Ο ΜΕΤΑΓΛΩΤΤΙΣΤΗΣ ΤΗΣ greek++

Χήτα Δανάη – ΑΜ: 4838

Τμήμα Μηχανικών Η/Υ Πανεπιστήμιο Ιωαννίνων

Μάιος 2025

Μάθημα: Μεταφραστές

ΠΕΡΙΕΧΌΜΕΝΑ:

1)Περιγραφή της γλώσσας greek++

Η γλώσσα προγραμματισμού **greek++** είναι μια απλή και εκπαιδευτική γλώσσα η οποία, παρά την απλότητά της, ενσωματώνει πολλά από τα στοιχεία που συναντώνται σε δημοφιλείς γλώσσες προγραμματισμού. Υποστηρίζει συναρτήσεις και διαδικασίες, προσφέροντας τη δυνατότητα για οργάνωση και επανάχρηση του κώδικα. Επίσης, επιτρέπει την αναδρομή, δηλαδή τη δυνατότητα μιας συνάρτησης να καλεί τον εαυτό της. Διαθέτει, ακόμα, όλες τις σημαντικές δομές ελέγχου που χρησιμοποιούνται συχνά, όπως τις δομές απόφασης (εάν-τότε-αλλιώς) και τις δομές επανάληψης (επανάλαβε-μέχρι, όσο, και για).

Η γλώσσα greek++ περιλαμβάνει ένα σύνολο από δεσμευμένες λέξεις, οι οποίες είναι οι εξής:

πρόγραμμα, δήλωση, εάν, τότε, αλλιώς, εάν_τέλος, επανάλαβε, μέχρι, όσο, όσο_τέλος, για, έως, με_βήμα, για_τέλος, διάβασε, γράψε, συνάρτηση, διαδικασία, διαπροσωπεία, είσοδος, έξοδος, αρχή_συνάρτησης, τέλος_συνάρτησης, αρχή_διαδικασίας, τέλος_διαδικασίας, αρχή_προγράμματος, τέλος_προγράμματος, ή, και, όχι, εκτέλεσε.

Το αλφάβητο της greek++ περιλαμβάνει:

- Μικρά και κεφαλαία γράμματα της ελληνικής και της λατινικής αλφαβήτου (Α…Ω, α…ω, A…Z, a…z)
- Αριθμητικά ψηφία (0…9)
- Σύμβολα των αριθμητικών πράξεων (+, -, *, /)
- Τελεστές συσχέτισης (<, >, =, <=, >=, <>)
- Σύμβολο ανάθεσης (:=)
- Σύμβολα διαχωρισμού (;, ,)
- Σύμβολα ομαδοποίησης ((,), [,])
- Σχόλια ({, })
- Σύμβολο περάσματος παραμέτρων με αναφορά (%)

Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από μια ακολουθία αριθμητικών ψηφίων, με προαιρετικό πρόσημο στην αρχή.

Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα, ψηφία και τον χαρακτήρα κάτω παύλα (_), αρχίζοντας πάντα με γράμμα. Κάθε αναγνωριστικό μπορεί να περιέχει το πολύ 30 χαρακτήρες. Αναγνωριστικά με περισσότερους από 30 χαρακτήρες θεωρούνται λανθασμένα.

Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται πλήρως από τον μεταγλωττιστή και μπορούν να χρησιμοποιούνται ελεύθερα για την ευανάγνωστη διαμόρφωση του κώδικα, υπό τον όρο να μην εμφανίζονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά ή σταθερές. Τα σχόλια πρέπει να περιλαμβάνονται ανάμεσα στα σύμβολα { και } και αγνοούνται επίσης από τον μεταγλωττιστή.

Οι τύποι δεδομένων που υποστηρίζει η greek++ είναι οι ακέραιοι και οι πραγματικοί αριθμοί. Η δήλωση των μεταβλητών γίνεται με την εντολή *δήλωση*, ακολουθούμενη από τα ονόματα των μεταβλητών, που χωρίζονται με κόμματα. Στο τέλος ακολουθεί άνω και κάτω τελεία (:) και ο τύπος των δεδομένων (π.χ., ακέραιος, πραγματικός), όπως φαίνεται παρακάτω. Επιπλέον επιτρέπεται η χρήση πολλαπλών γραμμών δηλώσεων.

δήλωση α, β : ακέραιος

Η προτεραιότητα των τελεστών στη γλώσσα greek++, από τη μεγαλύτερη στη μικρότερη, είναι:

- Μοναδικοί λογικοί τελεστές: όχι
- Πολλαπλασιαστικοί τελεστές: *, /
- Μοναδικοί προσθετικοί τελεστές: +, -
- Δυαδικοί προσθετικοί τελεστές: +, -
- Σχεσιακοί τελεστές: =, <, >, <>, <=, >=
- Λογικοί τελεστές: και, ή

Μορφή Προγράμματος

Κάθε πρόγραμμα στη greek++ ξεκινά με τη λέξη-κλειδί `πρόγραμμα` και ακολουθείται από το όνομά του. Η βασική δομή περιλαμβάνει δηλώσεις μεταβλητών, υποπρογράμματα (συναρτήσεις και διαδικασίες που μπορεί να είναι και φωλιασμένες), και τέλος τις εντολές του κυρίως προγράμματος, που τοποθετούνται ανάμεσα στα `αρχή_προγράμματος` και `τέλος_προγράμματος`.

πρόγραμμα id
  δηλώσεις
  υποπρογράμματα

αρχή_προγράμματος
εντολές
τέλος_προγράμματος

Περιγραφή του μεταγλωττιστή της Greek++

Ο μεταγλωττιστής της greek++ έχει υλοποιηθεί σε γλώσσα Python. Εκτελείται μέσω της γραμμής εντολών και δέχεται ως είσοδο αρχεία με κατάληξη `.gr` που περιέχουν προγράμματα γραμμένα στη γλώσσα greek++. Εάν το αρχείο δεν ακολουθεί τους κανόνες της γλώσσας, ο μεταγλωττιστής εμφανίζει διαγνωστικά μηνύματα λάθους και διακόπτει τη λειτουργία του.

Αν το πρόγραμμα είναι σωστό, ο μεταγλωττιστής παράγει:

- Ένα αρχείο ενδιάμεσου κώδικα (`.int`) που περιέχει τις τετράδες (quads) ενδιάμεσης αναπαράστασης.
- Ένα αρχείο assembly (`.asm`) το οποίο μπορεί να τρέξει σε επεξεργαστή MIPS.

Στάδια μεταγλώττισης

Ο μεταγλωττιστής ακολουθεί τα παρακάτω βασικά στάδια, που εφαρμόζονται διαδοχικά σε κάθε πρόγραμμα:

1. **Λεκτική ανάλυση:**
   Το πρόγραμμα διαβάζεται χαρακτήρα-χαρακτήρα και «σπάει» σε λεκτικές μονάδες (tokens), όπως αναγνωριστικά, αριθμοί, σύμβολα κτλ. Εδώ ανιχνεύονται και τα πρώτα λάθη, όπως άκυροι χαρακτήρες ή πολύ μεγάλα αναγνωριστικά.
2. **Συντακτική ανάλυση:**
   Έλεγχος αν η ακολουθία των tokens σχηματίζει συντακτικά ορθό πρόγραμμα, σύμφωνα με τη γραμματική της greek++. Υλοποιείται με τεχνικές αναδρομικής κατάβασης.
3. **Σημασιολογική ανάλυση:**
   Ελέγχεται αν τα ονόματα μεταβλητών, συναρτήσεων κτλ. δηλώνονται σωστά και δεν υπάρχουν διπλές δηλώσεις, λάθος χρήσεις παραμέτρων ή άλλες λογικές ασυμβατότητες. Ο πίνακας συμβόλων ενημερώνεται ανάλογα με τα scopes.
4. **Παραγωγή ενδιάμεσου κώδικα:**
   Το πρόγραμμα μετατρέπεται σε μια σειρά από τετράδες (quads), που απλοποιούν τη μεταφορά σε τελικό κώδικα.
5. **Παραγωγή τελικού κώδικα:**
   Οι τετράδες μεταφράζονται σε κώδικα Assembly για MIPS, με τη σωστή διαχείριση μνήμης, μεταβλητών, παραμέτρων και επιστροφών από υποπρογράμματα.

Αυτά τα στάδια εξασφαλίζουν ότι το πρόγραμμα θα μεταφραστεί σωστά και με ασφάλεια σε χαμηλότερου επιπέδου γλώσσα.

2) Λεκτική Ανάλυση

Η λεκτική ανάλυση είναι το πρώτο στάδιο της μεταγλώττισης στη greek++. Σε αυτό το στάδιο, το πρόγραμμα διαβάζεται χαρακτήρα προς χαρακτήρα και σπάει σε λεκτικές μονάδες (tokens), όπως λέξεις-κλειδιά, αναγνωριστικά, αριθμούς, σύμβολα πράξεων κ.ά.

Ο λεκτικός αναλυτής που υλοποιήσαμε σε Python (μέσα στη συνάρτηση lex()), διαβάζει το αρχείο εισόδου και, για κάθε τμήμα του κειμένου, εντοπίζει αν ανήκει σε κάποια από τις γνωστές κατηγορίες της γλώσσας. Για παράδειγμα, ξεχωρίζει αν κάτι είναι δεσμευμένη λέξη, αναγνωριστικό, αριθμητική σταθερά, σύμβολο τελεστή, διαχωριστικό, ή αν πρόκειται για σχόλιο (το οποίο αγνοείται).

Υλοποιήσαμε τον λεκτικό αναλυτή στη γλώσσα greek++ με χρήση ενός αυτόματου πεπερασμένων καταστάσεων το οποίο ορίσαμε μέσα στον πίνακα Trans_Diagram.

