PLY602–Μεταγλωττιστές Ι
Αναφορά Εργασίας

Ακαδημαϊκό Έτος 2018-2019

Μέλη Ομάδας
Ανδρουτσόπουλος Γεώργιος
ΑΜ: 2933
Καρβέλης Χριστόφορος
ΑΜ: 2989

Διδάσκων
Μανής Γεώργιος

1. Εισαγωγή

Στην εργασία του Ακαδημαϊκού Έτους 2018-2019 στο μάθημα των Μεταγλωττιστών Ι ζητήθηκε η ανάπτυξη του μεταγλωττιστή (compiler) για την γλώσσα **Starlet** η οποία είχε δημιουργηθεί από το διδάσκοντα για τους σκοπούς του μαθήματος. Η εργασία ήταν χωρισμένη σε τρεις φάσεις-στάδια. Το πρώτο στάδιο αφορούσε την υλοποίηση της λεκτικής και συντακτικής ανάλυσης του μεταγλωττιστή. Το δεύτερο στάδιο περιελάμβανε την παραγωγή του ενδιάμεσου κώδικα καθώς και την σημασιολογική ανάλυση. Τέλος στο τρίτο και τελευταίο στάδιο έγινε η παραγωγή του τελικού κώδικα σε συμβολική γλώσσα (assembly) MIPS. Στην παρούσα αναφορά περιγράφονται όλα αυτά τα στάδια και τονίζονται τα βασικά τους σημεία.
Αρχικά (**Ενότητα 2**) αναφέρονται οι διάφορες δομές δεδομένων που χρησιμοποιήθηκαν ώστε να αποθηκεύσουν τα απαραίτητα δεδομένα που χρειαζόταν ο μεταγλωττιστής για την ορθή του λειτουργία. Στην συνέχεια (**Ενότητα 3**) περιγράφονται οι κλάσεις που χρησιμοποιήθηκαν ώστε να ξεχωρίσουν τα πιο σημαντικά δομικά χαρακτηριστικά του μεταγλωττιστή, όπως οι οντότητες (entities) του πίνακα συμβόλων.
Στις **ενότητες 4 έως και 9** περιγράφονται οι βασικές λειτουργίες που κάνει ένας μεταγλωττιστής κατά την μεταγλώττιση ενός προγράμματος καθώς και το πως υλοποιήθηκαν αυτές στην δική μας περίπτωση.
Τέλος (**Ενότητα 10**) παρουσιάζεται ο τρόπος με τον οποίο έγινε η διαχείριση λαθών, που μπορεί να προκύψουν κατά την μεταγλώττιση ενός προγράμματος της συγκεκριμένης γλώσσας.
Ο μεταγλωττιστής αναπτύχθηκε σε γλώσσα προγραμματισμού **Python 3** λόγο της ευελιξίας που προσφέρει με την εύκολη διαχείριση δομών ευρετηρίου αλλά και για την δυνατότητα αντικειμενοστραφούς προγραμματισμού.

2. Δομές Δεδομένων του Μεταγλωττιστή

Οι συναρτήσεις του μεταγλωττιστή συνοδεύονται από ένα σύνολο δομών δεδομένων οι οποίες οργανώνουν τα δεδομένα που χρειάζεται ο μεταγλωττιστής κατά την λειτουργία του και κάνουν εύκολη την προσπέλασή τους. Κάποιες από τις δομές αυτές είναι οι εξής:

- **decision_table:** Πρόκειται για έναν δισδιάστατο πίνακα που αποτελεί την πιο σημαντική δομή όσον αφορά το κομμάτι της αρχικής επεξεργασίας των όρων του αρχείου πηγαίου κώδικα. Ο συγκεκριμένος πίνακας έχει ως στόχο της προσομοίωση του αυτομάτου καταστάσεων που χρησιμοποιείται κατά την λεκτική ανάλυση του μεταγλωττιστή. Περισσότερα για τον πίνακα αυτόν αναφέρονται στην **Ενότητα 4**.

- **keywords_map:** Πρόκειται για μία δομή ευρετηρίου η οποία ταιριάζει τα διαφορετικά keywords της γλώσσας που μπορεί να πάρει κάποιο tokentk (σύμβολο που διαβάζεται από το αρχείο πηγαίου κώδικα), με το id (idtk) που αντιστοιχεί στο καθένα.

- **statement_starters_set:** Η συγκεκριμένη δομή συνόλου κρατάει όλα τα keyword της γλώσσας τα οποία αποτελούν εναρκτήρια keywords κάποιας δομής-κώδικα της γλώσσας. Για παράδειγμα κρατάει το **if** από το **if-then[-else]-endif**, το **loop** από το **loop-endloop** και άλλα. Η δομή αυτή παίζει βασικό ρόλο κατά την διαχείριση των σφαλμάτων χρόνου μεταγλώττισης.

- **C_oper_map:** Πρόκειται για μία δομή ευρετηρίου η οποία αντιστοιχίζει τους τελεστές της γλώσσας **Starlet** σε τελεστές της γλώσσας **C**. Η χρήση της δομής αυτής γίνεται από την συνάρτηση **make_C_like_file()** η οποία είναι υπεύθυνη να παράξει το ισοδύναμο πρόγραμμα σε γλώσσα προγραμματισμού **C**, με βάση τον ενδιάμεσο κώδικα της **Starlet**.

- **symbols_map:** Πρόκειται για μία δομή ευρετηρίου η οποία χρησιμοποιείται κατά την λεκτική ανάλυση μαζί με τον πίνακα απόφασης (**decision_table**) για να καθορίσει την επόμενη μετάβαση στο αυτόματο με βάση το tokentk εισόδου. Το συγκεκριμένο ευρετήριο έχει ως κλειδιά τα διαφορετικά σύμβολα και τελεστές που μπορεί να αποδεχτεί η γλώσσα Starlet κατά το διάβασμα του αρχείου πηγαίου κώδικα. Σε κάθε ένα από τα κλειδιά αυτά αντιστοιχεί μία ακέραια τιμή και ένα id. Η ακέραια τιμή χρησιμοποιείται για να καθοριστεί η επόμενη μετάβαση στον πίνακα απόφασης αν διαβαστεί από τον λεκτικό αναλυτή το συγκεκριμένο σύμβολο. Το id (idtk) χρησιμοποιείται από τις διάφορες συναρτήσεις του μεταγλωττιστή ώστε να μπορούν να αναφερθούν στο σύμβολο αυτό.

- **mode_map:** Πρόκειται για ένα ευρετήριο που αντιστοιχίζει τους διαφορετικούς τύπους λαθών όπως αυτοί χρησιμοποιούνται από τις συναρτήσεις του μεταγλωττιστή, στο ακριβές όνομα αυτού του λάθους όπως θα εμφανιστεί στον χρήστη. Η δομή αυτή χρησιμοποιείται κατά την διαχείριση λαθών μεταγλώττισης από την **print_error_line()** ώστε να εμφανιστεί στον χρήστη ο τύπος του λάθους που προέκυψε.

- **par_mode_map:** Πρόκειται για ένα ευρετήριο που αντιστοιχίζει τους διαφορετικούς τύπους περάσματος παραμέτρων στις συνάρτησες της γλώσσας **Starlet** όπως αυτοί εμφανίζονται στον πηγαίο κώδικα, με τα id που έχουν επιλεχθεί για αυτούς στις τετράδες του ενδιάμεσου κώδικα.

- **mips_instr_map:** Πρόκειται για μία δομή ευρετηρίου η οποία αντιστοιχίζει τους διαφορετικούς τελεστές λογικών και αριθμητικών πράξεων της γλώσσας **Starlet** στις αντίστοιχες εντολές της συμβολικής γλώσσας **MIPS**. Η δομή αυτή χρησιμοποιείται κατά την παραγωγή του τελικού κώδικα ώστε με δεδομένο έναν συγκεκριμένο τελεστή μιας τετράδας του ενδιάμεσου κώδικα να παράγουμε την κατάλληλη εντολή **assembly** που αντιστοιχεί σε αυτόν τον τελεστή.

- **error_map:** Πρόκειται για ένα 3-επίπεδο που αποτελεί τον πυρήνα της διαχείρισης λαθών κατά την μεταγλώττιση. Το πρώτο του επίπεδο πραγματοποιεί έναν διαχωρισμό του συνόλου των λαθών που μπορεί να συμβούν στις τρεις βασικές κατηγορίες: λάθη κατά την **Λεκτική Ανάλυση**, λάθη κατά την **Συντακτική Ανάλυση**, λάθη κατά την **Σημασιολογική Ανάλυση**. Κάθε μία κατηγορία από αυτές διαχωρίζεται εκ νέου στο 2ο επίπεδο του ευρετηρίου ως προς το id του συμβόλου εισόδου που προκάλεσε το σφάλμα -αν πρόκειται για λάθος της Λεκτικής Ανάλυσης- ή ως προς την συνάρτηση του μεταγλωττιστή στην οποία εμφανίστηκε το λάθος - αν πρόκειται για λάθος της Συντακτικής ή της Σημασιολογικής Ανάλυσης. Τέλος το 3ο και τελευταίο επίπεδο του ευρετηρίου διαχωρίζει τα διαφορετικά λάθη του δεύτερου επιπέδου με βάση ένα id που έχει επιλεχθεί ώστε να χαρακτηρίσει επακριβώς το συγκεκριμένο λάθος. Αξίζει να αναφέρουμε ότι μπορεί να χρησιμοποιείται το ίδιο id λάθους όμως όχι στην ίδια κατηγορία λάθους του δευτέρου επιπέδου. Κάθε ένα από αυτά τα **id λάθους** ταιριάζουν σε ένα **μήνυμα λάθους** το οποίο χρησιμοποιείται από τις συναρτήσεις διαχείρισης λαθών (βλέπε **Ενότητα 10**) ώστε να ενημερωθεί ο χρήστης για τον λόγο που η μεταγλώττιση δεν ολοκληρώθηκε με επιτυχία. Επιπλέον κάποια από αυτά τα μηνύματα λάθους έχουν ειδικούς διαμορφωτές αλφαριθμητικών ώστε να εμφανίζουν στον χρήστη πληροφορίες όπως την συνάρτηση της γλώσσας **Starlet** στην οποία βρέθηκε μέσα το λάθος, όπου αυτό είναι εφικτό.

Οι υπόλοιπες δομές που υπάρχουν όπως ο πίνακας συμβόλων αναφέρονται στις αντίστοιχες ενότητες όπου συζητιούνται οι μηχανισμοί που τις χρησιμοποιούν.

3. Κλάσεις του Μεταγλωττιστή

Τα βασικότερα συστατικά ενός μεταγλωττιστή οργανώθηκαν σε κλάσεις της **Python** ώστε η διαχείριση τους και η αναφορά σε αυτά μέσα στο πρόγραμμα να είναι ξεκάθαρη. Οι κλάσεις που δημιουργήθηκαν είναι οι εξής:

- **class Quad :** Πρόκειται για την κλάση τα αντικείμενα της οποία αναπαριστούν τις τετράδες του ενδιάμεσου κώδικα. Αποτελείται από 5 πεδία, στο πρώτο (op) αποθηκεύεται ο τελεστής της τετράδας του ενδιάμεσου κώδικα που μπορεί να είναι ένας από τα {+, -, /, *, :=, >, <, =, >=, <=, <>, inp, out, begin_block, end_block, call, halt}. Ο τελεστής αυτός χαρακτηρίζει και τον τύπο της τετράδας (περισσότερα στην **Ενότητα 6**). Τα επόμενα τρία πεδία (term0, term1, term2) αφορούν τα άλλα τρία στοιχεία της τετράδας του ενδιάμεσου κώδικα και διαφέρουν ανάλογα με τον τελεστή της τετράδας. Τέλος το πεδίο label χρησιμοποιείται ώστε να συσχετίζεται η κάθε τετράδα με μία ετικέτα που περιέχει τον αύξον αριθμό δημιουργίας της συγκεκριμένης τετράδας.

