Λεκτικός αναλυτής

Γεώργιος Μανής
Πανεπιστήμιο Ιωαννίνων
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Φεβρουάριος 2022

Η λεκτική ανάλυση αποτελεί την πρώτη φάση της μεταγλώττισης. Κατά τη φάση αυτή, διαβάζεται το *αρχικό πρόγραμμα* (το οποίο συνηθίζεται να ονομάζεται και *πηγαίο πρόγραμμα*) και παράγονται οι λεκτικές μονάδες. Χρησιμοποιούμε τον όρο *λεκτική μονάδα* για να αναπαραστήσουμε οτιδήποτε έχει νόημα να θεωρηθεί ως αυτόνομο σύνολο συνεχόμενων χαρακτήρων που μπορεί να συναντηθεί σε ένα πρόγραμμα και βρίσκει σημασιολογία στην υπό υλοποίηση γλώσσα. Θα τις ονομάζαμε ίσως *λέξεις* σε μία φυσική γλώσσα, αλλά σε ένα πρόγραμμα μας ενδιαφέρει να εντοπίσουμε και ακολουθίες χαρακτήρων, όπως `==`, `+=`, `(`, `}`, άρα ο όρος *λέξη* δεν είναι δόκιμος.

Η λεκτική ανάλυση συνήθως γίνεται με τη χρήση αυτοματοποιημένων εργαλείων. Πολύ γνωστό εργαλείο ανάπτυξης λεκτικών αναλυτών, και από τα πρώτα που έγιναν διαθέσιμα και χρησιμοποιήθηκαν ευρέως, είναι το μέτα-εργαλείο *lex*. Το μεταγενέστερο *antlr* είναι, επίσης, μία πολύ καλή επιλογή, περισσότερο ώριμη προσέγγιση και ευκολότερη για τον προγραμματιστή. Θα ασχοληθούμε αρκετά με αυτό στο αντίστοιχο κεφάλαιο, παρακάτω (κεφ. **??**), όταν θα εξετάσουμε εργαλεία αυτοματοποιημένης ανάπτυξης. Σε σημερινές γλώσσες προγραμματισμού, όπως η Java και η Python, υπάρχουν διαθέσιμες βιβλιοθήκες που απλοποιούν σημαντικά την υλοποίηση ενός λεκτικού αναλυτή. Σε τέτοιες βιβλιοθήκες θα αναφερθούμε επίσης, στο ίδιο κεφάλαιο.

Στο κεφάλαιο αυτό μας ενδιαφέρει να μελετήσουμε τις βασικές έννοιες και μηχανισμούς της λεκτικής ανάλυσης, έχοντας ως κύριο στόχο την κατανόηση όλων των λεπτομερειών της ανάπτυξης ενός λεκτικού αναλυτή. Έτσι, χωρίς να χρησιμοποιηθεί κανένα εργαλείο αυτοματοποιημένης ανάπτυξης, αλλά χρησιμοποιώντας μια γλώσσα προγραμματισμού υψηλού επιπέδου ως το μόνο μέσο υλοποίησης, θα δούμε βήμα-βήμα την ανάπτυξη ενός λεκτικού αναλυτή, χωρίς να θεωρήσουμε ως δεδομένη κάποια ενότητα κώδικα.

1.1 Ο λεκτικός αναλυτής στη διαδικασία της μετάφρασης

Στο εισαγωγικό κεφάλαιο είδαμε την αλληλουχία των φάσεων της ανάπτυξης ενός μεταγλωττιστή. Εκεί, η λεκτική ανάλυση αποτελούσε χρονικά την πρώτη φάση της ανάπτυξης. Εάν, όμως, σχεδιάζαμε ένα διάγραμμα το οποίο να δείχνει τη δομή ενός μεταγλωττιστή με βάση τη διάρθρωση των μονάδων λογισμικού που το αποτελούν, τότε θα εμφανιζόταν σαν μία μονάδα λογισμικού η οποία καλείται από τον συντακτικό αναλυτή.

Ο λεκτικός αναλυτής υλοποιείται σαν μία συνάρτηση, η οποία διαβάζει έναν-έναν τους χαρακτήρες από την είσοδο (το αρχικό πρόγραμμα) και κάθε φορά που καλείται επιστρέφει την επόμενη διαθέσιμη λεκτική μονάδα. Στην περίπτωση που ο λεκτικός αναλυτής αναγνωρίσει κάποιο σφάλμα, τότε οφείλει να πληροφορήσει σχετικά τον συντακτικό αναλυτή ή να διακόψει τη μεταγλώττιση και να επιστρέψει στον χρήστη ένα διαφωτιστικό μήνυμα λάθους. Η αλληλεπίδραση του λεκτικού αναλυτή με το περιβάλλον του φαίνεται στο σχήμα 1.1, όπου έχει επιλεγεί η δεύτερη λύση, όπως και στο υπόλοιπο κεφάλαιο.