Ο λεκτικός αναλυτής φροντίζει επίσης να εντοπίζει και να αναφέρει λάθη που αφορούν τις λεκτικές μονάδες, όπως:

- Αναγνωριστικά με περισσότερους από 30 χαρακτήρες
- Μη επιτρεπτούς χαρακτήρες
- Αριθμούς εκτός επιτρεπτού ορίου
- Σχόλια που δεν έχουν κλείσει σωστά

Άνοιγμα του αρχείου

Αρχικά ξεκινάμε εισάγοντας το module sys για να διαβάσουμε το όνομα αρχείου από τη γραμμή εντολών και ανοίγουμε το αρχείο πηγαίου κώδικα που δίνεται ως πρώτο όρισμα. Η μεταβλητή lines ξεκινά από 1 και μετράει τις γραμμές κώδικα, ώστε να μπορούμε να εμφανίζουμε ακριβή μηνύματα σφαλμάτων.

```
1  import sys
2
3
4
5  infile = open(sys.argv[1])
6  lines = 1
```

ΔημΙΟυργία συνάρτησης charToInt

Δημιουργούμε την συνάρτηση charToInt, η οποία έχει ως σκοπό να μετατρέψει έναν χαρακτήρα σε έναν αριθμό που αντιστοιχεί σε μία συγκεκριμένη κατηγορία χαρακτήρων, ώστε ο λεκτικός αναλυτής να μπορεί να χειριστεί κάθε κατηγορία με βάση τις μεταβάσεις του αυτόματου. Με αυτό τον τρόπο, ο λεκτικός αναλυτής μπορεί να διαχειριστεί κατηγορίες χαρακτήρων αντί για κάθε μεμονωμένο χαρακτήρα ξεχωριστά.

Πιο συγκεκριμένα πραγματοποιεί την αντιστοίχηση όπως περιγράφουμε παρακάτω:

- Οι **λευκοί χαρακτήρες** όπως το κενό (`' '`), το tab (`'\t'`) και το newline (`'\n'`) αντιστοιχούν στην κατηγορία 0. Αυτοί οι χαρακτήρες δεν παράγουν tokens αλλά χρησιμοποιούνται για διαχωρισμό.
- Αν ο χαρακτήρας είναι κενή συμβολοσειρά (`''`), που σημαίνει το **τέλος του αρχείου (EOF)**, επιστρέφει την κατηγορία 19.
- Τα **κεφαλαία λατΙΝΙκά γράμματα** (`'A'` έως `'Z'`), τα **πεζά λατΙΝΙκά** (`'a'` έως `'z'`), τα **κεφαλαία ελληνΙκά** (`'Α'` έως `'Ω'`), τα **πεζά ελληνΙκά** (`'α'` έως `'ω'`), καθώς και τα ελληνικά γράμματα με τόνο (`'ά', 'έ', 'ή', 'ί', 'ό', 'ύ', 'ώ'`) ταξινομούνται όλα στην κατηγορία 1. Αυτή η κατηγορία χρησιμοποιείται για την αναγνώριση **αναγνωρΙστΙκών καΙ λέξεων-κλεΙδΙών**.
- Οι **ψηφία** (`'0'` έως `'9'`) ανήκουν στην κατηγορία 2, ώστε να αναγνωρίζονται ως αριθμητικές σταθερές.
- Οι **τελεστές** όπως το συν (`'+'`), μείον (`'-'`), πολλαπλασιασμός (`'*'`), διαίρεση (`'/'`), ισότητα (`'='`), μικρότερο (`'<'`), μεγαλύτερο (`'>'`) και το σύμβολο ανάθεσης (`':'`) ταξινομούνται σε κατηγορίες 3 έως 10 αντίστοιχα, για να αναγνωρίζονται και να διαχειρίζονται σωστά.
- Τα **σύμβολα ομαδοποίησης** και διαχωριστικά, όπως άγκιστρα (`'{', '}'`), άνω και κάτω τελείες, ερωτηματικό, παρένθεση, άγκιστρα τετραγωνικά (`'[', ']'`), ερωτηματικά, κόμμα, ερωτηματικό και ποσοστό, αντιστοιχούν στις κατηγορίες 11 έως 21.
- Ο χαρακτήρας **`'_'`** ταξινομείται στην κατηγορία 20, καθώς μπορεί να χρησιμοποιηθεί σε αναγνωριστικά.
- Οποιοσδήποτε άλλος χαρακτήρας που δεν ανήκει σε αυτές τις κατηγορίες κατατάσσεται στην κατηγορία 22, που θεωρείται **σφάλμα** και θα χειριστείται κατάλληλα από τον λεκτικό αναλυτή.

Με αυτή τη μέθοδο, η συνάρτηση `charToInt` διευκολύνει την ομαδοποίηση χαρακτήρων και επιτρέπει στο αυτόματο να μεταβαίνει στις σωστές καταστάσεις, κάνοντας πιο αποδοτική και οργανωμένη τη λεκτική ανάλυση.

```
def charToInt(ch):  1 usage
    if ch in [' ', '\t', '\n']:
        return 0
    elif ch == '':
        return 19
    elif ch in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ':
        return 1
    elif ch in 'abcdefghijklmnopqrstuvwxyz':
        return 1
    elif ch in 'ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ':
        return 1
    elif ch in 'αβγδεζηθικλμνξοπρσςτυφχψω':
        return 1
    elif ch in 'άέήίόύώ':
        return 1
    elif ch in '0123456789':
        return 2
    elif ch == '+':
        return 3
    elif ch == '-':
        return 4
    elif ch == '*':
        return 5
    elif ch == '/':
        return 6
    elif ch == '=':
        return 7
    elif ch == '<':
        return 8
    elif ch == '>':
        return 9
    elif ch == ':':
        return 10
    elif ch == '{':
        return 11
    elif ch == ';':
        return 12
    elif ch == ',':
        return 13
    elif ch == '(':
        return 14
    elif ch == ')':
        return 15
    elif ch == '[':
        return 16
    elif ch == ']':
        return 17
    elif ch == '%':
        return 18
    elif ch == '_':
        return 20
    elif ch == '}':
        return 21
    else:
        return 22

```

Πίνακας μεταβάσεων Trans_Diagram – Συνάρτηση init()

Αρχικά ορίζεται το μέγεθος του πίνακα: 7 καταστάσεις (γραμμές) και 23 κατηγορίες χαρακτήρων (στήλες). Ο πίνακας γεμίζει αρχικά με μηδενικά (κατάσταση 0). Στην συνέχεια ο πίνακας καθορίζει σε ποια κατάσταση πρέπει να μεταβεί ο λεκτικός αναλυτής ανάλογα με το χαρακτήρα που διαβάζει. Για παράδειγμα, στην κατάσταση 0 (αρχική), αν ο χαρακτήρας είναι γράμμα (στήλη 1), ο αυτόματος μεταβαίνει στην κατάσταση 1 (αναγνωριστικά). Αν είναι αριθμός (στήλη 2), μεταβαίνει στην κατάσταση 2 (ακέραιοι αριθμοί). Αν ο χαρακτήρας είναι το σύμβολο +, επιστρέφεται το token addtk. Οι καταστάσεις 3, 4, 5 και 6 αφορούν ειδικές περιπτώσεις όπως τελεστές σύγκρισης, ανάθεση και σχόλια.

```
def init():  1 usage
    global Trans_Diagram, greekWords

    col_num = 23
    Trans_Diagram = [[0 for j in range(col_num)] for i in range(7)]

    Trans_Diagram[0][0] = 0
    Trans_Diagram[0][1] = 1
    Trans_Diagram[0][2] = 2
    Trans_Diagram[0][3] = 'addtk'
    Trans_Diagram[0][4] = 'subtk'
    Trans_Diagram[0][5] = 'multk'
    Trans_Diagram[0][6] = 'divtk'
    Trans_Diagram[0][7] = 'equaltk'
    Trans_Diagram[0][8] = 3
    Trans_Diagram[0][9] = 4
    Trans_Diagram[0][10] = 5
    Trans_Diagram[0][11] = 6
    Trans_Diagram[0][12] = 'semicolontk'
    Trans_Diagram[0][13] = 'commatk'
    Trans_Diagram[0][14] = 'leftpartk'
    Trans_Diagram[0][15] = 'rightpartk'
    Trans_Diagram[0][16] = 'leftbrackettk'
    Trans_Diagram[0][17] = 'rightbrackettk'
    Trans_Diagram[0][18] = 'percenttk'
    Trans_Diagram[0][19] = 'eoftk'
    Trans_Diagram[0][20] = 'errortk'
    Trans_Diagram[0][21] = 'errortk'
    Trans_Diagram[0][22] = 'errortk'

    for i in range(col_num):
        Trans_Diagram[1][i] = 'idtk'
    Trans_Diagram[1][1] = 1
    Trans_Diagram[1][2] = 1
    Trans_Diagram[1][20] = 1

    for i in range(col_num):
        Trans_Diagram[2][i] = 'integertk'
    Trans_Diagram[2][2] = 2

    for i in range(col_num):
        Trans_Diagram[3][i] = 'lessthantk'
    Trans_Diagram[3][7] = 'lessequaltk'
    Trans_Diagram[3][9] = 'nonequaltk'

```

```
    for i in range(col_num):
        Trans_Diagram[4][i] = 'morethantk'
    Trans_Diagram[4][7] = 'moreequaltk'

    for i in range(col_num):
        Trans_Diagram[5][i] = 'error1tk'
    Trans_Diagram[5][7] = 'assigntk'

    for i in range(col_num):
        Trans_Diagram[6][i] = 6
    Trans_Diagram[6][21] = 0
    Trans_Diagram[6][19] = 'error2tk'

    greekWords = ['πρόγραμμα', 'δήλωση', 'εάν', 'τότε', 'αλλιώς', 'εάν_τέλος', 'επανάλαβε', 'μέχρι', 'όσο', 'όσο_τέλος',
                  'για', 'έως', 'με_βήμα', 'για_τέλος', 'διάβασε', 'γράψε', 'συνάρτηση', 'διαδικασία', 'είσοδος', 'έξοδος',
                  'διαπροσωπεία', 'αρχή_συνάρτησης', 'τέλος_συνάρτησης', 'αρχή_διαδικασίας', 'τέλος_διαδικασίας',
                  'αρχή_προγράμματος', 'τέλος_προγράμματος', 'ή', 'και', 'όχι', 'εκτέλεσε']
```

