυτή μπορεί να "κρέμεται" σαν δομή διασυνδεδεμένης λίστας, από μια θέση που σχετίζεται με την επιγραφή L στον Πίνακα Συμβόλων. Ο Assembler μόλις συναντήσει την εντολή "L:ADD X" ψάχνει την λίστα των εντολών που αναφέρονται στην L και βάζει την διεύθυνση της L:ADD X στο πεδίο διεύθυνσης όλων αυτών των εντολών. Επόμενες αναφορές στην L χρησιμοποιούν την διεύθυνση της L που είναι πλέον γνωστή.

Η "Απόσταση" του backpatching είναι συνήθως μικρή για τους περισσότερους μεταγλωττιστές. Σαν απόσταση στο παραπάνω παράδειγμα του Assembler μπορούμε να θεωρήσουμε την απόσταση μεταξύ των εντολών GOTO L και L:ADD X.

'Ασκηση Αυτοαξιολόγησης 2 / Κεφ. 1

Η μείωση του αριθμού των περασμάτων σε ένα μεταγλωττιστή στοχεύει να βοηθήσει στην κατασκευή:

α) πιο" μικρού" μεταγλωττιστή

β) πιο " τμηματοποιημένου" (άρα πιο εύκολα να γραφεί) μεταγλωττιστή

γ) πιο "γρήγορου" μεταγλωττιστή όσον αφορά τον χρόνο που χρειάζεται για να μεταφράσει το πρόγραμμά σας

- **δ)** πιο " γρήγορου" όσον αφορά τον χρόνο που χρειάζεται για να εκτελεσθεί το πρόγραμμά σας , μετά την μετάφρασή του από τον μεταγλωττιστή

ε) πιο εύκολου (ευκολία να κατασκευασθεί) και πιο συντηρήσιμου μεταγλωττιστή.

Ποιά από τις πέντε παραπάνω περιπτώσεις θεωρείται ότι είναι η σωστή; Αιτιολογήστε την επιλογή σας.

ΕΝΟΤΗΤΑ 1.4 ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

Ο Λεκτικός Αναλυτής διαβάζει έναν-έναν τους χαρακτήρες του αρχικού προγράμματος και τους χωρίζει σε μια ακολουθία ατομικών λεκτικών μονάδων που

καλούνται tokens και που το καθένα παριστάνει μια σειρά χαρακτήρων που μπορούν να διαχειρίζονται σαν μοναδική λογική ενότητα. Π.χ. στην εντολή Fortran:

IF (5.EQ.MAX) GO TO 100

αναγνωρίζονται τα οκτώ tokens:

IF, **(**, **5**, **.EQ.**, **MAX**, **)**, **GOTO**, **100**

Κάθε token αποτελείται από δύο στοιχεία, τον **τύπο** (type) και την **τιμή** (Value) του η οποία μπορεί να είναι κενή για κάποιο τύπο (π.χ. σύμβολα στίξης, keywords).

Ο Λεκτικός Αναλυτής και ο Συντακτικός Αναλυτής συχνά ανήκουν στο ίδιο πέρασμα. Στην περίπτωση αυτή ο Λεκτικός Αναλυτής λειτουργεί είτε κάτω από τον Συντακτικό Αναλυτή σαν υπορουτίνα αυτού είτε και οι δύο μαζί σαν συν-ρουτίνες (coroutines). Ο Συντακτικός Αναλυτής ζητά από τον Λεκτικό Αναλυτή το επόμενο token όποτε χρειάζεται ένα, και ο Λεκτικός Αναλυτής του επιστρέφει ένα κώδικα για το token που βρήκε. Αν το token είναι identifier ή άλλο token το οποίο μπορεί να έχει κάποια τιμή τότε ο Λεκτικός Αναλυτής επιστρέφει στον Συντακτικό Αναλυτή και την **τιμή του**. Η τιμή αυτή είναι συνήθως ένας **δείκτης** που δείχνει την θέση του Πίνακα Συμβόλων όπου βρίσκεται η τιμή (value) του token. Ο δείκτης αυτός, ουσιαστικά επιστρέφεται στον Λεκτικό Αναλυτή από τις ρουτίνες που χειρίζονται τον πίνακα συμβόλων και οι οποίες κλήθηκαν από τον Λεκτικό Αναλυτή.

Για να προσδιορίσει ο Λεκτικός Αναλυτής το επόμενο token χρειάζεται να ψάξει αρκετούς χαρακτήρες πέρα από τον τρέχοντα χαρακτήρα. Π.χ. στην εντολή:

```
IF ( 5.EQ.MAX ) GO TO 100
     ↑
```

Το βέλος δείχνει τη θέση της αρχής του επόμενου token μετά από τα "IF", "(" που θεωρούμε ότι έχουν ήδη αναγνωρισθεί από τον Λεκτικό Αναλυτή και σταλεί στον Συντακτικό Αναλυτή.

Δραστηριότητα 1 / Κεφάλαιο 1

Για να αναγνωρισθεί το "5" σαν token ο Λεκτικός Αναλυτής πρέπει να ψάξει τους χαρακτήρες μέχρι και τον "Q". Μπορείτε να εξηγήσετε γιατί;

Ο Λεκτικός Αναλυτής αναγνωρίζοντας το "5" βλέπει τον επόμενο χαρακτήρα να είναι η τελεία ".", άρα δεν μπορεί να καταλήξει αν πρόκειται για την ακέραια σταθερά "5" ή για την πραγματική σταθερά "5.", προχωράει επομένως στον επόμενο

χαρακτήρα "E" και τώρα έχει μπροστά του την συμβολοσειρά "5.E". Η συμβολοσειρά αυτή είναι πιθανό να είναι μια πραγματική σταθερά σε εκθετική μορφή αρκεί ο επόμενος χαρακτήρας να είναι πρόσημο (+|-) ή ψηφίο. Πρέπει λοιπόν να διαβάσει και τον επόμενο χαρακτήρα πριν καταλήξει για τον τύπο του token που προσπαθεί να αναγνωρίσει. 'Ετσι, διαβάζει υποχρεωτικά και τον χαρακτήρα "Q" για να αναγνωρίσει τελικά ότι πρόκειται για την ακέραια σταθερά "5".

