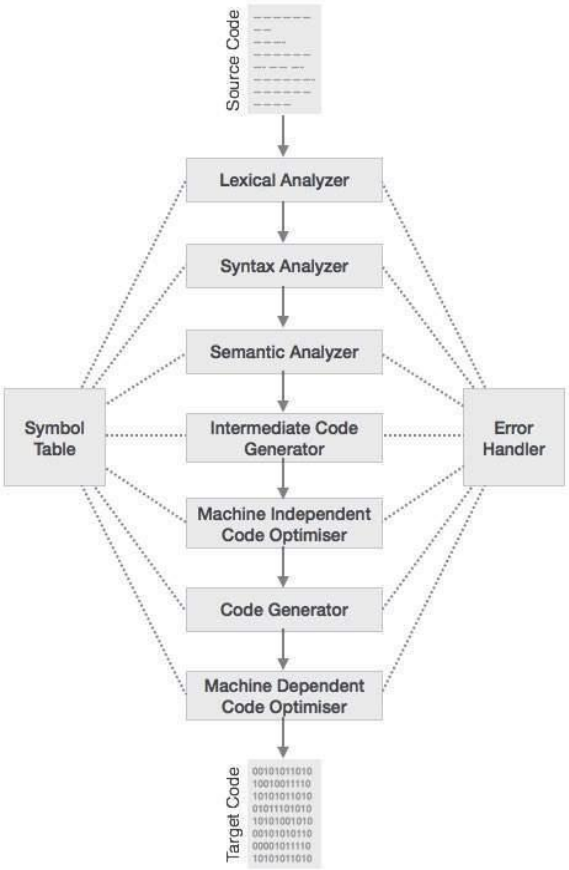
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΥΥ802 - ΜΕΤΑΦΡΑΣΤΕΣ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ



ΜΕΛΗ ΟΜΑΔΑΣ
ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΝΤΩΝΙΟΥ-ΧΑΤΖΗΓΙΩΣΗΣ - 4769
ΜΙΧΟΥ ΝΑΤΑΛΙΑ - 4922

ΙΩΑΝΝΙΝΑ - ΜΑΙΟΣ 2024

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

Η παρούσα εργασία έχει ως στόχο την δημιουργία ενός πλήρους μεταφραστή σε RISC-V assembly για την εκπαιδευτική γλώσσα cpy. Ο μεταφραστής, γραμμένος σε γλώσσα Python, πρέπει τελικά να μπορεί να πάρει ως είσοδο ένα πρόγραμμα σε cpy και να δημιουργήσει στο τέλος ένα αρχείο με κώδικα σε RISC-V assembly, καθώς και για όλα τα ενδιάμεσα στάδια, το οποίο μπορεί να τρέξει κανονικά σε προσομοιωτή και να δώσει τα σωστά αποτελέσματα.

Η σχεδίαση και υλοποίηση του μεταφραστή έγινε σε στάδια όπως μας ζητήθηκε. Συγκεκριμένα τα στάδια αυτά είναι:

1. Σχεδίαση και υλοποίηση Λεκτικού Αναλυτή (Lexer)
2. Σχεδίαση και υλοποίηση Συντακτικού Αναλυτή (Parser)
3. Παραγωγή Ενδιάμεσου Κώδικα (Intermediate Code)
4. Παραγωγή Πίνακα Συμβόλων (Symbol Table)
5. Παραγωγή Τελικού Κώδικα (Final Code)

Στα πλαίσια της εκπαιδευτικής αυτής εργασίας δεν μας ζητήθηκε βελτιστοποίηση του παραγόμενου κώδικα σε κανένα από τα δύο στάδια οπού αυτό θα ήταν δυνατό (στάδια 3 και 5)

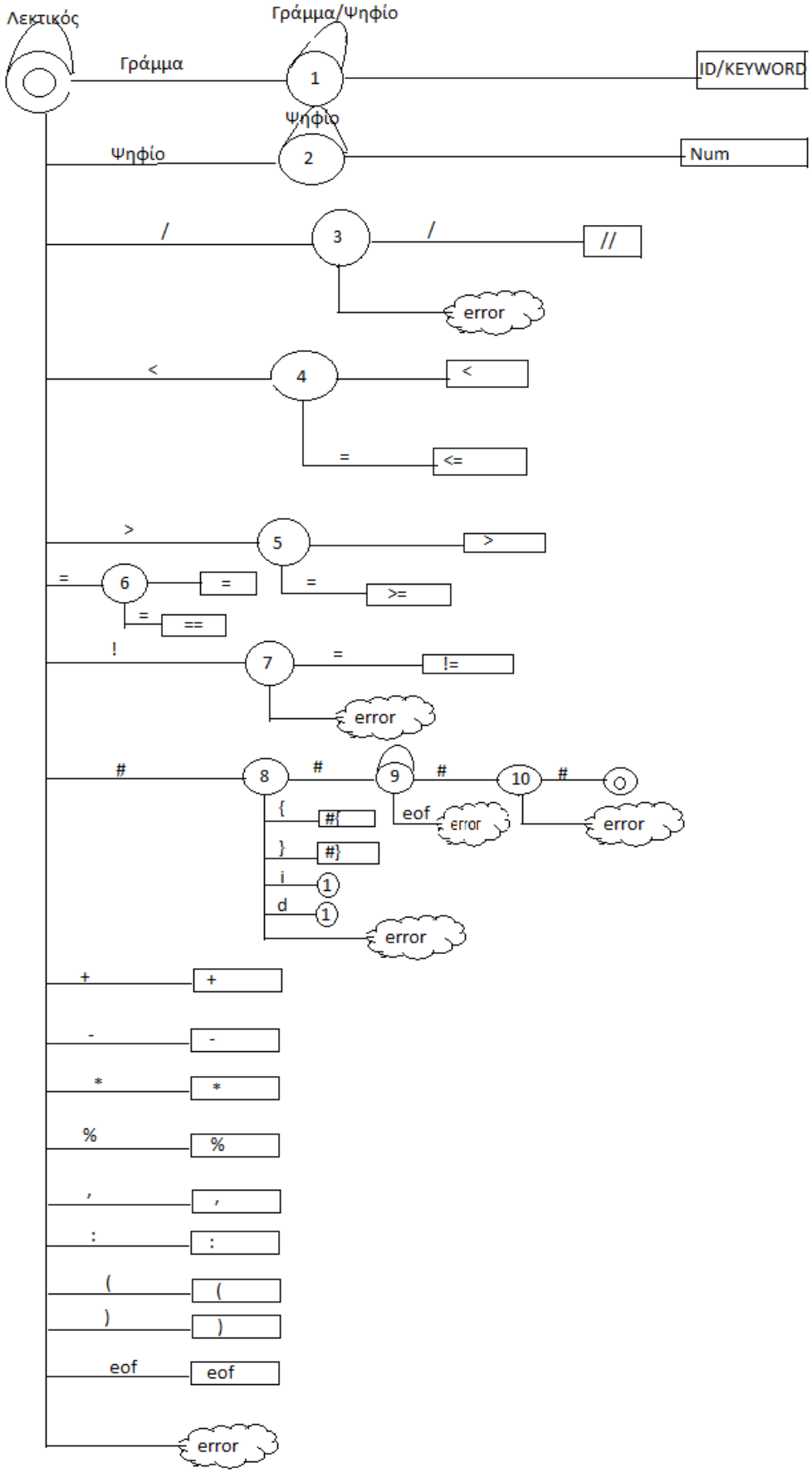
2. ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

2.1 Γενική Ιδέα

Στο κεφάλαιο αυτό παρουσιάζουμε τον Λεκτικό Αναλυτή που δημιουργήσαμε. Ο σκοπός του είναι να αναλύει τη γραμματική ενός αρχείου πηγαίου κώδικα και να παράγει μια σειρά από tokens, που αντιπροσωπεύουν τα αναγνωρισμένα σύμβολα της γλώσσας προγραμματισμού cpy. Αυτός ο λεκτικός αναλυτής είναι το πρώτο βήμα στη διαδικασία της μετάφρασης από υψηλού επιπέδου κώδικα σε εντολές που μπορεί να εκτελέσει ένας υπολογιστής.

Ο λεκτικός αναλυτής που έχουμε δημιουργήσει λειτουργεί μέσω ενός finite state machine (FSM), και επεξεργάζεται τον πηγαίο κώδικα χαρακτήρα προς χαρακτήρα. Ανάλογα με τον χαρακτήρα που διαβάζεται και την τρέχουσα κατάσταση του finite state machine, ο αναλυτής καθορίζει τον τύπο του token, καθώς και τη μετάβαση σε μια νέα κατάσταση. Τα αναγνωρισμένα tokens αποθηκεύονται σε μια λίστα και καταγράφονται στο αρχείο εξόδου. Αν βρεθεί κάποιο λάθος, καταγράφεται στο αρχείο εξόδου και ο αναλυτής συνεχίζει.
Αυτή η διαδικασία γίνεται μέσω ενός βρόχου που διατρέχει κάθε χαρακτήρα του αρχείου.

Ο σχεδιασμός του αυτομάτου που καθόρισε το FMS έγινε με βάση τις πληροφορίες που μας δόθηκαν στο μάθημα καθώς και τις διαφάνειες και το ηλεκτρονικό σύγγραμμα. Το σχεδιάγραμμα που δημιουργήσαμε για το αυτόματο μας παρατίθεται παρακάτω.



Εικόνα 1. Αυτόματο Κατάστασης

2.2 Περιγραφή των κλάσεων

Token
Για να αναπαραστήσουμε τα tokens, χρησιμοποιούμε την κλάση "Token", η οποία είναι αρκετά απλή. Περιλαμβάνει δύο πεδία:

- "Token": Το σύμβολο ή η λέξη που αναγνωρίζεται
- "tokenType": ο τύπος του token (π.χ. λέξη-κλειδί, αριθμός, identifier κλπ.)