ΔημΙΟυργία συνάρτησης lex()

Η lex() ξεκινά δηλώνοντας ότι θα χρησιμοποιήσει τις παγκόσμιες μεταβλητές lines (για αρίθμηση γραμμών), Trans_Diagram (τον πίνακα μεταβάσεων του αυτόματου), και greekWords (λίστα με τις λέξεις-κλειδιά της γλώσσας). Αρχικοποιεί την κατάσταση του αυτόματου (state) στο 0 (αρχική κατάσταση). Η μεταβλητή word είναι μια κενή συμβολοσειρά που θα χρησιμοποιηθεί για να συλλέξει τους χαρακτήρες που απαρτίζουν κάθε token.

```
def lex():  70 usages
    global lines, Trans_Diagram, greekWords

    state = 0
    word = ''
```

Στην συνέχεια ξεκινά η εκτέλεση του βρόχου, όσο η κατάσταση του αυτόματου είναι μικρότερη από 7 (δηλαδή βρίσκεται σε μια «ενδιάμεση» κατάσταση και όχι σε τελικό token ή σφάλμα). Διαβάζει τον επόμενο χαρακτήρα από το αρχείο (ch) και τον προσθέτει στη μεταβλητή word. Αν ο χαρακτήρας είναι αλλαγή γραμμής (\n), αυξάνει τον μετρητή γραμμών για σωστή αναφορά σφαλμάτων. Ο χαρακτήρας μετατρέπεται σε κατηγορία-στήλη με τη charToInt(ch)και η νέα κατάσταση του αυτόματου προκύπτει από τον πίνακα μεταβάσεων. Αν επιστρέψει πάλι στην αρχική κατάσταση (state == 0), μηδενίζει το word, δηλαδή αγνοεί ό,τι έχει διαβάσει μέχρι τώρα (αυτό γίνεται συνήθως για whitespace, σχόλια κλπ).

```
    while str(state)[0] < '7':
        ch = infile.read(1)
        word = word + ch
        if ch == '\n':
            lines = lines + 1

        col = charToInt(ch)
        state = Trans_Diagram[state][col]

        if str(state)[0] == '0':
            word = ''
```

Εάν η κατάσταση που βρέθηκε είναι σφάλμα (errortk, error1tk, error2tk), τότε περνάμε στον έλεγο και την διαχείρηση σφαλμάτων. Εκεί η lex() τυπώνει ένα κατάλληλο μήνυμα σφάλματος με τη γραμμή όπου συνέβη το λάθος και τερματίζει την εκτέλεση του προγράμματος. Αυτά τα σφάλματα αφορούν άγνωστους χαρακτήρες, λάθος χρήση του : χωρίς να ακολουθεί =, και μη κλεισμένα σχόλια αντίστοιχα.

```
156         if state == 'errortk':
157             print('Line %d: Unexptected character'%(lines))
158             exit(0)
159         elif state == 'error1tk':
160             print('Line %d: : should be followed by ='%(lines))
161             exit(0)
162         elif state == 'error2tk':
163             print('Line %d: : eof inside comments'%(lines))
164             exit(0)
165
```

Στην συνέχεια, όταν το αυτόματο τερματίσει στην κατάσταση idtk (αναγνωριστικό), έχει ήδη διαβάσει έναν παραπάνω χαρακτήρα που δεν ανήκει στο αναγνωριστικό (συνήθως ένα κενό ή άλλο σύμβολο). Αν αυτός ο χαρακτήρας ήταν αλλαγή γραμμής, μειώνουμε το lines κι έπειτα αφαιρούμε τον τελευταίο χαρακτήρα από το word για να μείνει ακριβώς το αναγνωριστικό. Στην συνέχεια επαναφέρουμε τον δείκτη αρχείου μία θέση πίσω για να μην χαθεί ο επόμενος χαρακτήρας στο επόμενο κάλεσμα της lex(). Ελέγχουμε αν το αναγνωριστικό ξεπερνά τους 30 χαρακτήρες και αν ναι, εμφανίζεται μήνυμα λάθους και το πρόγραμμα τερματίζει. Τέλος, αν το word που βρέθηκε ανήκει στις λέξεις-κλειδιά, το state αλλάζει ώστε να επιστραφεί το κατάλληλο token (π.χ. πρόγραμματk).

Αντίστοιχα, όταν η κατάσταση είναι ακέραιος αριθμός (integertk) ή τελεστές σύγκρισης (lessthantk, morethantk), εφαρμόζεται η ίδια λογική, δηλαδή αν ο τελευταίος χαρακτήρας είναι αλλαγή γραμμής, διορθώνουμε το μετρητή γραμμών. Έπειτα φαιρούμε τον τελευταίο χαρακτήρα που δεν ανήκει στο token και κάνουμε rewind το αρχείο κατά 1 θέση.

Τελικά, η συνάρτηση επιστρέφει το είδος του token (δηλαδή το τελικό state, π.χ. 'idtk', 'πρόγραμματk', 'addtk', 'integertk' κλπ) και το κείμενο του token (word) που διαβάστηκε.

```
    if state == 'idtk':
        if ch == '\n':
            lines = lines - 1

        word = word[:-1]
        infile.seek(infile.tell()-1, whence: 0)

        if len(word) > 30:
            print('Line %d: : variable above lenght'%(lines))
            exit(0)

        if word in greekWords:
            state = word + 'tk'
    elif state == 'integertk' or state == 'lessthantk' or state == 'morethantk':
        if ch == '\n':
            lines = lines - 1
        word = word[:-1]
        infile.seek(infile.tell()-1, whence: 0)

    return state, word
```

3)Συντακτική Ανάλυση

Ο συντακτικός αναλυτής της Greek++ έχει ως στόχο να διαβάσει τη σειρά των λεκτικών μονάδων (tokens) που του παραδίδει ο λεκτικός αναλυτής (lex()), και να ελέγξει αν το πρόγραμμα ακολουθεί τους συντακτικούς κανόνες της γλώσσας. Κάθε συντακτικός κανόνας υλοποιείται ως ξεχωριστή συνάρτηση στον κώδικα και αυτές οι συναρτήσεις καλούν η μία την άλλη, ακολουθώντας τη δομή του προγράμματος, μέχρι να φτάσουμε στο τέλος του αρχείου ή στο σύμβολο τερματισμού κάθε κανόνα. Εάν το πρόγραμμα δεν ακολουθεί σωστά κάποιο συντακτικό κανόνα, ο αναλυτής τερματίζει αμέσως και εμφανίζει ένα κατανοητό μήνυμα λάθους, στο οποίο φαίνεται ο τύπος του σφάλματος, η γραμμή στην οποία βρέθηκε και ποιο ήταν το πρόβλημα.

Στην αρχή του προγράμματος ελέγχουμε αν δόθηκε αρχείο εισόδου στη γραμμή εντολών. Αν όχι, τερματίζει με μήνυμα. Στην συνέχεια η inputCheck() χρησιμοποιείται για να ελέγξει ότι το επόμενο token είναι αυτό που περιμένει ο συντακτικός κανόνας. Αν δεν ταιριάζει, τερματίζει με αναφορά γραμμής και περιγραφή λάθους.

```
197
198  def inputCheck(actualtk, actual, expectedtk):  41 usages
199      global lines
200      if actualtk != expectedtk:
201          print("Line %d: Expected '%s' but got '%s'"%(lines, expectedtk[:-2], actual))
202          exit(0)
203
204
205  if len(sys.argv) < 2:
206      print("Please provide input file")
207      exit(0)
```

Ένας από τους κανόνες του συντακτικού αναλυτή είναι ο κανόνας program, ο οποίος ελέγχει αν ακολουθούνται σωστά τα βασικά στοιχεία στην αρχή και στο τέλος ενός Greek++ προγράμματος, όπως φαίνεται παρακάτω:

```
209  def program(state, word):  1 usage
210      inputCheck(state, word,  expectedtk: 'πρόγραμματk')
211      state, word = lex()
212      inputCheck(state, word,  expectedtk: 'idtk')
213      progname = word
214      state, word = lex()
215      state, word = programblock(state, word, progname)
216
217      inputCheck(state, word,  expectedtk: 'eoftk')
218      print("Program compiled with no errors");
219
```

Γενικότερα, όλοι οι κανόνες που υπάρχουν στην γραμματική της greek++ στον κώδικα μας είναι υλοποιημένες συναρτήσεις που όλες μαζί αποτελούν τον Συντακτικό Αναλυτή.

4)Παραγωγή ενδιάμεσου κώδικα

Η παραγωγή ενδιάμεσου κώδικα στην υλοποίηση της Greek++ γίνεται μέσω της δημιουργίας και διαχείρισης τετράδων (quads), οι οποίες αποτελούν ένα ευέλικτο μέσο απεικόνισης των βασικών εντολών του προγράμματος σε μορφή κατάλληλη για περαιτέρω μεταγλώττιση ή μετατροπή σε γλώσσα-στόχο.

Ο ενδιάμεσος κώδικας αποτελείται από τετράδες, δηλαδή δομές δεδομένων με έναν τελεστή (op), που δηλώνει τη λειτουργία (π.χ. +, -, *, /, :=, jump, κλπ), και τρία τελούμενα (x, y, z), που μπορεί να είναι ονόματα μεταβλητών, σταθερές, ή κενές θέσεις (ανάλογα με τον τελεστή).

Κάθε τετράδα είναι αριθμημένη (δηλαδή έχει έναν μοναδικό αριθμό/ετικέτα), και η εκτέλεσή τους γίνεται σειριακά με βάση αυτή την αρίθμηση, εκτός αν κάποιος τελεστής ελέγχει τη ροή.

