ΜΕΤΑΦΡΑΣΤΕΣ

ΟΜΑΔΑ

Καραγιαννίδης Χρήστος , ΑΜ : 4375

Κατσιλέρου Νεφέλη – Ελένη , ΑΜ : 4385

GLOBAL μεταβλητές

unique_num: κρατάει τον αριθμό της τετράδας που βρισκόμαστε τώρα. Αυτόν τον αριθμό επιστρέφει κάθε φορά που καλείται η nextquad (χωρίς να τον αυξάνει). Όταν καλείται η genquad φτιάχνει την τετράδα με τον αριθμό αυτόν και μετά τον αυξάνει κατά 1.

all_quads: είναι μια λίστα στην οποία προσθέτουμε κάθε φορά την τετράδα που δημιουργείται με την genquad. Όταν καλείται η backpatch διατρέχει αυτή την λίστα για να δει ποιες τετράδες ταυτίζονται με τους αριθμούς που της δόθηκαν ως όρισμα. Χρησιμοποιείται από την c() και την print_listOfAllQuads() προκειμένου να φτιάξουν τα c και int αρχεία αντίστοιχα. Τέλος χρησιμοποιείται και από την finalCode για να διαβάσει τις τετράδες του ενδιάμεσου και να παράγει τον τελικό.

temp_vars: είναι μια μεταβλητή που την αρχικοποιούμε στο 0 και αυξάνει κάθε φορά κατά 1 όταν θέλουμε να δημιουργήσουμε καινούρια προσωρινή μεταβλητή έτσι ώστε να μπορούμε να φτιάξουμε τα: T_1, T_2 κοκ στη συνάρτηση new_temp του ενδιάμεσου κώδικα.

listOfTempVars: είναι μια λίστα που κρατάει όλες τις προσωρινές μεταβλητές που έχουν δημιουργηθεί στη συνάρτηση new_temp του ενδιάμεσου κώδικα. Χρησιμοποιείται από τη συνάρτηση c() έτσι ώστε με βάση το μήκος της να γράψουμε κάθε προσωρινή μεταβλητή που περιέχει αυτή η λίστα στο αρχείο cFile.

listOfDeclarations: Κρατάει τα ονόματα από όλα τα declare που διάβασε η varlist() (συνάρτηση του syntax_analyzer) ώστε όταν δημιουργήσουμε το αρχείο .c να γράψει την γραμμή int i,j,… ; για την αρχικοποίηση των int μεταβλητών στην γλώσσα C.

declarationCounter: Μετρούσε πόσα declarations έχουν γίνει ώστε να διατρέξει μετά την λίστα όμως εν τέλει δεν χρησιμοποιήθηκε και για να μετρήσουμε το πλήθος των declarations χρησιμοποιήσαμε την len(listOfDeclarations).

symb: έχει το αρχείο test. symb στο οποίο γράφει η printSymbTable τα περιεχόμενα του πίνακα συμβόλων.

cfile: έχει το αρχείο test.c στο οποίο γράφει η c() (μέσω της CreateCFile()) τα περιεχόμενα του ενδιάμεσου κώδικα.

AllScopes: περιέχει όλα scopes. Προσθέτουμε σε αυτήν κάθε scope ακριβώς πριν το κάνουμε delete ώστε στον τελικό κώδικα να μπορέσουμε να βρούμε που ανήκει η κάθε οντότητα καθώς εκείνη την στιγμή το topScope = None αφού έχουν όλα διαγραφεί

topScope: ένας δείκτης στο scope με το μεγαλύτερο nestingLevel αυτή τη στιγμή. Αρχικά είναι None γιατί δεν υπάρχει κανένα scope για να δείχνει.

Λεκτικός Αναλυτής

Η φάση της μεταγλώττισής ξεκινάει από την λεκτική ανάλυση . Η λεκτική ανάλυση είναι η διαδικασία που μετατρέπει μια ακολουθία από χαρακτήρες σε μια ακολουθία από λεκτικές μονάδες (tokens). Μια συνάρτηση ή ένα πρόγραμμα που πραγματοποιεί λεκτική ανάλυση ονομάζεται λεκτικός αναλυτής(lexical analyzer) ο οποίος συχνά καλείται από έναν συντακτικό αναλυτή όπως θα δούμε και στη συνέχεια . Μια λεκτική μονάδα (token) είναι ένα αντικείμενο το οποίο περιέχει τρία πεδία με το πρώτο από αυτά να είναι η συμβολοσειρά που αναγνώρισε την οποία την ονομάσαμε recognized_string , το δεύτερο εντάσσει τη συμβολοσειρά σε μια κατηγορία που θα «ταιριάζει» με τον συντακτικό αναλυτή και το ονομάσαμε family π.χ. identifier, groupSymbol, number . Οι κατηγορίες αυτές που ουσιαστικά έχουν νόημα στη φάση του συντακτικού αναλυτή αλλάζουν ανάλογα με τις ανάγκες της γλώσσας που πρόκειται να χρησιμοποιήσουμε . Το τελευταίο πεδίο ενός αντικειμένου token είναι ο αριθμός της γραμμής στην οποία αναγνωρίστηκε και το ονομάσαμε line_number. Αυτό είναι επίσης χρήσιμο στην φάση του συντακτικού αναλυτή καθώς επιστρέφει έτσι εύστοχα μηνύματα για τα σφάλματα που μπορεί να εντοπίσει κατά την υλοποίηση του . Για να δούμε πιο συγκεκριμένα πως ο λεκτικός αναλυτής δημιουργεί και επιστρέφει στον συντακτικό αναλυτή τα token χρειαζόμαστε ένα διάγραμμα καταστάσεων το οποίο καλείται πεπερασμένο αυτόματο . Κάθε αυτόματο αποτελείται από την αρχική κατάσταση(start), κάποιες ενδιάμεσες και τελικές . Το αυτόματο ανάλογα με το τι θα εμφανιστεί στην είσοδο θα μεταβεί και στην κατάλληλη κατάσταση με

βάση τους κανόνες της γλώσσας που ακολουθεί . Στην είσοδο μπορεί να εμφανιστεί κάποιο ψηφίο , κάποιο γράμμα ή ψηφίο , χαρακτήρες όπως + , - , * , / , (,) , {,} , [,] καθώς επίσης και λογικοί τελεστές όπως := , < , > = , <> . Κάθε μια από τις προηγούμενες περιπτώσεις έχει τον δικό της τρόπο μεταχείρισης και ακολουθεί τις δίκες της καταστάσεις μέσα στο αυτόματο . Για παράδειγμα όταν εμφανίζεται ο τελεστής ισότητας οδηγούμαστε απευθείας σε τελική κατάσταση δηλαδή στην relOperator που είναι η κατηγορία στην οποία ανήκει . Όμως για τους υπόλοιπους λογικούς τελεστές δεν μπορούμε να πάμε κατευθείαν σε τελική κατάσταση διότι είναι πιθανό να ακολουθεί και άλλο σύμβολο όπως το '=' ή '<' ή '>' οπότε θα πρέπει να αναγνωριστεί η λεκτική μονάδα '<='και '>='και '<>' . Θέλουμε όμως κάθε φορά να μπορούμε να ξεχωρίσουμε ποιο είναι το σύμβολο που αναγνωρίσαμε δηλαδή αν είναι το '>' και όχι το '>=' ή το '<>' . Γι' αυτό το λόγο θα πρέπει να διαβάσουμε έναν χαρακτήρα ο οποίος είτε ανήκει στην επόμενη λεκτική μονάδα είτε είναι ο λευκός . Όταν είναι ο λευκός δεν υπάρχει πρόβλημα ενώ αν ανήκει στην επόμενη λεκτική μονάδα πρέπει να ακολουθήσουμε την εξής διαδικασία για να μην υπάρχει περίπτωση να χαθεί . Αυτό που μπορούμε να κάνουμε είναι για παράδειγμα αν έχουμε την έκφραση β<αγ να «κρυφοκοιτάξουμε» τι υπάρχει μετά τον λογικό τελεστή , δηλαδή να πάμε τον δείκτη μια θέση μπροστά να αναγνωρίσουμε ότι υπάρχει το 'α' άρα να βεβαιωθούμε ότι ο λογικός τελεστής είναι το < και στη συνέχεια να επιστρέψουμε τον δείκτη μια θέση πίσω προκειμένου όταν ξανακληθεί ο λεκτικός αναλυτής η αναγνώριση της εισόδου να αρχίζει από το 'α' και να μη χαθεί . Η ίδια διαδικασία θα ακολουθηθεί και στην αναγνώριση των κατηγοριών identifier , keyword και number . Σε περίπτωση που βρισκόμαστε στην αρχική κατάσταση και στην είσοδο βρεθεί ένας λευκός χαρακτήρας τότε παραμένουμε στην κατάσταση start . Ουσιαστικά αγνοούμε τους λευκούς χαρακτήρες και συνεχίζουμε παρακάτω στην ανάγνωση της εισόδου . Εάν όμως ο λευκός χαρακτήρας που συναντήσαμε είναι η αλλαγή γραμμής τότε θα πρέπει να ενημερώσουμε τον μετρητή των γραμμών , ο οποίος χρησιμεύει στο να επιστρέψουμε το πεδίο line_number που αναφέραμε παραπάνω στον συντακτικό αναλυτή. Εάν δεν φτάσουμε σε τελική κατάσταση , ο λεκτικός αναλυτής δεν τερματίζει και δεν επιστρέφει αποτέλεσμα . Γενικά ένας λεκτικός αναλυτής δεν επεξεργάζεται συνδυασμούς λεκτικών μονάδων αλλά αυτό αποτελεί έργο του συντακτικού αναλυτή . Για παράδειγμα , ένας lexical_analyzer αναγνωρίζει κάθε αγκύλη σαν ξεχωριστή λεκτική μονάδα , αλλά δεν ελέγχει αν κάθε αγκύλη που ανοίγει αντιστοιχεί σε μια αγκύλη που κλείνει . Αυτό σημαίνει ότι σε αυτή τη φάση υπάρχουν σφάλματα που

σχετίζονται με τη διάσχιση του αυτομάτου . Μπορεί ενώ βρισκόμαστε στην αρχική κατάσταση να εμφανιστεί στην είσοδο ένας χαρακτήρας που δεν ανήκει στη γλώσσα με αποτέλεσμα να μην ταιριάζει σε καμία από τις επιλογές των καταστάσεων . Σε αυτή τη περίπτωση εμφανίζουμε μήνυμα σφάλματος για μη έγκυρο χαρακτήρα . Επίσης σφάλματα μπορούμε να εντοπίσουμε εάν ενώ βρισκόμαστε στην κατάσταση για το assignment και έχουμε διαβάσει το ‘:’ , περιμένουμε να έρθει το ‘=’ και έρθει ένας άλλος χαρακτήρας , εάν έχουμε ανοίξει σχόλια και δεν τα έχουμε κλείσει πριν το τέλος του αρχείου , εάν μετά από ψηφίο δούμε γράμμα , καθώς επίσης και εάν το μήκος ενός identifier είναι μεγαλύτερο από 30 χαρακτήρες και μια ακέραια αριθμητική σταθερά δεν βρίσκεται μέσα στο επιτρεπτό εύρος τιμών που είναι από -2^32 έως 2^32 . Σε όλες τις παραπάνω περιπτώσεις ο έλεγχος δεν είναι επιτυχημένος και τότε ο λεκτικός αναλυτής θα οδηγηθεί σε κατάσταση σφάλματος.

Να σημειωθεί ότι δεν χρησιμοποιήσαμε κλάσεις για την υλοποίηση του λεκτικού αναλυτή. Πιο συγκεκριμένα η υλοποίηση μας περιγράφεται ως εξής :

Αρχικοποιούμε έναν πίνακα με όλα τα γράμματα (uppercase and lowercase) του αγγλικού αλφαβήτου ώστε να αναγνωρίζουμε αν ένας χαρακτήρας που διαβάσαμε είναι γράμμα και συνεπώς όλο το token είναι identifier/keyword.

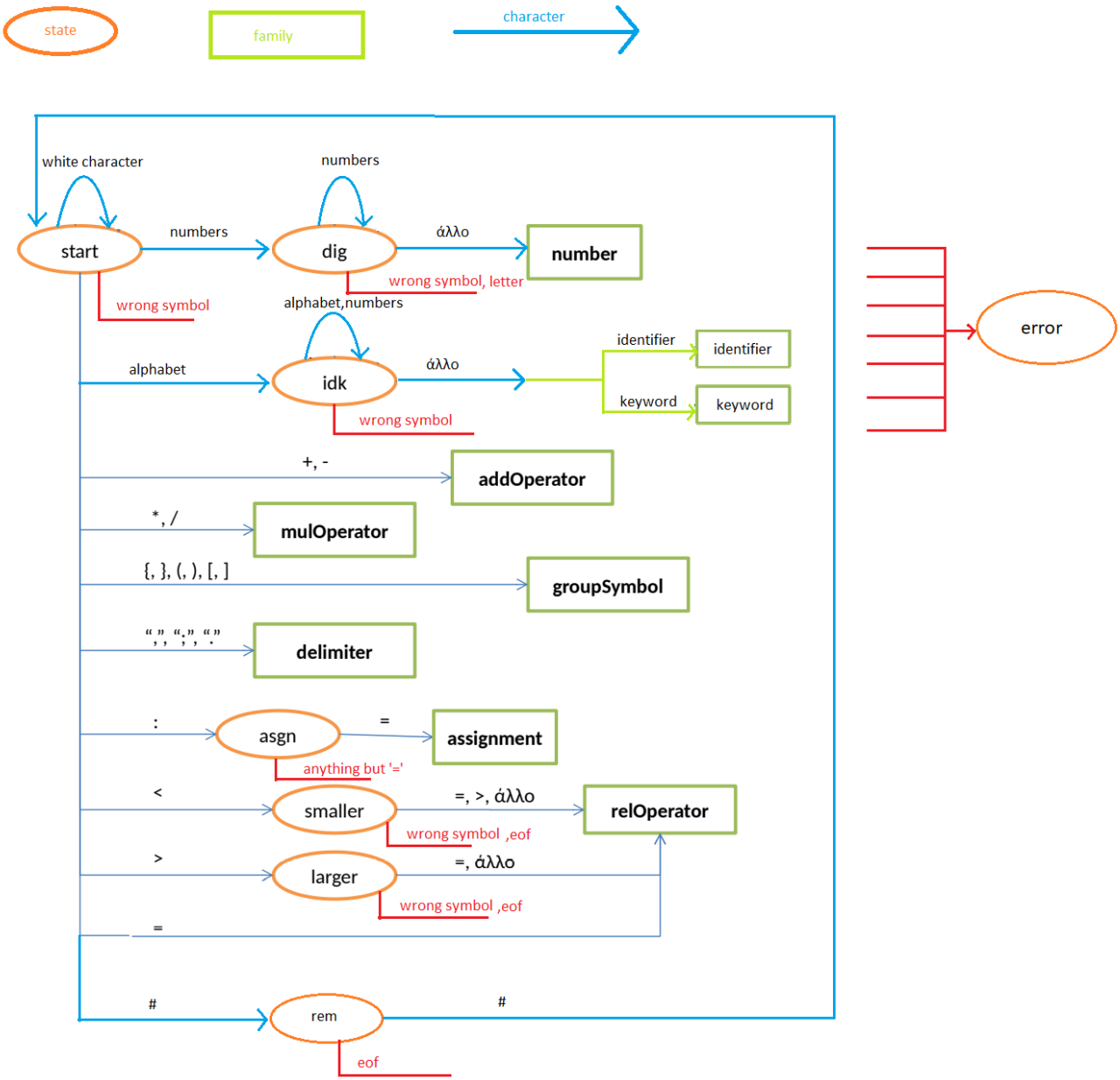
Αντιστοίχως το ίδιο κάναμε για τους αριθμούς για να ξέρουμε αν πρόκειται για ένα token που είναι αριθμός.

Επίσης το ίδιο κάναμε και για τα keywords ώστε να γνωρίζουμε αν το αλφαριθμητικό token που διαβάσαμε είναι ένα από τα δεσμευμένα keywords της Cimple.

Ο πίνακας conversionToFamily χρησιμοποιήθηκε στην 1η φάση της εργασίας για να μετατρέπει τους αριθμούς από το κάθε token που διαβάζαμε στο αντίστοιχο Family type που ανήκουν πχ token_comma = 62 η θέση του στον πίνακα conversionToFamily θα ήταν 62-50 = 12 άρα 13η θέση και συνεπώς είναι delimiter. Ο πίνακας αυτός πλέον δεν χρησιμοποιείται γιατί στην πορεία της εργασίας δεν χρειαζόταν να κάνουμε print τα αποτελέσματα του λεκτικού/συντακτικού αναλυτή.