Μετά την αναγνώριση του "5" ο δείκτης του Λεκτικού Αναλυτή δείχνει την τελεία του EQ, και επιστρέφει στον Συντακτικό Αναλυτή το ζευγάρι : τύπος "constant" (σταθερά) και τιμή "5" ή ένα δείκτη στον Πίνακα Συμβόλων.

Μετά την πλήρη αναγνώριση της εντολής από τον Λεκτικό Αναλυτή η σειρά των tokens μοιάζει με την εξής:
[if,] [(,] [const, 341] [eq,] [id, 729] [ ),] [goto,] [label, 554]
όπου ο δείκτης μέσα στις τετραγωνικές παρενθέσεις δίνει την σχετική θέση στον Πίνακα Συμβόλων όπου φυλάσσονται οι πληροφορίες για τις σταθερές, τις μεταβλητές και τις επιγραφές (constants, variables, labels).

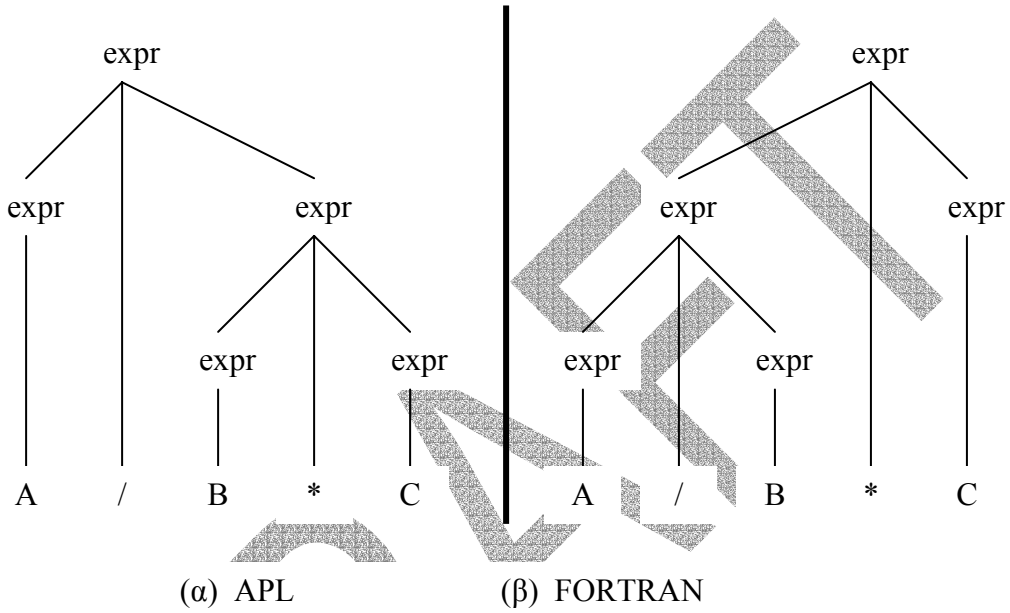
ΕΝΟΤΗΤΑ 1.5 ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

Δύο είναι οι βασικές λειτουργίες του Συντακτικού Αναλυτή. Ελέγχει ότι τα tokens που παίρνει από τον Λεκτικό Αναλυτή υπακούουν στις συντακτικές προδιαγραφές της γλώσσας και επιπλέον σχηματίζει και κάποια δενδρική μορφή για τις επόμενες φάσεις. Π.χ. σε Fortran πρόγραμμα η έκφραση A+/B, φθάνει στον Συντακτικό Αναλυτή σαν [id,n1] [+,] [/,] [id, n2] ο οποίος εντοπίζει δύο συνεχόμενους τελεστές (+, /) και απορρίπτει την έκφραση.

Η δενδρική μορφή (Parse Tree – Δένδρο Ανίχνευσης) π.χ. της έκφρασης **A/B*C** έχει δύο δυνατές παραστάσεις (α) και (β) (Σχήμα 1.4). Το ποια από τις δύο θα χρησιμοποιηθεί είναι συνάρτηση των προδιαγραφών της γλώσσας και πιο συγκεκριμένα, της συντακτικής δομής της γλώσσας. Ακριβέστερα, είναι συνάρτηση της προτεραιότητας και προσεταιριστικότητας των τελεστών όπως αυτές εκφράζονται μέσα από την τυπική σύνταξη της γλώσσας. Στο σχήμα 1.4 το (α), η έκφραση υπολογίζεται σαν A/(B*C) ενώ στο (β) σαν (A/B)*C. Το (α) είναι σωστό με βάση την σύνταξη της γλώσσας APL, και το (β) λάθος. Ενώ με βάση την σύνταξη της γλώσσας Fortran το (α) είναι λάθος και το (β) είναι σωστό. 'Οπως θα δούμε αργότερα οι

context-free γραμματικές (grammars) είναι ιδιαίτερα χρήσιμες για τον καθορισμό συντακτικών δομών. Και ακόμη, είναι δυνατόν για συγκεκριμένους τύπους context-free γραμματικών, να κατασκευασθούν αυτόματα οι αντίστοιχοι Συντακτικοί Αναλυτές. Στο σχήμα 1.5 φαίνεται το Δένδρο Ανίχνευσης της εντολής

if (5 **eq** MAX) **goto** 100, μέρος των περιεχομένων του Πίνακα Συμβόλων και τα tokens.