Στην υλοποίηση, χρησιμοποιούνται διάφορα είδη τετράδων:

- **Αριθμητικές πράξεις:**
  op, x, y, z → Το op μπορεί να είναι +, -, *, /. Τα x και y είναι τελούμενα (μεταβλητές ή σταθερές) και z είναι η μεταβλητή όπου αποθηκεύεται το αποτέλεσμα.
- **Εκχώρηση τιμής:**
  :=, x, _, z → Μεταφέρει την τιμή του x στο z.
- **Άλματα και συνθήκες:**
  jump, _, _, label → Άνευ όρου άλμα στην τετράδα με label.
  relop, x, y, label → Εάν η relop (π.χ. <, >, <=, >=, =, <>) ισχύει μεταξύ x και y, πήγαινε στην τετράδα με label.
- **Διαχείριση ενοτήτων:**
  begin_block, name, _, _ → Έναρξη προγράμματος/συνάρτησης/διαδικασίας.
  end_block, name, _, _ → Λήξη προγράμματος/συνάρτησης/διαδικασίας.
  halt, _, _, _ → Τερματισμός προγράμματος.
- **Διαχείριση παραμέτρων και κλήσεων:**
  par, x, mode, _ → Καθορισμός παραμέτρου συνάρτησης (mode: CV για τιμή, REF για αναφορά, RET για επιστροφή τιμής).
  call, name, _, _ → Κλήση διαδικασίας/συνάρτησης με όνομα name.
  retv, x, _, _ → Επιστροφή τιμής x από συνάρτηση.

Για την διαχείριση των τετράδων και την παραγωγή του κώδικα χρησιμοποιούμε κάποιες βοηθητικές συναρτήσεις, οι οποίες είναι:

- nextquad()
  - επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
- genquad(op, x, y, z)
  - δημιουργεί την επόμενη τετράδα (op, x, y, z)
- newtemp()
  - δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή
  - οι προσωρινές μεταβλητές είναι της μορφής

    T_1, T_2, T_3 ...
- emptylist()
  - δημιουργεί μία κενή λίστα ετικετών τετράδων
- makelist(x)
  - δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x
- merge($list_1$, $list_2$)
  - δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών $list_1$, $list_2$
- backpatch(list,z)
  - η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο
  - η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z

Τις υλοποιήσαμε ως εξής:

```
668  def nextquad():  30 usages
669      return len(quads)
670
671  def genquad(op, x, y, z):  32 usages
672      global quads
673
674      quads.append([str(nextquad()), op, str(x), str(y), str(z)])
675
676  def newTemp():  4 usages
677      global temps, scopes
678
679      temps = temps + 1
680      t = 't@' + str(temps)
681      addEntity([t, 'temp', scopes[0][3]])
682      return t
683
684  def emptylist():  2 usages
685      return []
686
687  def makelist(x):  11 usages
688      return [x]
```

```
689
690  def merge(list1, list2):  7 usages
691      list1.extend(list2)
692      return list1
693
694  def backpatch(list1, z):  14 usages
695      global quads
696
697      for i in list1:
698          quads[i][4] = str(z)
```

Τις συναρτήσεις αυτές τις χρησιμοποιούμε μέσα στις συναρτήσεις που έχουν δημιουργηθεί για την συντακτική ανάλυση. Κάποια παραδείγματα είναι τα εξής:

- Στην περίπτωση της εκχώρησης τιμής σε μια μεταβλητή, αφού αναγνωριστεί το σύμβολο εκχώρησης (:=), ακολουθεί η ανάλυση της έκφρασης στη δεξιά πλευρά και παράγεται μία τετράδα μέσω της συνάρτησης genquad. Η τετράδα έχει ως τελεστή το ':=' και εκφράζει ότι το αποτέλεσμα της έκφρασης eplace πρέπει να εκχωρηθεί στη μεταβλητή id. Με αυτό τον τρόπο, κάθε εντολή ανάθεσης στο πρόγραμμα μετατρέπεται αυτόματα σε τετράδα στον ενδιάμεσο κώδικα.

```
381  def assignment_stat(id):  1 usage
382      state, word = lex()
383      inputCheck(state, word,  expectedtk: 'assigntk')
384      state, word = lex()
385      state, word, eplace = expression(state, word)
386      genquad( op: ':=', eplace,  y: '_', id)
387      return state, word
388
```

- Στην υλοποίηση της δομής if, γίνεται αρχικά ανάλυση της συνθήκης και δημιουργούνται λίστες τετράδων για τα αποτελέσματα αληθές ή ψευδές. Με την backpatch ενημερώνονται οι τετράδες ώστε να οδηγούν στην κατάλληλη συνέχεια του προγράμματος. Επιπλέον, δημιουργείται τετράδα άλματος (jump) για τη διαχείριση της εναλλακτικής διαδρομής (else), ενώ όλες οι εντολές μέσα στο if και το else παράγουν τις δικές τους τετράδες με σωστή αλληλουχία. Έτσι, η ροή του προγράμματος ελέγχεται πλήρως από τον ενδιάμεσο κώδικα που παράγεται σε αυτό το στάδιο.

```
389  def if_stat():  1 usage
390      state, word = lex()
391      state, word, btrue, bfalse = condition(state, word)
392      inputCheck(state, word,  expectedtk: 'τότετk')
393      backpatch(btrue, nextquad())
394      state, word = lex()
395      state, word = sequence(state, word)
396      ifList = makelist(nextquad())
397      genquad( op: 'jump',  x: '_',  y: '_',  z: '_')
398      backpatch(bfalse, nextquad())
399
400      if state == 'αλλιώςτk':
401          state, word = elsepart(state, word)
402      inputCheck(state, word,  expectedtk: 'εάν_τέλοςτk')
403      backpatch(ifList, nextquad())
404      state, word = lex()
405
406      return state, word
```

- Στην επαναληπτική δομή while, η παραγωγή του ενδιάμεσου κώδικα ξεκινά σημειώνοντας την αρχή του βρόχου με τη μεταβλητή bquad (μέσω nextquad). Για κάθε επανάληψη ελέγχεται η συνθήκη και χρησιμοποιείται η backpatch ώστε να γίνεται το κατάλληλο άλμα είτε στην αρχή της επανάληψης (αν η συνθήκη ισχύει), είτε στην έξοδο από τον βρόχο (αν δεν ισχύει). Τέλος, παράγεται τετράδα τύπου 'jump' για την επιστροφή στην αρχή του βρόχου. Με αυτό τον τρόπο διασφαλίζεται ότι ο ενδιάμεσος κώδικας απεικονίζει σωστά τη λογική της επανάληψης του αρχικού προγράμματος.

```
413     def while_stat():  1 usage
414         bquad = nextquad()
415         state, word = lex()
416         state, word, btrue, bfalse = condition(state, word)
417         inputCheck(state, word,  expectedtk: 'επανάλαβετk')
418         backpatch(btrue, nextquad())
419         state, word = lex()
420         state, word = sequence(state, word)
421         genquad( op: 'jump',  x: '_',  y: '_', bquad)
422         backpatch(bfalse, nextquad())
423         inputCheck(state, word,  expectedtk: 'όσο_τέλοςtk')
424         state, word = lex()
425
426         return state, word
```

Και οι υπόλοιπες συναρτήσεις του συντακτικού αναλυτή έχουν τροποποιηθεί για να παράγουν τον ενδιάμεσο κώδικα ανάλογα με την παραπάνω διαδικασία ακολουθώντας τα πρότυπα του pdf του μαθήματος.

5) Πίνακας Συμβόλων – ΣημασιΟλογΙκή Ανάλυση

Η σημασιολογική ανάλυση αποτελεί το στάδιο του μεταγλωττιστή που ελέγχει αν το πρόγραμμα – εκτός από συντακτικά σωστό – έχει και νοηματική (σημασιολογική) ορθότητα. Δηλαδή, ελέγχει αν οι δηλώσεις, οι τύποι, οι μεταβλητές, οι παράμετροι, οι εκχωρήσεις κλπ. χρησιμοποιούνται με σωστό τρόπο βάσει των κανόνων της γλώσσας και πραγματοποιείται μέσω του πίνακα συμβόλων και των συναρτήσεων που τον διαχειρίζονται.

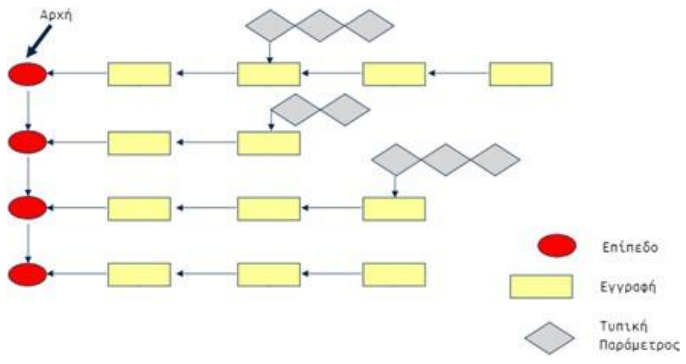
Ο πίνακας συμβόλων αποτελεί ένα από τα βασικότερα εργαλεία κάθε μεταγλωττιστή και χρησιμοποιείται για την αποθήκευση και διαχείριση πληροφοριών σχετικά με τις δηλώσεις και τη χρήση των μεταβλητών, των συναρτήσεων, των διαδικασιών, των παραμέτρων και των προσωρινών μεταβλητών ενός προγράμματος. Η ύπαρξή του είναι απαραίτητη ώστε να πραγματοποιείται σωστή σημασιολογική ανάλυση και να επιτρέπεται η παραγωγή ενδιάμεσου και τελικού κώδικα.

Στον δικό μου κώδικα, ο πίνακας συμβόλων υλοποιείται ως μια λίστα από scopes (πεδία ορατότητας), όπου κάθε scope είναι μια λίστα που περιέχει:

1. Το όνομα του scope (δηλαδή το όνομα του προγράμματος ή της συνάρτησης/διαδικασίας).
2. Μια λίστα από entities (οντότητες) που περιλαμβάνει όλες τις μεταβλητές, παραμέτρους, προσωρινές μεταβλητές, κ.λπ. που δηλώνονται στο συγκεκριμένο scope.
3. Το nesting level (το επίπεδο φωλιάσματος του scope).
4. Το frame length (το μέγεθος που απαιτείται για τις τοπικές και προσωρινές μεταβλητές αυτού του scope στη στοίβα εκτέλεσης).