Στον πίνακα transitions η πρώτη διάσταση του πίνακα είναι η τρέχουσα κατάσταση του μη πεπερασμένου αυτομάτου και η δεύτερη διάσταση παίρνει τιμή από τον χαρακτήρα που μόλις διαβάστηκε (για κάθε λέξη διαβάζονται οι χαρακτήρες της ένας-ένας και αποτιμάται τι είναι στο τέλος).Η επόμενη κατάσταση που θα πάμε στο αυτόματο καθορίζεται από την τρέχουσα μας

κατάσταση και το ποιον χαρακτήρα διαβάσαμε (current_state = transitions[current_state][token_character_read])



start = state_start =0

dig = state_digits =2

idk = state_letter_or_digit =1

asgn = state_assignment = 5

smaller = state_less = 3

larger = state_greater = 4

rem = state_comment = 6

numbers = [0,1,2...,9]

alphabet = [a,b,...,z, A,B,...,Z]

white_character = white_character = 0

+ = plus = 3

- = minus = 4

* = multiplication_operator = 5

/ = division_operator = 6

= = equal_sign = 7

< = less = 8

> = greater = 9

: = colon = 10

{ = left_angle_bracket = 11

} = right_angle_bracket = 12

, = comma = 13

; = semicolon = 14

) = right_parenthesis = 15

(= left_parenthesis = 16

end of file = eof = 17

wrong symbol = wrong_symbol = 18

. = full_stop = 19

[= left_square_bracket = 20

] = right_square_bracket = 21

= hashtag = 22

lex_analyzer() : Αρχικοποιεί την τρέχουσα κατάσταση στη κατάσταση start . Όσο η τρέχουσα κατάσταση δεν είναι αρνητική (error) και είναι κάποια από τις 7 υπάρχουσες (state_start ,state_letter_or_digit ,state_digits ,state_less ,state_greater ,state_assignment ,state_comment) διαβάζει έναν-έναν τους χαρακτήρες από το αρχείο και ανάλογα με το τι χαρακτήρας είναι αναθέτουμε σε μια μεταβλητή τον τύπου του χαρακτήρα που διαβάστηκε. Στη συνέχεια ελέγχουμε αν το μήκος ενός identifier (το οποίο είναι αρχικά κενό) είναι μικρότερο από 30 (όπως ζητείται) τότε βάζουμε στο recongized string τον χαρακτήρα που διαβάσαμε από το αρχείο . Αλλάζουμε την τρέχουσα κατάσταση ανάλογα με τον χαρακτήρα που διαβάσαμε και αν η καινούρια τρέχουσα κατάσταση έχει χρειαστεί να κρυφοκοιτάξει τον επόμενο χαρακτήρα για να αποτιμηθεί τότε πηγαίνουμε τον file pointer μία θέση πίσω (-1) . Επίσης καθώς διαβάζει τους χαρακτήρες ελέγχει αν κάποιος από αυτούς ανήκει στα δεσμευμένα keywords* και αν ναι τότε αναθέτουμε στην τρέχουσα κατάσταση το αντίστοιχο token . Κάνουμε και έλεγχο για το αν μια ακέραια αριθμητική σταθερά βρίσκεται μέσα στο επιτρεπτό όριο τιμών . Καλεί την print_error() .

print_error(): Ελέγχει αν το current state βρίσκεται σε κάποιο από τα 6 error states(state_error_wrong_sym,state_error_wrong_assignment,state_error_letter_after_digit,state_error_never_closing_comment,state_error_out_of_bounds_number,state_error_more_than_30_characters), κάνει print το αντίστοιχο error και exit.

Συντακτικός Αναλυτής

Τη φάση της λεκτικής ανάλυσης ακολουθεί η φάση της συντακτικής ανάλυσης. Ο συντακτικός αναλυτής αποκτά μια ακολουθία από λεκτικές μονάδες από τον λεκτικό αναλυτή και επαληθεύει ότι η ακολουθία των λεκτικών μονάδων μπορεί να παραχθεί από τη γραμματική για την πηγαία γλώσσα. Αναμένουμε από τον συντακτικό αναλυτή να αναφέρει οποιαδήποτε συντακτικά σφάλματα εντοπίσει με έναν κατανοητό τρόπο όπως η υπόδειξη μηνύματος σφάλματος στην οθόνη .Εννοιολογικά ο συντακτικός αναλυτής κατασκευάζει ένα συντακτικό δέντρο και το περνά στο υπόλοιπο του μεταγλωττιστή για περαιτέρω επεξεργασία. Στην πραγματικότητα δεν είναι ανάγκη να κατασκευαστεί σαφώς κανένα συντακτικό δέντρο , απλώς πέρα από την αναγνώριση σφαλμάτων δίνει το περιβάλλον πάνω στο οποίο θα βασιστεί η επόμενη φάση , η παραγωγή του ενδιάμεσου κώδικα αλλά και η υπόλοιπη μεταγλώττιση μετά από αυτή. Χρησιμοποιούμε γραμματική χωρίς

συμφραζόμενα (αν και μια γραμματική με συμφραζόμενα μας παρέχει πολύ περισσότερες λειτουργίες) διότι είναι πιο εύκολη στη συγγραφή και την υλοποίηση της . Μια γραμματική χωρίς συμφραζόμενα αποτελείται από τερματικά , μη-τερματικά, ένα αρχικό σύμβολο και παράγωγες. Εμπλουτίζουμε βέβαια την γραμματική αυτή με ευχετικές τεχνικές για να εξασφαλίσουμε ότι θα ικανοποιηθούν οι απαιτήσεις μας, στο επόμενο στάδιο ανάπτυξης. Κατά τη συντακτική ανάλυση όταν ένας κανόνας έχει την εναλλακτική να ενεργοποιήσει περισσότερους από έναν κανόνες τότε η επιλογή που θα κάνει καθορίζεται από την επόμενη λεκτική μονάδα που υπάρχει διαθέσιμη στην είσοδο. Ας δούμε για παράδειγμα τον κανόνα statement :

Statement -> assignStat

| ifstat

| whileStat

| …

| printStat

| ε

Στον κανόνα αυτόν , ο συντακτικός αναλυτής έχει να επιλέξει για το αν θα τον ενεργοποιήσει μέσω του κανόνα assignStat ή του ifStat ή οποιουδήποτε άλλου κανόνα υπάρχει για επιλογή. Η επιλογή αυτή λοιπόν καθορίζεται από το ποια είναι η λεκτική μονάδα στην είσοδο , δηλαδή αν είναι το while τότε θα ενεργοποιηθεί ο κανόνας whileStat. Αυτή είναι η διαδικασία που ακολουθείται σε περιπτώσεις που είναι εμφανές ποια είναι η λεκτική μονάδα που πρέπει να αναζητηθεί . Υπάρχουν όμως και περιπτώσεις όπως ο κανόνας term που δεν αναγνωρίζεται εύκολα η επιθυμητή λεκτική μονάδα . Σε αυτή τη περίπτωση λοιπόν , κοιτάμε μέσα στους λεκτικούς κανόνες που περιέχουν αυτές οι συναρτήσεις για να μπορέσουμε να καταλάβουμε αν θα αναζητήσουμε δεύτερο κανόνα ή όχι με βάση την επόμενη προς αναγνώριση λεκτική μονάδα στην είσοδο. Επίσης έχει χρειαστεί να διαφοροποιήσουμε το τετριμμένο σύμβολο που ορίζει την προτεραιότητα των πράξεων στις λογικές και στις αριθμητικές παραστάσεις . Έχουμε ορίσει δηλαδή τις αγκύλες [,] για τις λογικές παραστάσεις και τις παρενθέσεις (,) για τις αριθμητικές. Αυτό συνέβη διότι, αν η επόμενη λεκτική μονάδα που πρέπει να αναγνωρίσουμε στην είσοδο είναι το άνοιγμα παρένθεσης και έχουμε έναν κανόνα ο οποίος περιέχει σαν επιλογή

για παράδειγμα και το condition και το expression reloperator expression δεν θα μπορούμε να αποφασίσουμε ποιον δρόμο θα ακολουθήσουμε αφού και οι δυο επιλογές μπορούν να δημιουργήσουν συμβολοσειρές που αρχίζουν με αριστερή παρένθεση (. Με αυτό τον μετασχηματισμό η γραμματική μας έχει μια πιο ευανάγνωστη μορφή. Στη συνέχεια θα δούμε πως μετατρέψαμε τους κανόνες τις γραμματικής σε κώδικα. Αρχικά υλοποιήσαμε μια συνάρτηση για κάθε κανόνα της γραμματικής και έπειτα γράφουμε σε κώδικα το δεξί μέλος του κανόνα . Πιο αναλυτικά , για κάθε μη τερματικό σύμβολο ελέγχουμε ότι πράγματι συμπίπτει με την επόμενη λεκτική μονάδα στην είσοδο και καλούμε την αντίστοιχη συνάρτηση που έχουμε υλοποιήσει. Αν περάσει ο έλεγχος και όντως το τερματικό σύμβολο που συναντήσαμε συμπίπτει με την επόμενη λεκτική μονάδα τότε καταναλώνουμε την λεκτική μονάδα και προχωράμε στην αναγνώριση της επόμενης. Η κατανάλωση της λεκτικής μονάδας πραγματοποιείται με τη συνάρτηση get_token() που έχουμε υλοποιήσει και με αυτό το τρόπο καλείται ο λεκτικός αναλυτής ώστε το αντικείμενο token που περιέχει τα πεδία recognized_string, family και line_number να αποκτήσει την επόμενη λεκτική μονάδα στην είσοδο. Εάν από την άλλη , το τερματικό σύμβολο δεν συμπίπτει με την επόμενη λεκτική μονάδα τότε είτε φτάνουμε σε κατάσταση σφάλματος και εμφανίζεται το απαραίτητο μήνυμα είτε αν υπάρχει η επιλογή του κενού , την ακολουθούμε ελπίζοντας το σύμβολο που ζητάμε να αναγνωριστεί από τον επόμενο κανόνα. Τέλος μπορούμε να πούμε ότι ο συντακτικός και ο λεκτικός αναλυτής έχουν μια σχέση εξάρτησης καθώς ο δεύτερος επιστρέφει αντικείμενα της κλάσης token στον πρώτο. Δεν υλοποιήσαμε τον συντακτικό αναλυτή με κλάσεις, όμως δημιουργήσαμε όλες τις συναρτήσεις που αναφέρονται στην αντίστοιχη διαφάνεια του μαθήματος , καθώς επίσης τη μέθοδο get_token που εξηγήσαμε τη λειτουργια της παραπάνω , τη μέθοδο error() για τα συντακτικά σφάλματα και την syntax_analyzer που υλοποιεί την αναδρομική κατάβαση. Πιο συγκεκριμένα :

Syntax_analyzer(): Καλεί την συνάρτηση program() για να ξεκινήσει ο συντακτικός αναλυτής.

Get_token(): καλεί μια φορά τη lexical_analyzer() διαβάζοντας την επόμενη λέξη αποθηκεύοντας τον αριθμό γραμμής της λέξης στη μεταβλητή number.

Error(): Παίρνει σαν όρισμα έναν αριθμό με τον οποίο επιλέγουμε ποιο είναι το αντίστοιχο error και τον αριθμό γραμμής κάνοντας και τα δυο print και exit.

actualparitem() : Αν η τρέχουσα λέξη είναι το in_token (πχ. Function(**in x,inout y**)) τότε διαβάζει την επόμενη λέξη και καλεί τη συνάρτηση expression για να αποτιμήσει αυτή την έκφραση(x).Αν είναι το inout_token τότε διαβάζει την επόμενη λέξη και αν αυτή η λέξη είναι ένα identifier (αλφαριθμητικό) τότε διαβάζει την επόμενη λέξη.

actualparlist(): Καλεί την actualparitem() τουλάχιστον μια φορά και όσο υπάρχει κόμμα μετά από το item (που συμβολίζει ότι υπάρχει παραπάνω από μια παράμετρος) διαβάζει την επόμενη λέξη και καλεί ξανά την actualparitem().

Addoperator(): Αναγνωρίζει τα σύμβολα “+” και ”-” .

AssignStat(): Αν διαβάσει ένα identifier token (αλφαριθμητικό) τότε διαβάζει την επόμενη λέξη και αν αυτή είναι το assignment token “:=” διαβάζει την επόμενη λέξη και καλεί την expression() για να αποτιμήσει την έκφραση που ακολουθεί.

Programblock(): Καλείται μια φορά στην αρχή του προγράμματος αφού διαβάσουμε το όνομα του και περιμένει αρχικά ένα left angle bracket token “{“ διαβάζει την επόμενη λέξη και καλεί τις συναρτήσεις declarations() , subprograms() και block() . Αφού τελειώσουν την εκτέλεση τους όλες οι

εμφωλευμένες συναρτήσεις η συνάρτηση περιμένει ένα right angle bracket token "}" και μετά επιστρέφει .

ΣΗΜΕΙΩΣΗ : Η συγκεκριμένη συνάρτηση είναι η μοναδική που έχουμε αλλάξει το δεύτερο όρισμα στη συνάρτηση error και που έχουμε δυο περιπτώσεις για error στην ίδια if γιατί όταν ελέγχαμε το πρόγραμμα με κάποια τεχνητά λάθη που βάζαμε στο αρχείο .ci μας έβγαζε λάθος μήνυμα . Εν τέλει μπορεί να μην χρειαζόταν αυτό αλλά το καταλάβαμε εκ των υστέρων είχε και τις δυο elif αυτές.

Block(): Καλεί την statement() και όσο διαβάζει το semicolon token διαβάζει την επόμενη λέξη και ξανακαλεί την statement()

Boolfactor(): Ελέγχει 2 περιπτώσεις για το αν υπάρχει not στην αρχή της έκφρασης ή όχι. Είτε υπάρχει not είτε όχι το επόμενο token που προσπαθεί να διαβάσει είναι το left square bracket , μετά από αυτό καλεί την condition και όταν τελειώσει το condition περιμένει right square bracket. Αν στην αρχή της έκφρασης δεν διαβάσει ούτε το not ούτε το left square bracket σημαίνει ότι το expression που ακολουθεί είναι της μορφής 'expression' αντί για 'not[expression]' ή '[expression]' και συνεπώς καλεί την expression και την reloperator για να την αποτιμήσει. Δηλαδή οι 3 τύποι που μπορεί να είναι ένα boolfactor είναι πχ. not[X=5] (not και αγκύλη αμέσως μετά), [X=5 and Y>3] (ξεκίνημα με αγκύλη), Y>3 (χωρίς αγκύλη)

Boolterm(): Καλεί την boolfactor() και όσο διαβάζει το and token ξανακαλεί την boolfactor() για να αποτιμήσει πολλές boolean εκφράσεις.

CallStat(): Αν διαβάσει call token τότε πρέπει να πάρει έναν identifier (αλφαριθμητικό) και μετά άνοιγμα και κλείσιμο παρενθέσεων μέσα στις οποίες καλεί την actualparlist() για να πάρει τη λίστα με τις παραμέτρους .

Condition(): Καλεί την boolterm() και όσο διαβάζει το or token την ξανακαλεί.

Declarations(): Διαβάζει τις αρχικοποιήσεις που γίνονται στην αρχή του προγράμματος ,καλεί την varlist() για καθένα από τα declarations .

Elsepart(): Αν διαβάσει else token καλεί την statements().

Expression(): Καλεί την optionalsign() και term() τουλάχιστον μια φορά και αν μετά ακολουθούν "+" ή "-" (plus token ή minus token) καλεί την addoperator() και ξανά την term() για να διαβάσει τον επόμενο όρο.

Factor(): Κάνει αποτίμηση ενός παράγοντα από ένα γινόμενο/διαίρεση. Αν αυτός ο παράγοντας είναι αριθμός τότε απλά τον διαβάζει αν είναι παρένθεση

τότε καλεί την expression για να αποτιμήσει το εσωτερικό της παρένθεσης (και περιμένει το κλείσιμο της) ενώ αν είναι identifier(αλφαριθμητικό) τότε καλεί την idtail() γιατί τότε ίσως πρόκειται για τη τιμή που επιστρέφει μια συνάρτηση.

ForcaseStat(): Αν διαβάσει το forcase token τότε όσο διαβάσει το case token και την αριστερή παρένθεση καλεί την condition() για να δει τη συνθήκη του συγκεκριμένου case και με το που κλείσει η παρένθεση καλεί την statements() για να δει τι να κάνει στην περίπτωση που αποτιμηθεί σε true το συγκεκριμένο case .

Formalparitem(): Καλείται κατά την αρχικοποίηση μιας συνάρτησης. Ελέγχει για κάθε παράμετρο που ζητάει η εν λόγω συνάρτηση αν υπάρχει το in ή inout token πριν τη παράμετρο και όσο διαβάζει κόμμα “,” ανάμεσα στις παραμέτρους ξαναπεριμένει να διαβάσει in ή inout.

Idtail(): Μετά από ένα identifier ελέγχει αν υπάρχει άνοιγμα παρένθεσης όπου σε αυτή τη περίπτωση διαβάζει τις παραμέτρους της πλέον συνάρτησης(π.χ. function()) μέχρι να βρει δεξιά παρένθεση.

IfStat(): Αν βρει το if token τότε περιμένει αριστερή παρένθεση καλεί την condition() για την αποτίμηση της συνθήκης και αφού κλείσει η παρένθεση καλεί την statements() για να δει τι θα κάνει σε περίπτωση που είναι true η αποτίμηση και την elsepart() σε περίπτωση που υπάρχει else κάτω από την if.

IncaseStat(): Αν διαβάσει το incase token τότε όσο διαβάζει case token διαβάζει μια συνθήκη μέσα σε αριστερή και δεξιά παρένθεση και έπειτα καλεί την statements() σε περίπτωση που η συνθήκη αυτή είναι αληθής.

InputStat(): Αφού διαβάσει το input token περιμένει αριστερή παρένθεση και δεξιά παρένθεση και ανάμεσα τους ένα identifier.

Muloperator(): Αναγνωρίζει τα σύμβολα “/” και ”*” .

OptionalSign(): Αν διαβάσει plus token ή minus token καλεί την addoperator() .

PrintStat(): Αφού διαβάσει το print token περιμένει αριστερή παρένθεση, καλεί την expression() για την έκφραση μέσα στην παρένθεση και μετά περιμένει κλείσιμο παρένθεσης.

Program(): Καλείται στην αρχή του προγράμματος μια φορά και αφού διαβάσει το program token περιμένει το όνομα του προγράμματος και έπειτα καλεί την programblock() (η οποία με τη σειρά της καλεί πολλές ακόμα εμφωλευμένες

συναρτήσεις) και με το που τελειώσει περιμένει τελεία που συμβολίζει το τέλος του προγράμματος.

Reloperator(): Περιμένει το token ενός συγκριτικού τελεστή και αν δεν το πάρει επιστρέφει error.

ReturnStat(): Αφού διαβάσει return token καλεί την expression() για το εσωτερικό των παρενθέσεων.

Statement(): Ανάλογα με το τρέχον token καλεί την αντίστοιχη συνάρτηση .

Statements(): Αν διαβάσει αριστερή αγκύλη "{" καλεί την statement για κάθε μια πρόταση με ερωτηματικό ";" στο τέλος αλλιώς καλεί την statement() μια φορά γιατί χωρίς αριστερή αγκύλη "{" έχουμε ένα statement .

Subprogram(): Ελέγχει αν το υποπρόγραμμα είναι function ή procedure , μετά περιμένει το όνομα του ,μέσα στις παρενθέσεις καλεί την formalparitem() με τις παραμέτρους του υποπρογράμματος και μετά τη δεξιά παρένθεση καλεί την programblock() για το εσωτερικό του υποπρογράμματος.

Subprograms(): Όσο το τρέχον token είναι function ή procedure καλεί την subprogram().