Σχήμα 1.1: Ο λεκτικός αναλυτής

Όταν ο λεκτικός αναλυτής αναγνωρίσει μία λεκτική μονάδα, τότε την επιστρέφει στον συντακτικό αναλυτή, επιστρέφοντας του και τον έλεγχο. Μία λεκτική μονάδα είναι ένα αντικείμενο το οποίο περιέχει τρία πεδία. Το πρώτο είναι η συμβολοσειρά την οποία αναγνώρισε. Ας την ονομάσουμε *recognized_string*. Το δεύτερο εντάσσει τη συμβολοσειρά αυτή σε μία κατηγορία η οποία έχει νόημα για την συντακτική ανάλυση. Ας του δώσουμε το όνομα *family*. Τέτοιες κατηγορίες μπορεί να είναι οι ακόλουθες:

- *identifier*: ονόματα μεταβλητών, συναρτήσεων, διαδικασιών, σταθερών και ό,τι άλλο μπορεί να έχει ονομασία σε ένα πρόγραμμα. Στη φάση αυτή είναι πολύ νωρίς για να μπορέσουμε να διαχωρίσουμε αν η συμβολοσειρά που αναγνωρίσαμε αποτελεί όνομα μεταβλητής, συνάρτησης, διαδικασίας ή σταθεράς, ακόμα κι αν είναι νόμιμη (π.χ. δηλωμένη μεταβλητή ή όχι) και για τον λόγο αυτό δημιουργούμε μία γενική κατηγορία, κάτω από την οποία τοποθετούμε όλες αυτές τις περιπτώσεις.
- *number*: αριθμητικές σταθερές
- *keyword*: περιέχει τις λέξεις κλειδιά της γλώσσας, π.χ.: `while`, `if`, `else`
- *id*: περιέχει ό,τι θα μπορούσε να αποτελέσει ονομασία σε ένα πρόγραμμα, δηλαδή το όνομα μιας μεταβλητής, συνάρτησης, διδικασίας, συμβολικής σταθεράς ή το όνομα του προγράμματος
- *addOperator*: προσθετικοί αριθμητικοί τελεστές: `+`, `-`
- *mulOperator*: πολλαπλασιαστικοί αριθμητικοί τελεστές: `*`, `/`
- *relOperator*: λογικοί τελεστές π.χ.: `==`, `>=`, `<`, `<>`
- *assignment*: τελεστής εκχώρησης (`:=`)
- *delimiter*: διαχωριστές π.χ.: `,`, `.`, `;`
- *groupSymbol*: σύμβολα ομαδοποίησης π.χ.: `(`, `)`, `{`, `}`, `[`, `]`

Θα μπορούσαμε, φυσικά, να ορίσουμε πολλές ακόμα κατηγορίες, ανάλογα με τις ανάγκες της γλώσσας, όπως, η συμβολοσειρά, οι τελεστές σε bit (bitwise operators), μαθηματικές συναρτήσεις, κ.λ.π.

Ένα ακόμα πεδίο του αντικειμένου που αναπαριστά μια λεκτική μονάδα είναι ο αριθμός γραμμής στην οποία αναγνωρίστηκε. Η πληροφορία αυτή είναι χρήσιμη στον συντακτικό αναλυτή, έτσι ώστε να μπορεί να επιστρέψει εύστοχα μηνύματα σφάλματος, αν εντοπίσει κάποιο σφάλμα κατά τη συντακτική ανάλυση. Αν αυτή η πληροφορία δεν διατηρηθεί μέσα στο αντικείμενο που επιστρέφεται στον συντακτικό αναλυτή, τότε θα χαθεί,

αφού ο συντακτικός αναλυτής δεν χρησιμοποιεί άλλη πληροφορία, πέρα από αυτήν που του επιστρέφεται από τον συντακτικό αναλυτή. Ας το ονομάσουμε `line_number`.

Έτσι, μια λεκτική μονάδα μπορεί να αναπαρασταθεί με ένα αντικείμενο της κλάσης `Token`:

```
class Token:
# properties: recognized\_string, family, line_number
def __init__(recognized\_string, family, line_number):
    self.recognized\_string = recognized\_string
    self.family = family
    self.line_number = line_number
```

Ένα αντικείμενο της κλάσης `Token` περιέχει την πληροφορία που μεταφέρει ο λεκτικός αναλυτής στον συντακτικό αναλυτή. Για να δούμε πώς ο λεκτικός αναλυτής δημιουργεί και επιστρέφει στον συντακτικό αναλυτή τα `token` ας μελετήσουμε αναλυτικά την εσωτερική του λειτουργία.

1.2 Η εσωτερική λειτουργία ενός λεκτικού αναλυτή

Ο λεκτικός αναλυτής υλοποιείται από μία συνάρτηση η οποία καλείται από τον συντακτικό αναλυτή και κάθε φορά που καλείται επιστρέφει την επόμενη διαθέσιμη στην είσοδο λεκτική μονάδα. Αυτό σημαίνει ότι σε κάθε του κλήση, ο λεκτικός αναλυτής σημειώνει το σημείο στο οποίο σταματάει την ανάγνωση του αρχικού προγράμματος και την επόμενη φορά που καλείται συνεχίζει την ανάγνωση από το σημείο αυτό.