LexicalAnalyzer
Η κλάση "LexicalAnalyzer" είναι υπεύθυνη για την ανάλυση του αρχείου πηγαίου κώδικα και την παραγωγή των tokens. Η κλάση αυτή παίρνει μέσω argument το όνομα του αρχείου που θα αναλυθεί, ορίζει τις καταστάσεις του αυτόματου πεπερασμένου καταστάσεων (finite state machine) που χρησιμοποιείται για την ανάλυση και εκτελεί την λεκτική ανάλυση μέσω ενός loop που διατρέχει το αρχείο εισόδου μέχρι να συναντήσει EOF. Επιπλέον, δημιουργεί ή αν υπάρχει ήδη ανοίγει το αρχείο εξόδου (lexicalAnalyzer.txt), στο οποίο καταγράφονται τα αποτελέσματα της λεκτικής ανάλυσης

2.3 Καταστάσεις του Finite State Machine

Το finite state machine έχει διάφορες καταστάσεις που καθορίζουν την τρέχουσα διαδικασία αναγνώρισης ενός token. Οι καταστάσεις αυτές καθώς και οι αντίστοιχες καταστάσης του αυτόματου είναι:

stateBegin: Αρχική κατάσταση (0)
Στην αρχική κατάσταση, ο αναλυτής ελέγχει τον χαρακτήρα και αποφασίζει σε ποια κατηγορία ανήκει. Ανάλογα με την κατηγορία, αλλάζει την κατάσταση. Παραμένει σε αυτή την κατάσταση όσο συναντά λευκούς χαρακτήρες.

stateLetter: Αναγνωρίζει αλφαβητικούς χαρακτήρες και λέξεις (1)
Αν ο χαρακτήρας είναι γράμμα ή ψηφίο, προστίθεται στο τρέχον token. Αν το token υπερβαίνει τους 30 χαρακτήρες, αναγνωρίζεται ως λάθος.

stateDigit: Αναγνωρίζει αριθμητικούς χαρακτήρες (2)
Αν ο χαρακτήρας είναι ψηφίο, προστίθεται στο τρέχον token. Αν ο αριθμός υπερβαίνει τα όρια -32767 και 32767, αναγνωρίζεται ως λάθος.

stateDivision: Αναγνωρίζει τη διαίρεση (3)
Αν αναγνωριστεί το σύμβολο '//' μεταβαίνει στην κατάσταση αποδοχής. Αν βρει μόνο '/' αναγνωρίζεται ως λάθος.

stateLess, stateMore, stateEqual, stateDifferent: Αναγνωρίζουν συγκριτικές πράξεις (αντίστοιχα 4, 5, 6, 7)

Αυτές οι καταστάσεις χειρίζονται τις συγκριτικές πράξεις (`<`, `>`, `==`, `!=`). Μόλις αναγνωριστεί ένα από τα παραπάνω σύμβολα η αντίστοιχη κατάσταση μεταβαίνει στην κατάσταση αποδοχής.

stateSharp: Αναγνωρίζει σχόλια και σύμβολα ομαδοποίησης (8)
Αν αναγνωρίσει τα σύμβολα ομαδοποίησης '#{', '#}' μεταβαίνει στην κατάσταση αποδοχής. Αν αναγνωρίσει '##' ξεκινάει την ανάγνωση σχολίων.

stateCommentBegin (9)
Διαβάζει έναν έναν όλους τους χαρακτήρες μέχρι να συναντήσει ξανά '##' χωρίς να τους καταγράφει. Εάν συναντήσει EOF μεταβαίνει σε κατάσταση stateERROR.

stateCommentEnd (10)
Αναγνωρίζει ## για το τέλος των σχολίων, αλλιώς μεταβαίνει σε κατάσταση stateERROR.

stateERROR: Αναγνωρίζει λάθη.
Όταν εντοπίζεται ένα λάθος, το καταγράφει και επαναφέρει την κατάσταση στην αρχική.

stateOK: Κατάσταση αποδοχής.
Δημιουργεί αντικείμενο τύπου Token και το προσθέτει στη λίστα με τα tokens. Επίσης καταγράφει το αναγνωρισμένο token στο αρχείο εξόδου και επιστρέφει στην αρχική κατάσταση

stateEOF
Αν ο αναλυτής συναντήσει EOF (εκτός σχολίων), μεταβαίνει στην κατάσταση αποδοχής και ολοκληρώνει επιτυχώς την λεκτική ανάλυση.

Σε περίπτωση που ο επόμενος χαρακτήρας είναι '+' (συν), '-' (μείον), '*' (αστερίσκος/επί), '%', ',' (κόμμα), ':' (άνω κάτω τελεία), '(' (αριστερή παρένθεση), '(δεξιά παρένθεση) αναγνωρίζεται και ο αναλυτής επιστρέφει στην αρχική κατάσταση.

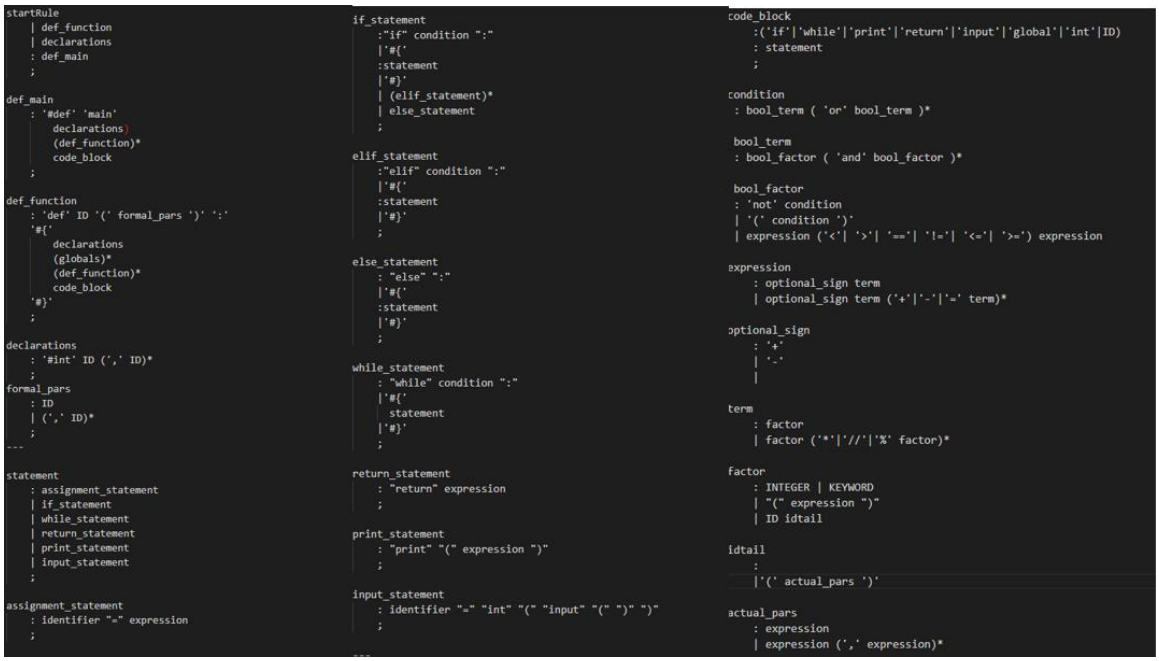
3. ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

3.1 Γενική Ιδέα

Στο κεφάλαιο αυτό παρουσιάζουμε τον συντακτικό αναλυτή. Ο συντακτικός αναλυτής ελέγχει την ορθότητα του προγράμματος βάσει μιας προκαθορισμένης γραμματικής και καθοδηγεί την παραγωγή κώδικα στις επόμενες φάσεις. Ο στόχος του είναι να διασφαλίσει ότι το πρόγραμμα που έχει γραφεί στη γλώσσα CPY είναι συντακτικά σωστό, ακολουθώντας τους κανόνες και τις δομές της γραμματικής της γλώσσας. Ο συντακτικός αναλυτής χρησιμοποιεί τα tokens που έχουν παραχθεί από τον λεξικό αναλυτή και τα οργανώνει σε μια δομή που είναι εύκολα διαχειρίσιμη για περαιτέρω επεξεργασία και εκτέλεση.

Ο συντακτικός αναλυτής ακολουθεί μια αναλυτική προσέγγιση για να επαληθεύσει την κάθε συντακτική μονάδα του προγράμματος, από τις δηλώσεις και τις εντολές, μέχρι τις εκφράσεις και τις παραμέτρους συναρτήσεων. Σε περίπτωση που εντοπιστεί κάποιο συντακτικό λάθος,

Σταματάει την ανάλυση του προγράμματος και αναφέρει στον προγραμματιστή το λάθος και το σημείο που υπήρξε αυτό.



```
startRule
    | def_function
    | declarations
    : def_main
    ;

def_main
    : '#def' 'main'
        declarations)
        (def_function)*
        code_block
    ;

def_function
    : 'def' ID '(' formal_pars ')' ':'
    '#{'
        declarations
        (globals)*
        (def_function)*
        code_block
    '#}'
    ;

declarations
    : '#int' ID (',' ID)*
    ;
formal_pars
    : ID
    | (',' ID)*
    ;
---

statement
    : assignment_statement
    | if_statement
    | while_statement
    | return_statement
    | print_statement
    | input_statement
    ;

assignment_statement
    : identifier "=" expression
    ;
```

```
if_statement
    :"if" condition ":"
    |'#{'
    :statement
    |'#}'
    | (elif_statement)*
    | else_statement
    ;

elif_statement
    :"elif" condition ":"
    |'#{'
    :statement
    |'#}'
    ;

else_statement
    : "else" ":"
    |'#{'
    :statement
    |'#}'
    ;

while_statement
    : "while" condition ":"
    |'#{'
     statement
    |'#}'
    ;

return_statement
    : "return" expression
    ;

print_statement
    : "print" "(" expression ")"
    ;

input_statement
    : identifier "=" "int" "(" "input" "(" ")" ")"
    ;
```

```
code_block
    :('if'|'while'|'print'|'return'|'input'|'global'|'int'|ID)
    : statement
    ;

condition
 : bool_term ( 'or' bool_term )*

 bool_term
 : bool_factor ( 'and' bool_factor )*

 bool_factor
 : 'not' condition
 | '(' condition ')'
 | expression ('<'| '>'| '=='| '!='| '<='| '>=') expression

expression
    : optional_sign term
    | optional_sign term ('+'|'-'|'=' term)*

optional_sign
    : '+'
    | '-'
    |

term
    : factor
    | factor ('*'|'//'|'%' factor)*

factor
    : INTEGER | KEYWORD
    | "(" expression ")"
    | ID idtail

idtail
    :
    |'(' actual_pars ')'

actual_pars
    : expression
    | expression (',' expression)*
```

Εικόνα 2. Γραμματική Γλώσσασ cpy

3.2 Περιγραφή των κλάσεων

Η κλάση που χρησιμοποιείται για την υλοποίηση του συντακτικού αναλυτή είναι η SyntaxAnalyzer και είναι υπεύθυνη για την λήψη και ανάλυση των tokens από τον λεκτικό αναλυτή κσθώς και για την ενσωμάτωση του ενδιάμεσου κώδικα και του πίνακα συμβόλων στην λειτουργία της.