- **class Entity :** Πρόκειται για την κλάση τα αντικείμενα της οποίας αναπαριστούν τους διαφορετικούς τύπους οντοτήτων που μπορεί να βρίσκονται μέσα στον πίνακα συμβόλων. Η κλάση έχει ως πεδία το όνομα, τον τύπο και το βάθος φωλιάσματος που έχει μία οντότητα στον πηγαίο κώδικα. Χωρίς να θέλουμε να μπούμε σε πολύ τεχνικές λεπτομέρειες απλός αναφέρουμε ότι η συγκεκριμένη κλάση κληρονομεί την κλάση του **module ABC** της **Python** ώστε να γίνει abstract και όλες όσες την κληρονομούν να υλοποιούν την μέθοδο **to_string()** για λόγους **debugging mode**.

- **class Variable :** Πρόκειται για μία υποκλάση της κλάσης Entity και συγκεκριμένα για την κλάση τα αντικείμενα της οποία αποτελούν τις **οντότητες-μεταβλητών** που μπορεί να υπάρχουν στον πίνακα συμβόλων. Η κλάση αυτή κληρονομεί από την κλάση Entity τα βασικά πεδία που πρέπει να έχει κάθε οντότητα (κάθε τύπου) και τα συμπληρώνει με τα πεδία var_type και offset. Το πεδίο **var_type** έχει ως σύνολο τιμών το {**Temponary**, **Regular**} και αναφέρεται στο είδος της μεταβλητής. Το πεδίο **offset** περιέχει την απόσταση που θα έχει η θέση μνήμης που θα αποθηκευτεί η μεταβλητή αυτή, από τον δείκτη στοίβας της συνάρτησης (ή του προγράμματος) η οποία θα ορίσει την μεταβλητή αυτή.

- **class Function :** Πρόκειται για μία υποκλάση της κλάσης Entity και συγκεκριμένα για την κλάση τα αντικείμενα της οποία αποτελούν τις **οντότητες-συναρτήσεων** που μπορεί να υπάρχουν στον πίνακα συμβόλων. Η κλάση αυτή κληρονομεί από την κλάση Entity τα βασικά πεδία που πρέπει να έχει κάθε οντότητα (κάθε τύπου) και τα συμπληρώνει με τα πεδία start_quad, arguments, framelength. Το πεδίο **start_quad** περιέχει τον αριθμό της ετικέτας της τετράδας η οποία είναι, η πρώτη τετράδα από όλες τις τετράδες της συνάρτησης, που θα εκτελεστεί κατά την κλήση της συνάρτησης αυτής στην οποία αναφέρεται η συγκεκριμένη οντότητα. Το πεδίο αυτό παίζει βασικό ρόλο κατά την παραγωγή του ενδιάμεσου κώδικα ώστε να ορίζεται κατάλληλα το σημείο του κώδικα στο οποίο θα πρέπει να γίνει άλμα κατά τον χρόνο εκτέλεσης ώστε να τρέξει η συγκριμένη συνάρτηση όταν αυτή καλείται. Το πεδίο **arguments** αποτελεί μία λίστα με όλα τα ορίσματα με τα οποία καλείται η συνάρτηση. Η κλάση **Argument** επεξηγείται παρακάτω. Τέλος το πεδίο **framelength** αναφέρεται στο συνολικό μέγεθος της συνάρτησης, δηλαδή στον χώρο που θα δεσμεύσει αυτή όταν φορτωθεί στην στοίβα. Και αυτό το πεδίο είναι εξαιρετικά σημαντικό για την παραγωγή του τελικού κώδικα, καθώς υποδεικνύει το πόσος χώρος στοίβας θα πρέπει να δεσμευτεί αλλά και το πόσο θα πρέπει να κατέβει

ο δείκτης στοίβας ώστε να πάει στην αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης κατά την κλήση της.

- **class Parameter :** Πρόκειται για μία υποκλάση της κλάσης Entity και συγκεκριμένα για την κλάση τα αντικείμενα της οποία αποτελούν τις **οντότητες-παραμέτρων** που μπορεί να υπάρχουν στον πίνακα συμβόλων. Η κλάση αυτή κληρονομεί από την κλάση Entity τα βασικά πεδία που πρέπει να έχει κάθε οντότητα (κάθε τύπου) και τα συμπληρώνει με τα πεδία par_mode, offset. Το πεδίο **par_mode** έχει ως σύνολο τιμών το {**in**, **inout**, **inandout**} και αναφέρεται στον τρόπο περάσματος της παραμέτρου μιας συνάρτησης. Το πεδίο **offset** περιέχει την απόσταση που θα έχει η θέση μνήμης που θα αποθηκευτεί η παράμετρος αυτή, από τον δείκτη στοίβας της συνάρτησης στην οποία βρίσκεται η παράμετρος αυτή.

- **class Argument :** Πρόκειται για την κλάση τα αντικείμενα της οποίας αναπαριστούν τα ορίσματα τα οποία μπορεί να έχει μία οντότητα τύπου συνάρτησης. Τα αντικείμενα αυτής της κλάσης έχουν μόνο ένα πεδίο το **par_mode** το οποίο όπως και στην κλάση **Parameter** έχει ως σύνολο τιμών το {**in**, **inout**, **inandout**} και αναφέρεται στον τρόπο περάσματος του ορίσματος αυτού, στην συνάρτηση στην οποία ανήκει. Τα αντικείμενα αυτής της κλάσης χρησιμοποιούνται κατά την κλήση μίας συνάρτησης σε συνδυασμό με τα αντικείμενα της κλάσης **Parameter** για να γίνει έλεγχος σχετικά με το αν η κλήση της συνάρτησης είναι ορθή.

- **class Scope :** Πρόκειται για την κλάση τα αντικείμενα της οποίας αποτελούν τα στοιχεία με τα οποία δομείται ένας πίνακας συμβόλων. Κάθε ένα από αυτά τα αντικείμενα αποτελούν μία διαφορετική γραμμή του πίνακα συμβόλων και υπάρχει ένα τέτοιο για κάθε διαφορετικό βάθος φωλιάσματος. Τα αντικείμενα αυτής της κλάσης περιέχουν αποτελούνται από τα πεδία name, entities, framelength.
  Το πεδίο **name** περιέχει το όνομα του **Scope**, αν για παράδειγμα το **Scope** που έχει δημιουργηθεί είναι μίας συνάρτησης με όνομα **f** τότε και αυτό θα έχει όνομα **f**.
  Το πεδίο **entities** αποτελεί μία λίστα από αντικείμενα της κλάσης **Entity** τα οποία αποτελούν και τις οντότητες (κάθε τύπου) που βρίσκονται την δεδομένη χρονική στιγμή στο δεδομένο scope του πίνακα συμβόλων. Το πεδίο **framelength** εκφράζει το μέγεθος του συγκεκριμένου scope. Αν το scope αυτό δεν είναι του κύριου προγράμματος αλλά κάποιος συνάρτησης, τότε το framelength του scope ισούται με το framelength της συνάρτησης αυτής.

4. Λεκτική Ανάλυση

Η Λεκτική Ανάλυση αποτελεί μία από τις πιο βασικές λειτουργίες ενός μεταγλωττιστή. Η συγκεκριμένη λειτουργία πραγματοποιεί το διάβασμα και την αναγνώριση συμβόλων της γλώσσας **Starlet** τα οποία στην συνέχεια χρησιμοποιούνται από τις διάφορες συναρτήσεις του μεταγλωττιστή. Το εργαλείο που πραγματοποιεί την Λεκτική Ανάλυση ονομάζεται Λεκτικός Αναλυτής και η λειτουργία του περιγράφεται από το παρακάτω αυτόματο πεπερασμένων καταστάσεων.

Ορολογία:

Καταστάσεις Αυτομάτου,

- **STATE_START (P0):** Πρόκειται για την εναρκτήρια κατάσταση του αυτομάτου, από αυτή ξεκινάει το αυτόματο κάθε φορά που καλείται ο Λεκτικός Αναλυτής.

- **STATE_ALPHA (P1):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα χαρακτήρα και πρόκειται να επιστρέψει ένα όνομα μεταβλητής ή κάποιο keyword της γλώσσας.

- **STATE_DIGIT (P2):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα αριθμό. Το σύνηθες αποτέλεσμα αυτού του σεναρίου είναι η επιστροφή ακέραιου αριθμού ο οποίος θα βρίσκεται μέσα στα επιτρεπτά όρια.

- **STATE_LESS (P3):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα τον τελεστή ‘<’.

- **STATE_GREATER (P4):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα τον τελεστή ‘>’.

- **STATE_COLON (P5):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα τον τελεστή ‘:’.

- **STATE_SLASH (P6):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν έχει διαβάσει πρώτα τον τελεστή ‘/’. Η κατάσταση αυτή μπορεί να αποτελεί την έναρξη ενός **σχολίου γραμμής**, ενός **σχολίου πολλαπλών γραμμών** ή απλά έναν τελεστή διαίρεσης.

- **STATE_LINECOMMENT (P7):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν ξαναδιαβάζει τον τελεστή ‘/’, ενώ βρισκόταν στην κατάσταση **STATE_SLASH**. Αυτό σημαίνει ότι πλέον το αυτόματο αναγνωρίζει ότι βρίσκεται σε κατάσταση **σχολίου γραμμής**.

- **STATE_LINECOMMENT_CHECK (P8):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν ξαναδιαβάζει τον τελεστή ‘/’, ενώ βρισκόταν στην κατάσταση **STATE_LINECOMMENT**. Στην συνέχεια της κατάστασης αυτής γίνεται έλεγχος για εμφωλευμένα σχόλια, τα οποία απαγορεύονται.

- **STATE_MULTILINECOMMENT_OPEN (P9):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν διαβάζει τον τελεστή ‘*’ , ενώ βρισκόταν στην κατάσταση **STATE_SLASH**. Αυτό σημαίνει ότι πλέον το αυτόματο αναγνωρίζει ότι βρίσκεται σε κατάσταση **σχολίου πολλαπλών γραμμών**.

- **STATE_MULTILINECOMMENT_CHECK (P10):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν ξαναδιαβάζει τον τελεστή ‘/’, ενώ βρισκόταν στην κατάσταση **STATE_MULTILINECOMMENT_OPEN**. Στην συνέχεια της κατάστασης αυτής γίνεται έλεγχος για εμφωλευμένα σχόλια, τα οποία απαγορεύονται.

- **STATE_MULTILINECOMMENT_CLOSE (P11):** Πρόκειται για την κατάσταση στην οποία μεταβαίνει το αυτόματο όταν διαβάζει τον τελεστή ‘*’ , ενώ βρισκόταν στην κατάσταση **STATE_MULTILINECOMMENT_OPEN**. Στην συγκεκριμένη περίπτωση το αυτόματο αναμένει για έναν τελεστή ‘/’ ώστε να ολοκληρώσει το **σχόλιο πολλαπλών γραμμών** ή κάποιον άλλο σύμβολο ώστε να επιστρέψει στην **STATE_MULTILINECOMMENT_OPEN**.