Κάθε στοιχείο της λίστας αντιπροσωπεύει ένα scope και είναι πίνακας της μορφής: [scopename, [entities], parent_scope_index, offset]

- scopename: Όνομα του scope (π.χ. όνομα συνάρτησης, προγράμματος κλπ)
- [entities]: Λίστα με όλα τα entities (δηλαδή μεταβλητές, functions, temps κτλ) που δηλώθηκαν στο scope αυτό.
- parent_scope_index: Δείκτης στο parent scope
- offset: Μετρητής για τη θέση κάθε μεταβλητής στο activation record.



Σε κάθε βασικό σημείο της υλοποίησης (είσοδος νέου scope, προσθήκη μεταβλητής/συνάρτησης, αναζήτηση οντότητας), ο πίνακας συμβόλων λειτουργεί σαν εργαλείο για τη σημασιολογική ανάλυση, διασφαλίζοντας τα παρακάτω:

- Τη **μοναδικότητα** των δηλώσεων στο κάθε scope,
- Την **έγκυρη ορατότητα** μεταβλητών και συναρτήσεων,
- Την **ορθή αναφορά** σε μεταβλητές/συναρτήσεις που έχουν δηλωθεί,
- Την **αποφυγή διπλών δηλώσεων** ή αναφορών σε αδήλωτες οντότητες.

Με αυτόν τον τρόπο, το πρόγραμμα ελέγχεται σε βάθος για σημασιολογικά λάθη, οδηγώντας σε ένα πιο σωστό και αξιόπιστο τελικό κώδικα.

Προσθήκη νέου Scope: addScope(scopename)

Η συνάρτηση αυτή προσθέτει ένα νέο scope στην αρχή της λίστας scopes. Δηλαδή, κάθε φορά που "μπαίνουμε" σε νέα συνάρτηση/διαδικασία ή το κυρίως πρόγραμμα, δημιουργούμε καινούργιο scope, ώστε να διαχειριζόμαστε ξεχωριστά τις δηλώσεις μεταβλητών και παραμέτρων που ανήκουν MONO σε αυτό το scope. Ο δείκτης parent_scope_index μας βοηθά να μπορούμε να επιστρέψουμε στο προηγούμενο scope όταν τελειώσουμε, ενώ το offset ξεκινάει από 12 (δηλαδή τη θέση της πρώτης μεταβλητής).

```
706
707     def addScope(scopename):  3 usages
708         global scopes
709
710         scopes.insert( _index: 0,  _object: [scopename, [], len(scopes), 12])
711
```

Αφαίρεση Scope: deleteScope()

Όταν τελειώνει το scope μιας συνάρτησης/διαδικασίας ή του κυρίως προγράμματος, αφαιρούμε το αντίστοιχο scope από την αρχή της λίστας scopes. Έτσι, όλες οι εγγραφές που αφορούν το τοπικό scope "εξαφανίζονται", κάτι που αντιστοιχεί με την έξοδο από μια συνάρτηση ή μπλοκ και τη λήξη της ορατότητας των τοπικών μεταβλητών.

```
712     def deleteScope():  3 usages
713         global scopes
714
715         print(scopes)
716         print()
717         del scopes[0]
```

Προσθήκη Entity στον Πίνακα Συμβόλων: addEntity(entity)

Κάθε φορά που δηλώνεται νέα μεταβλητή, παράμετρος, προσωρινή μεταβλητή κλπ, καλείται η addEntity, η οποία προσθέτει την οντότητα στη λίστα entities του τρέχοντος scope (δηλαδή της πρώτης θέσης στη λίστα scopes). Επίσης, αυξάνει το offset κατά 4 για κάθε μεταβλητή (εκτός αν είναι function/procedure), ώστε να υπολογίζεται η θέση που θα πάρει στη μνήμη κάθε entity. Έτσι, κάθε scope "ξέρει" ακριβώς τις τοπικές μεταβλητές του.

```
719     def addEntity(entity):  4 usages
720         global scopes
721
722         scopes[0][1].append(entity)
723         if entity[1] != 'func' and entity[1] != 'proc':
724             scopes[0][3] = scopes[0][3] + 4
725
```

Αναζήτηση Entity στον Πίνακα Συμβόλων: searchEntity(entityname)

Όταν πρέπει να βρούμε αν μία μεταβλητή ή συνάρτηση έχει δηλωθεί (π.χ. όταν χρησιμοποιείται ή γίνεται εκχώρηση τιμής), καλείται αυτή η συνάρτηση. Ψάχνει το entity από το πιο εσωτερικό (τελευταίο) προς το πιο εξωτερικό scope, μέχρι να βρει το entity με το όνομα entityname. Αν δεν το βρει, τυπώνει μήνυμα λάθους και τερματίζει το πρόγραμμα. Έτσι διασφαλίζεται η σωστή ορατότητα και η ύπαρξη των entities.

```
726     def searchEntity(entityname):
727         global scopes
728
729         for i in range(len(scopes)):
730             entities = scopes[i][1]
731             for j in range(len(entities)):
732                 entity = entities[j]
733                 if entity[0] == entityname:
734                     return entity
735
736         print("Line %d: '%s' is not declared"%(lines, entityname))
737         exit(0)
```

Εισαγωγή entities στο scope σε διάφορα σημεία της σύνταξης (π.χ. varlist, funcinput, funcoutput, newTemp)

- Η συνάρτηση varlist είναι υπεύθυνη για την αναγνώριση και εισαγωγή μεταβλητών στον πίνακα συμβόλων κατά τη διάρκεια της ανάλυσης των δηλώσεων. Κάθε φορά που δηλώνεται μια μεταβλητή, γίνεται προσθήκη της μέσω της addEntity, μαζί με τον τύπο της και τη διεύθυνση στο τρέχον scope. Έτσι, ο πίνακας συμβόλων ενημερώνεται ώστε η μεταβλητή να είναι γνωστή από το σημείο αυτό και μετά, και να μην επιτρέπεται διπλή δήλωση στο ίδιο scope. Επίσης, αυτό επιτρέπει τη σημασιολογική ανάλυση να ελέγχει αν μια μεταβλητή

έχει δηλωθεί πριν χρησιμοποιηθεί και να αποφεύγονται σφάλματα όπως χρήση αδήλωτης μεταβλητής..

```
235  def varlist(state, word, vartype):  5 usages
236      inputCheck(state, word,  expectedtk: 'idtk')
237      if vartype != '':
238          addEntity([word, vartype, scopes[0][3]])
239      state, word = lex()
240      while state == 'commatk':
241          state, word = lex()
242          inputCheck(state, word,  expectedtk: 'idtk')
243          state, word = lex()
244      return state, word
```

- Κάθε προσωρινή μεταβλητή που δημιουργείται καταγράφεται με ξεχωριστό όνομα και offset. Η συνάρτηση newTemp δημιουργεί και προσθέτει στον πίνακα συμβόλων μια νέα προσωρινή μεταβλητή κάθε φορά που προκύπτει ενδιάμεσο αποτέλεσμα από κάποια πράξη. Αυτές οι μεταβλητές χρησιμοποιούνται μόνο εντός του scope που δημιουργήθηκαν και δεν είναι ορατές έξω από αυτό. Η σημασιολογική ανάλυση εδώ διασφαλίζει ότι κάθε ενδιάμεσο αποτέλεσμα αποθηκεύεται σε ξεχωριστή θέση και πως δεν μπερδεύεται με κανονικές μεταβλητές του χρήστη, αποτρέποντας έτσι απρόβλεπτα σφάλματα στην παραγωγή και χρήση προσωρινών τιμών.

```
676  def newTemp():  4 usages
677      global temps, scopes
678
679      temps = temps + 1
680      t = 't@' + str(temps)
681      addEntity([t, 'temp', scopes[0][3]])
682      return t
```

- Οι συναρτήσεις funcinput και funcoutput διαχειρίζονται τα ορίσματα εισόδου και εξόδου αντίστοιχα, μιας συνάρτησης ή διαδικασίας. Χρησιμοποιούν εσωτερικά τη varlist, αλλά περνούν και τον κατάλληλο τύπο ορίσματος ('CV' για μεταβίβαση με τιμή, 'REF' για μεταβίβαση με αναφορά). Μέσα από τον πίνακα συμβόλων, αποθηκεύονται οι παράμετροι με τον σωστό τρόπο μεταβίβασης, ώστε στη συνέχεια να γίνεται σημασιολογικός έλεγχος για τη χρήση τους (π.χ. να μην αλλάζει μια τιμή που έχει δοθεί ως είσοδος με τιμή). Έτσι διασφαλίζεται ότι οι παράμετροι θα χρησιμοποιηθούν σωστά στη συνάρτηση και αποφεύγονται σημαντικά σημασιολογικά λάθη.

```
341  def funcinput(state, word):  2 usages
342      if state == 'εισοδοςtk':
343          state, word = lex()
344          state, word = varlist(state, word,  vartype: 'CV')
345      return state, word
346
347  def funcoutput(state, word):  2 usages
348      if state == 'έξοδοςtk':
349          state, word = lex()
350          state, word = varlist(state, word,  vartype: 'REF')
351      return state, word
```

Διαχείριση scopes κατά την είσοδο/έξοδο υποπρογραμμάτων (π.χ. funcblock, procblock)