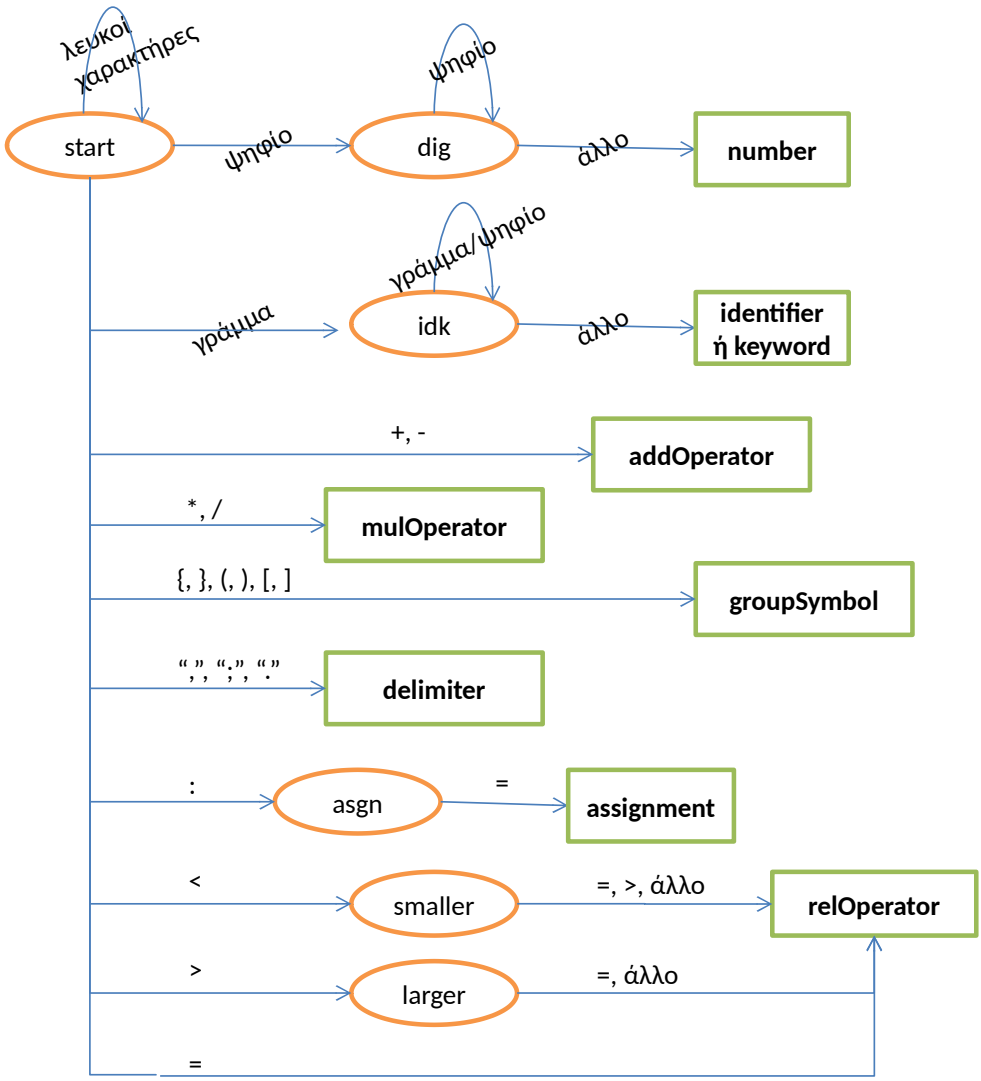
Η λειτουργία του βασίζεται σε ένα αυτόματο. Για την *C-imple* το αυτόματο που αναγνωρίζει τις λεκτικές μονάδες φαίνεται στο σχήμα 1.2. Κάθε τέτοιο αυτόματο αποτελείται από την αρχική κατάσταση, κάποιες ενδιάμεσες και κάποιες τελικές. Στο αυτόματο του σχήματος η αρχική κατάσταση είναι η κατάσταση *start*, οι μη τελικές καταστάσεις συμβολίζονται με έλλειψη, ενώ οι τελικές με παραλληλόγραμμο. Ας ακολουθήσουμε το αυτόματο να δούμε με ποιο τρόπο αναγνωρίζει τις λεκτικές μονάδες της *C-imple*.

Με την κλήση του λεκτικού αναλυτή το αυτόματο αρχικοποιείται στην κατάσταση *start*. Εάν στην είσοδο εμφανιστεί κάποιο ψηφίο, τότε το αυτόματο μεταβαίνει στην κατάσταση *dig*, η οποία είναι μη τελική. Όσο στην είσοδο εμφανίζονται ψηφία, τότε για κάθε ψηφίο που διαβάζεται, γίνεται μία μετάβαση, η οποία όμως δεν αλλάζει την κατάσταση του αυτομάτου, αφού είναι μετάβαση από την κατάσταση *dig* στην κατάσταση *dig*. Από την *dig* θα φύγει μόλις έρθει κάτι διαφορετικό, κάτι *άλλο* που δεν είναι ψηφίο. Όταν συμβεί αυτό το αυτόματο θα μεταβεί στην τελική κατάσταση *number* και θα έχει αναγνωρίσει μία αριθμητική (ακέραια) σταθερά. Στον συντακτικό αναλυτή θα επιστραφεί η αριθμητική σταθερά σαν `recognized_string`, σαν `family` η οικογένεια στην οποία ανήκει το `token`, που την περίπτωση αυτή είναι η οικογένεια `number` και φυσικά ο αριθμός γραμμής στον οποίο βρέθηκε η αριθμητική σταθερά, το *line_number*.