Τελικές ‘Καταστάσεις’ Αυτομάτου,

- **OK:** Πρόκειται για την τελική κατάσταση όπου ο Λεκτικός Αναλυτής τερματίζει την κλήση του και επιστρέφει ότι ακριβώς είχε συγκεντρώσει ο buffer του μέχρι εκείνη την στιγμή.

- **OKR:** Πρόκειται για την τελική κατάσταση όπου ο Λεκτικός Αναλυτής τερματίζει την κλήση του και επιστρέφει ότι όλα τα περιεχόμενα που έχει συγκεντρώσει ο buffer του εκτός από το τελευταίο σύμβολο που διάβασε. Το σύμβολο αυτό επιστρέφει στην ροή δεδομένων εισόδου και πρόκειται να είναι το πρώτο σύμβολο που θα διαβάσει ο Λεκτικός Αναλυτής όταν ξανακαλεστεί.

- **Error:** Πρόκειται για την τελική κατάσταση όπου ο Λεκτικός Αναλυτής ανιχνεύει κάποιο λάθος στην ροή των δεδομένων εισόδου και έτσι καλεί τον διαχειριστή σφαλμάτων μεταγλωττιστή, ώστε να αναγνωρίσει αυτό το λάθος και να τερματίσει την συνολική διαδικασία της μεταγλώττισης.

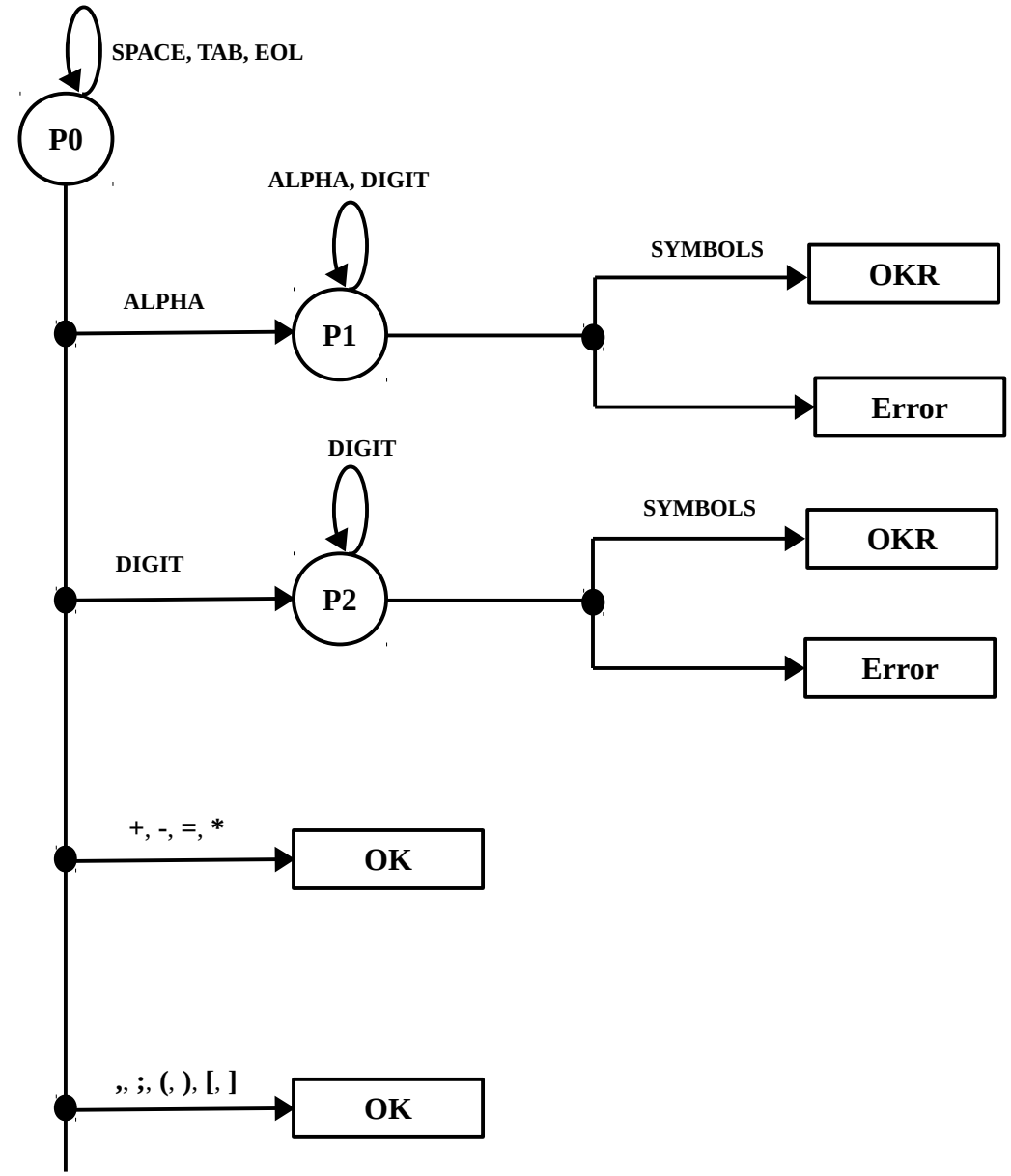
Παρατηρήσεις Λειτουργίας Αυτομάτου,

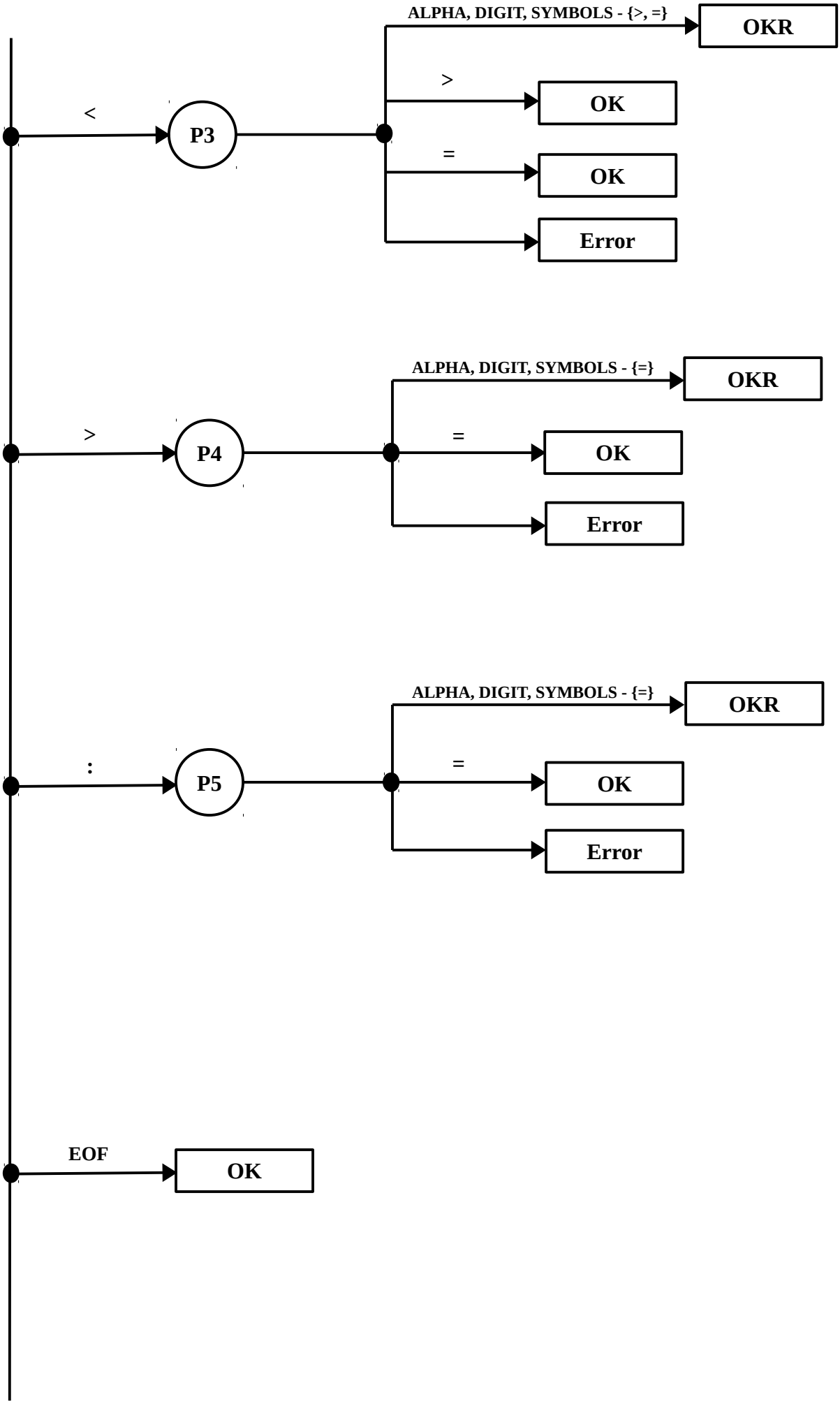
Ο Λεκτικός αναλυτής βλέπει όλο το αρχείο πηγαίου κώδικα σαν μία ροή δεδομένων εισόδου. Η ροή αυτή αποτελείται από σύμβολα που μπορεί να είναι αριθμοί, γράμματα του αλφάβητου της γλώσσας, τελεστές ή ειδικοί χαρακτήρες {**EOF**, ‘**n**’,’**t**’, ‘ ’}. Κάθε φορά που ο Λεκτικός Αναλυτής καλείται, διαβάζει από αυτή την ροή δεδομένων μέχρι να βρεθεί σε μία από τις παραπάνω τελικές καταστάσεις {**OK**, **OKR**, **Error**}. Όποτε διαβάζει ένα καινούργιο σύμβολο της ροής, το αποθηκεύει σε ένα **buffer** που διαθέτει και μεταβαίνει σε μία κατάσταση (διαφορετική ή την ίδια). Αν η κατάσταση αυτή είναι τελική τότε ανάλογα του είδους της αδειάζει κατάλληλα αυτόν τον buffer. Αν πρόκειται για κάποια από τις ενδιάμεσες καταστάσεις {**P0**,...,**P11**} τότε απλά προσθέτει το σύμβολο στον buffer και διαβάζει το επόμενο από την ροή δεδομένων. Σε όποια ακμή δεν προσδιορίζεται το σύμβολο εισόδου υπονοείται ότι η μετάβαση γίνεται για κάθε σύμβολο εκτός από εκείνα για τα οποία υπάρχει συγκεκριμένη ακμή στην δεδομένη κατάσταση.