Η κλάση SyntaxAnalyzer παίρνει σαν όρισμα τον πίνακα με τα tokens που έχει φτιάξει η κλάση Tokens και αρχικοποιούμε τον συντακτικό αναλυτή. Για να ξεκινήσουμε την λογική το κάνουμε από κάτω προς τα πάνω καλώντας την startRule, η συνάρτηση αυτή ελέγχει αρχικά εάν τα tokens ξεκινάνε μια από τις τρεις επιτρεπτές περιπτώσεις, όρισμα συνάρτησης, όρισμα μεταβλητής, όρισμα της main. Η λογική δουλεύει αναδρομικά ώστε το πρόγραμμα να είναι συντακτικά ορθό μόνο εάν έχει καλεστεί η main συνάρτηση, αλλιώς ξανά καλούμε την startRule καθώς μπορούμε να έχουμε πολλαπλές ορίσεις βοηθητικών συναρτήσεων ή μεταβλητών.

Εάν βρεθεί ορισμός μεταβλητής ορίζουμε την εμβέλεια στο σημείο που το αναγνωρίσαμε και μετά προχωράμε στην λογική της, που ελέγχουμε εάν ακολουθείται από λέξη που δεν ανήκει στις λέξεις κλειδιά ‘identifier’ καθώς και άμα υπάρχουν πάνω από μία μεταβλητές χωρισμένες πάντα με ‘,’.

Εάν βρεθεί ορισμός συνάρτησης καλούμε την def_function για να πραγματοποιήσει την λογική των βοηθητικών συναρτήσεων. Σε αυτό το σημείο ελέγχουμε τις παραμέτρους της συναρτήσεις και τις αποθηκεύουμε στον πίνακα συμβόλων με την κατάλληλη εμβέλεια κάθε φορά. Αφού

ορίσουμε και τις μεταβλητές είτε καθολικές είτε καινούριες, πάντα στην αρχή της συνάρτησης, καλούμε το μπλοκ του κώδικα το οποίο υλοποιεί την λογική της συνάρτησης. Εδώ επιτρέπουμε και εκφωλιασμένες συναρτήσεις καθώς και δημιουργούμε την εμβέλεια της συνάρτησης με τις αντίστοιχες τετράδες για την εκκίνηση και τερματισμό του κομματιού τις βοηθητικής συνάρτησης. Στο σημείο αυτό επίσης επιστρέφουμε την πρώτη και τελευταία τετράδα που ανήκει στην συνάρτηση και μεταφράζουμε το κομμάτι αυτό με όσες τετράδες έχει σε assembly μέσω της final_code που είναι η κλάση για τον τελικό κώδικα.

Εάν βρεθεί ορισμός main καλούμε την def_main η οποία λογική της είναι παρόμοια με αυτή των βοηθητικών συναρτήσεων με την διαφορά πως εδώ δεν έχουμε παραμέτρους για την main και καθώς δεν επιστρέφει κάτι στον πίνακα συμβόλων ορίζεται σαν Procedure και όχι σαν συνάρτηση. Επιπλέον πριν τελειώσουμε το πρόγραμμα ελέγχουμε εάν το token έχει φτάσει σε end of file όπου σημαίνει ότι έχουμε ελέγξει όλα τα tokens χωρίς να υπάρχει κάποιο λάθος οπότε καλούμε την syntaxCorect για να τερματίσουμε το πρόγραμμα.

Το μπλόκ του κώδικα καλείται για να ξεκινήσει η ροή του της λογικής κάθε συνάρτησης, ελέγχουμε εάν το token είναι κάποιο από τις λέξεις κλειδιά ή μια μεταβλητή και καλούμε τα statements που διαφοροποιούν την κατάσταση καθώς και ποια λογική θέλουμε να ακολουθήσουμε. Οι βασικές καταστάσεις χωρίζονται σε

- Ορισμό μεταβλητών ή πράξεων, δηλαδή είτε θα έχουμε αναφορά σε καθολικές μεταβλητές οπότε τις αρχικοποιούμε ή θα έχουμε κάποια πράξη όπου το αποτέλεσμα το αποθηκεύουμε σε μια από τις ορισμένες μεταβλητές.
- Ορισμό συνθήκη ελέγχου κατάστασης ‘if’ που μας μεταβαίνει στην ροή τις συνθήκης. Η λογική συνθήκης λειτουργεί σε δύο διαφορετικά βήματα. Πρώτα πρέπει να ορίσουμε την λογική συνθήκης. Αυτό το πραγματοποιείται μέσω της condition συνάρτησης. Στο επόμενο βήμα αφού έχει βρεθεί η συνθήκη καλούμε την statements για να υλοποιήσουμε την λογική μέσα στην δομή ελέγχου. Επιπλέον ελέγχουμε εάν υπάρχουν και άλλες δομές ελέγχου ‘elif’ και ‘else’. Σε αυτή την φάση υλοποιείται και η λογική άλματος για τις τετράδες. Οι λογική αυτή πραγματοποιείται δημιουργώντας αρχικά τις κατάλληλες τετράδες άλματος και στην συνέχεια μέσω της backpatch ενημερώνουμε τις τετράδες αυτές για να τον σωστό προορισμό με βάση το αποτέλεσμα της συνθήκης για την αληθή και ψευδή περίπτωση. Αυτό γίνεται καθώς επιστρέφουμε από την condition τον αριθμό της τετράδας των συνθηκών ώστε να την ενημερώσουμε και να βάλουμε για προορισμό την επόμενη τετράδα που δημιουργούμε καθώς αυτή θα είναι ο προορισμός.
- Ορισμό επαναληπτικού βρόχου ‘while’. Η κατάσταση βρόχου λειτουργεί παρόμοια με την δομή ελέγχου. Δηλαδή πρώτα πρέπει τα ελέγξουμε την συνθήκη για την οποία θα πραγματοποιείται η λογική του βρόχου. Για να το πραγματοποιήσουμε αυτό θα καλέσουμε πάλι την condition και την statements όμοια με πριν. Επιπλέον θα φτιάξουμε τις κατάλληλες τετράδες τις οποίες μετά θα ενημερώσουμε κατάλληλα με βάση την αληθή και ψευδή κατάσταση.
- Ορισμό κατάσταση επιστροφής, return_statement. Η συνάρτηση αυτή ευθύνεται για την κατάσταση επιστροφής τιμής και δημιουργεί μια τετράδα από την έκφραση που την ακολουθεί.
- Ορισμός κατάστασης εκτύπωσης. Ομοίως με πριν αφού πραγματοποιήσει τους απαραίτητους ελέγχους καλεί την expression για να πάρει την εκφράσει που πρέπει να εκτυπώσει και δημιουργεί μια τετράδα με την έκφραση αυτή.
- Ορισμός κατάστασης εισόδου. Στην κατάσταση αυτή δημιουργούμε μια τετράδα για την είσοδο δεδομένων μετα από τους κατάλληλους λεκτικούς και συντακτικούς ελέγχους, το όρισμα το παίρνουμε σαν παράμετρο και είναι η μεταβλητή στην οποία αποθηκεύουμε τα δεδομένα από την είσοδο.

Αυτές είναι οι βασικές καταστάσεις για την υλοποίηση της λογικής μιας συνάρτησης. Για την ορθή λειτουργία χρησιμοποιούνται και κάποιες επιπλέον βοηθητικές συναρτήσεις που η κάθε μια χρησιμοποιεί την άλλη σαν εργαλείο σε μια λογική κατάβασης. Ξεκινώντας πρώτα από την condition. Η συνάρτηση condition αναλύει λογικές εκφράσεις που συνδέονται με τον τελεστή or. Χρησιμοποιεί τη συνάρτηση bool_term για να αναλύσει κάθε όρο της έκφρασης και ενημερώνει τις λίστες cond_true και cond_false για τις αληθείς και ψευδείς διακλαδώσεις αντίστοιχα.

Η συνάρτηση bool_term αναλύει λογικούς όρους που συνδέονται με τον τελεστή and. Χρησιμοποιεί τη συνάρτηση bool_factor για να αναλύσει κάθε παράγοντα του όρου και ενημερώνει τις λίστες B_true και B_false.

Η συνάρτηση bool_factor αναλύει τους λογικούς παράγοντες. Μπορεί να περιλαμβάνει τον τελεστή not, παρενθέσεις ή συγκριτικές εκφράσεις.

Η συνάρτηση expression αναλύει αριθμητικές και λογικές εκφράσεις. Επεξεργάζεται προαιρετικά πρόσημα και όρους που συνδέονται με τους τελεστές +, -, και =.

Η συνάρτηση optional_sign αναλύει προαιρετικά πρόσημα (+ ή -).

Η συνάρτηση term αναλύει όρους που συνδέονται με τους τελεστές *, //, και %.

Η συνάρτηση factor αναλύει παράγοντες που μπορεί να είναι αριθμοί, μεταβλητές, εκφράσεις σε παρενθέσεις ή αναγνωριστικά.

Η συνάρτηση idtail αναλύει αναγνωριστικά και χειρίζεται την περίπτωση κλήσης συναρτήσεων.

Η συνάρτηση actual_pars αναλύει τις πραγματικές παραμέτρους που δίνονται σε μια συνάρτηση κατά την κλήση της.

Τέλος η συνάρτηση syntaxCorect ελέγχει αν το τέλος του αρχείου έχει φτάσει (δηλαδή αν ο currentToken είναι τύπου EOF) και καταγράφει ότι η σύνταξη είναι σωστή, γράφοντας τα σχετικά στοιχεία σε ένα αρχείο και εκτυπώνοντας τα ίδια στοιχεία στην κονσόλα.

3.3 Μέθοδοι Κλάσης:

1. __init__(self, tokens)

- **Περιγραφή**: Αρχικοποιεί την κλάση με τη λίστα των tokens και τη συμβολοπίνακα.
- **Λεπτομέρειες**:
  - Αποθηκεύει τα tokens.
  - Θέτει τον δείκτη token στο 0.
  - Ορίζει το τρέχον token.
  - Δημιουργεί έναν νέο συμβολοπίνακα.

2. nextToken(self)

- **Περιγραφή:** Μεταβαίνει στο επόμενο token στην λίστα των tokens.
- **Λεπτομέρειες:**

- Αυξάνει τα tokens κατά ένα.
- Εάν δεν υπάρχουν άλλα tokens ορίζει το τρέχον στο κενό.