Αν από την αρχική κατάσταση, αντί για ψηφίο έρθει γράμμα, τότε σύμφωνα με το αυτόματο θα μεταβούμε στην κατάσταση *idk*. Όμοια με την προηγούμενη περίπτωση, το αυτόματο θα παραμείνει στην κατάσταση *idk*, όσο στην είσοδο εμφανίζεται γράμμα ή ψηφίο. Μόλις εμφανιστεί κάτι διαφορετικό, κάτι που δεν είναι ούτε γράμμα ούτε ψηφίο, το αυτόματο μεταβαίνει στην τελική κατάσταση *identifier/keyword*.

Από το όνομά της κατάστασης μπορούμε να καταλάβουμε ότι εδώ δεν έχουμε αναγνωρίσει (τουλάχιστον όχι ακόμα) κάποιο αναγνωριστικό(identifier), δηλαδή το όνομα μιας μεταβλητής, μίας συνάρτησης, κλπ., αφού στην τελική κατάσταση αυτή θα καταλήξει το αυτόματο και στην περίπτωση που θα συναντήσει κάποια λέξη κλειδί της γλώσσας. Έτσι, πριν βιαστούμε να επιστρέψουμε στον συντακτικό αναλυτή το αποτέλεσμα, πρέπει πρώτα να γίνει ο έλεγχος αν αυτό που αναγνωρίσαμε είναι λέξη κλειδί, που ανήκει στην κατηγορία *keyword* ή είναι πράγματι ένας *intentifier* και ως τέτοιον πρέπει να τον χαρακτηρίσουμε.

Στο υπόλοιπο αυτόματο τα πράγματα είναι πιο ομαλά. Αν από την αρχική κατάσταση έρθει κάποιος από τους χαρακτήρες `+`, `-` τότε θα μεταβούμε αμέσως στην τελική κατάσταση *addOperator*. Αν έρθει κάποιο από τα `*`, `/` τότε πηγαίνουμε στην κατάσταση `mulOperator`. Αν έρθει κάποιο από τα `(`, `)`, `{`, `}`, `[`, `]`, τότε θα μεταβούμε στην τελική κατάσταση *groupSymbol*, ενώ αν έρθει ένα από τα `,`, `;„` `.`, τότε καταλήγουμε στην *delimiter*.



Σχήμα 1.2: Το αυτόματο λειτουργίας του λεκτικού αναλυτή

Ξεχωριστή κατηγορία αποτελεί το σύμβολο εκχώρησης, το οποίο απαιτεί μία μη τελική κατάσταση (την *asgn*) πριν φτάσει στην τελική *assignment*, αφού αποτελείται από δύο ακριβώς χαρακτήρες.

Παρόμοια περίπτωση είναι αυτή με τους λογικούς τελεστές. Εκτός από τον τελεστή ισότητας, εμφάνιση του οποίου μας οδηγεί αμέσως στην τελική κατάσταση *relOperator*, για τους υπόλοιπους λογικούς τελεστές πρέπει να είμαστε πιο προσεκτικοί. Αν αναγνωρίσουμε, για παράδειγμα, το σύμβολο <, αυτό δεν σημαίνει ότι μπορούμε να πάμε με ασφάλεια σε τελική κατάσταση, διότι είναι πιθανό να ακολουθεί το σύμβολο = ή το σύμβολο >, οπότε η λεκτική μονάδα που θα πρέπει να αναγνωριστεί θα είναι <= την πρώτη φορά και <> τη δεύτερη.

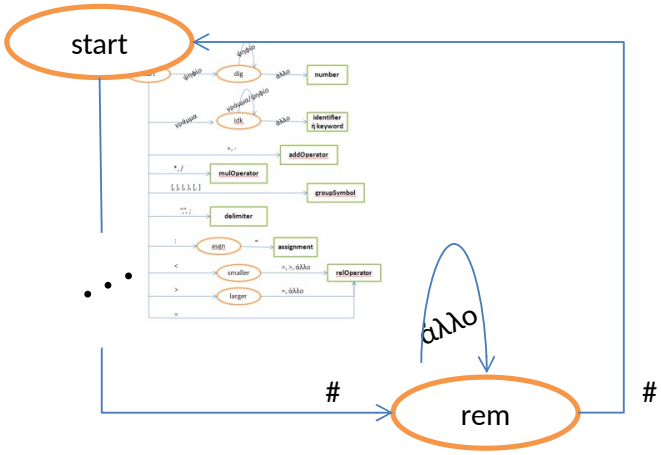
Μάλιστα, στην περίπτωση αυτή θα πρέπει να προσέξουμε και κάτι ακόμα. Ενώ στις περισσότερες περιπτώσεις, όποιο σύμβολο διαβάζουμε το ενσωματώνουμε σε κάποια λεκτική μονάδα, εδώ για να καταλάβουμε αν το σύμβολο που αναγνωρίσαμε ήταν το < και όχι το <> ή το <=, θα πρέπει να διαβάσουμε έναν χαρακτήρα ο οποίος ή είναι λευκός χαρακτήρας ή ανήκει στην επόμενη λεκτική μονάδα. Αν είναι λευκός χαρακτήρας δεν δημιουργείται πρόβλημα. Αν όμως ανήκει στην επόμενη λεκτική μονάδα πρέπει να είμαστε προσεκτικοί ώστε να μην τον χάσουμε.