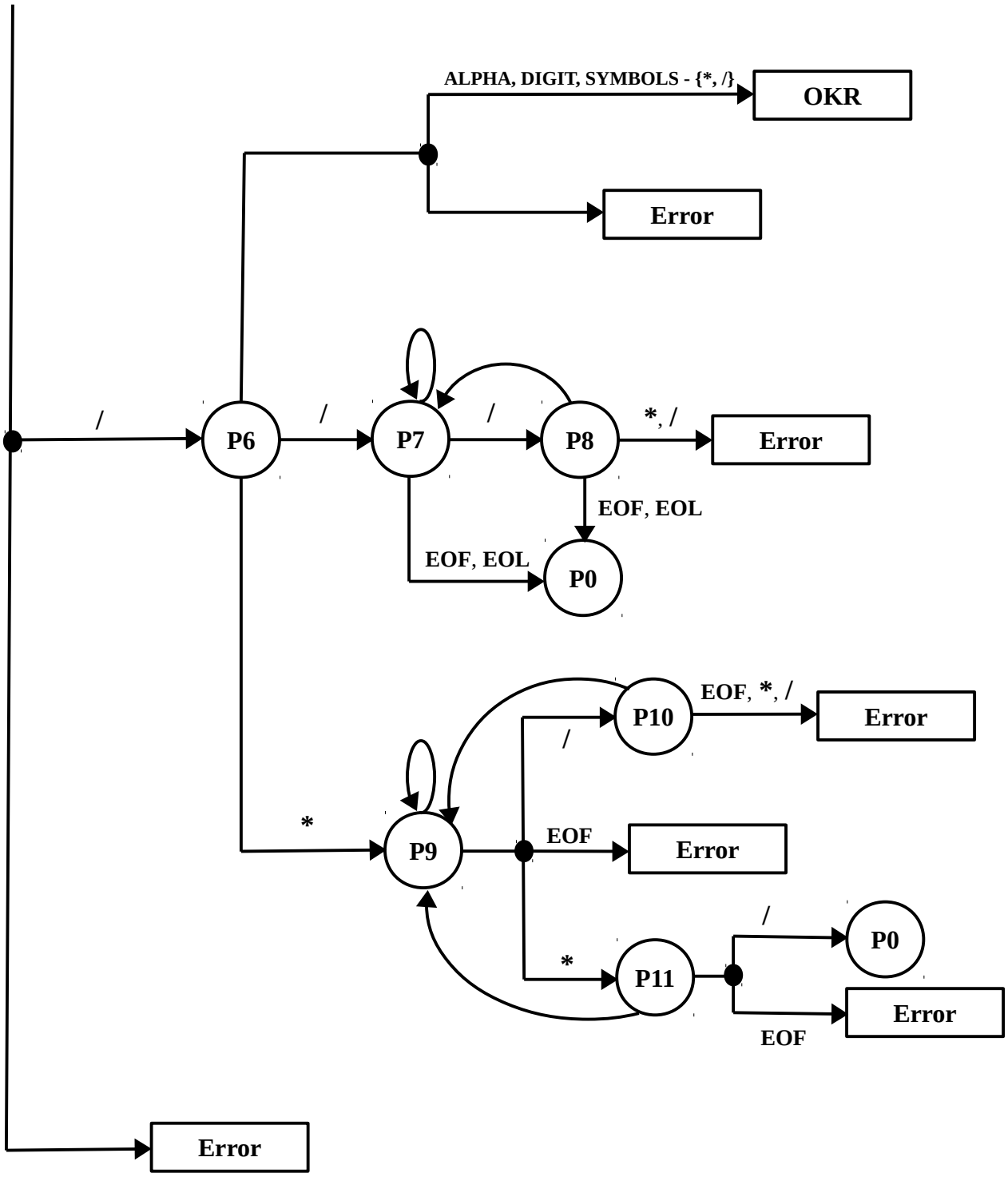
Ειδικοί συμβολισμοί Αυτομάτου,

- **ALPHA :** Το σύνολο όλων των γραμμάτων του αλφάβητου της γλώσσας. Δηλαδή τα '**a**', ..., '**z**' και τα '**A**', ..., '**Z**'
- **DIGIT :** Το σύνολο όλων των ακέραιων αριθμών που βρίσκονται στο διάστημα {**–32767**, ..., **32767**}
- **SYMBOLS :** Το σύνολο {' ', '**\t**', '**\n**', '+', '-', '*****', '/', '<', '>', '=', '**:**', ',', '**;**', '**(**', '**)**', '**[**', '**]**', **EOF**}
- **EOL :** Συμβολισμός του χαρακτήρα αλλαγής γραμμής '**\n**'.
- **EOF :** Συμβολισμός του τέλους αχρείου.

Το Αυτόματο:




ALPHA, DIGIT, SYMBOLS - {>, =}
OKR
<
P3
>
OK
=
OK
Error
ALPHA, DIGIT, SYMBOLS - {=}
OKR
>
P4
=
OK
Error
ALPHA, DIGIT, SYMBOLS - {=}
OKR
:
P5
=
OK
Error
EOF
OK


ALPHA, DIGIT, SYMBOLS - {*, /}
OKR
Error
/
P6
/
P7
/
P8
*, /
Error
EOF, EOL
EOF, EOL
P0
EOF, *, /
P10
Error
/
*
P9
EOF
Error
/
P0
*
P11
Error
EOF
Error

Ο πίνακας απόφασης:

STATES\SYMBOLS	SPACE/TAB	ALPHA	DIGIT
STATE_START	STATE_START	STATE_ALPHA	STATE_DIGIT
STATE_ALPHA	OKR	STATE_ALPHA	STATE_ALPHA
STATE_DIGIT	OKR	ERROR	STATE_DIGIT
STATE_LESS	OKR	OKR	OKR
STATE_GREATER	OKR	OKR	OKR
STATE_COLON	OKR	OKR	OKR
STATE_SLASH	OKR	OKR	OKR
STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_LINECOMMENT_CHECK	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CHECK	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CLOSE	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN

STATES\SYMBOLS	'+'	'-'	'*'
STATE_START	OK	OK	OK
STATE_ALPHA	OKR	OKR	OKR
STATE_DIGIT	OKR	OKR	OKR
STATE_LESS	OKR	OKR	OKR
STATE_GREATER	OKR	OKR	OKR
STATE_COLON	OKR	OKR	OKR
STATE_SLASH	OKR	OKR	STATE_MULTILINECOMMENT_OPEN
STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_LINECOMMENT_CHECK	STATE_LINECOMMENT	STATE_LINECOMMENT	ERROR
STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_CLOSE
STATE_MULTILINECOMMENT_CHECK	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	ERROR
STATE_MULTILINECOMMENT_CLOSE	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN

STATES\SYMBOLS	'/'	'<'	'>'
STATE_START	STATE_SLASH	STATE_LESS	STATE_GREATER
STATE_ALPHA	OKR	OKR	OKR
STATE_DIGIT	OKR	OKR	OKR
STATE_LESS	OKR	OKR	OK
STATE_GREATER	OKR	OKR	OKR
STATE_COLON	OKR	OKR	OKR
STATE_SLASH	STATE_LINECOMMENT	OKR	OKR
STATE_LINECOMMENT	STATE_LINECOMMENT_CHECK	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_LINECOMMENT_CHECK	ERROR	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_CHECK	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CHECK	ERROR	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CLOSE	STATE_START	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN

STATES\SYMBOLS	';'	'('	')'
STATE_START	OK	OK	OK
STATE_ALPHA	OKR	OKR	OKR
STATE_DIGIT	OKR	OKR	OKR
STATE_LESS	OKR	OKR	OKR
STATE_GREATER	OKR	OKR	OKR
STATE_COLON	OKR	OKR	OKR
STATE_SLASH	OKR	OKR	OKR
STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_LINECOMMENT_CHECK	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT
STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CHECK	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CLOSE	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN

STATES\SYMBOLS	'['	']'	**EOL ('\n')**
STATE_START	OK	OK	STATE_START
STATE_ALPHA	OKR	OKR	OKR
STATE_DIGIT	OKR	OKR	OKR
STATE_LESS	OKR	OKR	OKR
STATE_GREATER	OKR	OKR	OKR
STATE_COLON	OKR	OKR	OKR
STATE_SLASH	OKR	OKR	OKR
STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_START
STATE_LINECOMMENT_CHECK	STATE_LINECOMMENT	STATE_LINECOMMENT	STATE_START
STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CHECK	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CLOSE	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN	STATE_MULTILINECOMMENT_OPEN

STATES\SYMBOLS	**EOF**	**OTHER**
STATE_START	OK	ERROR
STATE_ALPHA	OKR	ERROR
STATE_DIGIT	OKR	ERROR
STATE_LESS	OKR	ERROR
STATE_GREATER	OKR	ERROR
STATE_COLON	OKR	ERROR
STATE_SLASH	OKR	ERROR
STATE_LINECOMMENT	STATE_START	STATE_LINECOMMENT
STATE_LINECOMMENT_CHECK	STATE_START	STATE_LINECOMMENT
STATE_MULTILINECOMMENT_OPEN	ERROR	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CHECK	ERROR	STATE_MULTILINECOMMENT_OPEN
STATE_MULTILINECOMMENT_CLOSE	ERROR	STATE_MULTILINECOMMENT_OPEN

Ο παραπάνω πίνακας αναπαριστά το αυτόματο καταστάσεων σε μορφή κατανοητή από τον μεταγλωττιστή. Στον κώδικα ο πίνακας αυτός έχει το όνομα **decision_table** και ο Λεκτικός Αναλυτής υλοποιείται από την συνάρτηση **lex()**.

5. Συντακτική Ανάλυση

Η συντακτική ανάλυση έγινε με βάση την γραμματική της γλώσσας **Starlet.** Στην συνέχεια παραθέτονται οι κανόνες της γραμματικής καθώς και οι συναρτήσεις που τους υλοποιούν.

- <program> ::= **program id** <block> **endprogram**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def program()**, πρόκειται για την πρώτη συνάρτηση που εκτελείται με την έναρξη της μεταγλώττισης.

- <block> ::= <declarations> <subprograms> <statements>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def block(name, return_list = None)**, πρόκειται για την συνάρτηση που εκτελείται για κάθε διαφορετικό μπλοκ κώδικα υπάρχει στο αρχείο πηγαίου κώδικα. Άρα η συνάρτηση αυτή καλείται από το κύριο πρόγραμμα αλλά και από τις συναρτήσεις. Είναι σημαντικό να τονίσουμε ότι βάσει της υλοποίησης, η συνάρτηση αυτή θα καλεστεί πρώτα από το μπλοκ κώδικα με το μεγαλύτερο βάθος φωλιάσματος.

- <declarations> ::= (**declare** <varlist>**;**)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def declarations()**, πρόκειται για την συνάρτηση που εκτελείται για κάθε δήλωση μεταβλητών γίνεται μέσα στο πρόγραμμα με την εντολή **declare**.

- <varlist> ::= **ε** | **id** (**,** **id**)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def varlist()**, πρόκειται για την συνάρτηση που συνδυάζεται με την declarations() ώστε να δηλωθεί ένα σύνολο από μεταβλητές σε ένα συγκεκριμένο μπλοκ κώδικα.

- <subprograms> ::= (<subprogram>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def subprograms()**, πρόκειται για την συνάρτηση η οποία επιτρέπει τον ορισμό πολλαπών συναρτήσεων. Όπως βλέπουμε στον κανόνα, αλλά και στην αντίστοιχη συνάρτηση, καλείται επαναληπτικά η συνάρτηση subprogram().

- <subprogram> ::= **function id** <funcbody> **endfunction**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def subprogram()**, πρόκειται για την συνάρτηση με την οποία γίνεται ο ορισμός μίας συνάρτησης. Η συνάρτηση αυτή καλείται όσες φορές χρειάζεται από τη συνάρτηση subprograms(), ώστε να οριστούν όλες οι συναρτήσεις οι οποίες υπάρχουν σε ένα μπλοκ.