(α) APL (β) FORTRAN

Σχήμα 1.4. Δένδρα Ανίχνευσης της A / B * C

'Ασκηση Αυτοαξιολόγησης 3 / Κεφ. 1

Κατασκεύασε όλα τα πιθανά Δένδρα Ανίχνευσης της αριθμητικής έκφρασης:

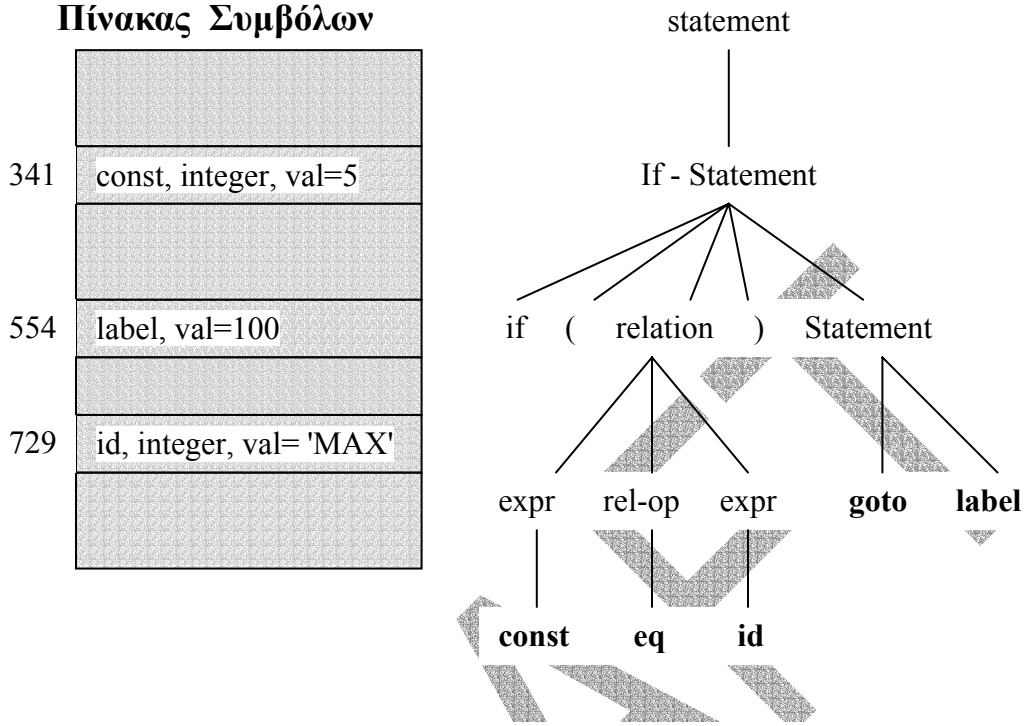
A * B / C

ΕΝΟΤΗΤΑ 1.6 ΔΗΜΙΟΥΡΓΙΑ ΕΝΔΙΑΜΕΣΟΥ ΚΩΔΙΚΑ

Η φάση της δημιουργίας ενδιάμεσου κώδικα μετασχηματίζει το Δένδρο Ανίχνευσης (parse tree) σε μια αναπαράσταση του αρχικού προγράμματος σε κάποια ενδιάμεση γλώσσα.

Ο Κώδικας τριών Διευθύνσεων (Three-address Code), είναι μια δημοφιλής μορφή ενδιάμεσης γλώσσας, όπου μια τυπική εντολή έχει τη μορφή:

A:=B **op** C



Tokens: [if,] [(,] [const, 341] [eq,] [id, 729] [),] [goto,] [label, 554]

Σχήμα 1.5. Ανίχνευση της εντολής **if** (5 **eq** MAX) **goto** 100

Το Δένδρο Ανίχνευσης (β) του Σχήματος 1.4 θα μπορούσε να μετασχηματιστεί στην εξής σειρά εντολών τριών-διευθύνσεων:

T_1:= A / B

T_2:= T_1 * C

όπου T_1 και T_2 είναι ονόματα προσωρινών μεταβλητών. Ακόμη η ενδιάμεση γλώσσα χρειάζεται να έχει και απλές εντολές JMP (ή GOTO), υπό συνθήκη και χωρίς συνθήκη, π.χ. **if** A **relop** B **goto** L). Υψηλότερου επιπέδου εντολές τύπου WHILE-DO, ή IF-THEN-ELSE, μεταφράζονται σε χαμηλότερες εντολές συνθήκης (conditional statements) τριών-διευθύνσεων.

Παράδειγμα 2 / Κεφ. 1 Η ακόλουθη εντολή WHILE,

While A>B **&** A<=2*B-5 **do** A:=A+B;

έχει την εξής αντιστοιχία σε σειρά από tokens:

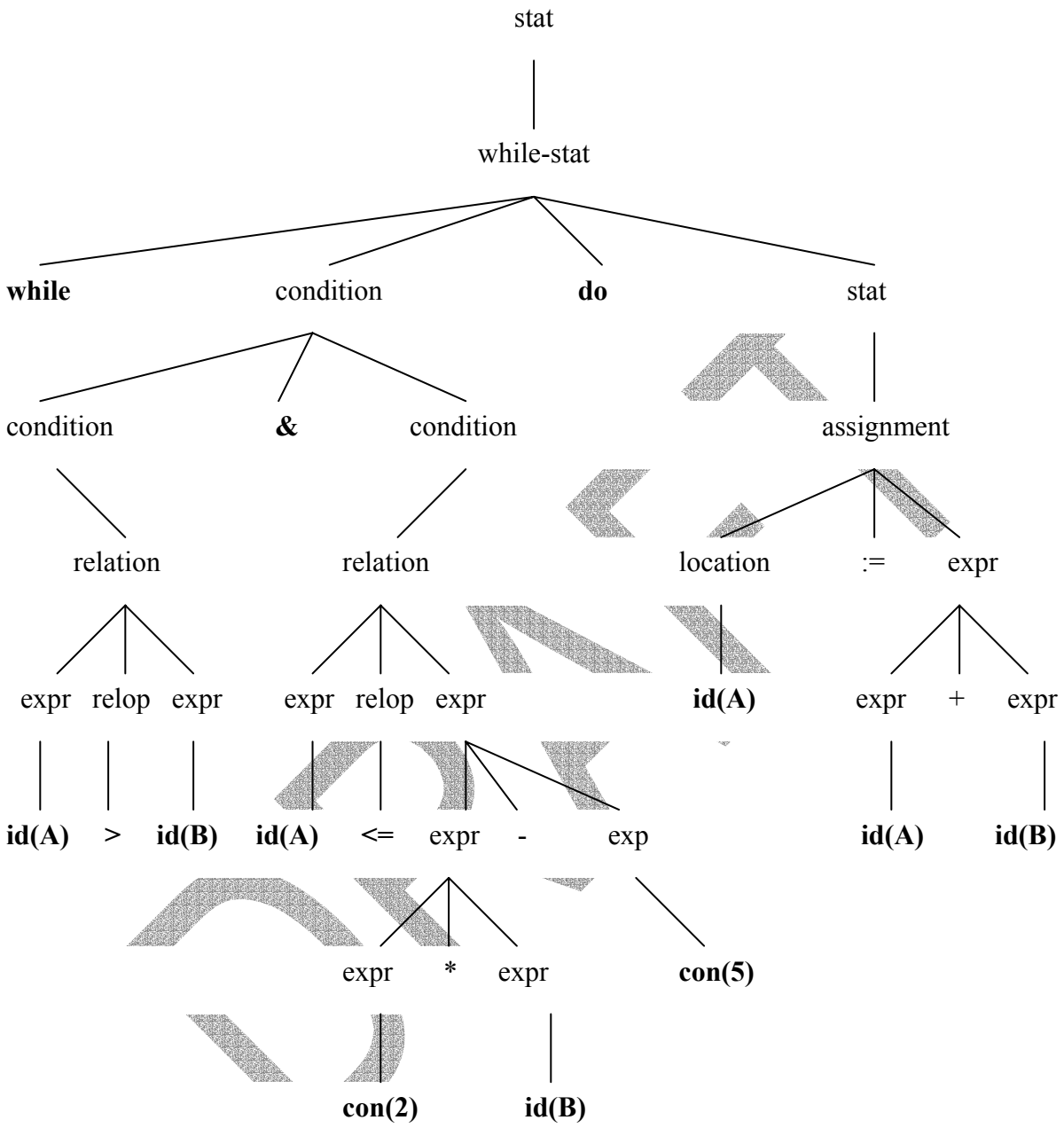
[**while**,] [**id**, n_1] [**relop**, **>**] [**id**, n_2] [**comprelop**, **&]** [**id**, n_1] [**relop**,**<=**]
[**con**, n_3] [*,] [**id**, n_2] [-,] [**con**, n_4] [**do**,] [**id**, n_1] [**=:**,] [**id**, n_1] [+,]
[**id**, n_2] [**;**,]