SwitchcaseStat(): Άμα διαβάσει switchcase token τότε για κάθε case token που διαβάζει περιμένει αριστερή παρένθεση καλεί την condition() για αποτίμηση συνθήκης και όταν κλείσει η παρένθεση καλεί την statements() για τις εκφράσεις που θα εκτελεστούν στο εκάστοτε case αν είναι true. Με το που τελειώσουν τα cases περιμένει να διαβάσει το default token ακολουθούμενο πάλι από την statements() για το τι να κάνει στην default περίπτωση.

Term(): Καλεί την factor() τουλάχιστον μια φορά και όσο διαβάζει divide ή multiply token καλεί την muloperator() και την factor().

Varlist(): Διαβάζει ένα ή περισσότερα αλφαριθμητικά τα οποία χωρίζονται με κόμμα.

WhileStat(): Αν βρει το while token τότε περιμένει αριστερή παρένθεση καλεί την condition() για την αποτίμηση της συνθήκης και αφού κλείσει η παρένθεση καλεί την statements() για να δει τι θα κάνει σε περίπτωση που είναι true η αποτίμηση.

CountFileLines(): Επιστρέφει το πλήθος των γραμμών του αρχείου που δίνεται σαν input.

Ενδιάμεσος Κώδικας

Επόμενο στάδιο είναι η μετατροπή του κώδικα σε μια ενδιάμεση γλώσσα , η οποία είναι και αυτή μια γλώσσα υψηλού επιπέδου. Η ενδιάμεση αναπαράσταση αποτελεί μέσο επικοινωνίας ανάμεσα στο εμπρόσθιο και το οπίσθιο τμήμα του μεταγλωττιστή και ένα από τα σημαντικά πλεονεκτήματα της είναι ότι απλοποιεί σε μεγάλο βαθμό την σχεδίαση του. Μπορούμε να θεωρήσουμε ότι ο ενδιάμεσος κώδικας παίρνει ως είσοδο το δέντρο της συντακτικής ανάλυσης που αναφέραμε παραπάνω και μας δίνει ως αποτέλεσμα το αρχικό πρόγραμμα μεταφρασμένο σε ενδιάμεση αναπαράσταση.

Ένα πρόγραμμα σε ενδιάμεση γλώσσα αποτελείται από μια σειρά από τετράδες οι οποίες είναι αριθμημένες με έναν μοναδικό αριθμό προκειμένου να μπορούμε να αναφερθούμε σε κάθε τετράδα χρησιμοποιώντας τον αριθμό της ως ετικέτα. Επίσης θα πρέπει το σύνολο των τετράδων να είναι διατεταγμένο δηλαδή να γνωρίζουμε για κάθε τετράδα ποια είναι η επόμενη της. Κάθε τετράδα αποτελείται από έναν τελεστή που καθορίζει την ενέργεια που πρόκειται αν γίνει π.χ. ‘+’ και τρία τελούμενα στα οποία θα εφαρμοστή αυτή η ενέργεια. Υπάρχει περίπτωση εάν η ενέργεια που πρόκειται να εκτελεστεί χρειάζεται λιγότερα από τρία τελούμενα κάποια από αυτά να μείνουν κενά καθώς και αν χρειάζεται περισσότερα από τρία να χρησιμοποιήσουμε παραπάνω από μια τετράδα είτε δηλαδή δυο αυτόνομες είτε η μια ως συνέχεια της άλλης. Οι τετράδες του ενδιάμεσου κώδικα αποθηκεύονται σε καμία κατάλληλη δομή στη μνήμη. Για να το πέτυχουμε αυτό στην python δημιουργούμε μια κλάση Quad η οποία έχει τα πεδία που αναφέραμε παραπάνω. Χρειαζόμαστε να έχουμε την ελευθέρια όχι μόνο να παράγουμε νέες τετράδες , αλλά να επιστρέφουμε και να τροποποιούμε τετράδες που έχουν ήδη παραχθεί σε μεταγενέστερο στάδιο. Η διατεταγμένη λίστα που περιέχει αντικείμενα των τετράδων αυτών αποτελεί το μέσο επικοινωνίας της φάσης της παραγωγής ενδιάμεσου κώδικα και της παραγωγής τελικού κώδικα , αφού ο τελικός κώδικας παράγεται με βάση αυτή τη λίστα και πληροφορίες που αντλεί από τον πίνακα συμβόλων.

Στη συνέχεια θα περιγράψουμε τις εντολές της ενδιάμεσης γλώσσας που χρησιμοποιήσαμε για τη μετατροπή του αρχικού κώδικα σε ενδιάμεσή αναπαράσταση. Αρχικά οι εντολές begin_block και end_block οριοθετούν την αρχή και το τέλος του ενδιάμεσου κώδικα που παρήχθη για το κυρίως πρόγραμμα , μια συνάρτηση ή μια διαδικασία. Έπειτα έχουμε εντολή

εκχώρησης (:=) , αριθμητικής πράξης (op), εντολή άλματος (jump), εντολή λογικού άλματος(conditional_jump) και κλήση συνάρτησης ή διαδικασίας(call). Ο τρόπος κλήσης μιας συνάρτησης και μιας διαδικασίας δεν διαφέρουν , οι μόνοι διαχωρισμού που γίνονται αφορούν το πέρασμα των παραμέτρων.

par, name, mode, _ όπου name το όνομα της παραμέτρου, ενώ το mode είναι ο τρόπος περάσματος που γι' αυτό έχουμε τρεις επιλογές στη C-imple:

- cv: πέρασμα με τιμή
- ref: πέρασμα με αναφορά
- ret: επιστροφή τιμής συνάρτησης

Οι μεταβλητές που έχουμε χρησιμοποιήσει μέχρι στιγμής είναι οι μεταβλητές που έχουμε δηλώσει μέσα στο πρόγραμμα μας , όμως πολλές φορές δημιουργείται η ανάγκη να αποθηκεύσουμε τιμές οι οποίες χρειάζονται σε περιπτώσεις όπως είναι η επιστροφή τιμής συνάρτησης. Οι προσωρινές μεταβλητές χρειάζονται όνομα για να αναφερόμαστε σε αυτές π.χ. T_1 και να εξασφαλίσουμε ότι αυτό το όνομα δεν έχει χρησιμοποιηθεί ήδη ούτε θα ξαναχρησιμοποιηθεί μελλοντικά, ένα τύπο αν απαιτείται από τη γλώσσα και χώρο στη μνήμη για να αποθηκευτούν. Στην ενδιάμεσή αναπαράσταση θα ασχοληθούμε μόνο με τα ονόματα που θα πάρουν αυτές οι προσωρινές μεταβλητές. Το όνομα που έδωσα παραπάνω ως παράδειγμα δεν είναι τυχαίο καθώς τέτοιας μορφής πρέπει να είναι τα ονόματα προκειμένου να διασφαλίσουμε την μοναδικότητα τους, δηλαδή να ξεκινάνε με γράμμα , το T κατά προτίμηση , το οποίο ακολουθείται από κάτω παύλα '_' μιας και δεν ανήκει στο αλφάβητο της Cimple άρα είμαστε σίγουροι ότι δεν έχει χρησιμοποιηθεί από τον προγραμματιστή και τέλος έναν αριθμό ο οποίος αυξάνεται κατά ένα ,κάθε φορά που φτιάχνουμε μια νέα προσωρινή μεταβλητή και εξασφαλίζει ότι δεν έχει ξαναδημιουργηθεί ίδια.

Μια άλλη εντολή είναι η είσοδος - έξοδος(in, out) και ο τερματισμός προγράμματος(halt).

Στη συνέχεια βρίσκουμε της βοηθητικές συναρτήσεις του ενδιάμεσου κώδικα οι οποίες είναι πολύ χρήσιμες για την σχεδίαση του διότι εξυπηρετούν ενέργειες οι οποίες επαναλαμβάνονται συχνά. Οι βοηθητικές συναρτήσεις είναι οι εξής :

- genQuad(op, x,y,z): δημιουργεί μια τετράδα ,στην θέση 0 βάζει τον αριθμό της ο οποίος είναι μοναδικός (και μεγαλύτερος από όλες τις προηγούμενες τετράδες και μικρότερος από τις επόμενες).Στις θέσεις 1-

4 βάζει με την σειρά τα ορίσματα op,x,y,z και αυξάνει την global μεταβλητή unique_num ώστε ο επόμενος που θα καλέσει την nextquad() να πάρει τον σωστό αριθμό. Τέλος προσθέτει την τετράδα που μόλις φτιάχτηκε στον global πίνακα που περιέχει όλες τις τετράδες και επιστρέφει την τετράδα.

- nextQuad (): επιστρέφει την global μεταβλητή unique_num.
- newTemp (): Αυξάνει το πλήθος των προσωρινών μεταβλητών (global temp_vars) κατά 1. Φτιάχνει ένα string το οποίο είναι μια ένωση του 'T_' και του αριθμού που κρατάει το πλήθος των προσωρινών μεταβλητών (πχ. Αν temp_vars =10 τότε φτιάχνει την temp μεταβλητή T_10)
- emptyList (): Δημιουργεί και επιστρέφει μια κενή λίστα
- makeList(x): Δημιουργεί μια νέα λίστα και την επιστρέφει έχοντας σαν μοναδικό στοιχείο της το x που είναι μια μεταβλητή.
- mergeList (list1, list2): Δημιουργεί μια λίστα και συνενώνει τις λίστες list1 και list2 σε αυτήν
- backpatch (list, z): Διαβάζει μία μία όλες τις τετράδες που υπάρχουν μέσα στην list (η οποία περιέχει τα id/unique_num των τετράδων που θέλουμε να κάνουμε backpatch) που δίνεται ως όρισμα και για κάθε του all quads αν το id της τετράδας από το all quads ταυτίζεται με το id του αντικειμένου της list που εξετάζουμε τώρα τότε κάνει backpatch στην 4η θέση του την τιμή z που δόθηκε ως όρισμα.

Το επόμενο βήμα είναι να αναζητήσουμε εκείνα τα σημεία της γραμματικής δηλαδή και του συντακτικού αναλυτή στα οποία πρέπει να εισάγουμε τις σημασιολογικές ρουτίνες που θα παράγουν ενδιάμεσο κώδικα. Έτσι, κάθε κανόνας με βάση τα δεδομένα που θα συλλέξει από τα μη τερματικά σύμβολα και με βάση τα τερματικά σύμβολα που θα αναγνωρίσει : πρώτον θα παράγει ενδιάμεσο κώδικα ,όπου απαιτείται και δεύτερον θα προετοιμάσει και θα προωθήσει πληροφορία στον κανόνα που την κάλεσε. Αναφορικά με την παραγωγή κώδικα καταλαβαίνουμε ότι μια αριθμητική παράσταση πρέπει να παράγει κώδικα όταν εκτελείται μια πρόσθεση, ένας πολλαπλασιασμός , αν υπάρχει εκχώρηση τιμής σε μια μεταβλητή , λόγω κάπου υπολογισμού ή λόγω αναγνώρισης τερματικού συμβόλου.

Όσον αφορά τις λογικές παραστάσεις , μπορούν να εμφανιστούν μέσα σε μια λογική συνθήκη μέσα σε ένα πρόγραμμα C-imple για παράδειγμα στις εντολές if ή while. Σε αυτή τη περίπτωση δεν μπορούμε να κρατήσουμε τα αποτελέσματα από τους κανόνες σε μεταβλητές όπως κάναμε στις αριθμητικές

παραστάσεις καθώς επίσης σε αντίθεση με πριν , ο ενδιάμεσος κώδικας παράγεται μόνο σε έναν κανόνα του δεξιού μέλους ενώ όλοι οι υπόλοιποι διαχειρίζονται πληροφορία και την επεξεργάζονται πριν περάσουν το δικό τους αποτέλεσμα προς τα πάνω. Γενικότερα , εάν έχουμε για παράδειγμα μια λογική έκφραση όπως α>β and δ<ε και εφαρμόσουμε την ενδιάμεση αναπαράσταση θα διαπιστώσουμε ότι στο τέλος θα έχουμε αρκετές τετράδες οι οποίες δεν είναι συμπληρωμένες ούτε έχουμε την κατάλληλη πληροφορία αυτή τη στιγμή για να τις συμπληρώσουμε . Αυτό συμβαίνει συνήθως σε περιπτώσεις που έχουμε την εντολή άλματος (jump). Η λύση στο πρόβλημα αυτό είναι η πληροφορία να περάσει σαν αποτέλεσμα στον κανόνα που ενεργοποίησε, τον κανόνα της λογικής συνθήκης και εκείνος να τις συμπληρώσει κατάλληλα με τα σημεία στα οποία πρέπει να γίνουν τα λογικά άλματα. Το θέμα αυτό όπως καταλαβαίνουμε παρουσιάζεται σε κάθε έναν από τους κανόνες που συμμετέχουν στην αποτίμηση μιας λογικής έκφρασης , οπότε θα πρέπει ο κανόνας να επιστρέψει σαν αποτέλεσμα τις τετράδες που έχουν μείνει ασυμπλήρωτες , ώστε να συμπληρωθούν αργότερα από άλλους κανόνες όταν θα γνωρίζουμε πλέον το που πρέπει να γίνει το λογικό άλμα. Κάθε κανόνας που καλείται φτιάχνει για τον κανόνα που τον κάλεσε δυο λίστες :

1. Τη λίστα true η οποία αποτελείται από όλες τις τετράδες που έχουν μείνει ασυμπλήρωτες . Οι τετράδες αυτές πρέπει να συμπληρωθούν με την ετικέτα εκείνης της τετράδας που θα μεταβεί ο έλεγχος αν η συνθήκη ισχύει .
2. Τη λίστα false , η οποία κι’ αυτή αποτελείται από όλες τις τετράδες που έχουν μείνει ασυμπλήρωτες και αντίστοιχα θα συμπληρωθούν με την ετικέτα εκείνης της τετράδας που θα μεταβεί ο έλεγχος αν η λογική συνθήκη δεν ισχύει.

Αφού οι εσωτερικές λίστες κάθε κανόνα π.χ. Q(1).true και Q(2).false γεμίσουν με τις συμπληρωμένες πλέον τετράδες , δημιουργούνται άλλες λίστες π.χ. B.true και B.false οι οποίες μεταφέρουν στον κανόνα που τις ενεργοποίησε όσες τετράδες δεν μπόρεσε ο ίδιος να συμπληρώσει αρχικά. Αυτή η διαδικασία ισχύει για περιπτώσεις που εξετάζουμε συνθήκες όπως είναι η or και η and. Όταν όμως εξετάζουμε τη σύγκριση δυο αριθμητικών εκφράσεων η λογική που ακολουθούμε είναι διαφορετική. Σε αυτή τη περίπτωση , από την πληροφορία ότι δυο αριθμητικές παραστάσεις συγκρίνονται , δεν προκύπτει και το που θα μεταβεί ο έλεγχος είτε η σύγκριση δώσει ως αποτέλεσμα αληθές είτε δώσει ψευδές. Έτσι , οι ετικέτες των δυο τετράδων που δημιουργούνται θα περάσουν σαν αποτέλεσμα έναν κανόνα π.χ. R, μέσω των λιστών R.true και R.false

αντίστοιχα ,οι οποίες δεν προέρχονται από κάποιον άλλον κανόνα όπως συνέβαινε παραπάνω, ώστε να γίνει η συμπλήρωση σε υψηλότερο επίπεδο.

Μελετάμε επίσης το πως κατασκευάζεται σχέδιο ενδιάμεσου κώδικα για τις τρεις βασικές δομές της γλώσσας : τη δομή επανάληψης while , τη δομή απόφασης if και τη δομή επιλογής switchcase. Καθώς επίσης και τις δομές incase και forcase. Θα δούμε ενδεικτικά τη δομή επανάληψης while :

whileStat -> while (condition) statements, ακολουθεί περίπου ίδια διαδικασία όσον αφορά τις λίστες condition. true και condition. false που δημιουργούνται. Πέρα όμως από το πως θα διαχειριστούμε τις λίστες , πρέπει να σκεφτούμε πως αφού εκτελεστούν οι εντολές των statements ο έλεγχος θα μεταβαίνει στην αρχή της λογικής συνθήκης ώστε αυτή να επανεξετάζεται. Αυτό θα μπορέσει να συμβεί κάνοντας δυο πράγματα. Το πρώτο είναι να σημειώνεται η ετικέτα της πρώτης τετράδας της συνθήκης όταν αυτή δημιουργείται και δεύτερον να δημιουργηθεί και η τετράδα η οποία θα κάνει το άλμα από το τέλος των statements στην αρχή της condition. Σε αυτό το σημείο αξίζει να σημειώσουμε ότι παίζει σημαντικό ρόλο στη συγκεκριμένη περίπτωση η σειρά με την οποία υλοποιούνται οι εντολές του ενδιάμεσου κώδικα για να διασφαλιστεί η σωστή εκτέλεση της δομής.

Τέλος στη φάση του ενδιάμεσου κώδικα , θα αναφερθούμε στις κλήσεις των συναρτήσεων. Όταν μια διαδικασία ή συνάρτηση θέλει να δει τις παραμέτρους της θα ξεκινήσει από την κλήση της και θα επιστρέφει προς τα πίσω .Η συνάρτηση μόλις ολοκληρωθεί επιστρέφει ένα αποτέλεσμα το οποίο πρέπει να αποθηκευτεί. Γι’ αυτό το λόγο χρειαζόμαστε μια μεταβλητή η οποία όμως θα είναι προσωρινή (η έννοια της έχει εξηγηθεί παραπάνω) και θα θεωρούμε ότι εκεί θα αποθηκεύεται το αποτέλεσμα της συνάρτησης.

Εξήγηση προσθηκών στις συναρτήσεις της syntax_analyzer :

Όλες οι συναρτήσεις της syntax_analyzer() κάνουν όλα όσα αναφέραμε πιο πάνω (στον συντακτικό αναλυτή) απλά για τις ανάγκες του ενδιάμεσου κώδικα εμπλουτίστηκαν κατάλληλα. Κάποιες συναρτήσεις πλέον επιστρέφουν τιμές ,κάποιες άλλες κρατάνε τις τιμές που επιστρέφουν σε άλλες μεταβλητές και κάποιες καλούν την genquad για να φτιάξουν την αντίστοιχη τετράδα που χρειαζόμαστε.

actualparitem() : Αν διαβαστεί in token (πάει να πει ότι έχουμε πέρασμα με τιμή) τότε κρατάει το αποτέλεσμα που επιστρέφει η expression σε μια μεταβλητή E και καλεί την genquad για την E βάζοντας ως 2ο όρισμα CV. Αν

όμως διαβάσει token inout τότε κρατάει το όνομα του identifier που ακολουθεί και καλεί την gen quad για αυτό το id με τύπο μεταβλητής REF καθώς έχουμε πέρασμα με αναφορά.