- <funcbody> ::= <formalpars> <block>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def funcbody(func_name, return_list)**, πρόκειται για την συνάρτηση η οποία καλείται από την subprogram() ώστε να διαβάσει τις τυπικές παραμέτρους της

συνάρτησης και να καλέσει την συνάρτηση block για να δομηθεί το μπλοκ κώδικα της συνάρτησης.

- <formalpars> ::= (<formalparlist>)

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def formalpars(func_name)**, πρόκειται για την συνάρτηση η οποία εκκινεί την διαδικασία διαβάσματος των τυπικών παραμέτρων της συνάρτησης.

- <formalparlist> ::= <formalparitem> (**,** <formalparitem>)* | **ε**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def formalparlist(func_name)**, πρόκειται για την συνάρτηση η οποία επιτρέπει το πέρασμα πολλαπλών τυπικών παραμέτρων σε μία συνάρτηση.

- <formalparitem> **::= in id | inout id | inandout id**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def formalparitem(func_entity)**, πρόκειται για την συνάρτηση η οποία καθορίζει τον τρόπο με τον οποίο περνάει μία παράμετρος στην συνάρτηση.

- <statements> ::= <statement> (**;** <statement>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def statements**(**exit_list** = **None**, **return_list** = **None**), πρόκειται για την συνάρτηση η οποία επιτρέπει την ύπαρξη πολλαπλών εντολών σε ένα μπλοκ κώδικα.

- <statement> ::= **ε** | <assignment-stat> | <if-stat> | <while-stat> |
  <do-while-stat> | <loop-stat> | <exit-stat> |
  <forcase-stat> | <incase-stat> | <return-stat> |
  <input-stat> | <print-stat>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def statement**(**exit_list** = **None**, **return_list** = **None**), πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή της γλώσσας οποιοδήποτε τύπου. Η συνάρτηση αυτή πρώτα αναγνωρίσει το είδος της εντολής και στην συνέχεια καλεί την κατάλληλη συνάρτηση που υλοποιεί τον κανόνα της εντολής που αναγνώρισε.

- <assignment-stat> ::= **id :=** <expression>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def assignment_stat()**, πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή ανάθεσης.

- <if-stat> ::= **if (**<condition>**) then** <statements> <elsepart> **endif**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def if_stat(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή τύπου **if**–**else**. Το **else** είναι προαιρετικό και για αυτό εκτελείται ειδικός κανόνας για αυτό.

- <elsepart> ::= **ε** | **else** <statements>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def elsepart(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία εντάσσει το κομμάτι **else** στην **if** που προηγήθηκε σε περίπτωσή που υπάρχει στον πηγαίο κώδικα το **else** keyword.

- <while-stat> ::= **while (**<condition>**)** <statements> **endwhile**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def while_stat(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή τύπου δομής επανάληψης **while**.

- <do-while-stat> ::= **dowhile** <statements> **enddowhile (**<condition>**)**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def do_while_stat(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή τύπου δομής επανάληψης **do-while**.

- <loop-stat> ::= **loop** <statements> **endloop**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def loop_stat(return_list)**, πρόκειται για την συνάρτηση η οποία ξεκινάει μία εντολή τύπου δομής επανάληψης **loop forever**.

- <exit-stat> ::= **exit**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def exit_stat(exit_list)**, πρόκειται για την συνάρτηση η οποία υπάρχει μέσα στους βρόχους τύπου **loop**, έτσι ώστε να δίνει την δυνατότητα τερματισμού του ατέρμονου βρόχου.

- <forcase-stat> ::= **forcase**
  (**when (**<condition>**) :** <statements>)*
  default: <statements> **enddefault**
  endforcase

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def forcase_stat(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία υλοποιεί την ειδική περίπτωση εντολής (forcase) που ορίζεται από τη γραμματική της γλώσσας.

- <incase-stat> ::= **incase**
  (**when (**<condition>**) :** <statements>)*
  endincase

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def incase_stat(exit_list, return_list)**, πρόκειται για την συνάρτηση η οποία υλοποιεί την ειδική περίπτωση εντολής (incase) που ορίζεται από τη γραμματική της γλώσσας.

- <return-stat> ::= **return** <expression>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def return_stat(return_list)**, πρόκειται για την συνάρτηση η οποία υλοποιεί την απαίτηση κάθε συνάρτησης της γλώσσας να επιστέφει κάποια ακέραια τιμή ως αποτέλεσμα.

- <input-stat> ::= **input id**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def input_stat()**, πρόκειται για την συνάρτηση η οποία επιτρέπει στα προγράμματα της γλώσσας να δέχονται από τον χρήστη είσοδο ακέραιων αριθμών.

- <print-stat> ::= **print** <expression>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def print_stat()**, πρόκειται για την συνάρτηση η οποία δίνει την δυνατότητα στα προγράμματα της γλώσσας να εμφανίζουν ακέραιες τιμές στην κονσόλα του χρήστη.

- <actualpars> ::= **(** <actualparlist> **)**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def actualpars(func_name)**, πρόκειται για την συνάρτηση η οποία εκκινεί την διαδικασία διαβάσματος των πραγματικών παραμέτρων κατά την κλήση μίας συνάρτησης.

- <actualparlist> ::= <actualparitem> (**,** <actualparitem>)* | **ε**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def actualparlist(arg_list, func_name)**, πρόκειται για την συνάρτηση η οποία επιτρέπει το πέρασμα πολλαπλών πραγματικών παραμέτρων σε μία συνάρτηση.

- <actualparitem> ::= **in** <expression> | **inout id** | **inandout id**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def actualparitem(arg_list, func_name)**, πρόκειται για την συνάρτηση η οποία καθορίζει τον τρόπο με τον οποίο περνάει μία παράμετρος στην συνάρτηση όταν γίνει η κλήση. Παρατηρούμε ότι μπορούμε να καλέσουμε μία συνάρτηση με ορίσματα, αριθμητικές σταθερές και εκφράσεις μόνο όταν ο τρόπος περάσματος τους είναι με τιμή (**in**).

- <condition> ::= <boolterm> (**or** <boolterm>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def condition()**, πρόκειται για την συνάρτηση η οποία επιτρέπει την δημιουργία σύνθετων λογικών εκφράσεων χωρισμένων με λογικούς τελεστές **or**.

- <boolterm> ::= <boolfactor> (**and** <boolfactor>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def boolterm()**, πρόκειται για την συνάρτηση η οποία επιτρέπει την δημιουργία σύνθετων λογικών εκφράσεων χωρισμένων με λογικούς τελεστές **and**.

- <boolfactor> ::= **not [**<condition>**]** | **[**<condition>**]** | <expression> <relational-oper> <expression>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def boolfactor()**, πρόκειται για την συνάρτηση η οποία επιτρέπει την δημιουργία σύνθετων λογικών εκφράσεων οι οποίες συνδυάζουν τελεστές **or** και **and** καθώς και **not** ώστε να ενώσουν διάφορες λογικές αλλά και αριθμητικές εκφράσεις.

- <expression> ::= <optional-sign> <term> (<add-oper> <term>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def expression()**, πρόκειται για την συνάρτηση η οποία επιτρέπει την δημιουργία αριθμητικών εκφράσεων. Οι αριθμητικές εκφράσεις αυτές μπορούν να περιέχουν σταθερές, μεταβλητές ή και κλήσεις συναρτήσεων.

- <term> ::= <factor> (<mul-oper> <factor>)*

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def term()**, πρόκειται για την συνάρτηση η οποία δημιουργεί αριθμητικές εκφράσεις χωρισμένες με τους τελεστές του πολλαπλασιασμού και της διαίρεσης.

- <factor> ::= **constant** | (<expression>) | **id** <idtail>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def factor()**, πρόκειται για την τερματική συνάρτηση μίας αριθμητικής έκφρασης όπου είτε θα ξανακαλεστεί η συνάρτηση δημιουργίας έκφρασης, ή θα γίνει η εξαγωγή από τον λεκτικό αναλυτή ενός αριθμού είτε θα γίνει μετάβαση στην συνάρτηση idtail() για εξαγωγή μεταβλητής ή κλήσης συνάρτησης.

- <idtail> ::= **ε** | <actualpars>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def idtail(var_name)**, πρόκειται για την συνάρτηση που αναγνωρίζει αν αυτό που διάβασε ο λεκτικός αναλυτής ήταν μεταβλητή ή κλήση συνάρτησης.

- <relational-oper> ::= = | <= | >= | > | < | <>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def relational_oper()**, πρόκειται για την συνάρτηση που αναγνωρίζει έναν λογικό τελεστή.

- <add-oper> ::= **+** | **-**

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def add_oper()**, πρόκειται για την συνάρτηση που αναγνωρίζει έναν εκ των τελεστών +, -.

- <mul-oper> ::= ***** | /

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def mul_oper()**, πρόκειται για την συνάρτηση που αναγνωρίζει έναν εκ των τελεστών *, /.

- <optional-sign> ::= **ε** | <add-oper>

  Η συνάρτηση που υλοποιεί τον συγκεκριμένο κανόνα είναι η **def optional_sign()**, πρόκειται για την συνάρτηση που αναγνωρίζει ένα πρόσημο αριθμού.

6. Παραγωγή Ενδιάμεσου Κώδικα

Ο ενδιάμεσος κώδικας της γλώσσας επιλέχθηκε να αποτελείται από τετράδες της μορφής, **op, term0, term1, term2**. Στον κώδικα του μεταγλωττιστή οι τετράδες αυτές αναπαρίστανται με τα αντικείμενα της κλάσης **Quad** που αναφέρθηκαν στην **Ενότητα 3**.
Η ιδέα που ακολουθήθηκε στον κώδικα είναι η προσθήκη κάθε τετράδας που παράγεται σε μία λίστα, η οποία λίστα στην συνέχεια θα χρησιμοποιείται για την παραγωγή του τελικού κώδικα και του αρχείου ενδιάμεσου κώδικα (αρχείο κατάληξης .int). Στην συνέχεια παρουσιάζεται ένας τυπικός τρόπος περιγραφής της παραγωγής κάθε είδους τετράδας του ενδιάμεσου κώδικα της γλώσσας, όμοιος με αυτόν που διδάχθηκε στο μάθημα.
Τα **{pj}** αποτελούν τα σημεία του κώδικα όπου γίνονται οι συγκεκριμένες ενέργειες.

Τετράδες (Quads):

- **Αρχή και τέλος μπλοκ κώδικα,**

  <block> ::= <declarations> <subprograms> **{p1}** <statements> **{p2}**

  {p1} : genquad(‘**begin_block**’, func_name, _, _)

  {p2} : if (idtk == “END_PROGRAM”){
  genquad(‘**halt**’, _, _, _)
  }
  genquad(‘**end_block**’, func_name, _, _)

  Το func_name, περνάει ως όρισμα στην block() από την program() ή την subprogram().