3. **tokenCheck(self, expectedToken)**

- **Περιγραφή**: Ελέγχει αν το τρέχον token είναι ίδιο με το αναμενόμενο.
- **Λεπτομέρειες:**
  - Αν το τρέχον token είναι το επιθυμητό τότε καλεί την nextToken().
  - Αν δεν ταιριάζει σταματάει το πρόγραμμα και ρίχνει εξαίρεση συντακτικού λάθους.

4. **startRule(self):**

- **Περιγραφή**: Αρχίζει την ανάλυση με τον αρχικό κανόνα της γραμματικής.
- **Λεπτομέρειες:**
  - Ελέγχει αν το τρέχον token είναι ανάμεσα στα token εκκίνησης (‘#int’, ‘def’, ‘#def’,) και καλεί τις αντίστοιχες μεθόδους (‘declarations’, ‘def_function’, ‘def_main’).
  - Επαναλαμβάνει αναδρομικά ώσπου να φτάσει στο τέλος του αρχείου ή να βρεθεί συντακτικό λάθος.

5. **def_main(self):**

- **Περιγραφή:** Ανάλυση και επεξεργασία της δήλωσης της κύριας συνάρτησης main στο πρόγραμμα.
- **Λεπτομέρειες:**
  - Ξεκινάει μια καινούρια Procedure την οποία την τοποθετεί στον πίνακα συμβόλων.
  - Εκκινεί ένα scope που θα έχει όλες τις απαραίτητες μεταβλητές.
  - Ελέγχει ένα πρώτα ορίζεται κάποια συνάρτηση ή νέα μεταβλητή μέσα της για να καλέσει τις κατάλληλες διαδικασίες.
  - Παράγει τις κατάλληλες τετράδες για τον ενδιάμεσο κώδικα.
  - Ενημερώνει τον πίνακα συμβόλων.
  - Καλεί την code_block για να υλοποιήσει τις υπόλοιπες λειτουργίες που πιθανών να υπάρχουν μέσα στην main.

6. **def_function(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται την δήλωση βοηθητικών συναρτήσεων.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά εάν υπάρχουν τα κατάλληλα tokens για την ροή του προγράμματος σύμφωνα με τον πίνακα συμβόλων.

- Ενημερώνει για νέες εκχωρήσεις στον πίνακα συμβόλων και ορατότητας καθώς και για την έξοδο από αυτά.
- Ελέγχει και επεξεργάζεται τις παραμέτρους της συναρτήσεις και τις εκχωρεί στον πίνακα συμβόλων.
- Ελέγχει και επεξεργάζεται πιθανές ορίσεις νέων μεταβλητών καθώς και τυχόν καθολικές μεταβλητές.
- Δημιουργεί ένα χρειάζεται εκφωλιασμένες συναρτήσεις αναδρομικά.
- Δημιουργεί νέες τετράδες είτε για εκκίνηση του block είτε για τερματισμό.
- Καλεί την code_block για την υλοποίηση της λογικής της βοηθητικής συνάρτησης.

7. **declarations(self)**

- **Περιγραφή:** Η συνάρτηση αυτή ορίζει νέες μεταβλητές.
- **Λειτουργία:**
  - Ελέγχει εάν το τρέχον token είναι λέξη 'identifier' και προχωράει στο επόμενο token.
  - Ελέγχει μετά εάν υπάρχει το ',' στην περίπτωση που έχουμε πολλαπλές ορίσεις μεταβλητών.
  - Δημιουργεί μια οντότητα 'Variable' για κάθε μεταβλητή που έχει ορίσει.

8. **formal_pars(self)**

- **Περιγραφή:** Ορίζει τις παραμέτρους μιας συνάρτησης.
- **Λειτουργία:**
  - Ελέγχει εάν το τρέχον τόκεν είναι 'identifier'.
  - Αποθηκεύει σε έναν προσωρινό πίνακα την παράμετρο.
  - Εάν το επόμενο 'token' είναι το ',' τότε σε έναν βρόχο ελέγχει για πιθανές άλλες παραμέτρους για να αποθηκεύσει στον πίνακα.
  - Επιστρέφει τις παραμέτρους κάθε συνάρτησης.

9. **statements(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται διάφορους τύπους δηλώσεων.
- **Λειτουργία:**
  - Ελέγχει εάν το τόκεν είναι τέτοιο ώστε να αναθέσει νέα τιμή και καλεί την assignment_statements
  - Καθορίζει τον τύπο της δήλωσης (π.χ. ανάθεση, if, while print, return, int) και καλεί other_statements

10. **assignment_statements(self)**

- **Περιγραφή**: Επεξεργάζεται δηλώσεις εκχώρησης.
- **Λεπτομέρειες:**
  - Αναγνωρίζει το όνομα της μεταβλητής στην αριστερή πλευρά της εκχώρησης.
  - Ελέγχει για το σύμβολο '=' και το αναγνωρίζει ως τελεστή εκχώρησης.
  - Αναγνωρίζει την έκφραση στη δεξιά πλευρά της εκχώρησης και την επεξεργάζεται καλώντας τη μέθοδο expression.
  - Παράγει τα κατάλληλα quadruples για την εκχώρηση της τιμής.

11. other_statements(self, result)

- **Περιγραφή:** Επεξεργάζεται άλλους τύπους δηλώσεων(if,while,print,return,int).
- **Λειτουργίες:**
  - Ελέγχει το πρώτο token της δήλωσης και καλεί τις αντίστοιχες μεθόδους **(**if_statements**,** while_statements**,** print_statements**,** return_statements**,** input statements**)** ανάλογα με τον τύπο της δήλωσης.

12. **if_statements(self):**

- **Περιγραφή:** Επεξεργάζεται τις δηλώσεις 'if'.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά άμα η λέξεις είναι η σωστές για την ροή της δήλωσης.
  - Καλεί την condition() για να διαβάσει την συνθήκη της 'if'.
  - Παράγει τις κατάλληλες τετράδες για την λογική τις 'if'.
  - Ενημερώνει τις τετράδες για τον προορισμό του άλματος για τις αληθείς και ψευδείς εκφράσεις.
  - Ελέγχει και καλεί εάν υπάρχουν 'elif', 'else' εκφράσεις.

13.**elif_statment(self):**

- **Περιγραφή:** Επεξεργάζεται την δήλωση 'elif'.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά άμα η λέξεις είναι η σωστές για την ροή της δήλωσης.

- Καλεί την condition() για να διαβάσει την συνθήκη της ‘elif’.
- Ενημερώνει τις τετράδες για τον προορισμό του άλματος για τις αληθείς και ψευδείς εκφράσεις.

14.**else_statment(self):**

- **Περιγραφή:** Επεξεργάζεται την δήλωση ‘else’.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά άμα η λέξεις είναι η σωστές για την ροή της δήλωσης.

15. **while_statements(self):**

- **Περιγραφή:** Επεξεργάζεται δηλώσεις while.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά άμα η λέξεις είναι η σωστές για την ροή της δήλωσης.
  - Αναγνωρίζει την έκφραση της συνθήκης και την επεξεργάζεται καλώντας τη μέθοδο condition.
  - Παράγει τα κατάλληλα quadruples για το μπλοκ κώδικα του while.
  - Ενημερώνει τις τετράδες για τον προορισμό του άλματος για τις αληθείς και ψευδείς εκφράσεις.

16. **return_statements(self):**

- **Περιγραφή:** Επεξεργάζεται δηλώσεις ‘return’.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά αν οι λέξεις ακολουθούν την ορθή ροή των κανόνων.
  - Αναγνωρίζει την έκφραση που πρέπει να επιστραφεί και την επεξεργάζεται καλώντας τη μέθοδο expression.
  - Παράγει τα κατάλληλα quadruples για την εκτύπωση της έκφρασης.

17. **print_statements(self):**

- **Περιγραφή:** Επεξεργάζεται δηλώσεις ‘print’.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά αν οι λέξεις ακολουθούν την ορθή ροή των κανόνων.
  - Αναγνωρίζει την έκφραση που πρέπει να εκτυπωθεί και την επεξεργάζεται καλώντας τη μέθοδο expression.

- Παράγει τα κατάλληλα quadruples για την εκτύπωση της έκφρασης.

18. **input_statements(self, id):**

- **Περιγραφή:** Επεξεργάζεται δηλώσεις ‘input’.
- **Λεπτομέρειες:**
  - Ελέγχει συντακτικά αν οι λέξεις ακολουθούν την ορθή ροή των κανόνων.
  - Αναγνωρίζει την έκφραση που πρέπει να εισαχθεί και την επεξεργάζεται καλώντας τη μέθοδο expression.
  - Παράγει τα κατάλληλα quadruples για την εκτύπωση της έκφρασης.

19. **code_block:**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται μπλοκ κώδικα που περιέχει δηλώσεις όπως if, while, print, return, input, global, int, ή αναγνωριστικά.
- **Λεπτομέρειες:**
  - Επαναλαμβάνει την αναγνώριση και επεξεργασία δηλώσεων όσο το τρέχον token είναι ένα από τα συγκεκριμένα keywords ή ένα αναγνωριστικό.
  - Χρησιμοποιεί τη μέθοδο statements για την επεξεργασία της κάθε δήλωσης.

20. **condition(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται συνθήκες λογικών εκφράσεων.
- **Λεπτομέρειες:**
  - Αρχικοποιεί τις λίστες cond_true και cond_false.
  - Αναγνωρίζει έναν λογικό όρο χρησιμοποιώντας τη μέθοδο bool_term.
  - Αν το τρέχον token είναι ‘or’, ελέγχει το ‘or’ token και επεξεργάζεται τον επόμενο λογικό όρο.
  - Χρησιμοποιεί τη μέθοδο Quadruple.backpatch για να διορθώσει τα quadruples που δημιουργήθηκαν.
  - Ενοποιεί τις λίστες cond_true και cond_false με τα αποτελέσματα των λογικών όρων.
  - Επιστρέφει τις τελικές λίστες cond_true και cond_false.

21. **bool_term(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται λογικούς όρους που συνδέονται με and.
- **Λεπτομέρειες:**
  - Αρχικοποιεί τις λίστες B_true και B_false.
  - Αναγνωρίζει έναν λογικό παράγοντα χρησιμοποιώντας τη μέθοδο bool_factor.
  - Αν το τρέχον token είναι and, ελέγχει το and token και επεξεργάζεται τον επόμενο λογικό παράγοντα.
  - Χρησιμοποιεί τη μέθοδο Quadruple.backpatch για να διορθώσει τα quadruples που δημιουργήθηκαν.
  - Ενοποιεί τις λίστες B_true και B_false με τα αποτελέσματα των λογικών παραγόντων.
  - Επιστρέφει τις τελικές λίστες B_true και B_false.