Addoperator(): Κρατάει στην μεταβλητή temp_add_op το identifier είτε είναι + είτε - και επιστρέφει αυτό το σύμβολο.

AssignStat(): Κρατάει το identifier στο οποίο θα αποθηκευτεί η έκφραση που ακολουθεί σε μια μεταβλητή. Αμέσως μετά το assignment token αποθηκεύεται το αποτέλεσμα που επιστρέφει η expression και καλεί την genquad για το id αυτό δίνοντας του την τιμή του Expression.

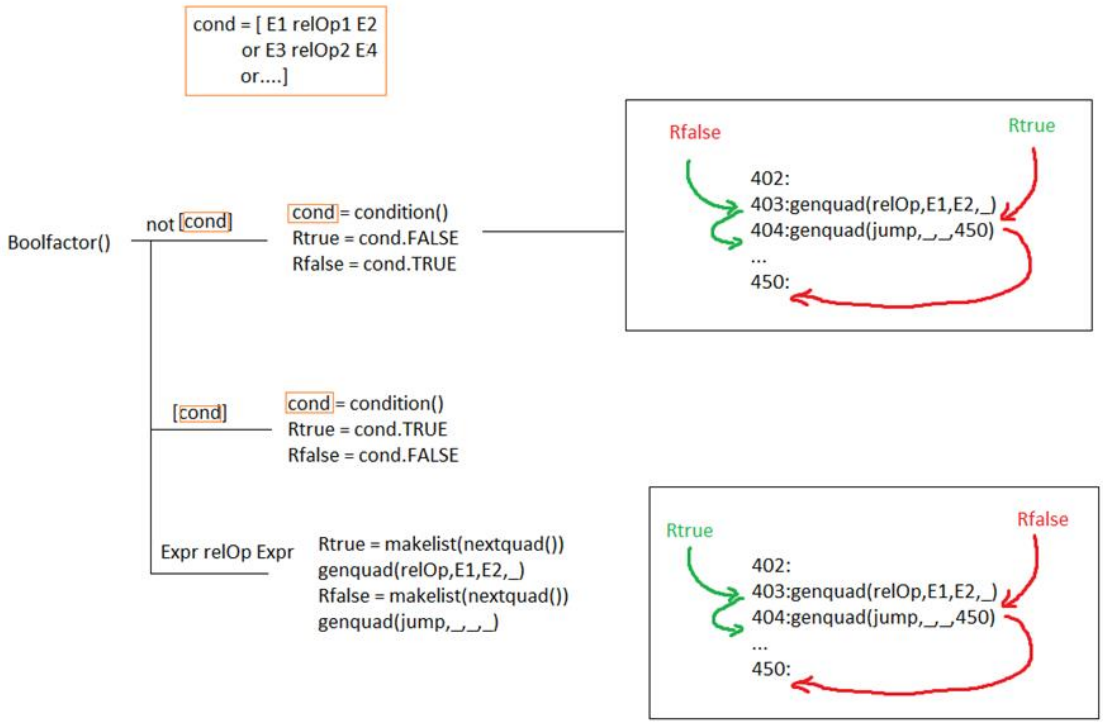
Programblock(): Καλεί την genquad για το begin block με όρισμα το name το οποίο δόθηκε ως όρισμα στην κλήση της ProgramBlock και είναι το όνομα του προγράμματος. Αφού διαβάσει το right angle bracket (σημαίνει ότι τέλειωσε το εκάστοτε block) ελέγχει αν το block που μόλις τέλειωσε είναι το block της main. Αν είναι τότε καλεί την genquad να φτιάξει την 4αδα με το halt και αμέσως μετά πάλι με την genquad φτιάχνει την 4αδα end_block η οποία θα έμπαινε έτσι κ αλλιώς (είτε ήταν το block της main είτε όχι).

Boolfactor(): Δημιουργεί δυο κενούς πίνακες Rtrue και Rfalse. Κρατάει τις τιμές που επιστρέφει η condition στην μεταβλητή cond και μετά αναλόγως με το πως είχε συνταχθεί η φράση αναθέτει στα Rtrue και Rfalse “τι να κάνουν” . Δηλαδή αν υπήρχε not στην έκφραση τότε αναθέτουμε στην Rtrue το cond[1] (που είναι το path που ακολουθούμε αν έβγαινε false ΑΛΛΑ ΑΝΤΙΣΤΡΕΦΟΝΤΑΙ ΤΑ PATHS) και στην Rfalse το cond[0] (που έχει το ‘path’ που θα ακολουθούσαμε αν έβγαινε true). Αν δεν υπήρχε το not τότε κανονικά αναθέτουμε στο Rtrue το cond[0] (true ‘path’ που επέστρεψε η condition) και στο Rfalse το cond[1] (που έχει το false ‘path’). Τέλος αν δεν υπήρχαν αγκύλες τότε κρατάμε το πρώτο expression, τον συγκριτικό τελεστή και το 2ο expression σε μια μεταβλητή τον καθένα. Βάζει στον πίνακα Rtrue την τιμή της επόμενης τετράδας (στην οποία θα πάμε αν αποτιμηθεί true η έκφραση μας) μετά καλεί την genquad δίνοντας ως ορίσματα τις 3 μεταβλητές που αποθηκεύσαμε νωρίτερα και βάζει στην λίστα Rfalse την τιμή της επόμενης τετράδας (στην οποία θα πάμε αν η έκφραση ήταν false) . Δηλαδή πχ Rtrue =[20]

[20,relop,x,y,_]

Rfalse = [21] . Άρα αν βγει true πάει στην τετράδα εκείνη αλλιώς φεύγει από

κάτω. Η συνάρτηση επιστρέφει τους πίνακες Rtrue και Rfalse.



Boolterm(): Κρατάει τις τιμές που επιστρέφει η boolfactor σε έναν πίνακα και δίνει την true τιμή της boolfactor στον πίνακα Qtrue και την false στον πίνακα Qfalse. Όσο διαβάζει και άλλον boolean όρο κάνει backpatch στον Qtrue που θα πρέπει να πάει αν βγει true που είναι η επόμενη τετράδα , κρατάει τον επόμενο boolean παράγοντα σε μια μεταβλητή, κάνει merge τον ήδη υπάρχον πίνακα Qfalse με την false τιμή που επέστρεψε η boolfactor() και δίνει την true τιμή που επέστρεψε η boolfactor στον πίνακα Qtrue. Αυτό επαναλαμβάνεται για κάθε and που υπάρχει και ενώνει boolean εκφράσεις. Η συνάρτηση επιστρέφει τα Qtrue και Qfalse.

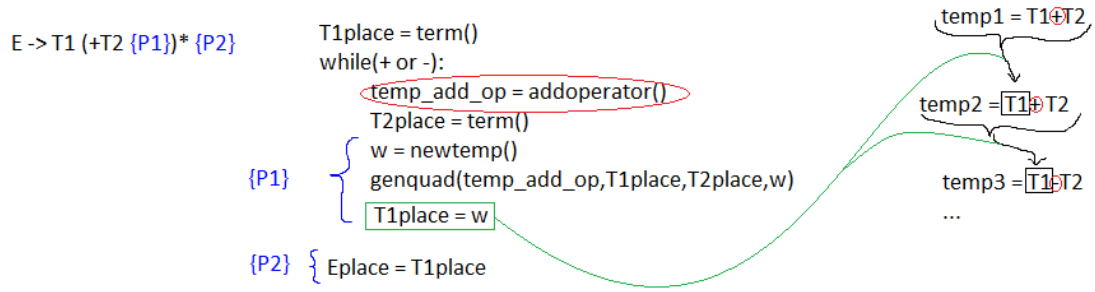
CallStat(): Κρατάει στην μεταβλητή id το identifier που διάβασε αμέσως μετά το call token και αφού κλείσει η παρένθεση με τις παραμέτρους τότε καλεί την genquad για να δημιουργήσει μια τετράδα στην οποία δίνει ως πρώτο όρισμα το call και ως 2ο το id της διαδικασίας που πρόκειται να κληθεί .

Condition(): Κρατάει ότι επιστρέφει η κλήση της boolterm() σε μια μεταβλητή και βάζει στον πίνακα Btrue το true μέρος αυτής της μεταβλητής και στον Bfalse το false. Όσο διαβάζει κι άλλον boolean όρο διαχωρισμένο από or , ‘στέλνει’ την Bfalse στην επόμενη τετράδα (με την χρήση της backpatch) , κρατάει σε μια

άλλη μεταβλητή το αποτέλεσμα του καινούριου όρου και κάνει merge το true μέρος που επέστρεψε ο καινούριος όρος με το ήδη υπάρχον Btrue. Τέλος θέτει ως Bfalse το false μέρος του καινούριου όρου. Η συνάρτηση επιστρέφει τα Btrue και Bfalse.

Declarations(): Αυξάνει έναν global counter ,,που μετράει το πόσες γραμμές υπάρχουν που κάνουν declare , κάθε φορά που διαβάζει ένα declare token και καλεί την varlist.

Expression(): Αφού η expression ελέγχει για προσθέσεις και αφαιρέσεις όρων, κρατάει τον πρώτο όρο σε μια μεταβλητή T1place και όσο διαβάζει '+' ή '-' , κρατάει το '+' ή '-' σε μια μεταβλητή temp_add_op, τον επόμενο όρο στην μεταβλητή T2place. Δημιουργεί μια προσωρινή μεταβλητή (w) , φτιάχνει μια τετράδα temp = T1 '+/-' T2 [genquad(operator,T1,T2,temp)] και θέτει ως πρώτο όρο όλης της επόμενης πράξης(αν υπάρχει) το αποτέλεσμα αυτηνής (δηλαδή την temp μεταβλητή). Η συνάρτηση επιστρέφει το αποτέλεσμα της έκφρασης (ανεξαρτήτως από το πόσες προσθέσεις η αφαιρέσεις έγιναν)



Factor(): Κρατάει στην μεταβλητή fact τον πρώτο όρο του γινομένου εάν είναι αριθμός , αλλιώς αν διαβάσει παρένθεση κρατάει στην μεταβλητή fact το αποτέλεσμα της έκφρασης που επιστρέφει η expression(). Αν όμως έχει διαβάσει ένα identifier κρατάει στην μεταβλητή fact το όνομα του και καλεί την idtail για αυτό το όνομα , κρατώντας το αποτέλεσμα της στην μεταβλητή fact πάλι.

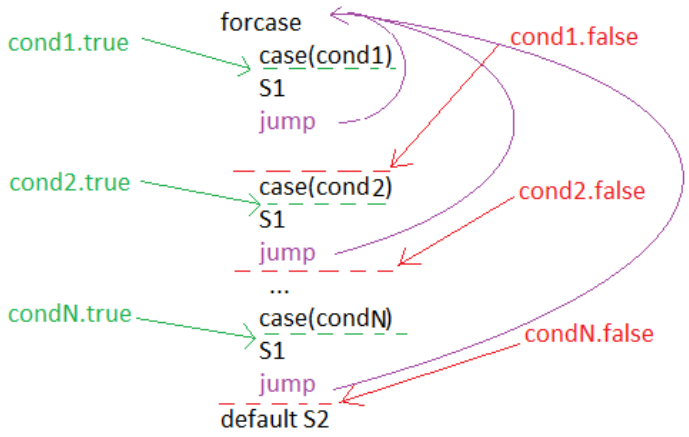
ForcaseStat(): Κρατάμε στην μεταβλητή firstCondQuad την πρώτη τετράδα της συνθήκης και έπειτα κρατάμε το τι επιστρέφει η condition για τα case δηλαδή το CondTrue και το CondFalse ανάλογα με το αν η συνθήκη είναι αληθής ή ψευδής αντίστοιχα. Με τη χρήση της backpatch στέλνει την αληθή συνθήκη (CondTrue) στην επόμενη τετράδα ώστε να μπούμε στα statements εκείνου του

case. Αφού εκτελεστούν τα statements εκείνου του case φτιάχνουμε μια τετράδα genquad(jump,_,_,firstCondQuad) ώστε να πάμε πάλι στην αρχή της forcase και να διατρέξουμε τα cases . Αν δεν εκτελεστεί κανένα από τα cases τότε εκτελούμε τα statements που βρίσκονται μετά την default.

```
forcase {P1}( (cond) {P2}S1 {P3})*
        default:  S2
```

```
{P1} {firstCondQuad =  nextquad()
      while(case)
          Condition = condition()
          CondTrue = Condition.true
          CondFalse = Condition.false
{P2}  { backpatch(CondTrue,nextquad())
```

```
{P3}  { genquad(jump,_,_,firstCondQuad)
      { backpatch(condFalse,nextquad())
```



Idtail(): Σε περίπτωση που υπάρχει παρένθεση μετά από ένα identifier (άρα πρόκειται για συνάρτηση και όχι μεταβλητή) , δημιουργεί μια προσωρινή μεταβλητή για την οποία δημιουργεί μια τετράδα με πέρασμα με αναφορά. Μετά φτιάχνει άλλη μια τετράδα για την κλήση της συνάρτησης. Η συνάρτηση επιστρέφει την προσωρινή μεταβλητή αν βρήκε παρένθεση (άρα πρόκειται για συνάρτηση) αλλιώς επιστρέφει το ίδιο το identifier το οποίο της δόθηκε ως όρισμα (γιατί δεν ήταν τελικά συνάρτηση αλλά μεταβλητή).

IfStat(): Μετά από ένα if token κρατάει το “που να πάει” που επιστρέφει η condition και βάζει το true μέρος στον πίνακα Btrue και το false στον Bfalse. Αν βγει true η συνθήκη θέλουμε να εκτελέσουμε τον κώδικα στο επόμενο statements άρα την ακριβώς επόμενη τετράδα (και συνεπώς κάνουμε backpatch το true στο NextQuad()) .Με το που τελειώσει το condition (το καταλαβαίνουμε γιατί διαβάσαμε δεξιά παρένθεση) και διαβάσουμε τα statements αυτής της if (με την κλήση της συνάρτησης statements()) ,

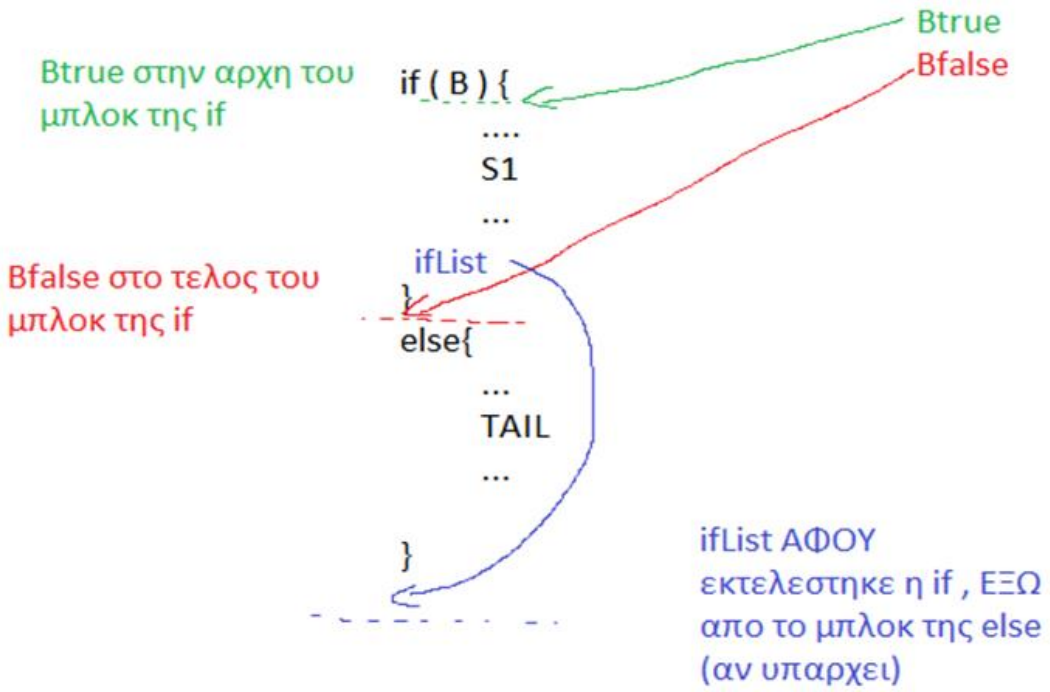
φτιάχνουμε μια κενή λίστα ifList που περιέχει τον αριθμό της επόμενης τετράδας. Κάνουμε backpatch την Bfalse στην επόμενη τετράδα (δηλαδή αν το condition στο if βγήκε false τότε με την backpatch αυτή πηγαίνουμε στην επόμενη τετράδα από το τέλος του block του if) , καλούμε την genquad για να κάνει jump έξω από την else (αφού εκτελέσαμε την if) όμως δεν ξέρουμε σε ποια τετράδα ακριβώς οπότε την αφήνουμε κενή. Καλούμε την elsepart() για την περίπτωση που υπάρχει else και αφού υπάρχει κάνουμε backpatch την jump που αφήσαμε κενή πριν στην επόμενη τετράδα αμέσως μετά το τέλος του block της else(αν υπάρχει).

ifstat ——————————

B = condition()
Btrue = B.true
Bfalse = B.false
backpatch(Btrue,nextQuad())

S1 = statements()
ifList = makelist(nextQuad())
genquad(jump,_,_,_)

TAIL = elsepart()
backpatch(ifList,nextquad())



IncaseStat(): Κρατάμε στην μεταβλητή firstCondQuad την επόμενη τετράδα ,η οποία βρίσκεται πριν από όλα τα cases , ώστε να μπορούμε να ξαναγυρίσουμε στην αρχή της Incase. Φτιάχνουμε μια προσωρινή μεταβλητή flag η οποία θα μας δείχνει αν έχει εκτελεστεί κάποιο statement. Αναθέτουμε αρχικά στο flag την τιμή 0 . Κρατάμε το αποτέλεσμα που επιστρέφει η κλήση της condition. Βάζουμε σε μια μεταβλητή CondTrue το true μέρος της προηγούμενης

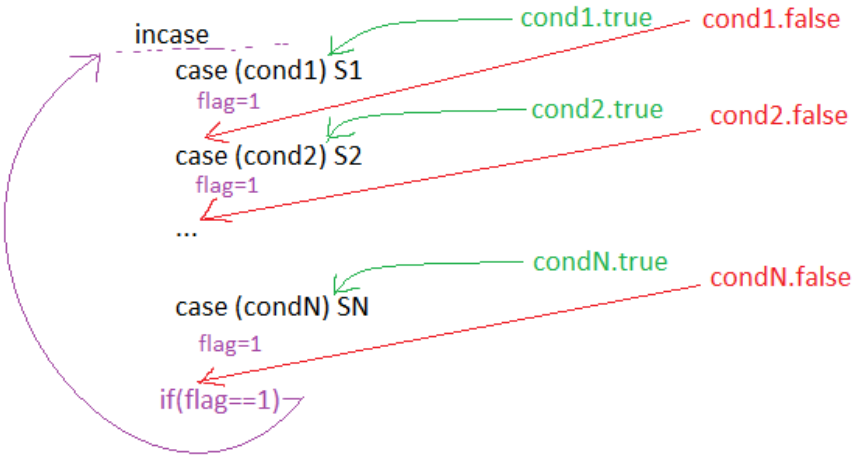
condition και σε μια μεταβλητή CondFalse το false. Κάνουμε backpatch την CondTrue στην αμέσως επόμενη τετράδα , δηλαδή στα statements του αντίστοιχου case. Αφού εκτελεστούν τα statements κάνουμε το flag=1 με τη δημιουργία μιας τετράδας genquad(:=,1,_,flag) και κάνουμε backpatch την CondFalse στην αμέσως επόμενη τετράδα δηλαδή αμέσως μετά τα statements του case που προηγήθηκε και πριν το condition του επόμενου. Μετά από όλα τα cases ελέγχουμε με την τετράδα genquad(=,flag,1,firstCondQuad) αν το flag == 1 και αν είναι πηγαίνουμε στην αρχή της incase στην τετράδα που κρατήσαμε με όνομα firstCondQuad.

```
incase {P1}( (cond) {P2} S1 {P3})*
      default {P4} S2
```

```
{P1} { firstCondQuad = nextquad()
       flag = newtemp()
       genquad(:=,0,_,flag)
  while(case)
       Condition = condition()
       CondTrue = Condition.true
       CondFalse = Condition.false
{P2} { backpatch(CondTrue,nextquad())

{P3} { genquad(:=,1,_,flag)
       backpatch(CondFalse,nextquad())

{P4} { genquad(=,flag,1,firstCondQuad)
```



InputStat(): Αφού διαβάσουμε το input token και ανοίξει η παρένθεση κρατάμε το id του input σε μια μεταβλητή και δημιουργούμε μια τετράδα “inp” με όρισμα αυτό το id.