Το Δένδρο Ανίχνευσης για την παραπάνω εντολή WHILE είναι αυτό του Σχήματος 1.7. Οι αλγόριθμοι που μετατρέπουν το Δένδρο Ανίχνευσης σε ενδιάμεση γλώσσα (κώδικα) θα συζητηθούν αργότερα. Προς το παρόν παρουσιάζεται στο Σχήμα 1.6 ο ενδιάμεσος κώδικας όπως περίπου θα είναι:

```
Q1:  if A > B goto Q2
     goto Q3
Q2:  T1 := 2 * B
     T2:= T1 - 5
     if A <= T2 goto Q4
     goto Q3
Q4:  A := A + B
     goto Q1
Q3: …
```

Σχήμα 1.6. Ενδιάμεσος κώδικας για την εντολή WHILE.

Οι περισσότεροι Μεταγλωττιστές δεν δημιουργούν Δένδρα Ανίχνευσης άμεσα, αλλά δημιουργούν τον ενδιάμεσο κώδικα καθώς προχωράει η συντακτική ανάλυση.



Σχήμα 1.7. Δένδρο Ανίχνευσης για την εντολή **WHILE**

ΕΝΟΤΗΤΑ 1.7 ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ (OPTIMIZATION)

Η φάση της βελτιστοποίησης δεν υπάρχει σε όλους τους Μεταγλωττιστές παρ' ότι μπορεί και να διπλασιάσει την ταχύτητα του τελικού προγράμματος.

Τοπική βελτιστοποίηση (Local optimization) μπορεί να εφαρμοστεί στον ενδιάμεσο κώδικα του Σχήματος 1.6.

Η ακολουθία:

```
if A > B goto Q2
goto Q3
Q2:
```

μπορεί να αντικατασταθεί από την εντολή:

```
if A <= B goto Q3
Q2:
```

Μια ακόμη πιο σημαντική βελτιστοποίηση αναφέρεται στον υπολογισμό των δεικτών των εκφράσεων, π.χ. η εκχώρηση:

A[I] := B[I] + C[I]

εφ' όσον η μνήμη του υπολογιστή προσπελαύνεται κατά bytes και υπάρχουν 4 bytes/word, χρειάζεται να υπολογίσει το 4*(I-1) τρεις φορές.

'Ενας Optimizing Compiler θα μπορούσε να τροποποιήσει τον ενδιάμεσο κώδικα και να χρησιμοποιήσει το 4*(I-1) σαν κοινή υποέκφραση η οποία υπολογίζεται μια μόνο φορά, πράγμα το οποίο δεν θα μπορούσε να κάνει ένας προγραμματιστής γιατί δεν του το επιτρέπει η υψηλού επιπέδου γλώσσα στην οποία προγραμματίζει.

Η βελτιστοποίηση των βρόχων (Loop Optimization) δίνει την μεγαλύτερη αύξηση ταχύτητας στα προγράμματα μιας και αυτά καταναλώνουν τον περισσότερο χρόνο μέσα σε βρόχους. Η τυπική λύση είναι να μετακινηθούν οι υπολογισμοί που δίνουν πάντα το ίδιο αποτέλεσμα μέσα στον βρόχο, έξω και ακριβώς πριν από τον βρόχο. Οι υπολογισμοί αυτοί ονομάζονται *loop invariants*.

ΕΝΟΤΗΤΑ 1.8 ΔΗΜΙΟΥΡΓΙΑ ΚΩΔΙΚΑ (CODE GENERATION)

Σαν παράδειγμα στην φάση αυτή, η εντολή ενδιάμεσου Κώδικα

A:=B+C (1)

μπορεί να μεταφραστεί στον εξής Κώδικα μηχανής:

```
LOAD   B
ADD    C          (2)
STORE  A
```

Δραστηριότητα 2 / Κεφάλαιο 1

Μια τέτοια όμως μετάφραση του ενδιάμεσου κώδικα σε κώδικα μηχανής δημιουργεί πολλά LOAD και STORE που πλεονάζουν. Η κατάσταση μπορεί να βελτιωθεί σημαντικά αν ο Δ.Κ. (Δημιουργός Κώδικα) διατηρεί πληροφορίες για το περιεχόμενο των Καταχωρητών (registers), του Accumulator και της "επόμενης χρήσης" των ονομάτων (identifiers) και δημιουργεί εντολές LOAD και STORE μόνο όταν χρειάζονται.