22. **bool_factor(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται λογικούς παράγοντες που μπορεί να περιέχουν not, παρενθετικές εκφράσεις ή συγκρίσεις.
- **Λεπτομέρειες:**
  - Αν το τρέχον token είναι not, ελέγχει το not token και αναγνωρίζει τη συνθήκη με τη μέθοδο condition.
  - Αν το τρέχον token είναι (, ελέγχει το άνοιγμα και το κλείσιμο των παρενθέσεων και αναγνωρίζει τη συνθήκη με τη μέθοδο condition.
  - Αν το τρέχον token είναι αναγνωριστικό ή αριθμός, αναγνωρίζει την έκφραση και τον τελεστή σύγκρισης, και δημιουργεί τα αντίστοιχα quadruples.
  - Επιστρέφει τις λίστες B_true και B_false.
  - Χρησιμοποιεί τη μέθοδο Quadruple.backpatch για να διορθώσει τα quadruples που δημιουργήθηκαν.
  - Ενοποιεί τις λίστες B_true και B_false με τα αποτελέσματα των λογικών παραγόντων.
  - Επιστρέφει τις τελικές λίστες B_true και B_false.

22. **expression(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται αριθμητικές εκφράσεις.
- **Λεπτομέρειες:**

- Αν το τρέχον token είναι μία δήλωση (if, elif, else, while, print, return, input), καλεί τη μέθοδο statements και στη συνέχεια την expression.
- Αν το τρέχον token είναι #}, ελέγχει το κλείσιμο του μπλοκ.
- Αναγνωρίζει το προαιρετικό πρόσημο και τον όρο της έκφρασης.
- Αν το τρέχον token είναι ένας αριθμητικός τελεστής (+, -, =), αναγνωρίζει τον επόμενο όρο και δημιουργεί τα αντίστοιχα quadruples.
- Επιστρέφει την τελική θέση της έκφρασης.
-

23. **optional_sign(self):**

- **Περιγραφή:** Αναγνωρίζει προαιρετικά πρόσημα + ή -.
- **Λεπτομέρειες:**
  - Αν το τρέχον token είναι + ή -, επιστρέφει το αντίστοιχο πρόσημο.
  - Επιστρέφει το πρόσημο ή κενή συμβολοσειρά αν δεν υπάρχει πρόσημο.

24. **term(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται όρους εκφράσεων που περιέχουν πολλαπλασιαστικούς τελεστές (*, //, %).
- **Λεπτομέρειες:**
  - Αναγνωρίζει τον παράγοντα της έκφρασης χρησιμοποιώντας τη μέθοδο factor.
  - Αν το τρέχον token είναι ένας πολλαπλασιαστικός τελεστής, αναγνωρίζει τον επόμενο παράγοντα και δημιουργεί τα αντίστοιχα quadruples.
  - Αν το τρέχον token είναι ένας πολλαπλασιαστικός τελεστής, αναγνωρίζει τον επόμενο παράγοντα και δημιουργεί τα αντίστοιχα quadruples.

25. **factor(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται παράγοντες εκφράσεων που μπορεί να είναι αριθμοί, keywords, παρενθετικές εκφράσεις ή αναγνωριστικά.
- **Λεπτομέρειες:**
  - Αν το τρέχον token είναι αριθμός ή keywords, επιστρέφει το token και προχωρά στο επόμενο.
  - Αν το τρέχον token είναι (, αναγνωρίζει την παρενθετική έκφραση.

- Αν το τρέχον token είναι αναγνωριστικό, επεξεργάζεται το υπόλοιπο του αναγνωριστικού με τη μέθοδο idtail.

26. **idtail(self, name):**

- **Περιγραφή:** Επεξεργάζεται το υπόλοιπο ενός αναγνωριστικού που μπορεί να είναι κλήση συνάρτησης.
- **Λεπτομέρειες:**
  - Αν το τρέχον token είναι (, αναγνωρίζει τις παραμέτρους της συνάρτησης με τη μέθοδο actual_pars.
  - Δημιουργεί τα κατάλληλα quadruples για την κλήση της συνάρτησης και την επιστροφή της τιμής.
  - Δημιουργεί τα κατάλληλα quadruples για την κλήση της συνάρτησης και την επιστροφή της τιμής.

27. **actual_pars(self):**

- **Περιγραφή:** Αναγνωρίζει και επεξεργάζεται τις πραγματικές παραμέτρους μιας συνάρτησης.
- **Λεπτομέρειες:**
  - Αναγνωρίζει τις παραμέτρους χρησιμοποιώντας τη μέθοδο expression.
  - Δημιουργεί τα κατάλληλα quadruples για τις παραμέτρους της συνάρτησης.
  - Επιστρέφει το αποτέλεσμα της τελευταίας αναγνωρισμένης παραμέτρου.

27. **syntaxCorect(self):**

- **Περιγραφή**: Ελέγχει αν η σύνταξη του προγράμματος είναι σωστή.
- **Λεπτομέρειες**:

Αν το τρέχον token είναι EOF, καταγράφει το τέλος του αρχείου και εκτυπώνει ότι η σύνταξη είναι σωστή.

4. ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

4.1 Γενική Ιδέα

Στην ενότητα αυτή παρουσιάζουμε την διαδικασία που ακολουθήσαμε για την παραγωγή ενδιάμεσου κώδικα. Σε αντίθεση με τις προηγούμενες δύο ενότητες, η παραγωγή ενδιάμεσου κώδικα δεν είναι ανεξάρτητη, όλες οι μέθοδοι που εμπλέκονται σε αυτή τη διαδικασία καλούνται σε συγκεκριμένα σημεία του κώδικα κατά τη διάρκεια την συντακτικής ανάλυσης.

Η προσέγγιση μας βασίστηκε στις οδηγίες που μας δόθηκαν κατά την διάρκεια των διαλέξεων καθώς και αυτές στο ηλεκτρονικό σύγγραμμα.

Συγκεκριμένα, στα πλαίσια της εργασίας αυτής, ο ενδιάμεσος κώδικας που παράγεται αποτελείται από τετράδες, η καθεμία εκ των οποίων έχει 4 πεδία, ένα πεδίο operation που αναπαριστά την ενέργεια που πρέπει να γίνει και 3 πεδία x, y, z τα οποία αναπαριστούν τα τελούμενα, πάνω στα οποία θα εφαρμοστεί η ενέργεια. Η τετράδες μας δημιουργούνται από την κλάση Quadruple. Επιπλέον, κάθε τετράδα έχει και ένα id, τον αύξοντα αριθμό της. Όλες οι τετράδες είναι αριθμημένες έτσι ώστε να είναι δυνατό να γίνουν άλματα μεταξύ τους όπου χρειάζεται και για να γνωρίζουμε πάντα πια τετράδα είναι η επόμενη της τρέχουσας.

```
class Quadruple:
    current_id = 0
    def __init__(self, operation, x, y, z):
        self.id = Quadruple.current_id
        Quadruple.current_id+=1
        self.operation = str(operation)
        self.x = str(x)
        self.y = str(y)
        self.z = str(z)

    def __str__(self):
        return f"({self.operation}, {self.x}, {self.y}, {self.z})"
```

Εικόνα 3. Κλάση Quadruple

4.2 Βοηθητικές Συναρτήσεις

Με βάση τις οδηγίες που μας δόθηκαν δημιουργήσαμε τις εξής βοηθητικές συναρτήσεις

- genQuad(operation, x, y, z): Δημιουργεί μια νέα τετράδα, το πρώτο πεδίο της οποίας είναι το operation (αντιστοιχεί στο operator) και τα τρία επόμενα τα x, y, z (αντιστοιχούν στα operand1, operand2, operand 3). Ο αριθμός της τετράδας που δημιουργείται προκύπτει αυτόματα από τον αριθμό της τελευταίας τετράδας που δημιουργήθηκε, συν ένα.

- nextQuad(): Επιστρέφει την ετικέτα της επόμενης τετράδας που θα δημιουργηθεί όταν κληθεί η genQuad.
- newTemp(): Επιστρέφει το όνομα της επόμενης προσωρινής μεβλητής.
- emptyList(): Δημιουργεί και επιστρέφει μια νέα κενή λίστα.
- makeList(label): Δημιουργεί και επιστρέφει μια νέα λίστα με μοναδικό στοιχείο την ετικέτα label.
- mergeList(list1, list2): Δημιουργεί μια λίστα και ενώνει τις list1 και list2 σε αυτή.
- backpath(list, z): Διαβάζει τις τετράδες τις λίστας list και για την τετράδα που αντιστοιχεί στην ετικέτα αυτή συμπληρώνει το τελευταίο της πεδίο με το z. Όταν συμπληρωθούν όλες οι τετράδες της λίστας, αυτή δεν χρειάζεται άλλο.

4.3 Διαδικασία Παραγωγής Ενδιάμεσου Κώδικα

Οι τετράδες του ενδιάμεσου κώδικα που δημιουργούνται μέσω του προγράμματος μας ομαδοποιούνται σε blocks για την διαχείρισή τους. Ένα μπλόκ δεν ξεκινάει μόλις ο συντακτικός αναλυτής συναντήσει την γραμμή δήλωσης μιας νέας συνάρτησης, καθώς τα μπλοκ δεν επιτρέπει να χωριστούν σε “κομμάτια”, κάτι το οποίο θα γινόταν σε περίπτωση εμφωλευμένων συναρτήσεων, Καθώς η δομή μιας συνάρτησης είναι εξ αρχής αυστηρά καθορισμένη (δήλωση τοπικών μεταβλητών, αναφορά καθολικών μεταβλητών, δήλωση εμφωλευμένων συναρτήσεων, κώδικας συνάρτησης), η εντολή για την παραγωγή της τετράδας για την έναρξη ενός μπλοκ τοποθετείται μετά και την δήλωση όλων των εμφωλευμένων συναρτήσεων.

```
    while self.currentToken.token == '#int':
        self.tokenCheck('#int')
        self.declarations()
    if self.currentToken.token == 'global':
        self.tokenCheck('global')
        if self.currentToken.tokenType == 'identifier':
            self.nextToken()
            # print(self.currentToken.token)
            while self.currentToken.token == ',':
                self.tokenCheck(',')
                if self.currentToken.tokenType == 'identifier':
                    self.nextToken()
    while self.currentToken.token == 'def':
        self.nextToken()
        self.def_function()

    self.symbol_table.update(func, starting_quad=Quadruple.nextquad())
    Quadruple.genquad("begin_block",programName,'_','_')
    beginBlock = Quadruple.beginBlock(quadrupleList)
    self.code_block()
    self.symbol_table.update(func, frame_length=self.symbol_table.stack_position[self.symbol_table.current_stack])
    Quadruple.genquad("end_block",programName,'_','_')
    endBlock = Quadruple.endBlock(quadrupleList)
    self.final_code.final_code(Quadruple.blockLength(beginBlock, endBlock, quadrupleList), self.symbol_table)
    self.symbol_table.exit_scope()
    if self.tokenCheck('#}'):
        return True
```

Εικόνα 3. Εκίνηση μπλοκ προγράμματος

Αφού έχει δημιουργηθεί και τετράδα για επιστροφή τιμή (όπως θα εξηγηθεί παρακάτω) και εφόσον δεν υπάρχουν άλλες εντολές στο σώμα της συνάρτησης, δημιουργείται και η τετράδα λήξης του block. Σε περίπτωση που το block είναι της main, πριν την τετράδα λήξης δημιουργείται ακόμη η τετράδα halt, που σηματοδοτεί το τέλος του προγράμματος, καθώς με βάση την περιγραφή της γλώσσας, η main βρίσκεται πάντα τελευταία.