Muloperator(): Κρατάει τα σύμβολα “*” και “/” σε μια μεταβλητή και την επιστρέφει .

PrintStat(): Αφού διαβάσει το print token και ανοίξει η παρένθεση καλούμε την expression για να μας επιστρέψει την έκφραση που ακολουθεί και να την αποθηκεύσουμε σε μια μεταβλητή. Έπειτα δημιουργούμε μια τετράδα τύπου “out” με όρισμα αυτή την έκφραση που αποθηκεύσαμε.

Program(): Καλείται στην αρχή του προγράμματος μια φορά και αφού διαβάσει το program token αποθηκεύει το όνομα του προγράμματος σε μια μεταβλητή και καλεί την programblock με ορίσματα πλέον το id αυτό και το FLAG=1 (που όπως είπαμε flag =1 έχει μόνο το main πρόγραμμα).

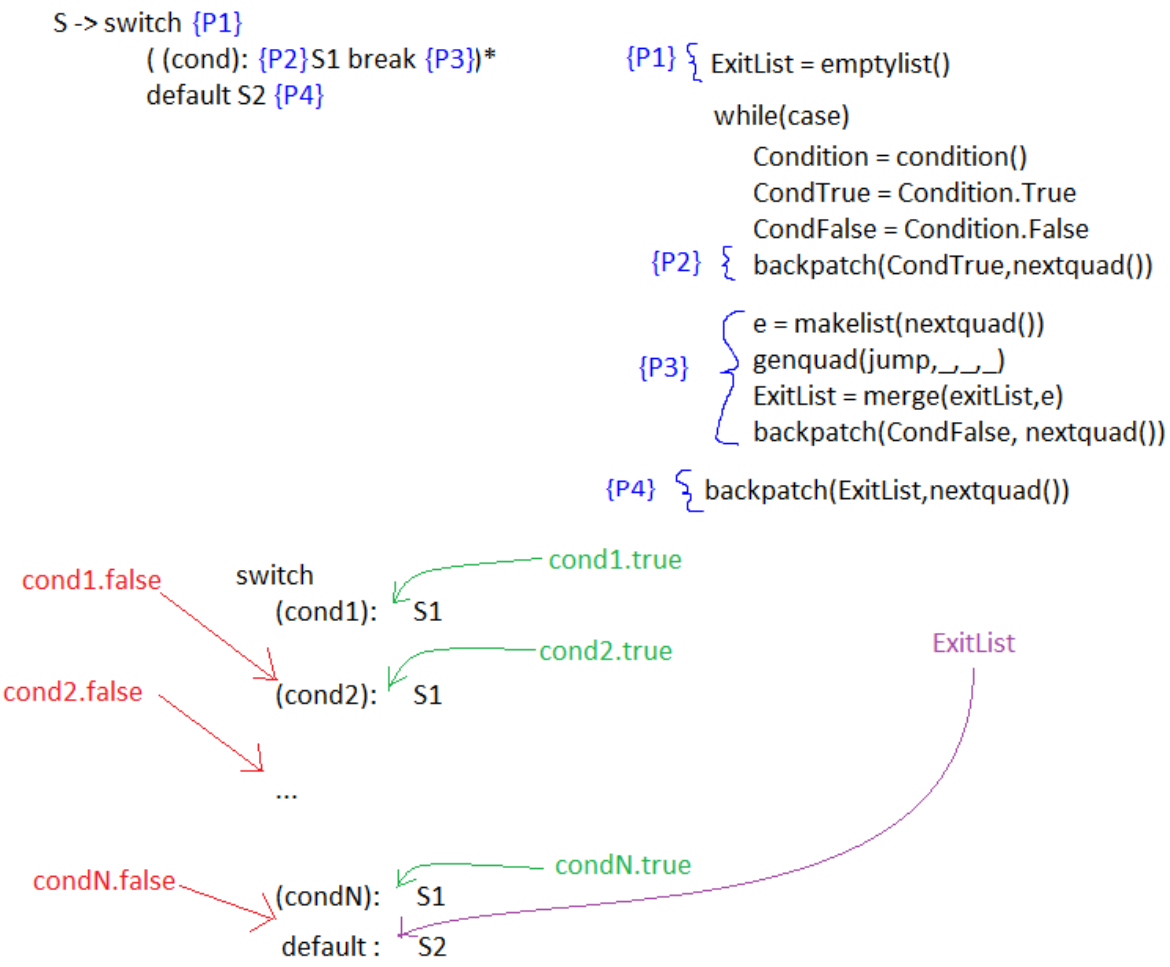
Reloperator(): Αποθηκεύει τον εκάστοτε συγκριτικό τελεστή σε μια μεταβλητή και την επιστρέφει.

ReturnStat(): Αφού διαβάσει το return token και άνοιγμα παρένθεσης καλεί την expression() και κρατάει ότι επιστρέφει αυτή σε μια μεταβλητή. Δημιουργεί μια τετράδα τύπου “retv” με όρισμα αυτό που μόλις αποθηκεύσαμε.

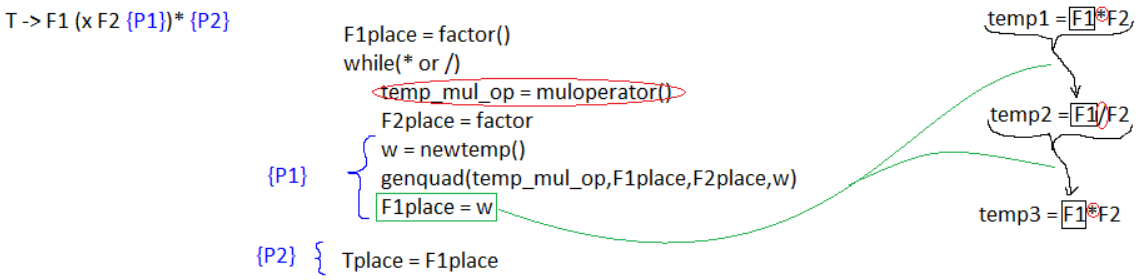
Subprogram(): Αφού διαβάσει function token ή procedure token κρατάει το id που ακολουθεί και μετά το κλείσιμο της παρένθεσης (των ορισμάτων) καλεί την programblock για το id που μόλις κρατήσαμε και με FLAG=0 γιατί δεν πρόκειται ποτέ για τη main.

SwitchcaseStat(): Αν διαβάσει το switchcase token φτιάχνει μια κενή λίστα η οποία θα συμπληρωθεί στο τέλος αφού δεν ξέρουμε ακόμα που θα πηγαίνει. Μόλις κάποιο από τα conditions βρεθεί αληθές ,εκτελεί τα αντίστοιχα statements. Κρατάμε το αποτέλεσμα που επιστρέφει η condition (δηλαδή το που θα πάμε αν βγει αντιστοίχως αληθής ή ψευδής η αποτίμηση) σε μια μεταβλητή με όνομα Condition και αναθέτουμε το true μέρος σε μια μεταβλητή CondTrue και το false σε μια CondFalse. Κάνουμε backpatch την condTrue στην επόμενη τετράδα αφού θέλουμε αν βγει αληθής να εκτελέσει το αντίστοιχο statement. Μετά κρατάμε τον αριθμό της επόμενης τετράδας ,την οποία και δημιουργούμε ώστε να είναι μια κενή jump με τη χρήση της genquad, και βάζουμε αυτόν τον αριθμό στην ExitList με την χρήση της merge. Κάνουμε backpatch την CondFalse στην επόμενη τετράδα δηλαδή αμέσως μετά το statements του case που προηγήθηκε και αμέσως πριν το condition του επόμενου. Τέλος κάνουμε backpatch την ExitList ,η οποία περιέχει τα jump από όλα τα cases, στην επόμενη τετράδα. Με αυτόν τον τρόπο αν δεν εκτελεστεί

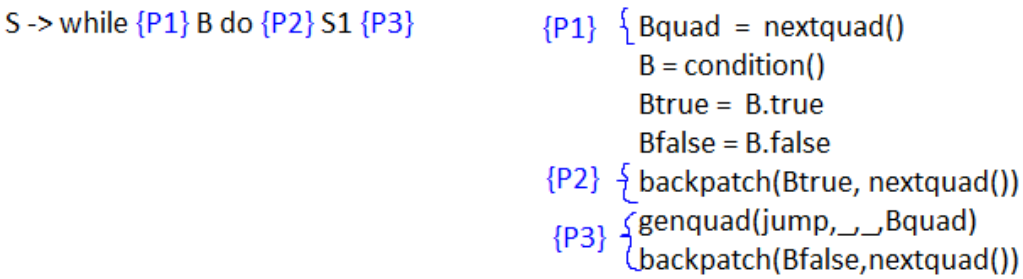
κανένα από τα cases που προηγήθηκαν εκτελείται μια φορά το statements αμέσως μετά την default και βγαίνουμε από την switchcase.

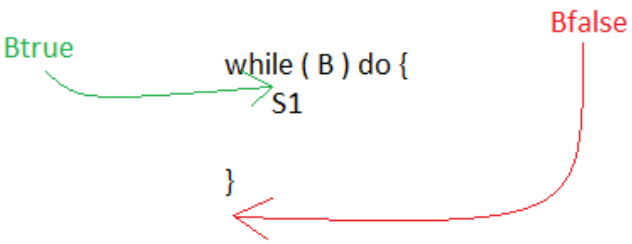


Term(): Κρατάει τον πρώτο όρο του γινομένου σε μια μεταβλητή F1 και όσο διαβάζει "*" ή "/" κρατάει σε μια μεταβλητή temp_mul_operator το σύμβολο του πολλαπλασιασμού ή της διαίρεσης , τον 2ο όρο στην μεταβλητή F2 , δημιουργεί μια καινούρια προσωρινή μεταβλητή temp και κάνει μια τετράδα για την πράξη temp = F1 'temp_mul_operator' F2 (genquad(temp_mul_operator,F1,F2,temp). Επιστρέφει το αποτέλεσμα όλου του γινομένου/διαίρεσης.



WhileStat(): Αφού διαβάσει το while token κρατάει τον αριθμό της επόμενης τετράδας σε μια μεταβλητή Bquad (δηλαδή πριν την while) , κρατάει ότι επιστρέφει η κλήση της condition σε μια μεταβλητή B και αναθέτει το true μέρος της σε μια μεταβλητή Btrue και το false σε μια Bfalse και κάνει backpatch την Btrue στην επόμενη τετράδα (δηλαδή αν είναι true η συνθήκη στέλνει την ροή στη πρώτη τετράδα του block της while). Με το που κλείσει η παρένθεση της condition κάνουμε μια τετράδα “jump” η οποία θα μας στείλει στην μεταβλητή που κρατήσαμε στην αρχή (που περιέχει τον αριθμό της τετράδας που αρχίζει η while[πριν το condition]) . Τέλος κάνουμε backpatch την Bfalse στην επόμενη τετράδα (δηλαδή αν δεν είναι αληθής η συνθήκη στην while να στείλει την ροή αμέσως μετά από τη τελευταία τετράδα του block της while).





Δημιουργία Cfile και IntFile:

Print_listOfAllQuads(): Ανοίγει ένα αρχείο με το όνομα test.int για γράψιμο και για κάθε μια τετράδα της global λίστας all_quads κάνει print μέσα σε αυτό (και στο terminal) τα 5 στοιχεία της κάθε 4αδας (5 γιατί έχει και το unique num)

CreateCFile() : Δημιουργεί ένα αρχείο test.c για γράψιμο . Γραφεί την main που είναι απαραίτητη σε όλα τα .c αρχεία . Καλεί την syntax_analyzer() , μετά την print_listOfAllQuads() για να δημιουργηθεί το αρχείο .int και μετά καλεί και την c για να γράψει μέσα στο αρχείο .c.

C (): Έχει 2 βασικές συναρτήσεις την printSemicolon() και την printLine()

PrintSemicolon(): κάνει write στο αρχείο c που δίνεται ως όρισμα ένα ελληνικό ερωτηματικό και ακολουθεί με \n και \t αναλόγως με το πόσα θα δοθούν ως όρισμα.

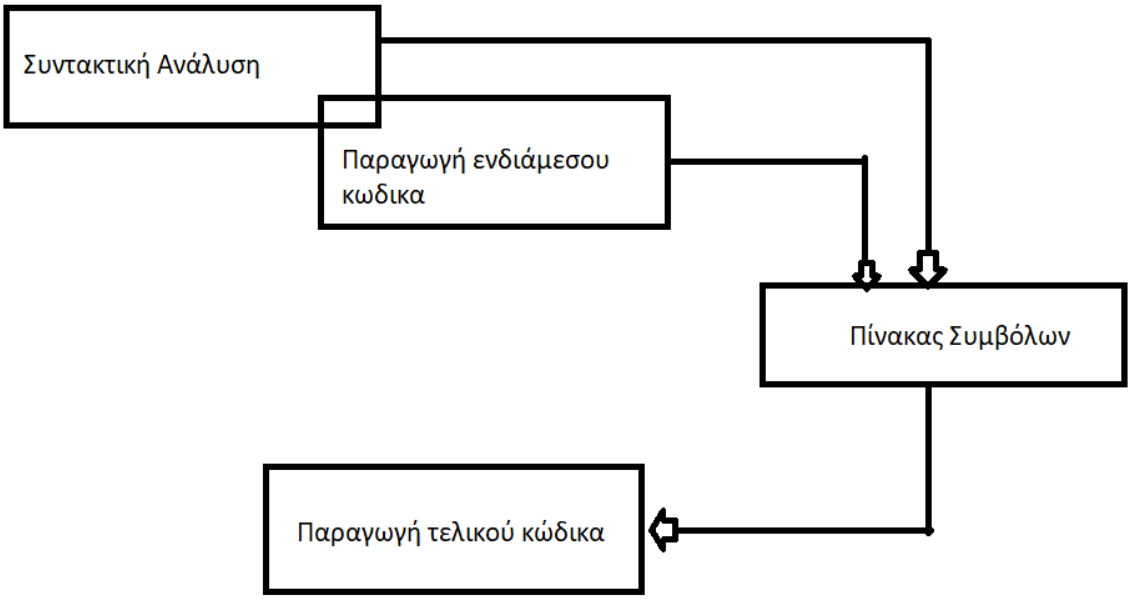
PrintLine(): γράφει στο αρχείο που δίνεται ως όρισμα “L_”+ αριθμό γραμμής+”: ”

Η c για κάθε ένα από τα στοιχεία της global μεταβλητής listOfDeclarations κρατάει το μέγεθος του εκάστοτε στοιχείου της σε μια μεταβλητή QuantityOfDeclarations και αν υπάρχει έστω και ένα στοιχείο μέσα στην listOfDeclarations (άρα το Quantity είναι >0) γράφει στο .c αρχείο την λέξη int και για κάθε μια μεταβλητή μέσα στο listOfDeclarations το όνομα της ακολουθούμενο από ένα κόμμα. Το ίδιο κάνει για την listOfTempVars. Έπειτα πηγαίνει στην λίστα με όλες τις τετράδες και για κάθε μια από αυτές ανάλογα με το σύμβολο με το οποίο ξεκινάνε γράφει την αντίστοιχη εντολή στην C .