Με αυτές τις υποθέσεις μπορείτε να εξηγήσετε πώς μπορεί να κατασκευασθεί "μικρότερος" κώδικας από τον (2) που δόθηκε παραπάνω;

(α) με την υπόθεση ότι το B βρίσκεται στον Accumulator και το A πρόκειται να χρησιμοποιηθεί σε επόμενη έκφραση (έχει επόμενη χρήση) αντί του (2) μπορώ να φτιάξω μόνο:

```
ADD    C        (3)
```

(β) αν δεν έχω πληροφορία για την επόμενη χρήση (next use) του A ή ξέρω ότι το A **δεν** έχει επόμενη εκχώρηση φτιάχνω:

```
ADD    C
STORE A
```

(γ) άλλοι συνδυασμοί είναι δυνατοί.

Σε ένα υπολογιστή με λίγους καταχωρητές υψηλής ταχύτητας, ο Δημιουργός Κώδικα θα προσπαθήσει να κάνει την βέλτιστη χρήση αυτών των καταχωρητών.

ΕΝΟΤΗΤΑ 1.9 ΔΙΑΧΕΙΡΙΣΗ ΠΙΝΑΚΩΝ ΚΑΙ ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΛΥΣΗ

'Ενας Μεταγλωττιστής χρειάζεται να διατηρεί πληροφορίες για όλα τα δεδομένα που εμφανίζονται στο πρόγραμμα. Π.χ. αν μια μεταβλητή παριστάνει έναν integer ή real αριθμό, το μέγεθος ενός Array, το πλήθος των παραμέτρων ενός υποπρογράμματος κ.λ.π.

Οι πληροφορίες αυτές συλλέγονται κατά τις αρχικές φάσεις -Λεκτικής Ανάλυσης και Συντακτικής Ανάλυσης - εισάγονται στον Πίνακα Συμβόλων, και έχουν διάφορες χρήσεις. Π.χ. αν υπάρχει η έκφραση A+B, όπου A είναι τύπου integer και B τύπου real, τότε αν το επιτρέπει η γλώσσα ο Μεταγλωττιστής πρέπει να φτιάξει κώδικα που να μετατρέπει τον A σε τύπου real, αλλιώς να καλέσει τις ρουτίνες χειρισμού λαθών για να αναφέρει το λάθος μίξης των τύπων.

Η παραπάνω λειτουργία εντάσσεται στην *Σημασιολογική Ανάλυση* (Semantic Analysis) η οποία εφαρμόζεται στον καθορισμό του τύπου των ενδιάμεσων αποτελεσμάτων, στον έλεγχο του τύπου των παραμέτρων και στον προσδιορισμό λειτουργίας που αναφέρεται από κάποιο τελεστή (π.χ. ο τελεστής + μπορεί να αναφέρεται σε ακέραια ή floating point πρόσθεση, σε λογικό "OR", ή και σε άλλη πράξη). Σημασιολογική Ανάλυση μπορεί να γίνει κατά τη φάση της Συντακτικής Ανάλυσης, της δημιουργίας ενδιάμεσου κώδικα, ή την τελική φάση της δημιουργίας κώδικα.

ΕΝΟΤΗΤΑ 1.10 ΔΙΑΧΕΙΡΙΣΗ ΛΑΘΩΝ (ERROR HANDLING)

Αποτελεί μια από τις πιο σημαντικές λειτουργίες του Μεταγλωττιστή. 'Οταν ο Μεταγλωττιστής ανακαλύψει κάποιο λάθος, πρέπει να το αναφέρει στον διαχειριστή λαθών ο οποίος εκδίδει κάποιο μήνυμα. Ακόμη ο Μεταγλωττιστής πρέπει να τροποποιήσει την είσοδο της φάσης στην οποία εμφανίστηκε το λάθος, ώστε η τελευταία να μπορέσει να συνεχίσει κανονικά.

'Ασκηση Αυτοαξιολόγησης 4 / Κεφ. 1

Εντοπίστε και διορθώστε τα λάθη που υπάρχουν στο παρακάτω κείμενο. "Ο Λεκτικός Αναλυτής διαβάζει το αρχικό πρόγραμμα σαν ένα σύνολο χαρακτήρων και

επιστρέφει τους χαρακτήρες αυτούς στον Συντακτικό Αναλυτή. Η δουλειά του Συντακτικού Αναλυτή, μετά την Σημασιολογική Ανάλυση είναι να ομαδοποιήσει τους χαρακτήρες σε tokens και τα tokens σε συντακτικές δομές σύμφωνα με τους συντακτικούς κανόνες της γραμματικής και να παράξει τον ενδιάμεσο κώδικα, υπό μορφή συντακτικού δένδρου, από το οποίο θα κατασκευασθεί ο τελικός κώδικας, ο οποίος στη συνέχεια μπορεί και να βελτιστοποιηθεί, από τον μεταγλωττιστή".

ΕΝΟΤΗΤΑ 1.11 ΕΡΓΑΛΕΙΑ ΜΕΤΑΓΛΩΤΤΙΣΤΩΝ

Διάφορα "εργαλεία" που έχουν αναπτυχθεί για να βοηθήσουν στην δημιουργία Μεταγλωττιστών ποικίλουν από Scanner και Parser **generators** σε πολύπλοκα συστήματα με διάφορα ονόματα όπως **Compiler-Compilers**, **Compiler generators**, ή **translator-writing systems**, και παράγουν ένα Μεταγλωττιστή από κάποια μορφή προδιαγραφών μιας πηγαίας γλώσσας και του υπολογιστή στον οποίο θα τρέχει ο Μεταγλωττιστής.

Οι προδιαγραφές στα συστήματα αυτά μπορούν να περιλαμβάνουν:

α. Περιγραφή της Λεκτικής και Συντακτικής δομής της γλώσσας.