Οι funcblock και procblock υλοποιούν τα scopes (πεδία ορατότητας) που δημιουργούνται κάθε φορά που δηλώνεται μια νέα συνάρτηση ή διαδικασία. Καταχωρούν το όνομα της συνάρτησης/διαδικασίας ως νέα οντότητα και δημιουργούν νέο scope, στο οποίο θα δηλωθούν τα ορίσματα και οι τοπικές μεταβλητές της συνάρτησης. Με αυτόν τον τρόπο ο πίνακας συμβόλων υποστηρίζει το σημασιολογικό έλεγχο για το ποιες μεταβλητές είναι ορατές μέσα σε κάθε συνάρτηση ή διαδικασία και αποφεύγονται συγκρούσεις ή λάθη που σχετίζονται με τη χρήση ονομάτων (name resolution). Όταν ολοκληρωθεί η λειτουργία της συνάρτησης/διαδικασίας, το scope διαγράφεται με τη deleteScope, εξασφαλίζοντας ότι οι τοπικές οντότητες δεν είναι πλέον προσβάσιμες εκτός του ορισμού τους.

```
283     def funcblock(state, word, funcname):  1 usage
284         global scopes
285
286         addEntity([funcname, 'proc', -1, -1])
287         addScope(funcname)
288         inputCheck(state, word,  expectedtk: 'διαπροσωπείατk')
289         state, word = lex()
290
291         state, word = funcinput(state, word)
292         state, word = funcoutput(state, word)
293
294         state, word = declarations(state, word);
295         state, word = subprograms(state, word);
296
297         scopes[1][1][len(scopes[1][1]) - 1][2] = nextquad()
298         genquad( op: 'begin_block', funcname,  y: '_',  z: '_')
299
300         inputCheck(state, word,  expectedtk: 'αρχή_συνάρτησηςτk')
301         state, word = lex()
302
303         state, word = sequence(state, word)
304         scopes[1][1][len(scopes[1][1]) - 1][3] = scopes[0][3]
305
306         deleteScope()
307         inputCheck(state, word,  expectedtk: 'τέλος_συνάρτησηςτk')
308
309         genquad( op: 'end_block', funcname,  y: '_',  z: '_')
310
311         state, word = lex()
312         return state, word
314     def procblock(state, word, procname):  1 usage
315
316         addEntity([procname, "proc", -1, -1])
317         addScope(procname)
318         inputCheck(state, word,  expectedtk: 'διαπροσωπείατk')
319         state, word = lex()
320
321         state, word = funcinput(state, word)
322         state, word = funcoutput(state, word)
323
324         state, word = declarations(state, word);
325         state, word = subprograms(state, word);
326         scopes[1][1][len(scopes[1][1]) - 1][2] = nextquad()
327         genquad( op: 'begin_block', procname,  y: '_',  z: '_')
328
329         inputCheck(state, word,  expectedtk: 'αρχή_διαδικασίαςτk")
330         state, word = lex()
331
332         state, word = sequence(state, word)
333         scopes[1][1][len(scopes[1][1]) - 1][3] = scopes[0][3]
334         deleteScope()
335         inputCheck(state, word,  expectedtk: 'τέλος_διαδικασίαςτk')
336         genquad( op: 'end_block', procname,  y: '_',  z: '_')
337
338         state, word = lex()
339         return state, word
```

6) Παραγωγή τελικού κώδικα

Η παραγωγή τελικού κώδικα γίνεται μέσα στη συνάρτηση deleteScope(), η οποία καλείται κάθε φορά που τελειώνει ένα scope (δηλαδή όταν τελειώνει ένα block της main, μιας συνάρτησης ή μιας διαδικασίας). Εκεί, όλες οι τετράδες που δημιουργήθηκαν σε αυτό το scope διατρέχονται και μεταφράζονται σε αντίστοιχες εντολές RISC-V, οι οποίες γράφονται στο αρχείο εξόδου Σημαντικό ρόλο παίζουν οι βοηθητικές συναρτήσεις gnlvcode, loadvr και storerv που εξειδικεύουν τη μετάφραση όσον αφορά τις μεταβλητές, τις παραμέτρους και τα scopes.

Η φάση της παραγωγής του τελικού κώδικα είναι η τελευταία φάση ενός μεταγλωττιστή και ακολουθεί τη φάση παραγωγής του ενδιάμεσου κώδικα (τετράδες). Σε αυτή τη φάση, ο μεταγλωττιστής μεταφράζει τις τετράδες σε οδηγίες σε χαμηλό επίπεδο, συμβατές με τον επεξεργαστή RISC-V. Ο παραγόμενος κώδικας αποθηκεύεται σε αρχείο με κατάληξη .asm και υλοποιεί όλες τις λειτουργίες του αρχικού προγράμματος, λαμβάνοντας υπόψιν τη διαχείριση της μνήμης (στοίβα), την απεικόνιση μεταβλητών, το πέρασμα παραμέτρων και τις κλήσεις υποπρογραμμάτων.

Ας δούμε μία προς μία τις σημαντικές συναρτήσεις που υλοποιούν τη χαρτογράφηση των τετραδών σε τελικό κώδικα.

- **gnlvcode(offset, dif)**
  Η συνάρτηση αυτή χρησιμοποιείται όταν θέλουμε να προσπελάσουμε μια μη τοπική μεταβλητή (δηλαδή μια μεταβλητή που δεν βρίσκεται στο τρέχον scope, αλλά σε κάποιον πρόγονο/πατέρα στο stack των scopes). Για παράδειγμα, μια διαδικασία μπορεί να έχει πρόσβαση σε μεταβλητές του περιβάλλοντός της.

```
869  def gnlvcode(offset, dif):  2 usages
870      outfile2.write('lw t0, -4(sp)\n')
871      for i in range(dif):
872          outfile2.write('lw t0, -4(t0)\n')
873
874      outfile2.write('addi t0, t0, -%d\n'%(offset))
875
```

1. outfile2.write('lw t0, -4(sp)\n'): Ξεκινάμε φορτώνοντας στον καταχωρητή t0 τη διεύθυνση του activation record του «πατέρα» (το -4(sp) κρατά τον static link κάθε activation record).
2. for i in range(dif): outfile2.write('lw t0, -4(t0)\n'): Αν το επίπεδο φωλιάσματος της ζητούμενης μεταβλητής απέχει περισσότερα επίπεδα, ακολουθούμε τα static links με επαναληπτικά lw t0, -4(t0) για να φτάσουμε στον σωστό πρόγονο.
3. outfile2.write('addi t0, t0, -%d\n'%(offset)): Τέλος, προσθέτουμε το offset της μεταβλητής (όπως το ξέρουμε από τον πίνακα συμβόλων), ώστε ο t0 να δείχνει ακριβώς στη μεταβλητή μας.

Αυτός ο μηχανισμός βασίζεται στην αρχή των activation records και static links που κρατά κάθε block, επιτρέποντας προσπέλαση σε μη τοπικές μεταβλητές.

- **loadvr(v,** r)
  Η συνάρτηση αυτή αναλαμβάνει να φορτώσει τη σωστή τιμή μιας μεταβλητής ή σταθεράς v στον καταχωρητή t<r>. Είναι ζωτικής σημασίας για να μεταφράσουμε τις τετράδες σε RISC-V, διότι σε κάθε πράξη πρέπει να έχουμε την τιμή σε κάποιο register.

```
def loadvr(v,r):  8 usages
    global scopes

    if v[0] in '0123456789-+':
        outfile2.write('li t%d, %s\n'%(r, v))
    else:
        entity, nestingLevel = searchEntity(v)
        if entity[1] == 'func' or entity[1] == 'proc':
            print("Line %d: '%s' cannot be used as variable"%(lines, entity[0]))
            exit(0)

        if nestingLevel == 0:
            outfile2.write('lw t%d, -%d(gp)\n'%(r, entity[2]))
        elif nestingLevel == len(scopes) - 1:
            if entity[1] != 'REF':
                outfile2.write('lw t%d, -%d(sp)\n'%(r, entity[2]))
            else:
                outfile2.write('lw t0, -%d(gp)\n'%(entity[2]))
                outfile2.write('lw t%d, (t0)\n'%(r))
        else:
            dif = (len(scopes) - 1) - nestingLevel - 1
            gnlvcode(entity[2], dif)
            if entity[1] != 'REF':
                outfile2.write('lw t%d, (t0)\n'%(r))
            else:
                outfile2.write('lw t0, (t0)\n')
                outfile2.write('lw t%d, (t0)\n'%(r))
```

1. Εάν το v είναι σταθερά, απλά κάνουμε li t<r>, v (load immediate).
2. Αλλιώς, ψάχνουμε το entity της μεταβλητής και το επίπεδο φωλιάσματος (με searchEntity).
3. Εάν το nestingLevel είναι 0, η μεταβλητή είναι **global** και προσπελαύνεται μέσω του gp (global pointer).
4. Εάν είναι τοπική (ίδιο scope με το τρέχον), προσπελαύνεται μέσω του sp. Αν είναι με αναφορά (REF), χρειάζεται ακόμα ένα επίπεδο indirection.
5. Εάν η μεταβλητή είναι μη τοπική (κάπου στο parent scope), χρησιμοποιούμε τη gnlvcode για να εντοπίσουμε σωστά τη θέση της στη στοίβα και φορτώνουμε με lw t<r>, (t0).

Ακολουθείται πιστά η λογική του περιβάλλοντος εκτέλεσης (activation records, static links, stack). Η συνάρτηση αυτή εγγυάται ότι, ανεξαρτήτως του που βρίσκεται η μεταβλητή, η τιμή της θα φτάσει σωστά στον κατάλληλο register.