- **Εκχώρηση,**

  <assignment-stat> ::= **id :=** <expression> **{p1}**

  {p1} : genquad(‘**:=**’, expression.Place, _, **id**)

- **Εντολή if-else,**

  <if-stat> ::= **if (**<condition>**) then** **{p1}** <statements> **{p2}** <elsepart> **endif** **{p3}**

  {p1} : backpatch(condition.True, nextquad())

  {p2} : if_list = makelist(nextquad())
  genquad(‘**jump**’, _, _, _)
  backpatch(condition.False, nextquad())

  {p3} : backpatch(if_list, nextquad())

- **Εντολή while,**

  <while-stat> ::= **while** **{p1}** **(**<condition>**)** **{p2}** <statements> **endwhile** **{p3}**

  {p1} : first_cond_quad = nextquad()

  {p2} : backpatch(condition.True, nextquad())

  {p3} : genquad(‘**jump**’, _, _, first_cond_quad)
  backpatch(condition.False, nextquad())

- **Εντολή dowhile,**

  <do-while-stat> ::= **dowhile** **{p1}** <statements> **enddowhile** **(**<condition>**)** **{p2}**

  {p1} : first_stat_quad = nextquad()

  {p2} : backpatch(condition.True, first_stat_quad)
  backpatch(condition.False, nextquad())

- **Εντολή loop,**

  <loop-stat> ::= **loop** **{p1}** <statements> **endloop** **{p2}**

  {p1} : first_stat_quad = nextquad()

  {p2} : genquad(‘**jump**’, _, _,first_stat_quad)

- **Εντολή forcase,**

  <forcase-stat> ::= **forcase** **{p1}**
  (**when (**<condition>**) : {p2}** <statements> **{p3}**)*
  default: <statements> **enddefault** **{p4}**
  endforcase **{p5}**

  {p1} : first_cond_quad = nextquad()
  jump_list = emptylist()

  {p2} : backpatch(condition.True, first_stat_quad)

  {p3} : jump_quad_list = makelist(nextquad())
  jump_list = merge(jump_list,jump_quad_list)
  genquad(‘**jump**’, _, _, _)
  backpatch(condition.False, nextquad())

  {p4} : genquad(‘**jump**’, _, _, first_cond_quad)

  {p5} : backpatch(jump_list, nextquad())

- **Εντολή incase,**

  <incase-stat> ::= **incase** **{p1}**
  (**when (**<condition>**) : {p2}** <statements> **{p3}**)*
  endincase {p4}

  {p1} : first_cond_quad = nextquad()
  incase_flag = newtemp()
  genquad(‘**:=**’,"**0**", _, incase_flag)

  {p2} : backpatch(condition.True, nextquad())

  {p3} : genquad(‘**:=**’,"**1**", _, incase_flag)
  backpatch(condition.False, nextquad())

  {p4} : genquad(‘**=**’, incase_flag, "**1**", first_cond_quad)

- **Εντολή return,**

  <return-stat> ::= **return** <expression> **{p1}**

  {p1} : genquad(‘**retv**’, expression.Place , _, _)

- **Εντολή input,**

  <input-stat> ::= **input id {p1}**

  {p1} : genquad(‘**inp**’, ‘**id**’ , _, _)

- **Εντολή output,**

  <print-stat> ::= **print** <expression> **{p1}**

  {p1} : genquad(‘**out**’, expression.Place , _, _)

- **Κλήση συνάρτησης,**

  <actualpars> ::= **({p1}** <actualparlist> **) {p2}**

  {p1} : arg_list = emptylist()arg_list = emptylist()

  {p2} : for arg_elem, argument **in (**arg_list, func_entity.arguments**){**
  genquad(‘**par**’,arg_elem[0], arg_elem[1], _)
  }

ret_var = newtemp()
genquad(‘**par**’ ,ret_var, "**RET**", _)
genquad(‘**call**’, func_name, _, _)

όπου, arg_elem ϵ {var_name, {**CV**, **CP**, **REF**}}.

<actualparlist> ::= <actualparitem> (**,** <actualparitem>)* | **ε**

--> Απλό πέρασμα παραμέτρων προς τα ‘κάτω’.

<actualparitem> ::= **in** <expression> | **inout id | inandout id**

--> Κάθε κλήση της, βάζει ένα ζεύγος τύπου arg_elem στην λίστα (arg_list) που της περνιέται ως όρισμα.

- **Κανόνας condition,**

<condition> ::= <boolterm^{1}> **{p1}** (**or** **{p2}** <boolterm^{2}> **{p3}**)*

{p1} : condition.True = boolterm^{1}.True
condition. False = boolterm^{1}.False

{p2} : backpatch(condition.False, nextquad())

{p3} : condition.True = merge(condition.True, boolterm^{2}.True)
condition. False = boolterm^{2}.False

- **Κανόνας boolterm,**

<boolterm> ::= <boolfactor^{1}> **{p1}** (**and** **{p2}** <boolfactor^{2}> **{p3}**)*

{p1} : boolterm.True = boolfactor^{1}.True
boolterm.False = boolfactor^{1}.False

{p2} : backpatch(boolterm.True, nextquad())

{p3} : boolterm.False = merge(boolterm.False, boolfactor^{2}.False)
boolterm.True = boolfactor^{2}.True

- **Κανόνας boolfactor,**

  <boolfactor> ::= **not [**<condition>**]** | **[**<condition>**]** | <expression> <relational-oper> <expression>

  (1) : <boolfactor> ::= **not [**<condition>**]** **{p1}**

  {p1} : boolfactor.True = condition.False
  boolfactor.False = condition.True

  (2) : <boolfactor> ::= **[**<condition>**]** **{p1}**

  {p1} : boolfactor.True = condition. True
  boolfactor.False = condition.False

  (3) : <boolfactor> ::= <expression^1> <relational-oper> <expression^2> **{p1}**

  {p1} : bftrue_list = makelist(nextquad())
  genquad(relational-oper.Place, expression^1.Place, expression^2.Place, _)
  bffalse_list = makelist(nextquad())
  genquad('**jump**', _, _, _)

- **Κανόνας expression,**

  <expression> ::= <optional-sign> <term^1> **{p1}** (<add-oper> <term^2> **{p2}**)* **{p3}**

  {p1} : if (is **sign){**
  tmp_var = newtemp()
  genquad(optional-sign.Place, term^1.Place, _, tmp_var)
  term^1.Place = tmp_var
  }

  {p2} : tmp_var = newtemp()
  genquad(optional-sign.Place, term^1.Place, term^2.Place, tmp_var)
  term^1.Place = tmp_var

  {p3} : expression.Place = term^1.Place

- **Κανόνας term,**

  <term> ::= <factor[1]> (<mul-oper> <factor[2]> **{p1}**)* **{p2}**

  {p1} : tmp_var = newtemp()
  genquad(mul-oper.Place, factor[1].Place, factor[2].Place, tmp_var)
  factor[1].Place = tmp_var

  {p2} : term.Place = factor[1].Place

- **Κανόνας factor,**

  <factor> ::= **constant** | **(**<expression>**)** | **id** <idtail>

  (1) : <factor> ::= **constant** **{p1}**

  {p1} : factor.Place = **constant**

  (2) : <factor> ::= **(**<expression>**)** **{p1}**

  {p1} : factor.Place = expression.Place

  (3) : <factor> ::= **id** <idtail> **{p1}**

  {p1} : factor.Place = idtail.Place

- **Κανόνας exit,**

  <exit-stat> ::= **exit** **{p1}**

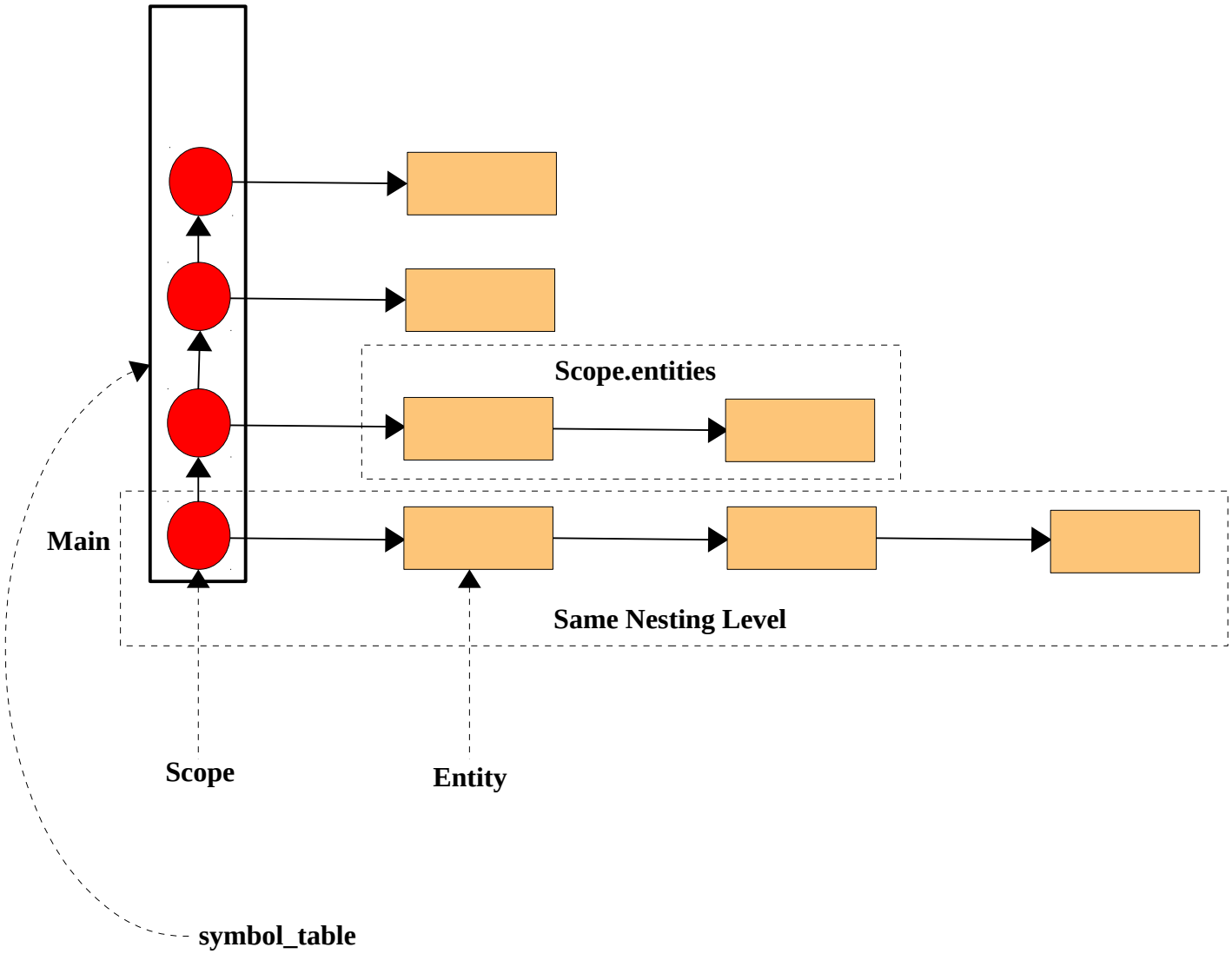
  {p1} : if (exit_list.exists()){
  exit_quad_list = makelist(nextquad())
  exit_list = merge(exit_list, exit_quad_list)
  genquad('**jump**', _, _, _)
  }

  --> ***Η exit() παίρνει ως όρισμα μία λίστα που θα έχει δημιουργηθεί από κάποια δομή επανάληψης .***

  --> ***Την exit_list την κάνει backpatch στο τέλος της, όποια δομή επανάληψης την χρησιμοποιεί.***

7. Πίνακας Συμβόλων

Ο πίνακας συμβόλων αποτελεί βασικό εργαλείο ώστε να γίνει η σημασιολογική ανάλυση και η παραγωγή του τελικού κώδικα. Η δομή που επιλέχτηκε για την αναπαράσταση του πίνακα συμβόλων στην υλοποίησή μας είναι μία λίστα από αντικείμενα της κλάσης Scope. Αναλυτικά η δομή παρουσιάζεται παρακάτω.