Η διαδικασία παραγωγής ενδιάμεσου κώδικα στη συνέχεια εξαρτάται από την γραμματική και επομένως από την δομή του συντακτικού αναλυτή. Η γραμματική μας έχει συνταχθεί με τέτοιο τρόπο ώστε οι κανόνες της να είναι ουσιαστικά ανεξάρτητοι μεταξύ τους, χαρακτηριστικό που μας επιτρέπει να χειριστούμε την παραγωγή ενδιάμεσου κώδικα για κάθε κανόνα χωριστά.

Οι κανόνες μας είναι έτσι δομημένοι ώστε με βάση την πληροφορία που συλλέγουν από τα μη τερματικά τους σύμβολα και τα τερματικά σύμβολα που αναγνωρίζουν να δημιουργούν ενδιάμεσο κώδικα ή να μεταφέρουν πληροφορία μεταξύ τους κατάλληλα. Όπου αυτό χρειάζεται, πληροφορία μεταφέρεται από έναν κανόνα σε αυτόν που τον κάλεσε ώστε τελικά όλοι οι κανόνες να έχουν την απαιτούμενη πληροφορία για να δημιουργήσουν κώδικα όταν αυτό απαιτείται.

ΑΠΛΑ STATEMENTS

Τα απλά statements, δηλαδή τα return, print και input, στο τέλος της συντακτικής τους ανάλυσης, καλούν την genQuad για δημιουργία της αντίστοιχης τετράδας του καθενός, όπως δόθηκαν οδηγίες στο μάθημα.

ΣΥΝΘΕΤΑ STATEMENTS

Assignment Statement

Το assignment statement αποτελεί ειδική περίπτωση καθώς απαιτείται ο υπολογισμός της έκφρασης του δεξιού μέλους του (μετά το =) και στην συνέχεια η παραγωγή της τετράδας του. Η πληροφορία που χρειάζεται για τον υπολογισμό αυτό παράγεται από την διαδικασία διαχείρισης αριθμητικών παραστάσεων, η οποία παρουσιάζεται παρακάτω.

Σύνθετα Statements (if, while)

Στις περιπτώσεις των if και while, απαιτείται δημιουργία τετράδων jump για τις διακλαδώσεις της ροής του προγράμματος ανάλογα με την ισχύ ή μη των συνθηκών τους.

Στην περίπτωση της if, μία τετράδα jump δημιουργείται αμέσως μετά τον έλεγχο της συνθήκης ώστε σε περίπτωση μη ισχύς να προβλεφθούν οι εντολές που ακολουθούν. Μια ακόμα δημιουργείται μετά από τις εντολές αυτές ώστε, σε περίπτωση ισχύεις, πιθανόν εντολές σε elif/else παρακάτω να προβλεφθούν. Η τετράδα-στόχος για καθεμία συμπληρώνεται αργότερα, με χρήση της backpatch.

Στην περίπτωση της while, μία τετράδα jump δημιουργείται αμέσως μετά την συνθήκη ώστε σε περίπτωση που αυτή δεν ισχύει το πρόγραμμα να βγει από το loop. Μια ακόμα δημιουργείται μετά της εντολές της while ώστε η ροή του προγράμματος να επιστρέφει στην τετράδα της συνθήκης για τον επόμενο έλεγχο.

ΛΟΓΙΚΕΣ ΠΑΡΑΣΤΑΣΕΙΣ

Οι λογικές παραστάσεις μέσα στον κώδικα αποτελούν μέρος κάποιου condition, επομένως η διαχείρισή τους ξεκινάει από την μέθοδο condition(). Καθώς οι λογικοί τελεστές μας έχουν προτεραιότητα not>and>or, η condition (χειρίζεται το or) καλεί την bool_term (χειρίζεται το and) και αυτή με την σειρά της την bool_factor (χειρίζεται το not). Η bool_factor καλεί πρώτα την expression για να χειριστεί το καθαρά αριθμητικό κομμάτι της συνθήκης όπως περιγράφεται παρακάτω.

Η bool_factor παράγει τετράδα για operations με operator κάποιο εκ των ['<', '>', '==', '!=', '<=', '>='] καθώς και τετράδα jump για την περίπτωση μη ισχύς των συγκρίσεων αυτών.

ΑΡΙΘΜΗΤΙΚΕΣ ΠΑΡΑΣΤΑΣΕΙΣ

Ομοίως, καθώς διαφορετικοί αριθμητικοί τελεστές έχουν διαφορετική προτεραιότητα, η συνάρτηση expression που χειρίζεται την πρόσθεση και αφαίρεση καλεί πρώτα την term που χειρίζεται τις πράξεις του πολλαπλασιασμού, της διαίρεσης καθώς και το modulo. Η συνάρτηση term καλεί την factor που χειρίζεται τους καθαρούς αριθμούς και μεταβλητές που μπορεί να βρεθούν σε μια παράσταση.

Η expression παράγει τετράδα για operations με operator κάποιο εκ των ['+','-', '=']και η term αντίστοιχα παράγει τετράδα για operations με operator κάποιο εκ των ['*', '//', '%']. Η επιλογή αυτή έγινε καθώς έτσι, με βάση την σειρά κλήσης, διατηρείται σωστά η προτεραιότητα των πράξεων.

ΣΥΝΑΡΤΗΣΕΙΣ

Καθώς ελέγχονται όλα τα στοιχεία μιας έκφρασης τελικά φτάνουμε στο σημείο πιθανού ελέγχου ενός identifier. Η factor τελικά καλεί την μέθοδο idtail, η οποία ελέγχει για την ύπαρξη παρενθέσεων αμέσως μετά τον identifier, ένδειξη ότι αυτός είναι όνομα συνάρτησης. Σε αυτή την περίπτωση παράγεται μια νέα προσωρινή μεταβλητή που θα αποτελέσει την μεταβλητή επιστροφής της συνάρτησης μαζί με την αντίστοιχη τετράδα καθώς και η τετράδα ώστε να κληθεί η εν λόγω συνάρτηση.

```
new_temp = Quadruple.newtemp()
self.symbol_table.insert(TemporaryVariable(new_temp, 'integer', None))
Quadruple.genquad('par',new_temp,'RET','_')
Quadruple.genquad('call',name,'_','_')
```

Εικόνα 4. Κλήση Συναρτήσεων

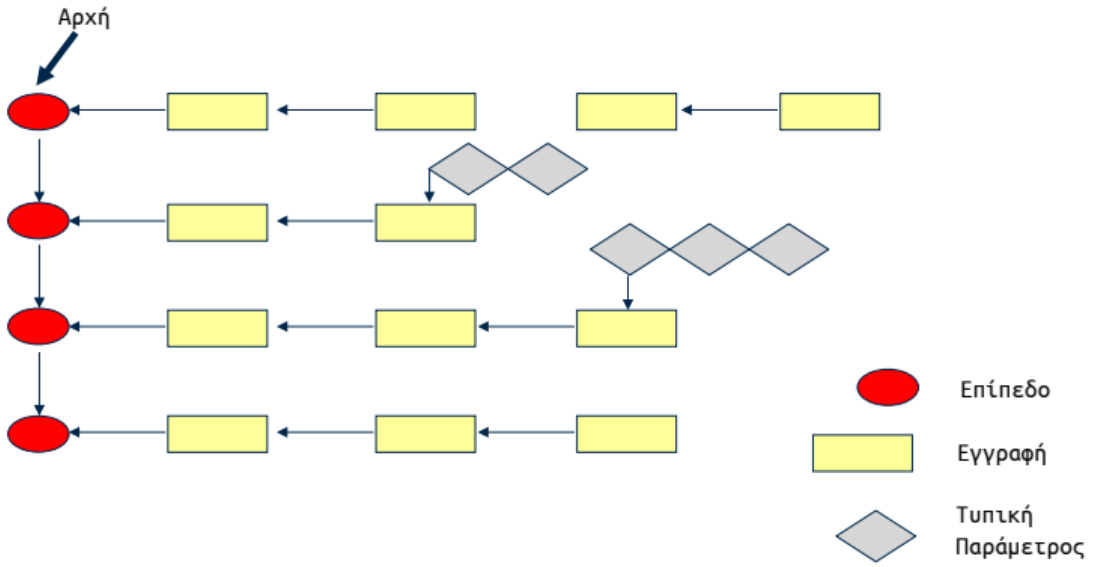
ΠΑΡΑΜΕΤΡΟΙ

Επιπλέον, πριν την παραγωγή της προσωρινής μεταβλητής, η idtail καλεί την μέθοδο actual_pars, η οποία παράγει μία τετράδα για κάθε παράμετρο της συνάρτησης. Καθώς η γλώσσα cpy ακολουθεί τους κανόνες μετάδοσης παραμέτρων της Python και αφού διαχειρίζεται μόνο ακέραιες μεταβλητές και δεν υποστηρίζει λίστες, όλες οι παράμετροι περνάνε με τιμή (cv).

```
Quadruple.genquad('par',res,'CV','_')
while self.currentToken.token == ',':
    self.tokenCheck(',')
    res =self.expression()
    Quadruple.genquad('par',res,'CV','_')
```

Εικόνα 5. Πέρασμα παραμέτρων

5. ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ



Εικόνα 6. Δομή Πίνακα Συμβόλων

5.1 Γενική Ιδέα

Στο κεφάλαιο αυτό παρουσιάζουμε την υλοποίηση του πίνακα συμβόλων. Στον πίνακα αυτό κατά την διάρκεια της συντακτικής ανάλυσης και της παραγωγής ενδιάμεσου κώδικα αποθηκεύονται πληροφορίες σχετικά με τα συμβολικά ονόματα που χρησιμοποιούμε στο πρόγραμμα (μεταβλητές, συναρτήσεις, παράμετροι κλπ), πληροφορίες που χρησιμεύουν αργότερα στην σημασιολογική ανάλυση και την παραγωγή τελικού κώδικα.