- **storerv(r, v)**
  Η συνάρτηση `storerv` αποθηκεύει την τιμή του καταχωρητή t<r> στη μεταβλητή v. Είναι το "αντίστροφο" της loadvr.

```
def storerv(r,v):  3 usages
    entity, nestingLevel = searchEntity(v)
    if entity[1] == 'func' or entity[1] == 'proc':
        print("Line %d: '%s' cannot be used as variable"%(lines, entity[0]))
        exit(0)
    if nestingLevel == 0:
        outfile2.write('sw t%d, -%d(gp)\n'%(r, entity[2]))
    elif nestingLevel == len(scopes) - 1:
        if entity[1] != 'REF':
            outfile2.write('sw t%d, -%d(sp)\n'%(r, entity[2]))
        else:
            outfile2.write('lw t0, -%d(gp)\n'%(entity[2]))
            outfile2.write('sw t%d, (t0)\n'%(r))
    else:
        dif = (len(scopes) - 1) - nestingLevel - 1
        gnlvcode(entity[2], dif)
        if entity[1] != 'REF':
            outfile2.write('lw t%d, (t0)\n'%(r))
        else:
            outfile2.write('sw t0, (t0)\n')
            outfile2.write('sw t%d, (t0)\n'%(r))
```

1. Βρίσκει τη θέση της μεταβλητής (στον πίνακα συμβόλων και στη στοίβα).
2. Αν είναι global: αποθηκεύει μέσω gp.
3. Αν είναι τοπική: αποθηκεύει μέσω sp.
4. Αν είναι μη τοπική: χρησιμοποιεί πάλι τη gnlvcode και αποθηκεύει μέσω t0.
5. Αν είναι με αναφορά, χρησιμοποιεί έναν επιπλέον δείκτη.

Αντίστοιχα με τη loadvr, ακολουθεί τη λογική του activation record και τη σωστή αποτύπωση του πού βρίσκεται κάθε μεταβλητή στη μνήμη.

Στο τέλος κάθε scope η deleteScope περνάει από όλες τις τετράδες που ανήκουν σε αυτό και για κάθε μια παράγει τις αντίστοιχες εντολές RISK-V.

Κύρια βήματα που ακολουθούνται στη deleteScope():

1. **Αναγνώριση των τετράδων που ανήκουν στο scope:**
   Μεταβλητή quadsPart παίρνει τις τετράδες που ανήκουν στο scope που ολοκληρώνεται.
2. **Για κάθε τετράδα:**
   Ελέγχεται ο τύπος της εντολής (π.χ. :=, +, jump, κλπ) και για κάθε τύπο, παράγονται οι αντίστοιχες εντολές σε assembly.
   Εδώ γίνονται οι κλήσεις σε βοηθητικές συναρτήσεις, όπως loadvr και storerv, για τη σωστή μεταφορά δεδομένων μεταξύ καταχωρητών και μνήμης, ανάλογα με το scope της κάθε μεταβλητής.
3. **Διαχείριση stack και επιστροφής:**
   Αν το scope είναι υποπρόγραμμα, τότε διαχειρίζεται καταχωρητές επιστροφής (ra), static links, κλπ.
   Αν το scope είναι το main πρόγραμμα, τότε παράγεται η ετικέτα Lmain.
4. **Διαγραφή scope από τον πίνακα scopes.**

Εκχώρηση (:=)

```
elif q[1] == ':=':
    loadvr(q[2], r 1)
    storerv( r 1, q[4])
```

- **Τι σημαίνει:** Μεταφέρει την τιμή του δεξιού μέρους (q[2]) στο αριστερό (q[4]).
- **loadvr:** Φέρνει τη σωστή τιμή (είτε από καταχωρητή, είτε από μνήμη, είτε από άλλο scope) στον t1.
- **storerv:** Αποθηκεύει την τιμή στον προορισμό (πάλι, είτε τοπικά, είτε σε static link, ανάλογα με το scope).

Αριθμητικές Πράξεις (+, -,* , /)

```
elif q[1] == '+' or q[1] == '-' or q[1] == '*' or q[1] == '/':
    loadvr(q[2], r 1)
    loadvr(q[3], r 2)
    if q[1] == '+':
        outfile2.write('add t1, t1, t2\n')
    elif q[1] == '-':
        outfile2.write('sub t1, t1, t2\n')
    elif q[1] == '*':
        outfile2.write('mul t1, t1, t2\n')
    elif q[1] == '/':
        outfile2.write('div t1, t1, t2\n')
    storerv( r 1, q[4])
```

- **Φόρτωση των δύο τελεστέων** σε t1, t2.
- **Εκτέλεση πράξης** στον t1 (π.χ. add t1, t1, t2).
- **Αποθήκευση αποτελέσματος** με storerv.

Άλματα και συγκρίσεις (jump, >, <, >=, <=, =, <>)

```
elif q[1] == 'jump':
    outfile2.write('j L%s\n'%(q[4]))
elif q[1] == '>' or q[1] == '<' or q[1] == '>=' or q[1] == '>=' or q[1] == '<>' or q[1] == '=':
    loadvr(q[2], r 1)
    loadvr(q[3], r 2)
    if q[1] == '>':
        outfile2.write('bgt t1, t2, L%s\n'%(q[4]))
    elif q[1] == '<':
        outfile2.write('blt t1, t2, L%s\n'%(q[4]))
    elif q[1] == '>=':
        outfile2.write('bge t1, t2, L%s\n'%(q[4]))
    elif q[1] == '<=':
        outfile2.write('bge t1, t2, L%s\n'%(q[4]))
    elif q[1] == '=':
        outfile2.write('beq t1, t2, L%s\n'%(q[4]))
    elif q[1] == '<>':
        outfile2.write('bne t1, t2, L%s\n'%(q[4]))
```

- **Jump:** Απλό άλμα στη διεύθυνση της ετικέτας.
- **Συγκρίσεις:** Φορτώνονται τα ορίσματα, συγκρίνονται, και αν η συνθήκη ισχύει γίνεται άλμα.

Εντολές εΙσόδου/εξόδου (inp, out)

```
747         if q[1] == 'out':
748             loadvr( v: 1, q[2])
749             outfile2.write('mv a0, t1\n')
750             outfile2.write('li a7, 1\n')
751             outfile2.write('ecall\n')
752 
753             outfile2.write('la a0, str_nl\n')
754             outfile2.write('li a7, 4\n')
755             outfile2.write('ecall\n')
756         elif q[1] == 'inp':
757             outfile2.write('li a7, 5\n')
758             outfile2.write('ecall\n')
759             outfile2.write('mv t1, a0\n')
760             storerv( r: 1, q[2])
```

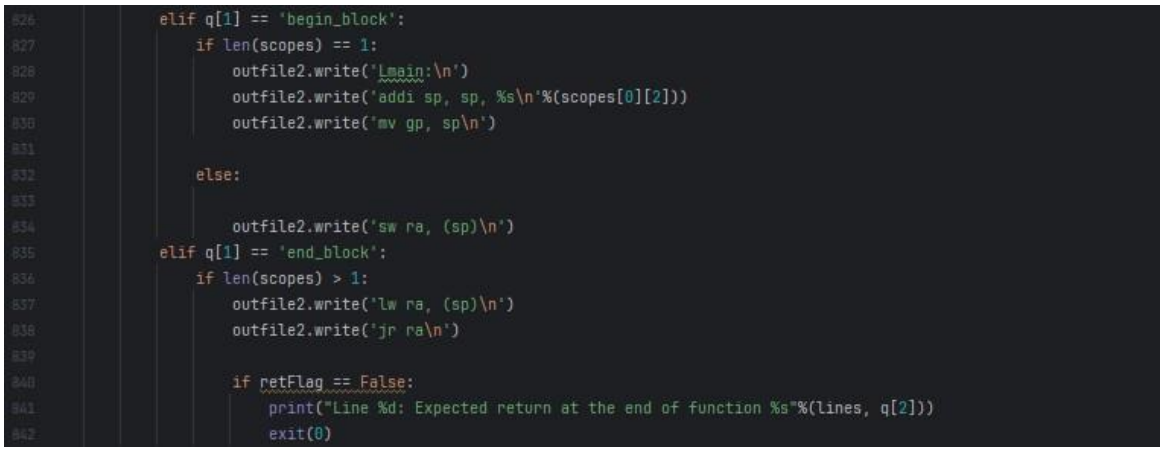
- **out:** Φορτώνεται η τιμή για εκτύπωση στον t1, μεταφέρεται στον a0 και γίνεται ecall για print.
- **inp:** Παίρνουμε τιμή με ecall και την αποθηκεύουμε στη μεταβλητή.

Κλήση υποπρογράμματος (call, par, retv)

```
793         elif q[1] == 'retv':
794             loadvr(q[2],  r: 1)
795             outfile2.write('lw t0, -8(sp)\n')
796             outfile2.write('sw t1, (t0)\n')
797         elif q[1] == 'par':
798             if i == 0:
799                 k = 0
800                 for q1 in quadsPart:
801                     if q1[0] > q[0] and q1[1] == 'call':
802                         entity, nestingLevel = searchEntity(q1[2])
803                         if entity[1] != 'func' and entity[1] != 'proc':
804                             print("Line %d: '%s' cannot be used as func/proc"%(lines, entity[0]))
805                             exit(0)
806                         outfile2.write('addi fp, sp, %d\n'%(entity[3]))
807                         break
816         elif q[1] == 'call':
817             i = 0
818             entity, nestingLevel = searchEntity(q[2])
819             if entity[1] != 'func' and entity[1] != 'proc':
820                 print("Line %d: '%s' cannot be used as func/proc"%(lines, entity[0]))
821                 exit(0)
822             outfile2.write('sw sp, -4(fp)\n')
823             outfile2.write('addi sp, sp, %d\n'%(entity[3]))
824             outfile2.write('jal L%d\n'%(entity[2]))
825             outfile2.write('addi sp, sp, -%d\n'%(entity[3]))
```

- **par:** Προετοιμασία παραμέτρων στο stack, ανάλογα με το είδος τους (CV, REF, RET).
- **call:** Ενημέρωση static link, αλλαγή sp, εκκίνηση υποπρογράμματος, επαναφορά sp.
- **retv:** Επιστροφή τιμής μέσω του κατάλληλου activation record.