Στο παράδειγμα `a>bc`, για να είμαστε βέβαιοι ότι ο λογικός τελεστής είναι το < και όχι κάποιος άλλος, πρέπει να διαβάσουμε και το `b`. Όταν όμως ξανακληθεί ο λεκτικός αναλυτής, η ανάγνωση της εισόδου πρέπει να αρχίσει από το `b`, το οποίο έχει ήδη διαβαστεί, και όχι από το `c`, το οποίο είναι το επόμενο σύμβολο στην είσοδο. Πρέπει δηλαδή, όταν χρησιμοποιήσουμε το `b`, με κάποιο τρόπο να το επιστρέψουμε πάλι στη θέση του

στην είσοδο ή να βρούμε μία εναλλακτική ισοδύναμη λύση.

Εκτός από τους λογικούς τελεστές που είδαμε παραπάνω, το ίδιο συμβαίνει στην αναγνώριση των κατηγοριών *identifier*, *keyword* και *number*, αφού και εκεί, για να αναγνωρίσουμε μία λεκτικη μονάδα, είμαστε υποχρεωμένοι να *κρυφοκοιτάξουμε* σε χαρακτήρα που δεν ανήκει στη λεκτική μονάδα, έτσι ώστε να γνωρίζουμε ότι η υπό αναγνώριση λεκτική μονάδα ολοκληρώθηκε. Στο αυτόματο του σχήματος 1.2 μπορείτε να εντοπίσετε τις περιπτώσεις αυτές εκεί που εμφανίζεται η λέξη *άλλο*.

Το μόνο σημείο το οποίο δεν έχουμε συζητήσει καθόλου ακόμα είναι οι λευκοί χαρακτήρες. Αν βρισκόμαστε στην αρχική κατάσταση και στην είσοδο βρεθεί ένας λευκός χαρακτήρας, τότε παραμένουμε στην αρχική κατάσταση. Πρακτικά, ο λευκός χαρακτήρας αγνοείται και συνεχίζουμε την ανάγνωση της εισόδου. Αφού δεν έχουμε φτάσει σε τελική κατάσταση, ο λεκτικός αναλυτής δεν τερματίζει και δεν επιστρέφει αποτέλεσμα. Αυτό θα γίνει όταν αναγνωριστεί λεκτική μονάδα. Όταν ο λευκός χαρακτήρας που συναντήσαμε είναι η *αλλαγή γραμμής*, τότε πρέπει να ενημερωθεί ο μετρητής γραμμών, ο οποίος μας χρησιμεύει για να επιστέψουμε στον συντακτικό αναλυτή το πεδίο `line_number` της κλάσης `Token`.