Πίνακας συμβόλων

Η πληροφορία που σχετίζεται με τα συμβολικά ονόματα που χρησιμοποιούνται στο πρόγραμμα που μεταγλωττίζουμε αποθηκεύεται σε μια δυναμική δομή , τον πίνακα συμβόλων. Η δομή αυτή παρακολουθεί την εξέλιξη της μεταγλώττισής και προσθέτει ή αφαιρεί πληροφορία από και σε αυτήν ώστε κάθε στιγμή η διαδικασία της μεταγλώττισής να περιέχει μόνο τις εγγραφές στις οποίες το υπό μεταγλώττιση πρόγραμμα έχει δικαίωμα να έχει πρόσβαση τη συγκεκριμένη στιγμή με βάση τους κανόνες εμβέλειας της γλώσσας. Σε έναν πίνακα συμβόλων αποθηκεύεται οποιαδήποτε πληροφορία σχετίζεται με τις διαδικασίες και τις συναρτήσεις , τις παραμέτρους , τα ονόματα των σταθερών

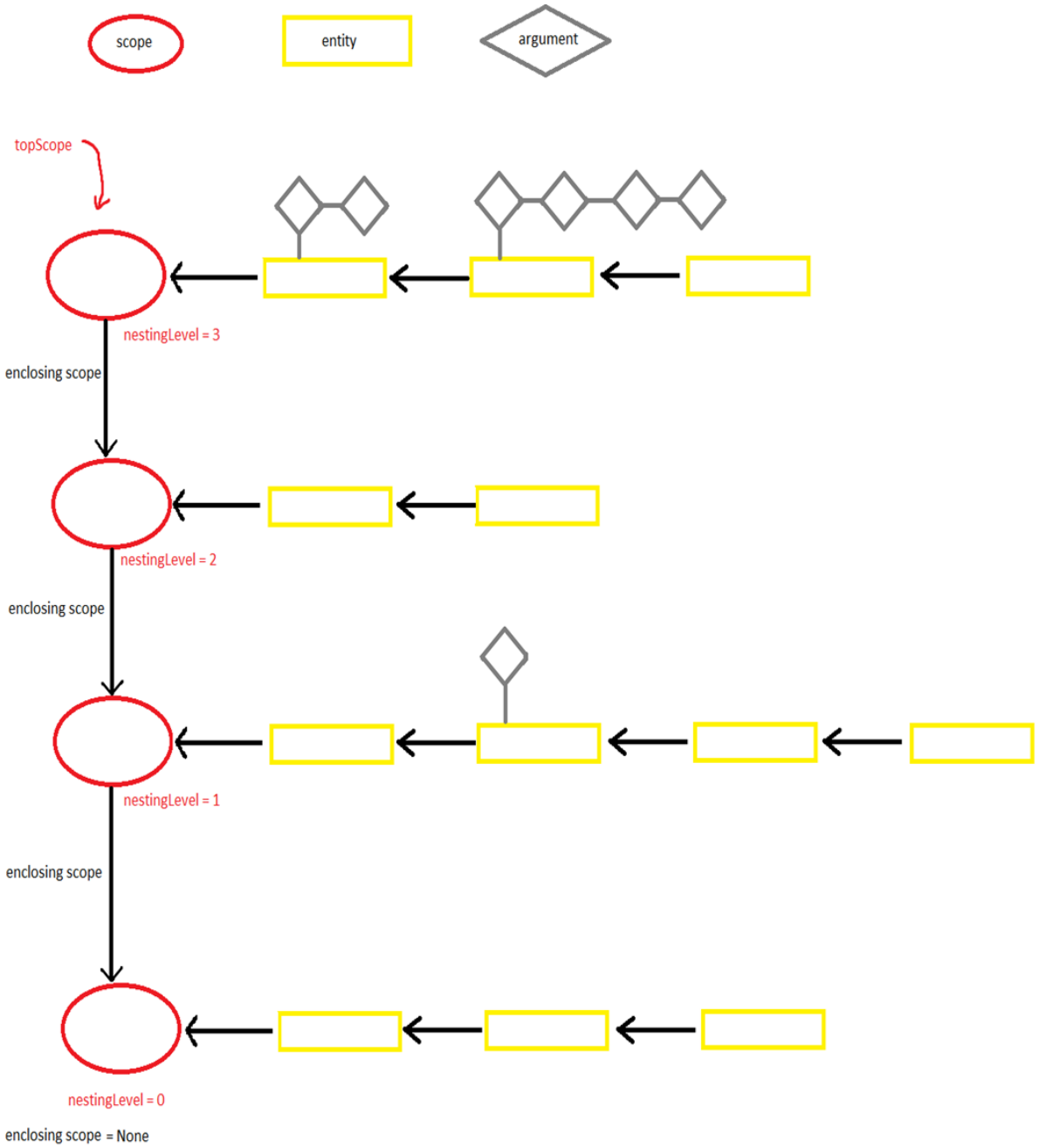
και τις μεταβλητές του προγράμματος. Όταν λέμε αποθηκεύεται πληροφορία εννοούμε ότι κρατάμε σε έναν πίνακα πληροφορία για τα συμβολικά ονόματα που εμφανίζονται στο πρόγραμμα. Επίσης για κάθε ένα στοιχείο του οποίου κρατάμε την πληροφορία υπάρχει διαφορετική εγγραφή στον πίνακα ανάλογα με το είδος του συμβολικού ονόματος όπως είναι λογικό. Η πληροφορία αυτή είναι διαθέσιμη για τη φάση της παραγωγής του τελικού κώδικα. Βέβαια πέραν του τελικού κώδικα , ο πίνακας συμβόλων παρέχει την πληροφορία που απαιτείται για τη σημασιολογική ανάλυση. Με τον όρο σημασιολογική ανάλυση αναφερόμαστε στη διαδικασία εκτέλεσης σημασιολογικών ελέγχων , δηλαδή ότι το πρόγραμμα είναι τόσο λεκτικά, όπως για παράδειγμα ότι τα αναγνωριστικά έχουν έγκυρα ονόματα , όσο και συντακτικά, όπως ότι οι δηλώσεις έχουν τη σωστή δομή , καλώς διαμορφωμένο. Έτσι, ένα μέρος της σημασιολογικής ανάλυσης υλοποιείται μέσα στον πίνακα συμβόλων. Η κύρια λειτουργία του πίνακα συμβόλων είναι ότι διαθέτει την πληροφορία που έχει συλλέξει από τις φάσεις της συντακτικής ανάλυσης και της παραγωγής ενδιάμεσου κώδικα για την παραγωγή του τελικού κώδικα. Στον τελικό κώδικα δεν καλούμε συνάρτηση που ανήκει στον ενδιάμεσο κώδικα και γι’ αυτό δεν υπάρχει βελάκι από την παραγωγή ενδιάμεσου κώδικα απευθείας στην παραγωγή τελικού κώδικα.



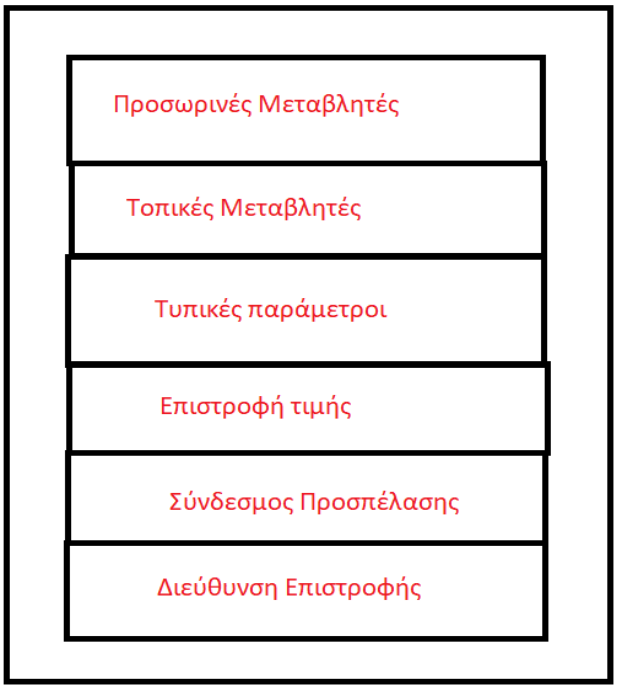
Αναφορικά με τις εγγραφές στον πίνακα συμβόλων , είναι απαραίτητο να κρατάμε όλη τη πληροφορία του αντικειμένου που εγγράφεται. Αυτό σημαίνει πως για παράδειγμα σε μια εγγραφή που αντιστοιχίζεται σε μια μεταβλητή θα αποθηκεύσουμε πέρα από το όνομα της και τον τύπο της καθώς έχει σημασία

Μια ακόμα πολύ σημαντική πληροφορία για κάθε μεταβλητή είναι η θέση στην οποία θα βρίσκεται στην μνήμη και αυτή η πληροφορία θα αναζητηθεί στον πίνακα συμβόλων κατά την παραγωγή τελικού κώδικα. Όμως το ποια ακριβώς θέση θα κρατάμε για κάθε μεταβλητή το βρίσκουμε με το εγγράφημα δραστηριοποίησης της συνάρτησης ή της διαδικασίας στην οποία υπάρχει αυτή η μεταβλητή. Το εγγράφημα δραστηριοποίησης ο χώρος που δεσμεύει μια συνάρτηση ή μια διαδικασία για να τοποθετήσει τα δεδομένα της στη μνήμη , ουσιαστικά υπολογίζεται η απόσταση της μεταβλητής από την αρχή του πίνακα συμβόλων. Σε αυτό το σημείο ορίζουμε την κλάση Variable με την οποία μπορούμε να εισάγουμε εγγραφές στον πίνακα συμβόλων και το τι πεδία θα περιέχει αυτή η κλάση εξηγείται περαιτέρω στη συνέχεια όπως φαίνεται και από το δεύτερο σχήμα που ακολουθεί. . Για μια διαδικασία θα πρέπει να αποθηκεύσουμε τη θέση της μνήμης στην οποία είναι αποθηκευμένη η διαδικασία και για μια παράμετρο θα πρέπει να αποθηκεύσουμε τον τύπο περάσματος της παραμέτρου δηλαδή αν γίνεται πέρασμα με τιμή ή με αναφορά. Για την παράμετρο, το πεδίο στο οποίο θα δώσουμε μεγαλύτερη βάση μιας και διαφέρει από την μεταβλητή είναι ο τύπος περάσματος της ο οποίος μπορεί να παίρνει τις τιμές cv(πέρασμα με τιμή) και ref(πέρασμα με αναφορά). Γι’ αυτό φτιάχνουμε μια υποκλάση με όνομα Parameter που έχει 2 πεδία το parMode για να κρατάει το CV/REF και το offset. Με το offset μπορούμε να γνωρίζουμε την απόσταση της παραμέτρου από την αρχή του εγγραφήματος δραστηριοποίησης προκειμένου να μπορούμε να βρούμε την θέση της παραμέτρου στη μνήμη. Οι δυο τύποι εγγραφής στον πίνακα συμβόλων που έμεινε να σχολιάσουμε είναι η συνάρτηση και η διαδικασία. Με αυτές τις δυο , μπορούμε να καταλάβουμε εάν υπάρχει κάποιο υποπρόγραμμα στον κώδικα. Γι’ αυτόν τον λόγο φτιάχνουμε μια υποκλάση με όνομα subprogram που έχει ένα πεδίο type που υποδηλώνει αν είναι function ή procedure. Επίσης χρειαζόμαστε και ένα πεδίο startingQuad όπου αποθηκεύουμε την ετικέτα της πρώτης εκτελέσιμης τετράδας στην οποία πρέπει η καλούσα συνάρτηση να κάνει άλμα προκειμένου να ξεκινήσει η εκτέλεση της κληθείσας , πράγμα που συμβαίνει στον ενδιάμεσο κώδικα. Το τελευταίο κοινό πεδίο μεταξύ μιας συνάρτησης και μιας διαδικασίας είναι το framelength. Το framelength κρατάει το μήκος του εγγραφήματος δραστηριοποίησης. Στη συνέχεια ορίζουμε άλλη μια κλάση την TemporaryVariable η οποία δεν είναι απαραίτητο να υπάρχει μιας και όσον αφορά τη λειτουργικότητα της δεν διαφέρει από τη Variable. Ο πίνακας συμβόλων αποτελείται από επίπεδα τα οποία ονομάζουμε scope. Scope

σημαίνει εμβέλεια και κάθε επίπεδο αντιστοιχεί στη μετάφραση μιας συνάρτησης. Αυτό σημαίνει πως όταν ξεκινάει για παράδειγμα μια μετάφραση δημιουργείται και ένα νέο επίπεδο scope και αντίστοιχα όταν τερματίζεται η μετάφραση μιας συνάρτησης , τότε αφαιρείται το επίπεδο της από τον πίνακα συμβόλων. Επίσης σε περίπτωση που έχουμε φωλιασμένες συναρτήσεις τότε δημιουργείται επίπεδο και για αυτές. Με το μηχανισμό της προσθαφαίρεσης επίπεδων επιτυγχάνουμε να κρατάμε μέσα στον πίνακα συμβόλων κάθε στιγμή μόνο την πληροφορία που πρέπει με βάση τους κανόνες της γλώσσας. Η δομή του πίνακα συμβόλων απεικονίζεται στο παρακάτω σχήμα.



Το επόμενο πράγμα που θα αναλύσουμε λίγο καλυτέρα είναι το εγγράφημα δραστηριοποίησης. Στο εγγράφημα δραστηριοποίησης , τοποθετούνται πολύ σημαντικές πληροφορίες για την εκτέλεση μιας συνάρτησης , όπως οι πραγματικές παράμετροι , οι τοπικές μεταβλητές και οι προσωρινές μεταβλητές. Η πρώτη θέση στο εγγράφημα αυτό καταλαμβάνεται από τη διεύθυνση επιστροφής της συνάρτησης και έχει κατάλληλο μέγεθος ώστε να αποθηκεύσουμε μια διεύθυνση στη μνήμη. Στη περίπτωση μας , κάθε θέση στο εγγράφημα δραστηριοποίησης θα αποτελείται από 4 bytes μιας και ο μοναδικός τύπος της Cimple , ο integer , είναι 4 bytes. Στη δεύτερη θέση του εγγραφήματος δραστηριοποίησης τοποθετείται ο σύνδεσμος προσπέλασης που είναι ουσιαστικά ένας δείκτης ο οποίος δείχνει στο εγγράφημα δραστηριοποίησης στο οποίο πρέπει να αναζητήσει η συνάρτηση μια μεταβλητή ή παράμετρο που δεν της ανήκει αλλά έχει πρόσβαση σε αυτή σύμφωνα με τους κανόνες εμβέλειας της γλώσσας. Στη Cimple κάθε συνάρτηση ή διαδικασία έχει πρόσβαση στις δίκες της παραμέτρους και μεταβλητές , στις τοπικές μεταβλητές και παραμέτρους του γονέα της και σε κάθε άλλη συνάρτηση που ανήκει στο γενεαλογικό δέντρο της . Η τρίτη θέση του εγγραφήματος δραστηριοποίησης είναι αυτή στην οποία δεσμεύουμε χώρο για να αποθηκευτεί η διεύθυνση της μεταβλητής στην οποία θα επιστραφεί η τιμή της συνάρτησης. Μέχρι στιγμής στις τρεις θέσεις, έχουμε αποθηκεύσει τις τρεις διευθύνσεις, δηλαδή συνολικά 12 bytes. Μετά θα τοποθετήσουμε τις παραμέτρους. Για κάθε μια παράμετρο δεσμεύουμε 4 bytes και αναλόγως ποιο θα είναι το πέρασμα της , στη θέση που είναι δεσμευμένη για την παράμετρο θα τοποθετηθεί η τιμή της ή η διεύθυνση της. Οι παράμετροι τοποθετούνται στη στοίβα με τη σειρά εμφάνισής τους στις τυπικές παραμέτρους της συνάρτησης ή της διαδικασίας. Τα ίδια ισχύουν και στη δέσμευση χώρου για τις μεταβλητές οι οποίες τοποθετούνται αμέσως μετά τις παραμέτρους. Η τελευταία ομάδα που λαμβάνει χώρο στην στοίβα είναι οι προσωρινές μεταβλητές όπου κι αυτές ανεξαρτήτως που δημιουργούνται χρειάζονται χώρο για να αποθηκευτεί η τιμή τους. Τις χειριζόμαστε όπως και τις τοπικές μεταβλητές. Σχηματικά η δομή του εγγραφήματος δραστηριοποίησης είναι η ακόλουθη.