β. Περιγραφή του τί έξοδο θα δημιουργεί κάθε δομικό στοιχείο της γλώσσας, (πχ κώδικας, διαχείριση λαθών) ,και

γ. Περιγραφή του στοχευόμενου υπολογιστή (target machine).

Οι βασικές βοήθειες-εργαλεία που παρέχονται από τους υπάρχοντες Compiler-Compilers είναι:

1. Γεννήτορες Λεκτικών Αναλυτών (Scanner Generators)
2. Γεννήτορες Συντακτικών Αναλυτών (Parser Generators), οι οποίοι προσφέρουν περισσότερη αξιοπιστία από τους Συντακτικούς Αναλυτές που κατασκευάζονται "με το χέρι".
3. Βοήθειες για Code Generation.

Συχνά ένας Compiler-Compiler διαθέτει μια υψηλού επιπέδου γλώσσα για τον ενδιάμεσο κώδικα, τον Assembler, ή τον κώδικα μηχανής. Ο χρήστης γράφει ρουτίνες σ' αυτή τη γλώσσα και στον τελικό Μεταγλωττιστή οι ρουτίνες καλούνται στις κατάλληλες στιγμές από τον αυτόματα δημιουργούμενο Συντακτικό Αναλυτή.

ΕΝΟΤΗΤΑ 1.12 BOOTSTRAPPING

'Ενας Μεταγλωττιστής χαρακτηρίζεται από τρεις γλώσσες: την πηγαία, την τελική και την γλώσσα στην οποία είναι γραμμένος. Οι γλώσσες αυτές μπορεί να είναι όλες διαφορετικές. Π.χ. ένας Μεταγλωττιστής μπορεί να τρέχει σε κάποια μηχανή και να παράγει τελικό κώδικα (Object Code) για μια άλλη μηχανή. Αυτός καλείται **Cross-Compiler** και παραδείγματα τέτοιων υπάρχουν σε μεγάλα συστήματα (μηχανές) που παράγουν κώδικα για να τρέξει σε μικρότερα (π.χ. σε PC).

Bootstrapping είναι η διαδικασία κατασκευής ενός Μεταγλωττιστή για μια γλώσσα X και ο οποίος είναι γραμμένος στη γλώσσα X(!) και παράγει κώδικα για μια μηχανή A.

Για την παρακάτω συζήτηση χρησιμοποιείται ο συμβολισμός:

$$C_Z^{X\,Y} \quad \text{ή} \quad C_Z^{X \to Y}$$

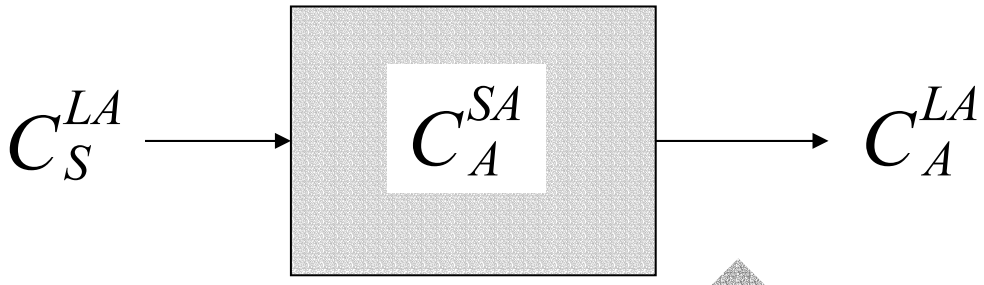
που σημαίνει: Μεταγλωττιστής για την γλώσσα X, γραμμένος στην γλώσσα Z ο οποίος παράγει κώδικα στην γλώσσα (μηχανής ή assembly) Y.

Υποθέτουμε ότι έχουμε μια γλώσσα L, για την οποία θέλουμε ένα Μεταγλωττιστή στη μηχανή A.

Γράφουμε κατ' αρχήν για τη μηχανή A ένα "μικρό" Μεταγλωττιστή $C_A^{S_L A}$ ο οποίος μεταφράζει προγράμματα γραμμένα σε ένα **υποσύνολο** S της L σε κώδικα μηχανής ή Assembly A **και είναι γραμμένος σε A**.

Κατόπιν γράφουμε ένα Μεταγλωττιστή $C_{S_L}^{LA}$ στο υποσύνολο S της L. 'Οταν ο $C_{S_L}^{LA}$ περάσει από τον $C_A^{S_L A}$ που **τρέχει** στη μηχανή A, το αποτέλεσμα είναι ένας Μεταγλωττιστής C_A^{LA} για ολόκληρη την γλώσσα L που μπορεί να τρέχει στη μηχανή A.

Διαγραμματικά:



Η διαδικασία αυτή μπορεί να επαναληφθεί σε 2-3 βήματα διαλέγοντας κάθε φορά και ένα μεγαλύτερο υποσύνολο της γλώσσας S.

Σύνοψη Κεφαλαίου 1

'Εχοντας πλέον μελετήσει το κεφάλαιο αυτό θα πρέπει να έχετε αποκτήσει μια συνοπτική μεν αλλά πλήρη εικόνα για το τι είναι οι μεταφραστές και ειδικότερα οι Μεταγλωττιστές και οι Διερμηνευτές που κατά κύριο λόγο μας ενδιαφέρουν.

Ξεκινήσαμε με την διαδικασία μετάφρασης–φόρτωσης-εκτέλεσης ενός πηγαίου προγράμματος έτσι ώστε να σχηματίσετε μια εικόνα των διαδικασιών που είναι απαραίτητες να γίνουν μέχρι να είναι σε θέση να εκτελεσθεί στον υπολογιστή ένα πρόγραμμα το οποίο θα έχετε γράψει σε κάποια γλώσσα προγραμματισμού. Στη συνέχεια είδατε επιγραμματικά τη δομή ενός μεταγλωττιστή να αποτελείται από ένα σύνολο δραστηριοτήτων που τις αναφέραμε σαν ‘φάσεις’ και οι οποίες είναι οι εξής: λεκτική ανάλυση, συντακτική ανάλυση, σημασιολογική ανάλυση, δημιουργία ενδιάμεσου κώδικα, βελτιστοποίηση ενδιάμεσου κώδικα, και δημιουργία τελικού κώδικα. Επειδή οι δραστηριότητες αυτές αν γίνονται μια-μια επιβαρύνουν πάρα πολύ τον χρόνο που απαιτείται για να ολοκληρωθεί η διαδικασία μεταγλώττισης, είδαμε πως μπορούμε να ενσωματώσουμε τις δραστηριότητες αυτές (φάσεις) σε ένα ή περισσότερα ‘περάσματα’ κάνοντας χρήση της τεχνικής της ‘οπισθομπάλωσης’. Την τεχνική αυτή την είδατε να εφαρμόζεται στην περίπτωση ενός Assembler.