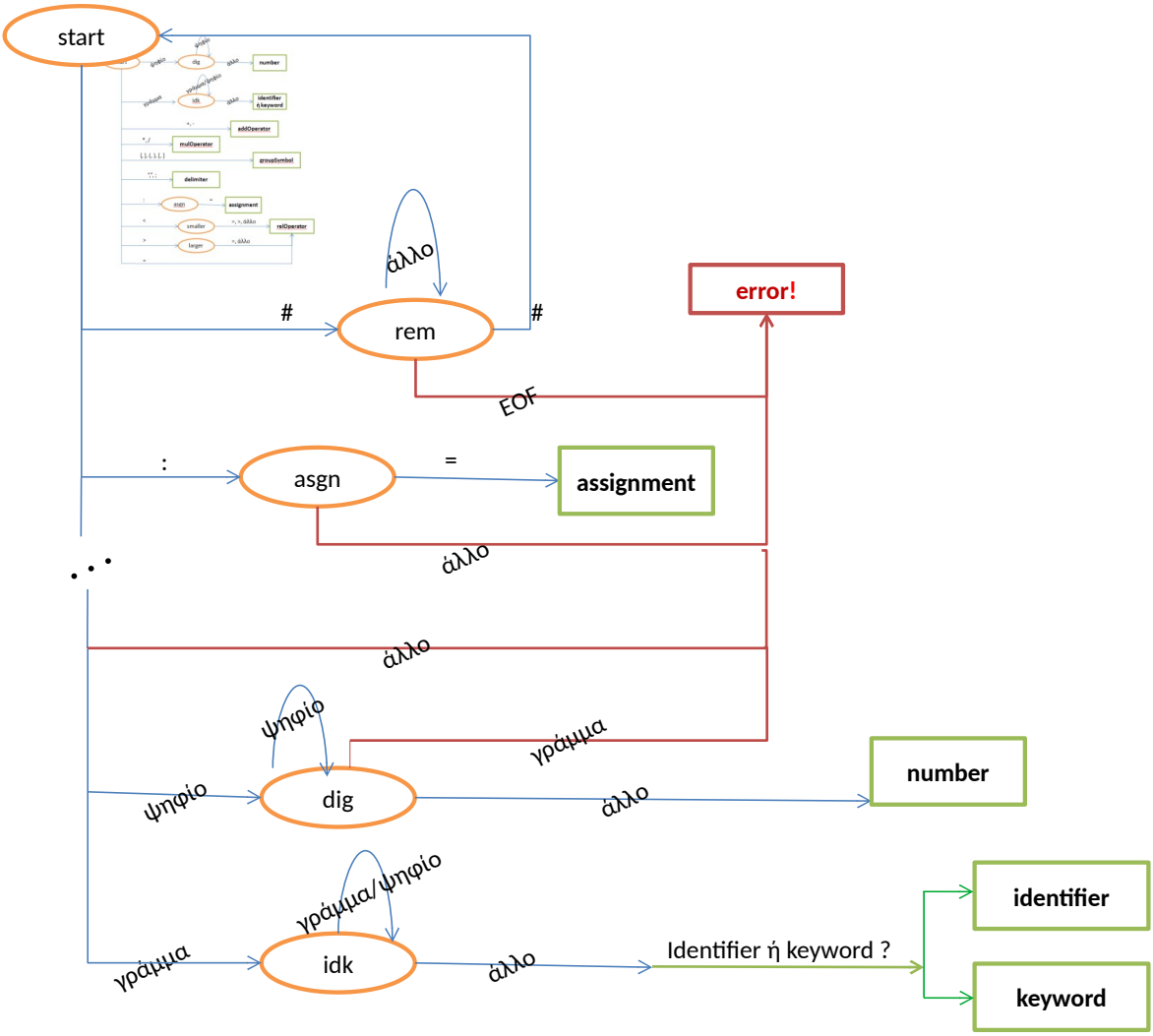
Σχήμα 1.3: Διαχείριση σχολίων από τον λεκτικό αναλυτή

Πριν κλείσουμε την περιγραφή της λειτουργίας του λεκτικού αναλυτή, ας αφιερώσουμε και μία παράγραφο στα σχόλια. Ο σχολιασμός του κώδικα πρέπει να είναι αναπόσπαστο τμήμα της συγγραφής του. Τα σχόλια τα διαχειρίζεται αποκλειστικά ο λεκτικός αναλυτής, τα αναγνωρίζει και τα αφαιρεί από το πρόγραμμα. Ο συντακτικός αναλυτής δεν χρειάζεται καν να πληροφορηθεί την ύπαρξή τους. Στην *C-imple* τα σχόλια αρχίζουν και τελειώνουν με τον χαρακτήρα #. Όταν βρισκόμαστε στην αρχική κατάσταση και εμφανιστεί ο χαρακτήρας # τότε μεταβαίνουμε στην κατάσταση *rem*. Εκεί παραμένουμε όσο στην είσοδο δεν εμφανίζεται πάλι ο χαρακτήρας #. Μόλις εμφανιστεί ο χαρακτήρας #, μεταβαίνουμε πάλι στην αρχική κατάσταση, όπου συνεχίζουμε με την αναγνώριση της επόμενης λεκτικής μονάδας. Αφού δεν έχουμε φτάσει σε τελική κατάσταση, ο λεκτικός αναλυτής δεν τερματίζει και δεν επιστρέφει αποτέλεσμα. Πρέπει πάλι να προσέξουμε με την αλλαγή γραμμής, αφού όταν βρισκόμαστε μέσα στα σχόλια, παρόλο που πρακτικά δεν αναγνωρίζουμε λεκτικές μονάδες πρέπει να ενημερώνουμε τον μετρητή γραμμών όταν συναντάμε αλλαγή γραμμής. Στο σχήμα 1.3 φαίνεται πώς θα ενσωματώσουμε τα σχόλια στο αυτόματο του σχήματος 1.2.

1.3 Διαχείριση σφαλμάτων

Στη φάση της λεκτικής ανάλυσης αναγνωρίζονται τα πρώτα από τα σφάλματα που μπορεί να ανακαλύψει ένας μεταγλωττιστής. Φυσικά, πρόκειται για σφάλματα τα οποία σχετίζονται με λεκτικές μονάδες. Σφάλματα που σχετίζονται με σύνταξη (π.χ. παράλειψη παρένθεσης) ή σημασιολογικά (π.χ. μία μεταβλητή δεν έχει δηλωθεί) θα εντοπιστούν σε φάσεις που θα ακολουθήσουν.