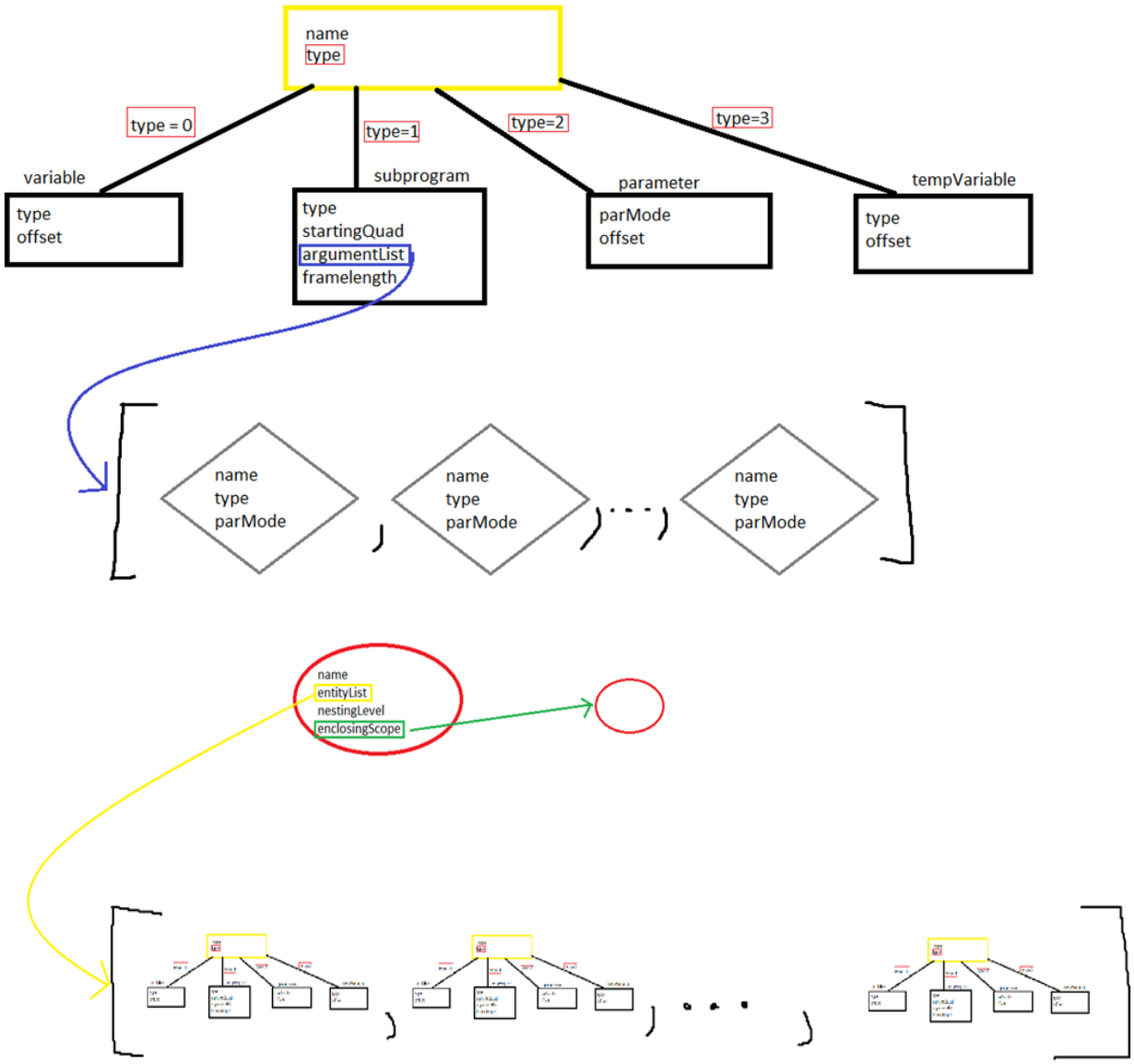


Ένας άλλος όρος που θα μας χρειαστεί στη συνέχεια είναι το μήκος του εγγραφήματος δραστηριοποίησης , το οποίο πρόκειται για τον συνολικό χώρο σε bytes που καταλαμβάνει το εγγράφημα δραστηριοποίησης στη στοίβα. Το μήκος αυτό γίνεται γνωστό αφότου τελειώσει η μετάφραση του ενδιάμεσου κώδικα για ένα υποπρόγραμμα καθώς όπως είναι λογικό πρέπει πρώτα να γνωρίζουμε τον αριθμό των παραμέτρων , των μεταβλητών και των προσωρινών μεταβλητών μιας συνάρτησης ή διαδικασίας. Επίσης να εξηγήσουμε και τον όρο offset ο οποίος έχει αναφερθεί και παραπάνω και είναι ουσιαστικά η απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης. Ας δώσουμε ένα παράδειγμα για να γίνει πιο ξεκάθαρο το εγγράφημα δραστηριοποίησης. Έχουμε μια συνάρτηση με 3 παραμέτρους , 1 τοπική μεταβλητή και 1 προσωρινή , τα offset που αντιστοιχούν στις οντότητες αυτές είναι 12,16,20,24 και 28. Το μήκος του εγγραφήματος δραστηριοποίησης είναι 32 και ξεκινάμε από τα 12 διότι τα 12 πρώτα bytes είναι δεσμευμένα για τη διεύθυνση επιστροφής, τον σύνδεσμο προσπέλασης και την επιστροφή τιμής.

Το ζητούμενο όπως έχουμε προαναφέρει είναι σε κάθε σημείο του τελικού κώδικα ο πίνακας συμβόλων να περιέχει ακριβώς εκείνες τις εγγραφές τις οποίες το πρόγραμμα που μεταφράζεται έχει δικαίωμα να προσπελάσει εκείνη τη συγκεκριμένη στιγμή με βάση του κανόνες της πηγαίας γλώσσας. Τώρα θα δούμε τι ενέργειες κάνει ένας πίνακας συμβόλων, καθώς προσθαφαιρούμε πληροφορία. Ένας πίνακας συμβόλων κάνει πρόσθεση νέας εγγραφής . Αυτή η πρόσθεση γίνεται όταν συναντάμε διαδικασίες όπως είναι η δήλωση

μεταβλητής, συνάρτησης ή διαδικασίας , παραμέτρου ή όταν δημιουργούμε μια νέα προσωρινή μεταβλητή. Η νέα εγγραφή που δημιουργείται προστίθεται στο ανώτερο επίπεδο του πίνακα συμβόλων , «πάνω-πάνω» , στο επίπεδο δηλαδή της συνάρτησης ή της διαδικασίας που μεταφράζεται τη στιγμή αυτή. Επίσης μπορεί να προσθέσει νέο επίπεδο, πράγμα που γίνεται με την εκκίνηση της μετάφρασης μια συνάρτησης ή διαδικασίας ή του κυρίως προγράμματος. Το νέο επίπεδο όπως είναι λογικό, δημιουργείται πάνω από το τελευταίο επίπεδο που έχει δημιουργηθεί μέχρι στιγμης. Απο τη στιγμή που ο πίνακας συμβόλων μπορεί να προσθέσει νέο επίπεδο, περιμένουμε ότι σαν λειτουργια θα έχει και την αφαίρεση του. Όντως η αφαίρεση ενός επιπέδου γίνεται όταν ολοκληρωνεται η μετάφραση μιας συνάρτησης ή διαδικασίας ή το κυρίως προγράμματος. Αφαιρειται όλο το επίπεδο μαζι με όλες τις εγγραφές του. Επιπλέον ο πίνακας συμβόλων ενημερώνει τα πεδία , όπως για παράδειγμα το framelength και το startingQuad μιας και κατά τη δημιουργία της εγγραφής δεν ήταν διαθέσιμη η πληροφορία για αυτά τα πεδία ακόμη. Μια τελευταία λειτουργια του πίνακα συμβόλων είναι η αναζήτηση μιας εγγραφής, η οποία γίνεται με το όνομα της εγγραφής. Η αναζήτηση ξεκινάει από το υψηλότερο επίπεδο και να δεν βρεθεί το όνομα που ψάχνει συνεχίζει σε χαμηλότερα ώσπου είτε να βρεθεί η ζητούμενη εγγραφή είτε να εξαντληθούν όλα τα επίπεδα στον πίνακα συμβόλων και να αναγνωριστεί σφάλμα.

Στον πίνακα συμβόλων μας , η κλάση Entity έχει τοποθετηθεί υψηλοτέρα στην ιεραρχία(σε σχέση με τις Variable, subprogram, parameter, tempVariable) ώστε να ενοποιήσει εννοιολογικά όλες τις εγγραφές. Απο την Entity κληρονομεί η Variable καθώς και οι άλλες υποκλάσεις. Με την σειρά της η κλάση parameter κληρονομεί από την κλάση entity. Οι προσωρινές μεταβλητές που αποφασίσαμε ότι θα έχουν δική τους κλάση TemporaryVariable θα κληρονομούν από την κλάση Variable.



Για τον πίνακα συμβόλων δημιουργήσαμε 3 classes μια για κάθε ένα αντικείμενο του πίνακα συμβόλων. Η κλάση scope είναι για τα επίπεδα του πίνακα συμβόλων , η κλάση entity είναι για τις εγγραφές - οντότητες και η argument είναι για τις τυπικές παραμέτρους.

class argument: Κρατάει 3 πεδία ανά argument ένα που περιέχει το όνομα της τυπικής παραμέτρου, ένα για τον τύπο της (δεν είναι απαραίτητο αφού έχουμε μόνο integers) και τέλος ένα πεδίο για τον τρόπο με τον οποίο «δίνεται» η μεταβλητή δηλαδή με τιμή ή με αναφορά.

class entity: Για κάθε entity κρατάει 2 πεδία, το όνομα του entity και τον τύπο του (μεταβλητή, υποπρόγραμμα, παράμετρος, προσωρινή μεταβλητή).Επιπλέον για κάθε αντικείμενο κλάσης “entity” αναλόγως με τον τύπο του ορίζονται και 4 υποκλάσεις μια για κάθε τύπο που αναφέραμε πιο πάνω. Η κάθε υποκλάση έχει τα δικά της πεδία οπότε εν τέλει ένα αντικείμενο τύπου entity έχει από 2 έως 6 πεδία ανάλογα με τον τύπο του. Στην πραγματικότητα κάθε αντικείμενο τύπου entity έχει 2 (+2+4+2+2) = 12 πεδία όμως δεν τα αξιοποιούμε όλα αφού οπότε χρειάζεται να αλλάξουμε κάποιο από τα πεδία ενός συγκεκριμένου entity ελέγχουμε πρώτα το πεδίο type του για να δούμε ποια δεδομένα να αλλάξουμε. Για παράδειγμα αν myEntity.type == 0 (variable) τότε κάνουμε myEntity.variable.offset = 16 ενώ αν myEntity.type == 3 κάνουμε myEntity.tempVariable.offset = 16.

class scope: Ένα αντικείμενο τύπου scope έχει ένα πεδίο για το όνομα του έναν πίνακα στον οποίο αποθηκεύει τις οντότητες που βρίσκονται σε αυτό το επίπεδο, ένα πεδίο να κρατάει το επίπεδο στο οποίο βρισκόμαστε (αριθμό) και τέλος έναν δείκτη σε ένα άλλο scope όπου ουσιαστικά μας δείχνει σε ποιο scope μέσα βρίσκεται το τρέχον scope (πρακτικά ποιος είναι ο «πατέρας»)

new_argument (): πηγαίνει στο τρέχον top Scope στην λίστα με τις οντότητες του στην υποκλάση subprogram στη λίστα με τα arguments και προσθέτει στο τέλος το αντικείμενο που της δίνεται ως όρισμα.

new_entity () : πηγαίνει στο top Scope στην λίστα του με τις οντότητες και προσθέτει την οντότητα που της δίνεται ως όρισμα.

newScope() : παίρνει ως όρισμα ένα όνομα . Δημιουργεί ένα καινούριο Scope με αυτό το όνομα , βάζει ως «πατέρα» αυτού του scope το μέχρι στιγμής top scope , αν το μέχρι στιγμής top scope δεν υπάρχει (είναι None/Null) τότε δίνει στο scope που μόλις δημιουργήσαμε το επίπεδο 0 αλλιώς δίνει ότι είχε ο «πατέρας» συν 1. Τέλος κάνει το scope που μόλις δημιουργήσαμε το top Scope

deleteScope() :κάνει top scope τον πατέρα του τρέχοντος top scope (Ουσιαστικά δεν καταστρέφει το αντικείμενο top Scope αλλά αλλάζοντας τον δείκτη που δείχνει σε αυτό το κάνει να χαθεί στην μνήμη)

addScopeToListOfAllScopes() : Βάζει το τρέχον topScope σε μια λίστα που περιέχει όλα τα scopes ανεξαρτήτως του nesting Level τους. Μπορεί παραπάνω από ένα scopes μέσα σε εκείνη την λίστα να έχουν το ίδιο nesting level.

calcStartingQuad() : Πάει στο scope που περιέχει το τρέχον top Scope (δηλαδή τον πατέρα του) στο τελευταίο στοιχείο της λίστας με τις οντότητες που έχει, στο πεδίο startingQuad της υποκλάσης subprogram της οντότητας entity και δίνει στο πεδίο starting quad την τιμή της επόμενης τετράδας.

calculateOffset() : Όλα τα entities ενός scope έχουν τουλάχιστον 12 θέσεις στην μνήμη offset οπότε κρατάμε την τιμή 12 ως την αρχή. Μετράμε πόσες οντότητες τύπου int parameter ή temp υπάρχουν σε αυτό το scope και για καθεμιά από αυτές αυξάνουμε το offset κατά 4 και τέλος επιστρέφουμε την τιμή του.

calcFramelength() : Πηγαίνουμε στο «πατέρα» του τρέχοντος top Scope , στο τελευταίο στοιχείο της λίστας οντοτήτων του και αφού ξέρουμε ότι είναι subprogram (γιατί μόνο με την κλήση subprogram (Function ή procedure) θα αλλάζαμε Scope , πηγαίνουμε στην υποκλάση subprogram του τελευταίου αντικειμένου entity και συγκεκριμένα στο πεδίο framelength και του δίνουμε την τιμή που μας επιστρέφει η κλήση της calculateOffset().

addParameters() : για κάθε ένα argument στο scope “πατέρα” του τρέχοντος top Scope , στο τελευταίο entity στη λίστα του με τα entities στην υποκλάση αυτού του αντικειμένου ,subprogram , στην λίστα με τα arguments , δημιουργούμε μια οντότητα που έχει το όνομα αυτού του argument και ως τύπο = 2 (parameter) και βάζουμε στα πεδία της υποκλάσης parameter τον τύπο περάσματος (0 cv , 1 ref) και το offset του με τη χρήση της συνάρτησης calculateOffset.

printSymbTable() : Η συνάρτηση αυτή αρχικά χρησιμοποιήθηκε για να κάνουμε print στο terminal τα αποτελέσματα του πίνακα συμβόλων και αργότερα δώσαμε ως όρισμα ένα αρχείο symbFile στο οποίο γράφτηκαν τα ίδια lines που γράφονταν στο terminal. Για κάθε ένα scope από το Top Scope μέχρι το scope με nestingLevel =0 για κάθε μια οντότητα του κάθε scope αναλόγως με το τι είναι κάνει print τα πεδία του και αφού το κάνει αυτό για κάθε οντότητα πηγαίνει στο scope «πατέρα».

Οι συναρτήσεις της syntaxAnalyzer που αλλάζουν

Οι παρακάτω συναρτήσεις έχουν όσες λειτουργίες είχαν και πριν (στον ενδιάμεσο κώδικα) απλά πλέον καλούν και κάποιες συναρτήσεις του πίνακα συμβόλων.

programblock(): Αφού κληθεί η programblock καλεί την newScope για να δημιουργήσει ένα νέο scope αφού κάθε συνάρτηση έχει το δικό της . Αν το blockFlag (το οποίο είχαμε αναφέρει ότι όταν είναι 1 συμβολίζει την κλήση της main) είναι 0 τότε καλούμε την addparameters() για τις παραμέτρους που ίσως να ακολουθούν την κλήση της συνάρτησης. Αν πάλι δεν βρισκόμαστε στην main τότε καλούμε και την calcStartingQuad για να επιστρέψει τον αριθμό της επόμενης τετράδας στην function/procedure από την οποία ξεκινάει αυτό το subprogram. Με το που τελειώσει η συνάρτηση αν έχουμε BlockFlag!=1 δηλαδή δεν έχουμε την main , καλούμε την calcFramelength() για να υπολογίσει το framelength της συνάρτησης που προηγήθηκε. Τέλος κάνουμε print τον πίνακα (στο terminal και στο αρχείο) , καλούμε την addScopeToListOfAllScopes() (θα μας χρησιμεύσει στον τελικό κώδικα) και τέλος κάνουμε delete το τρέχον top Scope κάνοντας top Scope και δίνοντας τον έλεγχο στον «πατέρα».

formalparitem(): Με το που διαβάσει ένα όρισμα συνάρτησης είτε είναι in (CV) είτε inout (REF) δημιουργεί ένα argument με όνομα ίδιο με το identifier που μόλις αποθηκεύσαμε και θέτει την τιμή parMode σε 0 για CV και 1 για REF (από -1 που είναι default) και καλεί την new_argument για να βάλει αυτό το αντικείμενο τύπου argument που μόλις δημιούργησε στην λίστα με τα arguments του υποπρογράμματος. Αυτό το κάνει για όσα ορίσματα έχουν δοθεί στο υποπρόγραμμα

subprogram (): δημιουργεί μια καινούρια οντότητα με ορίσματα το όνομα της και το 1 (δηλαδή ότι είναι υποπρόγραμμα) αν είναι function τότε βάζουμε στο πεδίο type της υποκλάσης subprogram το 1 (δηλαδή function) ενώ αν είναι procedure το 0.

varlist(): για κάθε identifier που θα διαβάσει δημιουργεί μια καινούρια οντότητα με ορίσματα το όνομα αυτό το identifier και τον αριθμό 0 που συμβολίζει variable. Στο πεδίο offset της υποκλάσης variable δίνουμε τον αριθμό που επιστρέφει η συνάρτηση calculateOffset.

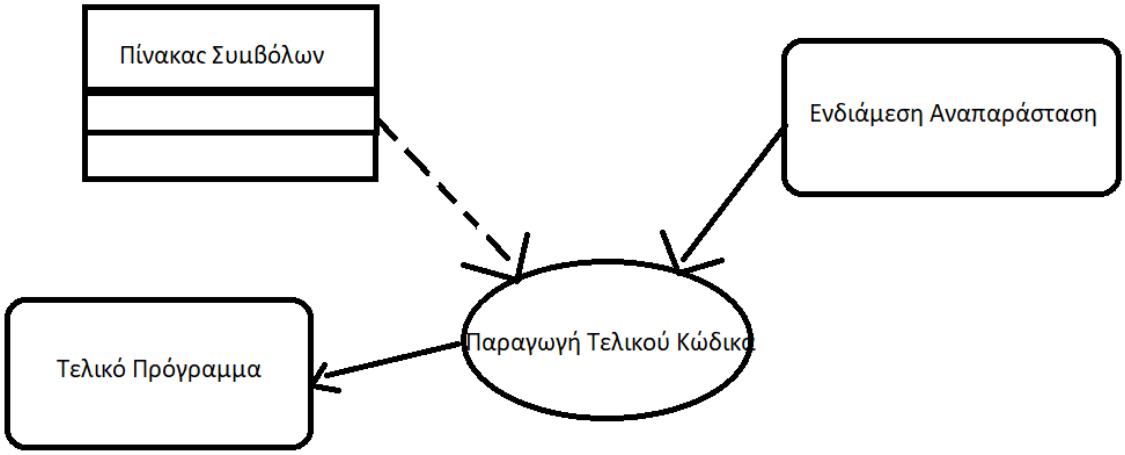
Οι συναρτήσεις του ενδιάμεσου κώδικα που αλλάζουν

newtemp(): κάνει ότι έκανε ήδη απλά πλέον δημιουργεί και ένα entity δίνοντας ως ορίσματα το όνομα της και τον τύπο του entity (=3) γιατί είναι προσωρινή μεταβλητή και βάζει στην υποκλάση tempVariable στο πεδίο offset την τιμή που επιστρέφει η τιμή της calculateOffset(). Τέλος βάζει την καινούρια

οντότητα στην λίστα με τις οντότητες του τρέχοντος topScope με την χρήση της new_entity()

Τελικός κώδικας

Η παραγωγη τελικού κώδικα είναι η τελευταια φάση παραγωγής κώδικα και προκυπτει από τον ενδιαμεσο κώδικα με χρηση του πίνακα συμβολων.Απο τον ενδιαμεσο κώδικα προκυπτει μια σειρά εντολων τελικού κώδικα , η οποία ανακτα πληροφορίες από τον πίνακα συμβολων.Για καλυτερη κατανοηση παραθετουμε το εξής σχήμα που περιγραφει την παραγωγη του τελικού κώδικα στη διαδικασια της μεταγλωττισης.