Ο πίνακας αποτελείται από επίπεδα (scopes) και εγγραφές (entities). Το αρχικό/μηδενικό scope του πίνακα δημιουργείται αμέσως με την εκκίνηση της διαδικασίας συντακτικής ανάλυσης καθώς σε αυτό αποθηκεύονται μεταξύ άλλων οι καθολικές μεταβλητές, που με βάση την εκφώνηση της εργασίας μας ορίζονται στην αρχή ενός προγράμματος.

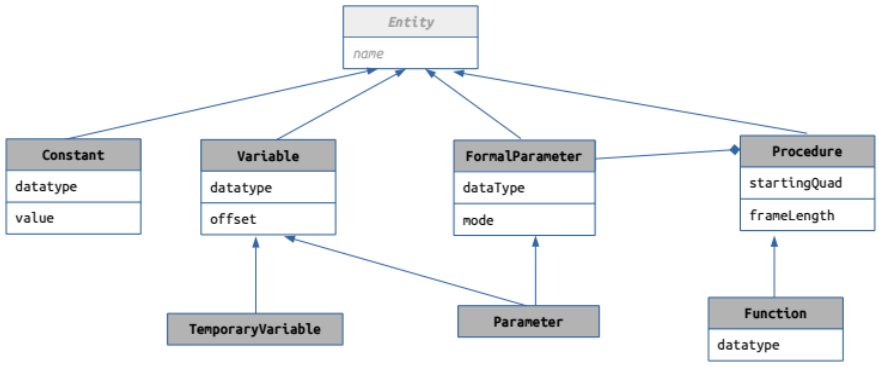
Στην συνέχεια, για κάθε συνάρτηση που ορίζεται εισάγουμε ένα νέο entity τύπου Function στον πίνακα και δημιουργούμε ένα νέο scope, στο οποίο εισάγονται οι πληροφορίες για την εν λόγω συνάρτηση όπως οι τοπικές της μεταβλητές, οι προσωρινές και καθολικές μεταβλητές που αυτή χρησιμοποιεί και οι παράμετροί της. Πληροφορίες όπως η αρχική της τετράδα και το μήκος του εγγραφήματος δραστηριοποίησης της βρίσκονται στο αρχικό scope του πίνακα, στα πεδία της οντότητας τύπου Function που αντιστοιχεί στην συνάρτηση

Μόλις ολοκληρωθεί η διαδικασία μετάφρασης της συνάρτησης, τυπώνουμε την τρέχουσα μορφή του πίνακα σε ένα εξωτερικό αρχείο (symbolTable.sym) και αφαιρούμε το scope που της αντιστοιχεί από τον πίνακα.

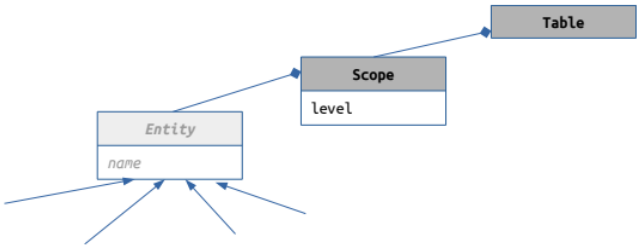
5.2 Περιγραφή των κλάσεων

Με μία ματιά στον κώδικα, είναι εμφανές ότι ο πίνακας συμβόλων έχει αρκετά περισσότερες κλάσεις από ότι τα υπόλοιπα τμήματα του κώδικα.

Για την αναπαράσταση των στοιχείων του πίνακα ακολουθήσαμε μια καθαρά αντικειμενοστραφή σχεδίαση, όπως αυτή παρουσιάζεται στο ηλεκτρονικό σύγγραμμα που μας δόθηκε. Έτσι δημιουργήσαμε την αφηρημένη κλάση Entity με μοναδικό πεδίο το name, το μόνο κοινό στοιχείο κάθε εγγραφής του πίνακα και στη συνέχεια, εκμεταλλευόμενοι την κληρονομικότητα των δημιουργήσαμε την ιεραρχική σχεδίαση που προτείνεται στο σύγγραμμα όπως φαίνεται στην παρακάτω εικόνα



Σχήμα 8.4: Η ιεραρχία των κλάσεων των εγγραφών του πίνακα συμβόλων.



5.3 Διαδικασία Προσθήκης Οντοτήτων στον Πίνακα Συμβόλων

Ανεξάρτητα από το είδος τους, οι οντότητες εισάγονται στον πίνακα μόλις ο συντακτικός αναλυτής αναγνωρίσει το όνομά τους. Αυτό όμως δεν είναι το τέλος της διαδικασίας καθώς,

ανάλογα με το είδος τους, πληροφορίες που χρειάζεται να συμπληρωθούν στον πίνακα δεν έχουν υπολογιστεί ακόμη.

Ξεκινώντας την συντακτική ανάλυση του προγράμματος, αν υπάρχουν καθολικές μεταβλητές οι οποίες θυμίζουμε ορίζονται πάντοτε στην αρχή του προγράμματος, η συνάρτηση startRule καλή την συνάρτηση enterScope του πίνακα για την δημιουργία του πρώτου scope. Σε περίπτωση που δεν υπάρχουν καθολικές μεταβλητές να δηλωθούν, η δημιουργία του πρώτου scope γίνεται μόλις οριστεί η πρώτη συνάρτηση.

Μόλις ξεκινήσει η μετάφραση μιας νέας συνάρτησης, καλείται ξανά η enterScope, αυτή τη φορά για την δημιουργία του scope της συνάρτησης. Σε αυτό εντάσσονται με την σειρά που εμφανίζονται στον κώδικα οι παράμετροι και οι τοπικές της μεταβλητές. Πριν την δημιουργία της πρώτης τετράδας ενδιάμεσου κώδικα που αντιστοιχεί στην συνάρτηση, καλείται η μέθοδος update και ενημερώνει το πεδίο starting_quad με την επόμενη τετράδα (καλώντας την nextQuad())

```
self.symbol_table.update(func, starting_quad=Quadruple.nextquad())
Quadruple.genquad("begin_block",programName,'_','_')
beginBlock = Quadruple.beginBlock(quadrupleList)
self.code_block()
self.symbol_table.update(func, frame_length=self.symbol_table.stack_position[sel
Quadruple.genquad("end_block",programName,'_','_')
endBlock = Quadruple.endBlock(quadrupleList)
self.final_code.final_code(Quadruple.blockLength(beginBlock, endBlock, quadruple
self.symbol_table.exit_scope()
if self.tokenCheck('#}'):
    return True
```

Εικόνα 7. Δημιουργία Εμβέλειας

Αφού ολοκληρωθεί η μετάφραση της συνάρτησης, καλείται ξανά η update για να ενημερώσει το πεδίο frame_length το οποίο μπορεί να υπολογιστεί ως η θέση του δείκτη στοίβας την στιγμή αυτή. Μετά και την δημιουργία της τελευταίας τετράδας ενδιάμεσου κώδικα για την συνάρτηση, καλείται η exit_scope, αφαιρεί το τρέχον scope από τον πίνακα και καθώς και το εγγράφημα δραστηριοποίησης της. Σε περίπτωση που δεν υπάρχουν πλέον scopes, δηλαδή ολοκληρώθηκε και η μετάφραση της main, επιστρέφει.

Κάθε φορά που ολοκληρώνεται η μετάφραση μιας συνάρτησης και πριν το αντίστοιχο scope αφαιρεθεί από τον πίνακα συμβόλων, καλείται από την exit_scope η μέθοδος print_state για να εκτυπωθεί η τρέχουσα μορφή του πίνακα.

6. ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

6.1 Γενική Ιδέα

Σε αυτή την ενότητα παρουσιάζουμε την υλοποίηση μας για την παραγωγή τελικού κώδικα, δηλαδή κώδικα RISC-V assembly.

Η παραγωγή τελικού κώδικα εξαρτάται σχεδόν αποκλειστικά στον ήδη παραχθέν ενδιάμεσο κώδικα, καθότι δεν είναι τίποτα περισσότερο από μετατροπή της κάθε τετράδας του στις αντίστοιχες εντολές assembly ώστε να επιτυγχάνεται το επιθυμητό αποτέλεσμα.

Για την διαδικασία αυτή δημιουργήσαμε την κλάση FinalCode που παίρνει ως ορίσματα την λίστα τετράδων και τον πίνακα συμβόλων και δημιουργεί (ή ανοίγει προς εγγραφή αν αυτό ήδη υπάρχει) το τελικό αρχείο .asm. Καθώς κάποιες γραμμές κώδικα βρίσκονται στην αρχή κάθε προγράμματος RISC-V assembly, ξεκινάμε γράφοντας τις στο αρχείο

Η παραγωγή τελικού κώδικα γίνεται σε blocks, όπως αυτά είναι καθορισμένα από τον ενδιάμεσο κώδικα, δηλαδή γίνεται για κάθε συνάρτηση μόλις ολοκληρωθεί η συντακτική της ανάλυση και η παραγωγή ενδιάμεσου κώδικα για αυτή. Για κάθε τετράδα του ενδιάμεσου κώδικα, παράγονται μερικές γραμμές τελικού κώδικα δηλαδή μερικές εντολές assembly RISC-V. Καθώς η assembly είναι γλώσσα χαμηλού επιπέδου, απαιτούνται περισσότερες εντολές της για να επιτευχθεί η διαδικασία που αναπαρίσταται με μια μόνο εντολή σε γλώσσες υψηλού επιπέδου.

Για να μπορέσουμε να ορίσουμε το block, δηλαδή τις τετράδες που πρέπει να χρησιμοποιηθούν δημιουργήσαμε στη κλάση Quadruple δύο πολύ απλές συναρτήσεις, beginBlock και endBlock, οι οποίες επιστρέφουν ακέραιες τιμές καθώς και την συνάρτηση blockLength.