Τα σφάλματα αυτής της φάσης προκύπτουν από συνθήκες που μπορεί να εμφανιστούν κατά τη διάσχιση



Σχήμα 1.4: Διαχείριση σφαλμάτων από τον λεκτικό αναλυτή

του αυτόματου. Θα χρησιμοποιήσουμε το αυτόματο για την *C-imple* (σχήμα 1.2) για να εντοπίσουμε τέτοιες περιπτώσεις. Κάθε γλώσσα, φυσικά, έχει τις δικές της ιδιαιτερότητες, οπότε η διαχείριση σφαλμάτων θα πρέπει να προσαρμοστεί ανάλογα. Ας δούμε, όμως, ποια σφάλματα είναι δυνατόν να εντοπίσουμε με βάση το αυτόματο του σχήματος 1.2. Το εμπλουτισμένο αυτόματο που προκύπτει φαίνεται στο σχήμα 1.4, το οποίο και θα αναλύσουμε.

- Ενώ βρισκόμαστε στην κατάσταση *start* εμφανίζεται στην είσοδο ένας χαρακτήρας που δεν ταιριάζει σε καμία από τις επιλογές που φαίνεται να υπάρχουν για την κατάσταση αυτή. Ο μόνος λόγος που μπορεί να συμβεί αυτό είναι γιατί εμφανίστηκε χαρακτήρας που δεν ανήκει στη γλώσσα. Στην περίπτωση αυτή, όχι μόνο έχουμε εντοπίσει σφάλμα, αλλά ξέρουμε ακριβώς και ποιο είναι αυτό. Μπορούμε να εμφανίζουμε ένα αρκετά διαφωτιστικό μήνυμα, κάτι σαν: *Εμφανίστηκε ο χαρακτήρας xx στη γραμμή yy, ο οποίος δεν ανήκει στη γλώσσα*. Παρατηρήστε ότι όταν εμφανίζεται μη νόμιμος χαρακτήρας, αυτό θα συμβεί πάντοτε όταν είμαστε στην αρχική κατάσταση *start*. Δείτε το ακόλουθο παράδειγμα, στο οποίο ως είσοδος εμφανίζεται η συμβολοσειρά `wh!le`. Η πρώτη κλήση του λεκτικού αναλυτή θα εντοπίσει τη λεκτική μονάδα `wh`, κρυφοκοιτάζοντας, αλλά όχι καταναλώνοντας τον χαρακτήρα `!`. Η δεύτερη κλήση του λεκτικού αναλυτή είναι αυτή που θα εντοπίσει το `!` και θα επιστρέψει το μήνυμα σφάλματος.

- Ενώ βρισκόμαστε στην κατάσταση *asgn* και αναμένουμε να έρθει ο χαρακτήρας `=`, εμφανίζεται κάποιος άλλος χαρακτήρας. Σε κάποια άλλη γλώσσα ίσως αυτό ήταν επιτρεπτό, οπότε θα έπρεπε να προσαρμο-

στούμε ανάλογα, όμως στην *C-imple* δεν είναι. Άρα εδώ εντοπίζουμε ένα ακόμα πιθανό σφάλμα, για το οποίο είμαστε πάλι σε θέση να εμφανίσουμε ένα διαφωτιστικό μήνυμα σφάλματος.

- Ενώ βρισκόμαστε στην κατάσταση *rem* έχουν ανοίξει δηλαδή σχόλια, εμφανίζεται τέλος του αρχείου εισόδου. Έχουμε δηλαδή σχόλια τα οποία ανοίγουν, αλλά δεν κλείνουν ποτέ.

- Τέλος, στη λεκτική ανάλυση θα κατατάσσαμε και την αναγνώριση λεκτικών μονάδων που, ενώ ξεκινούν από ψηφίο, μέσα στη λεκτική μονάδα εμφανίζονται γράμματα. Αυτό είναι κάτι που θα μπορούσε να αναγνωριστεί και να οδηγήσει σε σφάλμα και κατά τη διάρκεια της συντακτικής ανάλυσης. Είναι φανερό, όμως, ότι σαν σφάλμα είναι λεκτικό και πρέπει να ενταχθεί εδώ. Μάλιστα, το μήνυμα το οποίο έχουμε τη δυνατότητα να εμφανίσουμε τώρα είναι πολύ περισσότερο διαφωτιστικό.

- Σύμφωνα με την περιγραφή της *C-imple*, υπάρχουν δύο περιορισμοί, οι οποίοι θα πρέπει να ελεγχθούν: α) εάν το μήκος ενός *identifier* είναι το πολύ 30 χαρακτήρες και εάν μία ακέραια αριθμητική σταθερά βρίσκεται μέσα στο επιτρεπτό εύρος τιμών. Έτσι, στις καταστάσεις *identifier*, *keyword* και *number* του σχήματος 1.2 θα γίνουν οι απαραίτητοι έλεγχοι. Στην περίπτωση που οι έλεγχοι είναι επιτυχημένοι, τότε θα έχουμε επιτυχή αναγνώριση και επιστροφή του αποτελέσματος στον συντακτικό αναλυτή. Εάν ο έλεγχος δεν είναι επιτυχημένος, τότε ο λεκτικός αναλυτής θα οδηγείται σε κατάσταση σφάλματος.