Έναρξη καΙ τέλος block (begin_block, end_block)



```
        elif q[1] == 'begin_block':
            if len(scopes) == 1:
                outfile2.write('Lmain:\n')
                outfile2.write('addi sp, sp, %s\n'%(scopes[0][2]))
                outfile2.write('mv gp, sp\n')

            else:

                outfile2.write('sw ra, (sp)\n')
        elif q[1] == 'end_block':
            if len(scopes) > 1:
                outfile2.write('lw ra, (sp)\n')
                outfile2.write('jr ra\n')

                if retFlag == False:
                    print("Line %d: Expected return at the end of function %s"%(lines, q[2]))
                    exit(0)
```

- **begin_block:** Στο main παράγει την ετικέτα Lmain, διαχειρίζεται sp και gp.
- **end_block:** Στα υποπρογράμματα, φροντίζει για την επαναφορά ra και τη λήξη της συνάρτησης/διαδικασίας.

Τελικό βήμα

Στο τέλος, το scope διαγράφεται από τον πίνακα scopes με del scopes[0], ώστε να "καθαρίσει" η μνήμη και να προχωρήσει στο επόμενο scope.

6)Testing

Το test αρχείο που χρησιμοποιούμε είναι το εξής:

```
πρόγραμμα τεστ
    δήλωση α,β
    δήλωση γ

    συνάρτηση αύξηση(α,β)
        διαπροσωπεία
        είσοδος α
        έξοδος β

        συνάρτηση αύξηση2(α,β)
            διαπροσωπεία
            είσοδος α
            έξοδος β

                συνάρτηση αύξηση3(α,β)
                    διαπροσωπεία
                    είσοδος α
                    έξοδος β

                    αρχή_συνάρτησης
                        β := α + 1 ;
                        αύξηση3 := α + 1 { δεν μπαίνει ερωτηματικό
                                           είναι τέλος block }
                    τέλος_συνάρτησης

            αρχή_συνάρτησης{2}
                β := α + 1 ;
                αύξηση2 := α + 1 { δεν μπαίνει ερωτηματικό
                                   είναι τέλος block }
            τέλος_συνάρτησης

        διαδικασία τύπωσε_συν_1(x)
            διαπροσωπεία
            είσοδος x
            αρχή_διαδικασίας
                γράψε x+1

        τέλος_διαδικασίας
        αρχή_συνάρτησης {1}
            β := α + 1 ;
            αύξηση := α + 1 { δεν μπαίνει ερωτηματικό
                              είναι τέλος block }
        τέλος_συνάρτησης

    διαδικασία τύπωσε_συν_1(x)
        διαπροσωπεία
        είσοδος x

        συνάρτηση αύξηση3(α,β)
            διαπροσωπεία
            είσοδος α
            έξοδος β

            αρχή_συνάρτησης
                β := α + 1 ;
                αύξηση3 := α + 1 { δεν μπαίνει ερωτηματικό
                                   είναι τέλος block }
            τέλος_συνάρτησης

        διαδικασία τύπωσε_συν_1(x)
            διαπροσωπεία
            είσοδος x
            αρχή_διαδικασίας
                γράψε x+1
            τέλος_διαδικασίας

        αρχή_διαδικασίας
            γράψε x+1
        τέλος_διαδικασίας
    αρχή_προγράμματος
        α := 1 ;
        β := 2 + α * α / (2 - α - (2*α));
        γ := αύξηση(α,%β);
```

```
74
75          για α:=1 έως 8 με_βήμα -2 επανάλαβε
76              εκτέλεσε τύπωσε_συν_1(α)
77          για_τέλος;
78
79          β := 1;
80          εάν β<>22 ή όχι [β>=23 ή β<=24] τότε
81              β := β+1
82              αλλιώς
83                  β:=β-1
84              εάν_τέλος ;
85
86          β := 1;
87          εάν β<>22 ή όχι [β>=23 ή β<=24] τότε
88              β := β+1
89              εάν_τέλος ;{ όχι ερωτηματικό, είναι τέλος block }
90          όσο β<10 επανάλαβε
91              εάν β<>22 ή [β>=23 και β<=24] τότε
92              β := β+1
93              εάν_τέλος { όχι ερωτηματικό, είναι τέλος block }
94          όσο_τέλος; { θέλει ερωτηματικό
95                       χωρίζει εκτελέσιμες εντολές }
96          διάβασε β;
97          επανάλαβε
98              β := β + 1
99          μέχρι β<-100
100     τέλος_προγράμματος
101
```

Ενδιάμεσος Κώδικας:

```
1   β : begin_block , αύξηση3 , _ , _
2   1 : + , α , 1 , t@1
3   2 : := , t@1 , _ , β
4   3 : + , α , 1 , t@2
5   4 : retv , t@2 , _ , _
6   5 : end_block , αύξηση3 , _ , _
7   6 : begin_block , αύξηση2 , _ , _
8   7 : + , α , 1 , t@3
9   8 : := , t@3 , _ , β
10  9 : + , α , 1 , t@4
11  10 : retv , t@4 , _ , _
12  11 : end_block , αύξηση2 , _ , _
13  12 : begin_block , τύπωσε_συν_1 , _ , _
14  13 : + , x , 1 , t@5
15  14 : end_block , τύπωσε_συν_1 , _ , _
16  15 : begin_block , αύξηση , _ , _
17  16 : + , α , 1 , t@6
18  17 : := , t@6 , _ , β
19  18 : + , α , 1 , t@7
20  19 : retv , t@7 , _ , _
21  20 : end_block , αύξηση , _ , _
22  21 : begin_block , αύξηση3 , _ , _
23  22 : + , α , 1 , t@8
24  23 : := , t@8 , _ , β
25  24 : + , α , 1 , t@9
26  25 : retv , t@9 , _ , _
27  26 : end_block , αύξηση3 , _ , _
28  27 : begin_block , τύπωσε_συν_1 , _ , _
29  28 : + , x , 1 , t@10
30  29 : end_block , τύπωσε_συν_1 , _ , _
31  30 : begin_block , τύπωσε_συν_1 , _ , _
32  31 : + , x , 1 , t@11
33  32 : end_block , τύπωσε_συν_1 , _ , _
34  33 : begin_block , τεστ , _ , _
35  34 : := , 1 , _ , α
36  35 : * , α , α , t@12
37  36 : - , 2 , α , t@13
38  37 : * , 2 , α , t@14
39  38 : - , t@13 , t@14 , t@15
40  39 : / , t@12 , t@15 , t@16
41  40 : + , 2 , t@16 , t@17
42  41 : := , t@17 , _ , β
43  42 : par , α , CV , _
44  43 : par , β , REF , _
45  44 : par , t@18 , RET , _
46  45 : call , αύξηση , _ , _
47  46 : := , t@18 , _ , γ
48  47 : := , 1 , _ , α
49  48 : jump , _ , _ , 50
50  49 : + , α , -2 , α
51  50 : < , -2 , 0 , 53
52  51 : > , -2 , 0 , 55
53  52 : jump , _ , _ , 57
54  53 : >= , α , 8 , 57
55  54 : jump , _ , _ , 60
56  55 : <= , α , 8 , 57
57  56 : jump , _ , _ , 60
58  57 : par , α , CV , _
59  58 : call , τύπωσε_συν_1 , _ , _
60  59 : jump , _ , _ , 49
61  60 : := , 1 , _ , β
62  61 : <> , β , 22 , 67
63  62 : jump , _ , _ , 63
64  63 : >= , β , 23 , 70
65  64 : jump , _ , _ , 65
66  65 : <= , β , 24 , 70
67  66 : jump , _ , _ , 67
68  67 : + , β , 1 , t@19
69  68 : := , t@19 , _ , β
70  69 : jump , _ , _ , 72
71  70 : - , β , 1 , t@20
72  71 : := , t@20 , _ , β
73  72 : := , 1 , _ , β
74  73 : <> , β , 22 , 79
75  74 : jump , _ , _ , 75
76  75 : >= , β , 23 , 82
77  76 : jump , _ , _ , 77
78  77 : <= , β , 24 , 82
79  78 : jump , _ , _ , 79
80  79 : + , β , 1 , t@21
81  80 : := , t@21 , _ , β
82  81 : jump , _ , _ , 82
83  82 : < , β , 10 , 84
84  83 : jump , _ , _ , 94
85  84 : <> , β , 22 , 90
86  85 : jump , _ , _ , 86
87  86 : >= , β , 23 , 88
88  87 : jump , _ , _ , 93
89  88 : <= , β , 24 , 90
90  89 : jump , _ , _ , 93
91  90 : + , β , 1 , t@22
92  91 : := , t@22 , _ , β
93  92 : jump , _ , _ , 93
94  93 : jump , _ , _ , 82
95  94 : inp , β , _ , _
96  95 : + , β , 1 , t@23
97  96 : := , t@23 , _ , β
98  97 : < , β , -100 , 95
99  98 : jump , _ , _ , 99
100 99 : halt , _ , _ , _
101 100 : end_block , τεστ , _ , _
102
```

Πίνακας Συμβόλων:

Τελικός Κώδικας (μέχρι γραμμή 35):

```
1   .data
2   str_nl: .asciz "\n"
3   .text
4   j Lmain
5   # 0 : begin_block , αύξηση3 , _ , _
6   sw ra, (sp)
7   # 1 : + , α , 1 , t@1
8   lw t1, -12(sp)
9   li t2, 1
10  add t1, t1, t2
11  sw t1, -20(sp)
12  # 2 : := , t@1 , _ , β
13  lw t1, -20(sp)
14  lw t0, -16(gp)
15  sw t1, (t0)
16  # 3 : + , α , 1 , t@2
17  lw t1, -12(sp)
18  li t2, 1
19  add t1, t1, t2
20  sw t1, -24(sp)
21  # 4 : retv , t@2 , _ , _
22  lw t1, -24(sp)
23  lw t0, -8(sp)
24  sw t1, (t0)
25  # 5 : end_block , αύξηση3 , _ , _
26  lw ra, (sp)
27  jr ra
28  # 6 : begin_block , αύξηση2 , _ , _
29  sw ra, (sp)
30  # 7 : + , α , 1 , t@3
31  lw t1, -12(sp)
32  li t2, 1
33  add t1, t1, t2
34  sw t1, -20(sp)
35  # 8 : := , t@3 , _ , β
```