Είδατε επίσης την δομική και λειτουργική διαφορά ανάμεσα στους μεταγλωττιστές και τους διερμηνευτές. Είδατε δηλαδή ότι από άποψη δομής μπορούμε να θεωρήσουμε ότι και οι δυο αποτελούνται από δυο τμήματα το ‘εμπρόσθιο τμήμα’ και το ‘οπίσθιο τμήμα’. Το εμπρόσθιο τμήμα και στις δυο περιπτώσεις εκτελεί τις ίδιες λειτουργίες και καταλήγει στη δημιουργία του ‘ενδιάμεσου κώδικα υποθετικής μηχανής’(υποθετικού υπολογιστή). Αντίθετα το μεν οπίσθιο τμήμα του μεταγλωττιστή από αυτόν τον ενδιάμεσο κώδικα δημιουργεί τον τελικό κώδικα, το δε οπίσθιο τμήμα του διερμηνευτή κάνει εξομείωση της υποθετικής μηχανής και εκτέλεση του ενδιάμεσου κώδικα.

Στη συνέχεια περιγράψαμε με λίγα λόγια όλες τις φάσεις. 'Ετσι μάθατε ότι ο Λεκτικός Αναλυτής ομαδοποιεί τους χαρακτήρες του πηγαίου προγράμματος σε λεκτικές δομές που τις ονομάσαμε ‘tokens’ και ακόμη ότι ο ίδιος λειτουργεί σαν υπορουτίνα του Συντακτικού Αναλυτή. Ο Συντακτικός Αναλυτής ομαδοποιεί τα tokens σε ‘συντακτικές’ δομές οι οποίες μπορούν να αναπαρασταθούν με τη βοήθεια των ‘δένδρων ανίχνευσης’ ή με ‘κώδικα τριών διευθύνσεων’ (και τα δυο μπορείτε να τα θεωρήσετε σαν μορφές ενδιάμεσης γλώσσας -κώδικας- της υποθετικής μηχανής). Επάνω σε αυτόν τον ενδιάμεσο κώδικα γίνονται οι παρεμβάσεις που αφορούν την σημασιολογική ανάλυση (για παράδειγμα πρόσθεση επιπλέον κώδικα για την μετατροπή των τύπων των ‘εντελών’ σε μια αριθμητική έκφραση). Στη συνέχεια μάθατε μερικά είδη βελτιστοποίησης του ενδιάμεσου κώδικα όπως η ‘τοπική βελτιστοποίηση’ και η βελτιστοποίηση ‘βρόχων’ και είδατε πως μπορείτε να δημιουργήσετε ‘τελικό’ κώδικα μηχανής από τον ενδιάμεσο κώδικα λαμβάνοντας βεβαίως υπόψιν την ‘αρχιτεκτονική’ του ΄στοχευόμενου’ υπολογιστή. Η ‘διαχείριση’ του ‘Πίνακα Συμβόλων’ και η ‘Διαχείριση Λαθών’ είναι δυο άλλες πολύ σημαντικές λειτουργίες τόσο της μεταγλώττισης όσο και της διερμήνευσης. Αυτές δεν αναφέρονται σαν ξεχωριστές φάσεις αλλά σαν διεργασίες οι οποίες γίνονται σε κάθε μία από τις φάσεις.

Ακόμη, μάθατε και ότι υπάρχουν κάποια εργαλεία τα οποία μας βοηθάνε στην κατασκευή μεταγλωττιστών και διερμηνευτών και τα οποία αναφέρονται σαν ΄γεννήτορες’ λεκτικών και συντακτικών αναλυτών. Τέλος, είδατε και μια ειδική περίπτωση διαδικασίας κατασκευής ενός μεταγλωττιστή η οποία αναφέρεται σαν ‘bootstrapping’.

Απαντήσεις Ασκήσεων Αυτοαξιολόγησης του Κεφαλαίου 1

Απάντηση 'Ασκησης 1

Από άποψη δομής, ένας μεταγλωττιστής και ένας διερμηνευτής μπορούν να θεωρηθούν ότι αποτελούνται από δύο ξεχωριστά τμήματα, το εμπρόσθιο (front-end) και το οπίσθιο (back-end) τμήμα. Το εμπρόσθιο τμήμα είναι ίδιο και στις δύο περιπτώσεις. 'Ομως, το οπίσθιο μέρος ενός διερμηνευτή, διερμηνεύει / εκτελεί την ενδιάμεση μορφή κώδικα η οποία παρήχθη από το εμπρόσθιο μέρος του, ενώ σε ένα μεταγλωττιστή το οπίσθιο μέρος του παράγει το εκτελέσιμο πρόγραμμα το οποίο θα εκτελεσθεί από τον υπολογιστή όταν το ζητήσει ο χρήστης.

Απάντηση 'Ασκησης 2

α) Συχνά παίρνουμε αυτή την απάντηση που όμως δεν είναι η σωστή διότι ενώ αρκετές φορές γίνεται πράγματι πιο μικρός σε μέγεθος (κώδικας-αριθμός εντολών) ο μεταγλωττιστής εν τούτοις ο στόχος είναι αυτός της απάντησης (γ) δηλαδή να επιταχυνθεί ο συνολικός χρόνος μετάφρασης ενός προγράμματος από τον μεταγλωττιστή.