1.4 Παράδειγμα εκτέλεσης

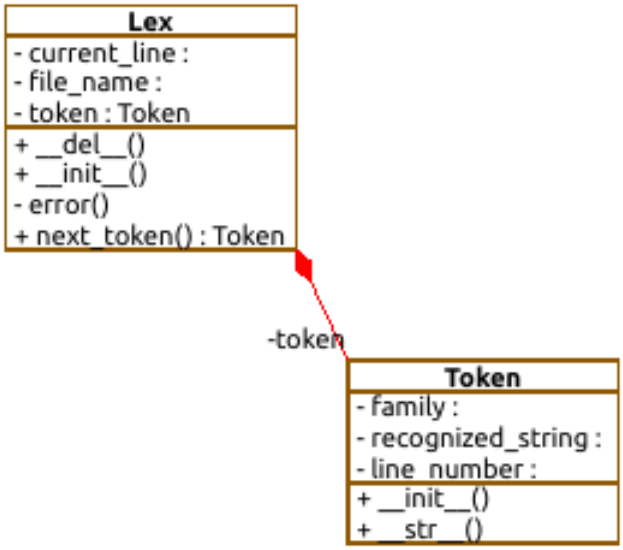
Το πρόγραμμα *factorial.ci* του κεφαλαίου **??** επιστρέφει τις ακόλουθες λεκτικές μονάδες:

```
program     family:"keyword",  line: 1
factorial   family:"id",  line: 1
{           family:"groupSymbol",  line: 2
declare     family:"keyword",  line: 4
x           family:"id",  line: 4
;           family:"delimiter",  line: 4
declare     family:"keyword",  line: 5
i           family:"id",  line: 5
,           family:"delimiter",  line: 5
fact        family:"id",  line: 5
;           family:"delimiter",  line: 5
input       family:"keyword",  line: 8
(           family:"groupSymbol",  line: 8
x           family:"id",  line: 8
)           family:"groupSymbol",  line: 8
;           family:"delimiter",  line: 8
fact        family:"id",  line: 9
:=          family:"assignment",  line: 9
1           family:"number",  line: 9
;           family:"delimiter",  line: 9
i           family:"id",  line: 10
:=          family:"assignment",  line: 10
1           family:"number",  line: 10
;           family:"delimiter",  line: 10
while       family:"keyword",  line: 11
(           family:"groupSymbol",  line: 11
i           family:"id",  line: 11
<=          family:"relOperator",  line: 11
x           family:"id",  line: 11
```

```
)           family:"groupSymbol",  line: 11
{           family:"groupSymbol",  line: 12
fact        family:"id",  line: 13
:=          family:"assignment",  line: 13
fact        family:"id",  line: 13
*           family:"mulOperator",  line: 13
i           family:"id",  line: 13
;           family:"delimiter",  line: 13
i           family:"id",  line: 14
:=          family:"assignment",  line: 14
i           family:"id",  line: 14
+           family:"addOperator",  line: 14
1           family:"number",  line: 14
;           family:"delimiter",  line: 14
}           family:"groupSymbol",  line: 15
;           family:"delimiter",  line: 15
print       family:"keyword",  line: 16
(           family:"groupSymbol",  line: 16
fact        family:"id",  line: 16
)           family:"groupSymbol",  line: 16
;           family:"delimiter",  line: 16
}           family:"groupSymbol",  line: 17
.           family:"delimiter",  line: 17
```

1.5 Διάγραμμα κλάσεων λεκτικού αναλυτή

Ένα πιθανό διάγραμμα κλάσεων του λεκτικού αναλυτή θα μπορούσε να περιέχει δύο κλάσεις. Η μία θα ορίζει την κλάση `Token`, για τη δημιουργεί αντικείμενα που θα αναπαριστούν λεκτικές μονάδες. Τα πεδία της είναι τα `recognized_string`, `family` και `line_number`, που έχουμε συζητήσει παραπάνω.



Σχήμα 1.5: Διάγραμμα κλάσεων του λεκτικού αναλυτή

Η δεύτερη κλάση είναι αυτή που θα δημιουργήσει το αντικείμενο του λεκτικού αναλυτή. Το πρώτο πεδίο της θα διατηρεί το όνομα του αρχείου του αρχικού προγράμματος, το δεύτερο τη γραμμή στην οποία βρισκόμαστε σε κάθε στιγμή της μετάφρασης και το τρίτο τη λεκτική μονάδα που δημιουργεί και θα επιστρέψει ως

αποτέλεσμα στον συντακτικό αναλυτή.

Το διάγραμμα κλάσεων του λεκτικού αναλυτή φαίνεται στο σχήμα 1.5. Σε κάθε φάση ανάπτυξης θα εμπλουτίζουμε το διάγραμμα αυτό, ώστε να φτάσουμε στο τέλος να έχουμε το συνολικό διάγραμμα του μεταφραστή.