Η beginBlock καλείται αμέσως μετά την δημιουργία της τετράδας (begin_block, _, _, _) για μία συνάρτηση και επιστρέφει το μήκος της λίστας των τετράδων quadrupleList[-1], καθώς και η τετράδα (begin_block, _, _, *)* είναι απαραίτητη για την παραγωγή τελικού κώδικα. Ομοίως, η endBlock καλείται αμέσως μετά την δημιουργία της τετράδας *(*end_block, _, _, _) και επιστρέφει το μήκος της λίστας τετράδων εκείνη την στιγμή. Η πληροφορία αυτή χρησιμοποιείται από την blockLength που επιστρέφει την λίστα quadrupleList[beginBlock:endBlock]. Έχοντας πλέον την πληροφορία αυτή καλείται η final_code για παραγωγή τελικού κώδικα.

6.2 Βοηθητικές Συναρτήσεις

Για την διαδικασία παραγωγής τελικού κώδικα, είναι απαραίτητη η υλοποίηση τριών βοηθητικών συναρτήσεων, οι οποίες στην πραγματικότητα είναι υπεύθυνες για το μεγαλύτερο κομμάτι της διαδικασίας. Και για τις τρεις ακολουθήσαμε ακριβώς τις οδηγίες που δόθηκαν στο ηλεκτρονικό σύγγραμμα.

gnlvCode(): Δημιουργεί τελικό κώδικα για την προσπέλαση πληροφορίας που βρίσκεται αποθηκευμένη στο εγγράφημα δραστηριοποίησης κάποιου προγόνου της συνάρτησης που μεταφράζεται την στιγμή που καλείται
loadvr(): Παράγει τελικό κώδικα ο οποίος διαβάζει μια μεταβλητή που είναι αποθηκευμένη στη μνήμη και την μεταφέρει σε ένα καταχωρητή
storerv(): Παράγει τελικό κώδικα ο οποίος αποθηκεύει στη μνήμη την τιμή μιας μεταβλητής η οποία βρίσκεται σε έναν καταχωρητή

Δημιουργήσαμε ακόμη την βοηθητική συνάρτηση createLabel, που χρησιμοποιεί ένα counter ώστε να τυπώνει για κάθε τετράδα ενδιάμεσου κώδικα την αντίστοιχη ετικέτα L(num) στον τελικό.
Στο σημείο αυτό θεωρούμε σκόπιμο να αναφερθούμε και σε δύο ακόμα βοηθητικές μεθόδους οι οποίες ορίζονται στην κλάση του Πίνακα Συμβόλων αλλά δεν χρησιμοποιούνται μέχρι το στάδιο παραγωγής τελικού κώδικα. Αυτές είναι η varlookup και η funclookup, οι οποίες εκτελούν την διαδικασία αναζήτησης μεταβλητών ή συναρτήσεων αντίστοιχα.

```
def varlookup(self, name):
    scopeNum = 0
    for scope in reversed(self.scopes):
        scopeNum += 1
        for entity in scope:
            if entity.name == name:
                return scopeNum, entity
    exit('Variable '+name+' not found')

def funclookup(self, name):
    for scope in reversed(self.scopes):
        for entity in scope:
            if entity.name == name:
                if entity.__class__.__name__ == 'Procedure' or entity.__class__.__name__ == 'Function':
                    return entity.starting_quad, entity.frame_length, entity.formalParameters
    exit('Function not found')
```

Εικόνα 8. varlookup, funclookup

Η μέθοδος varlookup αναζητά μία μεταβλητή ξεκινώντας από το τρέχων scope και μεταβαίνοντας στο προηγούμενο μέχρι να φτάσει και στο εξωτερικό δηλαδή το global. Επιστρέφει την απόσταση από το τρέχων scope του scope στο οποίο βρέθηκε η μεταβλητή, δηλαδή 1 αν βρέθηκε στο τρέχων scope, 2 αν βρέθηκε στο αμέσως προηγούμενο κοκ.

Η μέθοδος funclookup ακολουθεί παρόμοια διαδικασία. Αναζητά μία συνάρτηση ξεκινώντας από το τρέχων scope και μεταβαίνοντας στο προηγούμενο. Επιστρέφει την αρχική τετράδα, το frame length και την λίστα παραμέτρων της συνάρτηση.

6.3 Διαδικασία Παραγωγής Τελικού Κώδικα

Η βασική λογική της παραγωγής ενδιάμεσου κώδικα επιτυγχάνεται στην συνάρτηση final_code. Συγκεκριμένα οι διαδικασίες που επιτυγχάνει μπορούν να χωριστούνε στις εξής:

- **Διαχείριση της κύριας συνάρτησης (main)**:

- Αν η πρώτη τετράδα αφορά την κύρια συνάρτηση (main), αρχικοποιούνται οι δείκτες στοίβας (sp) και πλαισίου (fp) και δημιουργείται η ετικέτα Lmain για να σηματοδοτηθεί η έναρξη του κύριου προγράμματος.

 Δημιουργία Ετικετών:

- Για κάθε τετράδα δημιουργείται μια ετικέτα που αντιπροσωπεύει τη θέση της στο τελικό κώδικα.

 Διαχείριση Παραμέτρων και Κλήσεων Συναρτήσεων:

- Αν η τετράδα αφορά παράμετρο (par), αποθηκεύεται σε μια λίστα παραμέτρων.
- Αν η τετράδα αφορά κλήση συνάρτησης (call), διαχειρίζεται τη μεταφορά των παραμέτρων, και την κλήση της συνάρτησης με εντολή jal.

 Μετάφραση των Εντολών:

- Ανάλογα με την εντολή της τρεάδας (π.χ. ανάθεση, αριθμητικές πράξεις, συγκρίσεις, άλματα), μεταφράζεται σε αντίστοιχες εντολές assembly.
- Οι αριθμητικές πράξεις (π.χ., +, -, *, //, %) μεταφράζονται σε εντολές που χειρίζονται καταχωρητές και τιμές.
- Οι εντολές σύγκρισης και άλματος (<, >, ==, !=, <=, >=, jump) μεταφράζονται σε αντίστοιχες εντολές assembly για τον έλεγχο της ροής του προγράμματος.

 Εντολές Εισόδου/Εξόδου:

- Η εντολή out μεταφράζεται σε εντολές που γράφουν τιμές στην κονσόλα.
- Η εντολή inp μεταφράζεται σε εντολές που διαβάζουν τιμές από την κονσόλα και τις αποθηκεύουν στη στοίβα.

 Εντολές Τερματισμού και Επιστροφής:

- Η εντολή ret διαχειρίζεται την επιστροφή από μια συνάρτηση, αποθηκεύοντας τη διεύθυνση επιστροφής και επαναφέροντας τους δείκτες στοίβας.
- Η εντολή halt τερματίζει την εκτέλεση του προγράμματος.

 Τερματισμός Συνάρτησης:

- Στο τέλος της επεξεργασίας των τετραπλών, δημιουργούνται οι απαραίτητες ετικέτες και εντολές για τον τερματισμό των υποπρογραμμάτων και της κύριας συνάρτησης.

Έχοντας λάβει την λίστα τετράδων ενδιάμεσου κώδικα η οποίες πρέπει να μετατραπούν σε τελικό κώδικα, η finalCode Ξεκινάει την λειτουργία της. Αρχικά ελέγχει αν το μπλοκ είναι η main καθώς σε αυτή τη περίπτωση πρέπει να γίνει κατάλληλη προσαρμογή των pointers. Στην συνέχεια παράγεται ο κώδικας για το πέρασμα παραμέτρων. Σημειώνουμε ξανά ότι, καθώς η γλώσσα cpy δέχεται ως μοναδικό τύπο δεδομένων ακεραίους αριθμούς και δεν υποστηρίζει λίστες, όλες οι παράμετροι περνάν με τιμή.

Το επόμενο βήμα είναι η δημιουργία τελικού κώδικα για τον σύνδεσμο προσπέλασης και τον δείκτη στοίβας. Από αυτό το σημείο και μετά η διαδικασία είναι αρκετά απλή, καθώς ανάλογα με τις εντολές/πράξεις που ακολουθούν στο σώμα της συνάρτησης, παράγονται οι κατάλληλες εντολές τελικού κώδικα με την χρήση μιας μεγάλης δομής if/elif. Φτάνοντας στο τέλος του block, παράγονται και οι εντολές για μεταβίβαση ελέγχου και πλέον έχει ολοκληρωθεί η διαδικασία.

7. ΠΡΟΒΛΗΜΑΤΑ ΚΑΙ ΕΛΛΕΙΨΕΙΣ

Δυστυχώς δεν καταφέραμε να ολοκληρώσουμε επιτυχώς το στάδιο παραγωγής τελικού κώδικα καθώς ο τελικός κώδικας σε assembly που μεταφράζεται δεν λειτουργεί πάντα ομαλά, έχοντας πολλές φορές αποκλείσεις από τα επιθυμητά αποτελέσματα καθώς και ορισμένες φορές αδυναμία στο να κάνει compile των κώδικα. Σε ορισμένες περιπτώσεις δεν ολοκληρώνεται η παραγωγή τελικού κώδικα λόγο προβλημάτων στην αναζήτηση οντοτήτων στον Πίνακα Συμβόλων.

Τεστάροντας το πρόγραμμα με μικρά αρχεία-τεστ που δημιουργήσαμε με αυτό το σκοπό, Τα σημεία που έχουμε παρατηρήσει να υπάρχει απόκλιση είναι όταν χρησιμοποιούμε την ίδια global μεταβλητή σε παραπάνω από μια συναρτήσεις, καθώς αλλάζουν τα δεδομένα. Επιπλέον ένα άλλο πρόβλημα που παρουσιάστηκε είναι πως όταν καλούμε μια συνάρτηση και μετά τυπώνουμε το αποτέλεσμα, το αποτέλεσμα είναι επιθυμητό, ενώ όταν καλούμε πολλαπλές συναρτήσεις και μετά τυπώνουμε τα αποτελέσματα όλα μαζί, παρουσιάζονται πάλι αποκλίσεις στο στις τιμές. Θέματα τα οποία δυστυχώς δεν μπορέσαμε να διορθώσουμε στον χρόνο που μας δόθηκε.

Οποιεσδήποτε άλλες έλλειψης μπορεί να εντοπίσετε είναι επίσης θέμα έλλειψης χρόνου.

ΚΑΛΗ ΔΙΟΡΘΩΣΗ!