Ενέργειες πίνακα συμβόλων:

- **Εισαγωγή Scope,**
  γίνεται με την συνάρτηση **insert_scope()** προτού δημιουργηθεί ένα καινούργιο μπλοκ κώδικα.

- **Διαγραφή Scope,**
  γίνεται με την συνάρτηση **delete_scope()** αφού τελειώσει η δήλωση ένα μπλοκ κώδικα.

- **Εισαγωγή Entity,**
  γίνεται με την συνάρτηση **insert_entity()** κάθε φορά που συναντιέται η δήλωση μίας μεταβλητής ή συνάρτησης. Το **entity** αυτό προστίθεται στο τρέχον **scope**.

- **Αναζήτηση Entity σε όλα τα Scopes,**
  γίνεται με την συνάρτηση **lookup_enclosing_scopes(),** ξεκινώντας από το τρέχον **scope** και πηγαίνοντας προς τα πιο ‘κάτω’ **scopes** αναζητάμε ένα συγκεκριμένο **entity**.

- **Αναζήτηση Entity στο τρέχον Scope**
  γίνεται με την συνάρτηση **lookup_current_scope(),** όμοια με το παραπάνω αλλά μόνο για το τρέχον **Scope**.

8. Σημασιολογική Ανάλυση

Η φάση της Σημασιολογικής Ανάλυσης αφορά τη ικανοποίηση συγκεκριμένων απαιτήσεων της γλώσσας οι οποίες δεν προσδιορίζονται ρητά από την γραμματική της, αλλά είναι απαραίτητες ώστε να είναι ορθή η παραγωγή του τελικού κώδικα. Οι απαιτήσεις αυτές παρατίθενται παρακάτω.

- Κάθε συνάρτηση της γλώσσας πρέπει να έχει ένα **return**.
  Στην υλοποίησή μας αυτό επιτυγχάνεται περνώντας σε όλες τις συναρτήσεις του συντακτικού αναλυτή οι οποίες αφορούν μπλοκ κώδικα μία λίστα με όνομα return_list. Κανονικά έχει την λειτουργία μίας flag αλλά επειδή η Python δεν επιτρέπει πέρασμα με αναφορά επιλέχθηκε αυτή η λύση. Αυτή η λίστα δημιουργείται για πρώτη φορά στην συνάρτηση subprogram() και τίθεται στην τιμή False, δηλαδή με το που εκκινεί ένα μπλοκ συνάρτησης. Όταν στην συνέχεια ενώ βρισκόμαστε στο μπλοκ της συνάρτησης διαβαστεί από τον λεκτικό αναλυτή η εντολή **return**, μεταβαίνουμε στην συνάρτηση return_stat(). Εκεί η τιμή της γίνεται True, έτσι στην συνέχεια που θα επιστρέψει η ροή εκτέλεσης στην αρχική subprogram(), ο compiler θα καταλάβει ότι υπήρχε **return** την δεδομένη συνάρτηση.

- Μόνο οι συναρτήσεις μπορούν να έχουν **return**.
  Με βάση τα παραπάνω, καλεστεί η return() θα έχει τιμή **None** στην λίστα (αφού δεν υπήρχε συνάρτηση που να την δημιουργήσει) και έτσι θα ενημερώσει κατάλληλα τον διαχειριστή σφαλμάτων.

- Η εντολή **exit** υπάρχει μόνο σε βρόχους τύπου **loop**.
  Στην υλοποίησή μας αυτό επιτυγχάνεται περνώντας σε όλες τις συναρτήσεις του συντακτικού αναλυτή οι οποίες αφορούν μπλοκ κώδικα μία λίστα με όνομα exit_list. Η λίστα αυτή δημιουργείται μέσα στην συνάρτηση loop_stat(). Άρα αν καλεστεί η συνάρτηση exit_stat() και η λίστα αυτή δεν έχει δημιουργηθεί προηγουμένως (έχει τιμή **None**) τότε ο μεταγλωττιστής καταλαβαίνει ότι η **exit** αυτή δεν βρίσκεται μέσα σε **loop** και ενημερώσει κατάλληλα τον διαχειριστή σφαλμάτων.

- Κατά την δήλωση μίας μεταβλητής,
  ελέγχουμε αν στο βάθος φωλιάσματος που δηλώνεται δεν υπάρχει άλλη συνάρτηση ή μεταβλητή με το ίδιο όνομα.

- Κατά την δήλωση μίας συνάρτησης,
  ελέγχουμε αν στο βάθος φωλιάσματος που δηλώνεται δεν υπάρχει άλλη συνάρτηση ή μεταβλητή με το ίδιο όνομα.

- Κατά την χρήση μίας μεταβλητής,
  ελέγχουμε αν η μεταβλητή αυτή έχει δηλωθεί στο τρέχον ή σε κάποιο μικρότερο βάθος φωλιάσματος (χρήση της lookup_enclosing_scopes()) και ότι είναι δηλωμένη ως μεταβλητή ή παράμετρος.

- Κατά την χρήση μίας συνάρτησης,
  ελέγχουμε αν η συνάρτηση αυτή έχει δηλωθεί στο τρέχον ή σε κάποιο μικρότερο βάθος φωλιάσματος (χρήση της lookup_enclosing_scopes()) και ότι είναι δηλωμένη ως συνάρτηση. Επίσης ελέγχουμε και ότι τα ορίσματα περνάνε με βάση των τρόπο περάσματος των παραμέτρων της συνάρτησης.

9. Παραγωγή Τελικού Κώδικα

Η βασική ιδέα στην συγκεκριμένη φάση της μεταγλώττισης είναι ότι έχουμε μία λίστα με τετράδες οι οποίες δημιουργήθηκαν κατά την διάρκεια διαβάσματος ενός μπλοκ του κώδικα. Άρα διατηρούμε κάθε χρονική στιγμή μία λίστα στην οποία θα προσθέτουμε κάθε νέα τετράδα μέχρι την στιγμή που θα βρούμε μία τετράδα τύπου end_block. Την χρονική στιγμή αυτή και πριν διαγραφεί το τρέχον **Scope** από το πίνακα συμβόλων θα πρέπει να παράγουμε τελικό κώδικα για όλες τις τετράδες που βρίσκονται μέσα σε αυτή την λίστα. Στην συνέχεια παρουσιάζονται οι συναρτήσεις που συμβάλουν στην διαδικασία παραγωγής του τελικού κώδικα.

- **def gnvlcode(variable)**
  Η συγκεκριμένη συνάρτηση καλείται όταν ζητείται η τιμή μίας μεταβλητής η οποία όμως δεν είναι τοπική στην συνάρτηση που την καλεί. Έτσι αυτή η συνάρτηση παίρνει από τον σύνδεσμο προσπέλασης τον πατέρα της συνάρτησης που ζητάει την μεταβλητή και ξεκινώντας από το βάθος φωλιάσματός του κατεβαίνει προς τα χαμηλότερα βάθη φωλιάσματος μέχρι να βρει την μεταβλητή. Όταν βρεθεί στο βάθος φωλιάσματος που δηλώνεται η συγκεκριμένη μεταβλητή θέτει τον καταχωρητή $t0 ίσο με την διεύθυνση μνήμης που βρίσκεται η μεταβλητή αυτή.

- **def loadvr(variable, register)**
  Η συγκεκριμένη συνάρτηση φορτώνει σε έναν καταχωρητή (register) την τιμή της μεταβλητής variable. Διακρίνουμε τις εξής περιπτώσεις:

  - Η variable είναι σταθερά:
    li register, v
  - Η variable είναι καθολική μεταβλητή:
    lw register,-offset($s0)
  - Η variable είναι τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
    lw register,-offset($sp)
  - Η variable είναι τοπική παράμετρος τύπου REF:
    lw $t0,-offset($sp)
    lw register,($t0)
  - Η variable είναι μη τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
    gnlvcode()
    lw register,($t0)
  - Η variable είναι μη τοπική παράμετρος τύπου REF:
    gnlvcode()
    lw $t0,($t0)
    lw register,($t0)

- **def storerv(register, variable)**
  Η συγκεκριμένη συνάρτηση γράψει στην θέση μνήμης μίας μεταβλητής (variable) την τιμή του καταχωρητή register. Διακρίνουμε τις εξής περιπτώσεις:

  - Η variable είναι καθολική μεταβλητή:
    sw register,-offset($s0)
  - Η variable είναι τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
    sw register,-offset($sp)
  - Η variable είναι τοπική παράμετρος τύπου REF:
    lw $t0,-offset($sp)
    sw register,($t0)
  - Η variable είναι μη τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
    gnlvcode()
    sw register,($t0)
  - Η variable είναι μη τοπική παράμετρος τύπου REF:
    gnlvcode()
    lw $t0,($t0)
    sw register,($t0)

- **def compile_quads()**
  Πρόκειται για την συνάρτηση που διατρέχει τον πίνακα uncompiled_quads με τις τετράδες και καλεί για κάθε μία την συνάρτηση compile_quad() ώστε να γίνει εν τέλη η παραγωγή του κώδικά της. Είναι σημαντικό να παρατηρήσουμε ότι οι τετράδες που αφορούν παραμέτρους σε αντίθεση με τις υπόλοιπες δεν μεταφράζονται απευθείας. Αλλά αποθηκεύονται σε μία ουρά και μεταφράζονται όταν βρεθεί η τετράδα τύπου ‘call ‘ καθώς απαιτείται η γνώση του framelength της συνάρτησης που θα καλεστεί προτού γραφτούν οι τιμές των παραμέτρων της στο εγγράφημα δραστηριοποίησής της.

- **def compile_quad(quad, args)**
  Πρόκειται για την συνάρτηση η παράγει τον τελικό κώδικα μίας τετράδας που παίρνει ως είσοδο. Ανάλογα με το είδος της τετράδας γίνονται οι κατάλληλες κλήσεις της συνάρτησης gen_mips_instr(), με βάση την θεωρία που διδάχθηκε στο μάθημα. Συγκεκριμένα:

  - Τετράδα άλματος: jump, _, _, label
    j label

  - Τετράδα άλματος συνθήκης: relop, x, y, label
    loadvr(x,$t1)
    loadvr(y,$t2)
    branch, $t1, $t2, label

    όπου relop ϵ {=, <>, <, >, <=, >=} και
    branch ϵ {beq, bne, bgt, blt, bge, ble}

  - Τετράδα εκχώρησης: :=, x, _, z
    loadvr(x,$t1)
    storerv($t1,z)

- Τετράδα προσήμου: sign, x, _, z
  Αν sign = ‘-’ τότε,
  loadvr(x, $t1)
  subu $t1, $zero, $t1
  storerv($t1, z)
  Αν sign = ‘+’ τότε,
  loadvr(x, $t1)
  add $t1, $zero, $t1
  storerv($t1, z)

  όπου sign ϵ {+, -}

- Τετράδα αριθμητικής πράξης: op x,y,z
  loadvr(x,$t1)
  loadvr(y,$t2)
  op_instr $t1,$t1,$t2
  storerv($t1,z)

  όπου op ϵ {+, -, *, /} και
  op_instr ϵ {add, sub, mul, div}

- Τετράδα εξόδου: out x, _, _
  li $v0,1
  loadvr(x, $t1)
  move $a0, $t1
  syscall

  - Για αλλαγή γραμμής μετά από κάθε print:
  li $v0, 4
  la $a0, newline
  syscall