β) Ασφαλώς και δεν είναι έτσι. Το αντίθετο μάλιστα συμβαίνει, δηλαδή για να υλοποιηθεί η οπισθομπάλωση (μέσω της οποίας υλοποιείται η μείωση των περασμάτων), μάλλον λιγότερο τμηματοποιημένη γίνεται η δομή του μεταγλωττιστή.

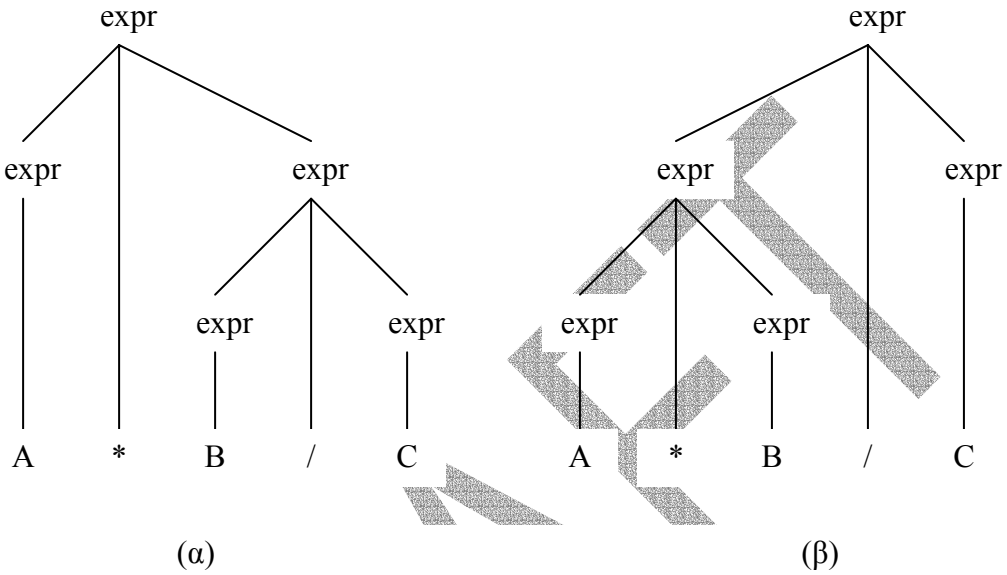
γ) Πολύ σωστά. Πράγματι ο στόχος είναι να μειωθεί ο χρόνος μετάφρασης του προγράμματός σας. Δηλαδή, να μειωθεί ο χρόνος που χρειάζεται ο μεταγλωττιστής να μεταφράσει το πρόγραμμά σας από πηγαίο σε τελικό κώδικα, πράγμα που επιτυγχάνεται από μεταγλωττιστές οι οποίοι κάνουν τα λιγότερα δυνατόν περάσματα πάνω στις διάφορες εσωτερικές μορφές του προγράμματός σας.

δ) Αν επιλέξατε αυτή την απάντηση, τότε μάλλον έχετε μπερδέψει τον χρόνο μετάφρασης με τον χρόνο εκτέλεσης ενός προγράμματος. Ο χρόνος μετάφρασης σχετίζεται με την μείωση των περασμάτων ενώ ο χρόνος εκτέλεσης σχετίζεται με την ύπαρξη ή μή της φάσης βελτιστοποίησης ενός μεταγλωττιστή.

ε) Αν επιλέξατε αυτή την απάντηση μάλλον διαβάσατε επιπόλαια και πιθανόν να πρέπει να μελετήσετε επιπλέον και την Θεματική Ενότητα "Τεχνολογία Λογισμικού".

Απάντηση 'Ασκησης 3

Για την παραπάνω έκφραση μπορείς να κατασκευάσεις τα παρακάτω δύο τουλάχιστον δένδρα ανίχνευσης. Π.χ.



Οι συντακτικοί κανόνες που περιγράφουν την σύνταξη μιας γλώσσας προγραμματισμού, είναι τέτοιοι που να επιτρέπουν την κατασκευή ενός μόνο δένδρου, αλλιώς λέμε ότι η γραμματική είναι διφορούμενη και επομένως δεν είναι κατάλληλη να χρησιμοποιηθεί, για την κατασκευή ενός μεταγλωττιστή .

Στην προκειμένη περίπτωση δεν σου έχει δοθεί σύνταξη της γραμματικής την οποία θα έπρεπε να χρησιμοποιήσεις, επομένως σωστά έφτιαξες δύο διαφορετικά δένδρα. Στο κεφάλαιο της Συντακτικής Ανάλυσης θα δούμε πώς γράφουμε μη διφορούμενες γραμματικές κατάλληλες για την κατασκευή μεταγλωττιστή.

Απάντηση 'Ασκησης 4

Στο κείμενο αυτό υπάρχουν σοβαρά και λιγότερο σοβαρά λάθη. Η φράση "επιστρέφει τους χαρακτήρες αυτούς στον Συντακτικό Αναλυτή ", περιέχει ένα πολύ σοβαρό λάθος. Ο Λεκτικός Αναλυτής δεν επιστρέφει χαρακτήρες αλλά, tokens τα οποία είναι κωδικοποιημένη μορφή των λεκτικών δομών της γλώσσας και τα οποία αντιπροσωπεύουν εν γένει σύνολα χαρακτήρων ομαδοποιημένα από τον Λεκτικό

Αναλυτή με βάση τους Λεκτικούς Κανόνες της γραμματικής. Δεύτερο σοβαρό λάθος είναι η φράση "… μετά την Σημασιολογική Ανάλυση". Η Σημασιολογική Ανάλυση ακολουθεί την Συντακτική Ανάλυση και δεν προηγείται αυτής. Λιγότερο σημαντικό λάθος είναι η φράση "… υπό μορφή Συντακτικού Δένδρου… ", ενώ το σωστό είναι "σε κάποια ενδιάμεση μορφή όπως τριάδες, τετράδες, κώδικας postfix , ή Δένδρου Ανίχνευσης.

Προσοχή επίσης, στη διάκριση μεταξύ Συντακτικού Δένδρου και Δένδρου Ανίχνευσης. Τα Συντακτικά Δένδρα είναι διαφορετικά από τα Δένδρα Ανίχνευσης.