  όπου newline έχει δηλωθεί στο .data τμήμα του assembly αρχείου ως: **newline: .asciiz "\n"**

- Τετράδα εισόδου: inp _, _, x
  li $v0,5
  syscall
  storerv($v0, x)

- Τετράδα επιστροφής τιμής συνάρτησης: retv _, _, x
  loadvr(x,$t1)
  lw $t0, -8($sp)
  sw $t1, ($t0)

- Παράμετροι συνάρτησης:

  Πριν την μετάφραση της πρώτης τετράδας παραμέτρου εκτελώ,
  add $fp,$sp,framelength
  Στην συνέχεια παράγω κανονικά τελικό κώδικα για τις τετράδες τύπου par ως εξής:

- par,x,CV, _ :

  loadvr(x,$t0)
  sw $t0, -(12+4i)($fp)

- par,x,CP, _ :

  loadvr(x,$t0)
  sw $t0, -(12+4i)($fp)

- par,x,REF, _:

  Αν η x είναι καθολική μεταβλητή:
  addi $t0, $s0, -offset
  sw $t0, -(12+4i) ($fp)

  Η x είναι τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
  addi $t0, $sp, -offset
  sw $t0, -(12+4i)($fp)

  Η x είναι τοπική παράμετρος τύπου REF:
  lw $t0, -offset($sp)
  sw $t0, -(12+4i)($fp)

  Η x είναι μη τοπική μεταβλητή ή παράμετρος αλλά όχι τύπου REF:
  gnlvcode(x)
  sw $t0, -(12+4i)($fp)

  Η x είναι μη τοπική παράμετρος τύπου REF:
  gnlvcode(x)
  lw $t0, ($t0)
  sw $t0, -(12+4i)($fp)

- par,x,RET, _:

  add $t0, $sp, -offset
  sw $t0, -8($fp)

όπου i ο αύξων αριθμός της παραμέτρου

- Τετράδα Κλήσης Συνάρτησης: call, _, _, f
  - Αν καλούσα και κληθείσα έχουν ίδιο βάθος φωλιάσματος,
    lw $t0, -4($sp)
    sw $t0, -4($fp)
  - Αλλιώς,
    sw $sp, -4($fp)

Στην συνέχεια:
add $sp, $sp, framelength
jal f
add $sp, $sp, -framelength

- Τετράδα αρχής μπλοκ συνάρτησης: begin_block, _, _, f
  sw, $ra, 0, $sp

- Τετράδα αρχής μπλοκ προγράμματος: begin_block, _, _, f
  addi $sp, $sp, framelength
  move $s0, $sp

- Τετράδα τέλους μπλοκ συνάρτησης: end_block, _, _, f
  lw $ra, ($sp)
  jr $ra

- Τετράδα τερματισμού προγράμματος: halt, _, _, _
  li $v0, 10
  syscall

- Πρώτη εντολή του assembly αρχείου:
  j Lmain

- **def gen_mips_instr(op, terms)**
  Πρόκειται για την συνάρτηση που παράγει μία εντολή assembly αρχιτεκτονικής MIPS σε αναπαράσταση αλφαριθμητικού και την προσθέτει στο σύνολο εντολών που θα γραφτούν στο αρχείο τελικού κώδικα.

Υλοποίηση περάσματος με αντιγραφή (CP):

Για την υλοποίηση δυνατότητας περάσματος παραμέτρων με αντίγραφο έγιναν οι εξής προσθήκες:

- **Στην compile_quads**, κρατείται μία λίστα με τις σχετικές θέσεις (τα i) όλων των παραμέτρων τύπου CP που θα γραφτούν στο εγγράφημα δραστηριοποίησης της κληθείσας συνάρτησης.

- **Στην compile_quad**, αρχικά κρατάμε τον stack pointer ($sp) της κληθείσας συνάρτησης που πλέον έχει επιστρέψει στο προσωρινό καταχωρητή $s1. Έπειτα για κάθε παράμετρο της λίστα παραμέτρων με πέρασμα τύπου CP (την οποία μας έχει περάσει ως όρισμα η compile_quads()) αποθηκεύουμε την τελική τιμή της από το εγγράφημα δραστηριοποίησης της κληθείσας συνάρτησης (χρήση του $s1) στον καταχωρητή $t1. Στην συνέχεια καλούμε απλά μία storerv('$t1', entity_par) ώστε να ενημερώσουμε την μεταβλητή που πέρασε ως παράμετρος κατά την κλήση με την νέα τιμή.

10. Έλεγχος Σφαλμάτων

Όσον αφορά τα σφάλματα που μπορεί να προκύψουν κατά την Λεκτική, Συντακτική ή Σημασιολογική Ανάλυση χρησιμοποιήθηκε ένας κεντρικοποιημένος τρόπος διαχείρισης τους που βασίζεται τόσο στην δομή 3-επίπεδου ευρετηρίου **error_map** που περιγράφηκε στην **Ενότητα 1,** όσο και στις εξής συναρτήσεις:

- **def is_valid_stat(idtk, tokentk)**

  Η λειτουργία της συγκεκριμένης συνάρτησης είναι να εξετάζει αν το ζεύγος (idtk, tokentk) που παίρνει ως όρισμα αφορά ένα έγκυρο ‘σύμβολο’ της γλώσσας, για την δεδομένη στιγμή της μεταγλώττισης όπου καλείται η συνάρτηση αυτή.
  Η συνάρτηση χρησιμοποιεί την δομή ευρετηρίου **statement_starters_set** ώστε αναγνωρίσει αν το tokentk που δέχεται είναι ένα εναρκτήριο keyword κάποιας δομής-κώδικα της γλώσσας (πχ. Το **if** από το **if**-[**else**]-**endif**).
  Συνήθως χρησιμοποιείται σε συνδυασμό με την **terminal_keyword_exist()**.
  Όταν γίνεται αυτό μπορούμε να διακρίνουμε τρεις περιπτώσεις πιθανών λαθών.
  Η πρώτη είναι, περίπτωση λάθους όπου λείπει το τερματικό keyword κάποιας δομής-κώδικα της γλώσσας (πχ. Το **endloop** από το **loop-endloop**).
  Η δεύτερη είναι η περίπτωση λάθους όπου δεν λείπει κάποιο τερματικό keyword, αλλά πριν από αυτό έχει προηγηθεί κάποιο άλλο keyword της γλώσσας (πχ. **program**) το οποίο είναι έγκυρο για την γλώσσα γενικά (άρα περνάει και από τον λεκτικό αναλυτή), αλλά δεν είναι έγκυρο για το συγκεκριμένο σημείο του προγράμματος. Και η τελευταία, είναι η περίπτωση όπου στις εντολές που προηγούνται κάποιου τερματικού keyword δομής-κώδικα λείπει το semicolon (‘;’) μίας ή περισσοτέρων από αυτές.

- **def terminal_keyword_exists(initial_keyword_id, terminal_keyword_id)**

  Η λειτουργία της συγκεκριμένης συνάρτησης είναι να δέχεται ένα εναρκτήριο (initial_keyword_id) και ένα τερματικό (terminal_keyword_id) idtk από keywords κάποιας δομής-κώδικα της γλώσσας και να ανιχνεύει τυχόν έλλειψη του τερματικού keyword στο πηγαίο κώδικα μου μεταγλωττίζεται. Ένα παράδειγμα της λειτουργίας της συνάρτησης αυτής βρίσκεται στην συνάρτηση **loop_stat()** όπου καλείται με ορίσματα ('LOOP_ID', 'ENDLOOP_ID'). Τότε η συνάρτηση διαβάζει πάλι από την αρχή όλο το source αρχείο κώδικα με επαναληπτική χρήση της **lex()** ώστε να ελέγξει αν ο αριθμός των **loop** ισούται με τον αριθμό των **endloop**, που υπάρχουν στο αρχείου κώδικα. Σε περίπτωση ισότητας το αποτέλεσμα είναι θετικό και έτσι προσδιορίζεται κάποιος άλλος τύπος λάθους, αλλιώς εμφανίζεται κατάλληλο μήνυμα λάθους στην κονσόλα περί έλλειψης του **endloop**.
  Επειδή το επαναδιάβασμα του αρχείου και θα προκαλέσει αλλαγές στις global μεταβλητές **idtk**, **tokentk** και **line_count** είναι σημαντικό να κρατούνται οι τρέχουσες τιμές τους πριν την κλήση αυτής της συνάρτησης ώστε το μήνυμα λάθους να είναι ακριβές. Όπως αναφέρθηκε και παραπάνω η συγκεκριμένη συνάρτηση συνδυάζεται συνήθως με την **is_valid_stat()** για έναν ολοκληρωμένο και ακριβή προσδιορισμό του τύπου σφάλματος.

- **def get_error_line()**

  Πρόκειται για μία συνάρτηση η οποία επιστρέφει την γραμμή του αρχείου πηγαίου

κώδικα που βρίσκεται την στιγμή της κλήσης της η μεταγλώττιση. Η συνάρτηση αυτή χρησιμοποιείται πριν την κλήση της **terminal_keyword_exist()** ώστε να αποθηκευτεί η τρέχουσα γραμμή, για τους λόγους που αφορούν την λειτουργία της τελευταίας και αναφέρθηκαν παραπάνω.

- **def error_line_is_setted(error_info)**

  Η συγκεκριμένη συνάρτηση υλοποιεί ένα έλεγχο σχετικά με το αν έχει προηγηθεί κλήση της **get_error_line()**. Χρησιμοποιείται έτσι ώστε να προσδιοριστεί και να τυπωθεί η σωστή γραμμή που εντοπίστηκε το λάθος. Το **error_info** που δέχεται σαν όρισμα πρόκειται για ένα τύπο (προσωρινού) ευρετηρίου που χρησιμοποιείται μεταξύ των συναρτήσεων διαχείρισης λαθών ώστε να γίνει η αντιστοίχιση στο σωστό μήνυμα λάθους.

- **def error_handler(mode, error_func, error_id, error_info)**

  Η συγκεκριμένη συνάρτηση είναι αυτή που καλείται σε κάθε σημείο της μεταγλώττισης προκύπτει κάποιο σφάλμα. Είναι υπεύθυνη για την εμφάνιση της γραμμής του αρχείου πηγαίου κώδικα όπου ανιχνεύτηκε το σφάλμα, αλλά και της περιγραφής αυτού του σφάλματος στην κονσόλα. Τέλος είναι η συνάρτηση που εκτελεί και άμεσο τερματισμό της διαδικασίας της μεταγλώττισης όταν κάποιο σφάλμα ανιχνευτεί.

- **def print_error_line(mode, error_info)**

  Η συγκεκριμένη συνάρτηση καλείται από την error_handler() ώστε να τυπώσει γραμμή του αρχείου πηγαίου κώδικα όπου ανιχνεύτηκε το σφάλμα. Χρησιμοποιώντας τα ορίσματα της εμφανίζει στην κονσόλα τον τύπο του λάθους και τις γραμμές του αρχείου γύρω από τις οποίες βρίσκεται αυτό.

- **def print_error_msg(mode, error_func, error_id, error_info)**

  Η συγκεκριμένη συνάρτηση καλείται από την error_handler() ώστε να τυπώσει μία λεπτομερή περιγραφή για το σφάλμα. Το μόνο που κάνει είναι η εξαγωγή του μηνύματος λάθους από το 3-επίπεδο ευρετηρίο **error_map** με βάση τα ορίσματα με τα οποία καλείται.