Η τελική γλώσσα που δημιουργείται από έναν μεταγλωττιστή είναι συνήθως η γλώσσα μηχανής ενός επεξεργαστή. Στη δική μας τη περίπτωση , θα παράγουμε για τη Cimple τελικό κώδικα σε assembly code του επεξεργαστή RISC-V.

Οι καταχωρητες RISC-V είναι οι εξής :

- Ο μηδενικος καταχωρητης(zero), ο οποιος έχει μονιμα την τιμή 0

- Ο δείκτης στοιβας γνωστός και ως stack pointer (sp) ο οποίος σημειωνει την αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης ή της διαδικασίας που κάθε στιγμή εκτελείται.
- Ο δείκτης πλαισίου γνωστός και ως frame pointer (fp) ο οποίος χρησιμοποιείται για να δείξουμε την αρχή ενός εγγραφήματος δραστηριοποίησης τη στιγμή που δημιουργείται.
- Οι προσωρινοί καταχωρητες , temporary registers(t0-t6) οι οποίοι χρησιμεύουν για παρα πολλούς λόγους.
- Οι καταχωρητές διατηρούμενων τιμων , saved registers (s1-s11) , οι οποίοι διατηρουνται ανάμεσα σε κλήσεις συναρτησεων και διαδικασιων.
- Οι καταχωρητές ορισματων συναρτησεων (a0-a7) , οι οποίοι χρησιμοποιούνται για να περαστούν παράμετροι ανάμεσα σε συναρτήσεις ή διαδικασίες.
- Η διεύθυνση επιστροφής ,return address (ra), όπου σε αυτόν τον καταχωρητή αποθηκευεται η διεύθυνση στην οποία πρέπει να επιστρεψει ο έλεγχος του προγράμματος όταν ολοκληρωθει η εκτέλεση μιας συνάρτησης ή μιας διαδικασιας.
- Ο μετρητής προγράμματος, program counter(pc), περιέχει τη διεύθυνση της εντολής που εκτελείται
- Ο καθολικός δείκτης , global pointer (gp), ο οποίος δείχνει στην αρχή των καθολικών μεταβλητων σε ένα πρόγραμμα.

Οι καταχωρητές μπορούν να πάρουν τιμή είτε με απευθείας εκχώρηση αριθμητικής σταθεράς η οποία γίνεται με την ψευδοεντολή li, είτε με μεταφορά δεδομένων από έναν καταχωρητη σε έναν άλλον με την ψευδοεντολή mv , είτε με τέλεση κάποιας αριθμητικής πράξης. Για την τέλεση κάποιας αριθμητικής πράξης υπάρχουν διαθέσιμες οι εξής εντολές : add για την πρόσθεση μεταξύ δυο καταχωρητών, sub για την αφαίρεση δυο καταχωρητών, mul για πολλαπλασιασμό και div για τέλεση της διαίρεσης.

Τώρα ήρθε η ώρα να εξετάσουμε τις εντολές lw και sw με τις οποίες έχουμε πρόσβαση στη μνήμη. Το l προέρχεται από τη λέξη load και συμβολίζει την ανάγνωση , ενώ το s προέρχεται από τη λέξη store και συμβολίζει την εγγραφή, ενώ το w ότι πρόκειται να διαβάσουμε ή να γράψουμε δεδομένα μεγέθους 4 bytes δηλαδή μιας λέξης. Για την πρόσβαση στη μνήμη χρησιμοποιούμε κυρίως τους καταχωρητές stack pointer (sp) , frame pointer (fp) και λαμβάνουμε ως βάση έναν συγκεκριμένο καταχωρητή , μετακινούμαστε κατά offset θέσεις και στο σημείο που μετακινηθήκαμε διαβάζουμε αν πρόκειται για lw ή γράφουμε

αν πρόκειται για sw την τιμή ενός άλλου καταχωρητή. Ένα παράδειγμα σύνταξης είναι το εξής :

- lw reg1, offset(reg2) όπου reg1 είναι ο καταχωρητης προορισμου στον οποιο θα γραψουμε, reg2 ο βασικος καταχωρητης και το offset είναι η απόσταση από τον reg2.
- sw reg1, offset(reg2) όπου reg1 είναι ο καταχωρητής πηγής, reg2 ο βασικός καταχωρητής και το offset είναι η απόσταση από τον reg2.

Ένας άλλος τρόπος πρόσβασης στη μνήμη είναι απευθείας μέσω καταχωρητή χωρίς να δηλωθεί κάποιο offset.

Στη συνέχεια θα αναλύσουμε τις εντολές διακλαδώσεων. Η εντολή για άλμα χωρίς συνθήκη είναι η j , ενώ οι εντολές αλμάτων υπό συνθήκη είναι αυτές με τις οποίες υλοποιούμε τις λογικές παραστάσεις. Οι εντολές αυτές είναι η branch if equal (beq) η οποία ελέγχει για ισότητα καταχωρητών , η branch if not equal(bne) η οποία ελέγχει για το αν δυο καταχωρητές δεν είναι ίσοι , η branch if less than (blt) η οποία ελέγχει για το αν ένας καταχωρητής είναι μικρότερος συγκριτικά με έναν άλλον , η branch greater than (bgt) που ελέγχει αν ένας καταχωρητής είναι μεγαλύτερος από έναν άλλον , η branch if less or equal than(ble) που ελέγχει ουσιαστικά για μικρότερο ή ίσο και τέλος η branch if greater than or equal than(bge) που ελέγχει για μεγαλύτερο ή ίσο μεταξύ δυο καταχωρητών. Ανάλογα με το αποτέλεσμα της σύγκρισης και το είδος της συνθήκης που εκφράζει η εντολή , πραγματοποιείται άλμα στην ετικέτα label ή όχι.

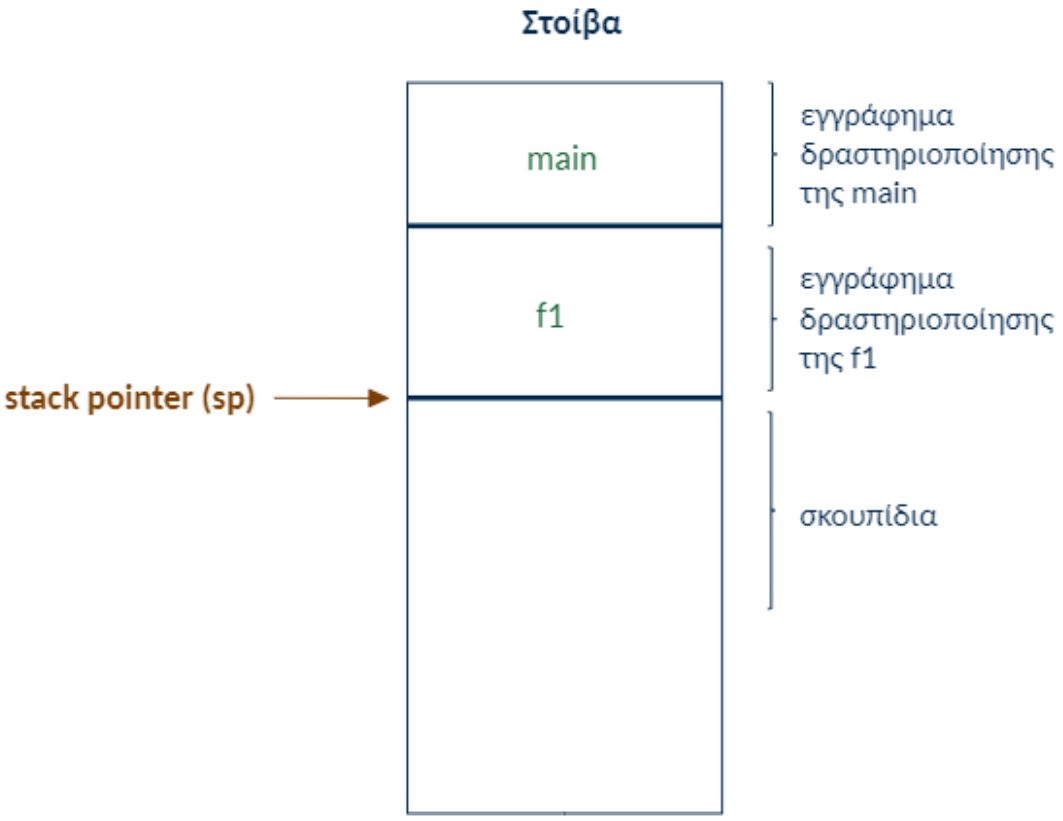
Για την κλήση συνάρτησης ή μιας διαδικασίας υποστηρίζεται η εντολή jal , η οποία παίρνει ως όρισμα μια διεύθυνση και εκτελεί άλμα στη διεύθυνση αυτή. Ταυτόχρονα τοποθετεί στον καταχωρητή return address (ra) τη διεύθυνση εντολής που ακολουθεί την jal στον υπό μετάφραση κώδικα.

Η είσοδος των δεδομένων από το πληκτρολόγιο γίνεται μέσω των καταχωρητών ορισμάτων (a0-a7). Μόλις αυτά τα ορίσματα τοποθετηθούν στους δυο καταχωρητές , καλείται η εντολή ecall ώστε να διαβαστούν τα δεδομένα. Εάν θέλουμε να ορίσουμε ότι πρόκειται να διαβαστεί ακέραιος αριθμός τότε γράφουμε li a7,5 δηλαδή τοποθετούμε την τιμή 5 στον καταχωρητή a7, ενώ αν θέλουμε να εμφανίσουμε στην οθόνη έναν ακέραιο αριθμό τότε τοποθετούμε στον καταχωρητή a7 την τιμή 1 li a7,1 και στον καταχωρητή a0 την τιμή που θέλουμε να εμφανίσουμε. Έπειτα καλούμε την ecall. Επίσης αν θέλουμε να αλλάξουμε γραμμή τότε θα ορίσουμε ένα

συμβολικό όνομα για τον χαρακτήρα αλλαγής γραμμής και στη συνέχεια τοποθετούμε στον καταχωρητή a7 τον αριθμό 4 li a7,4 και στον καταχωρητή a0 το συμβολικό όνομα που δώσαμε.

Για τον τερματισμό ενός προγράμματος χρησιμοποιούμε πάλι τους ιδίους καταχωρητές μόνο που τώρα στον a7 δίνουμε την τιμή 93, ενώ στον a0 τοποθετούμε αυτό που θέλουμε να επιστρέψουμε στο σύστημα ως αποτέλεσμα.

Ο χώρος που δεσμεύεται στη στοίβα αφορά πολύ σημαντικά δεδομένα για τη σωστή λειτουργία κάθε συνάρτησης ή διαδικασίας που εκτελείται , καθώς επίσης αφορά και τιμές ή διευθύνσεις μεταβλητών. Παρακάτω θα δούμε αναλυτικα τη δομή ενός εγγραφήματος δραστηριοποίησης , πού τοποθετείται στη στοίβα , ποια είναι η διάρκεια ζωής του και πως συνεισφέρει στη λειτουργία του προγράμματος. Όταν ξεκινάει η εκτέλεση ενός προγράμματος το λειτουργικό σύστημα δεσμεύει χώρο ο οποίος θα λειτουργήσει σαν στοίβα και εκεί τοποθετείται ο δείκτης στοίβας sp ο οποίος μη ξεχνάμε ότι θα πρέπει ανά πάσα στιγμή να δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης που εκτελείται τη δεδομένη στιγμή. Στη συνέχεια ο δείκτης στοίβας μετατοπίζεται τόσες θέσεις ,όσες θεσεις μνημης θελουμε να καταλάβει το εγγράφηκα δραστηριοποίησης δεσμεύοντας έτσι χώρο για το κυρίως πρόγραμμα. Αν το κυρίως πρόγραμμα καλέσει μια συνάρτηση τότε αυτή θα τοποθετηθεί μετά το εγγράφηκα δραστηριοποίησης του κυρίως προγράμματος. Η δέσμευση του χωρου θα γίνει με τη μετατοπιση του sp και προφανώς το εγγραφημα δραστηριοποίησης του κυρίως προγράμματος θα συνεχισει να υπάρχει στη στοιβα αφού η εκτέλεση του δεν έχει ολοκληρωθεί και θα επιστραφει σε αυτό όταν τελειώσει η εκτέλεση της συνάρτησης που καλέστηκε. Αυτή διαδικασία ακολουθείται για κάθε συνάρτηση που καλείται από μια άλλη. Σε περίπτωση που φτάσουμε σε μια συνάρτηση η οποία δεν καλεί κάποια άλλη τότε ο δείκτης θα επιστρέψει στην προηγούμενη θέση του και ο χώρος που κατείχε η τελευταία συνάρτηση αποδεσμεύεται χωρίς αυτό να σημαίνει ότι τα περιεχόμενα του έχουν σβηστεί. Για τα περιεχόμενα αυτά χρησιμοποιούμε τον όρο σκουπίδια. Όταν ολοκληρωθεί η εκτέλεση του κυρίως προγράμματος τότε και ο τελευταίος χώρος αποδεσμεύεται και στη συνέχεια όλος ο χώρος που καταλάμβανε αυτή τη εφαρμογή στη στοίβα επιστρέφεται στο λειτουργικό σύστημα. Σχηματικά η ολοκλήρωση της συνάρτησης και η επιστροφή δεσμευμένου χώρου στη στοίβα φαίνεται ως εξής :



Σε αυτό το σημείο θα δούμε λίγο πιο αναλυτικά αυτό που είχαμε αναφέρει και στην φάση του πίνακα συμβόλων , ότι οι τρεις πρώτες θέσεις του εγγραφήματος δραστηριοποίησης καταλαμβάνουν συνολικά 12 bytes .

1. Η διεύθυνση επιστροφής είναι η διεύθυνση στην οποία πρέπει να μεταβεί το πρόγραμμα μετά την ολοκλήρωσης της εκτελεσης της συνάρτησης ή της διαδικασιας.Καταλαμβανει τα πρώτα 4 bytes του εγγραφήματος δραστηριοποίησης.
2. Ο σύνδεσμος προσπελασης έχει τη διεύθυνση του εγγραφήματος δραστηριοποίησης του γονεα της συνάρτησης ή της διαδικασιας.Με αυτό τον σύνδεσμο η συνάρτηση ή η διαδικασία μπορεί να προσπελασει δεδομένα που ανηκουν στους προγόνους της.Καταλαμβανει τα bytes από τη θέση 4 εως και τη θέση 7 , συνολικα 4 bytes.
3. Η επιστροφή τιμης έχει τη διεύθυνση της μεταβλητής στην οποία επιθυμουμε να γράφει το αποτέλεσμα της συνάρτησης και καταλαμβανει τα bytes από τη θέση 8 εως και 11 δηλαδή συνολικά 4 bytes. Αν πρόκειται για διαδικασία , η θέση αυτή στο εγγραφημα δραστηριοποίησης έχει με σκουπιδια.

Οι τρεις αυτές πρωτες θεσεις συμπληρωνονται κατά την εκτέλεση του προγράμματος από κώδικα τον οποίο παραγει ο μεταγλωττιστης.

Όπως και στον πίνακα συμβόλων, ετσι και εδω, θα ορίσουμε καποιες βοηθητικες συναρτήσεις οι οποίες θα μας διευκολυνουν σε διάφορους τομεις.Θα τις αναφερουμε εδώ επιγραμματικα και παρακατω θα εξηγήσουμε διεξοδικα ποια είναι η λειτουργία τους και πως τις χρησιμοποιήσαμε. Οι βοηθητικές συναρτήσεις είναι η gnlvcode() , loadvr() και storerv().

Η gnlvcode() είναι μια συνάρτηση που παράγει τελικό κώδικα για την προσπέλαση των μεταβλητών ή των διευθύνσεων που είναι αποθηκευμένες σε κάποιο εγγράφημα δραστηριοποίησης διαφορετικό από αυτό της συνάρτησης που μεταφράζεται αυτή τη στιγμή. Αυτό μας επιτρέπεται από τους κανόνες της Cimple αφού κάθε συνάρτηση έχει δικαίωμα να προσπελάσει και μεταβλητές ή διευθύνσεις που ανήκουν σε εγγράφημα δραστηριοποίησης κάπου προγόνου της. Επίσης μιας και πρόγονος είναι και το κυρίως πρόγραμμα , η gnlvcode() μπορεί να προσπελάσει και τις καθολικές μεταβλητές, όμως δεν είναι κάτι που θα αξιοποιήσουμε εμείς κατά την υλοποίηση μας. Η συνάρτηση αυτή παίρνει ως όρισμα τη μεταβλητή της οποίας την τιμή ή την διεύθυνση θέλουμε να προσπελάσουμε και το αποτέλεσμα της εκτέλεσης δίνει είτε τη διεύθυνση της μεταβλητής που αναζητείται σε έναν καταχωρητή t0(αναζήτηση τιμής μιας μεταβλητής) είτε τη διεύθυνση της μνήμης στην οποία περιέχεται η διεύθυνση της μεταβλητής που αναζητείται σε έναν καταχωρητή t0(αναζήτηση διεύθυνση μιας μεταβλητής). Η gnlvcode() λειτουργεί ως εξής : Αναζητεί στον πίνακα συμβόλων το όνομα της μεταβλητής που της δίνεται σαν παράμετρος, από το ποιο επίπεδο βρέθηκε η μεταβλητή αυτή , η gnlvcode() συμπεραίνει πόσα επίπεδα επάνω στο γενεαλογικό δέντρο της συνάρτησης θα πρέπει να ανέβει ώστε να φτάσει στο εγγράφημα δραστηριοποίησης που έχει την πληροφορία που αναζητεί. Αρχικά μεταβαίνει στον γονέα της συνάρτησης , η διεύθυνση του εγγραφήματος δραστηριοποίησης του βρίσκεται αποθηκευμένη στο σύνδεσμο προσπέλασης στη θέση -8(sp). Για να ανέβουμε αυτά τα επίπεδα χρησιμοποιούμε έναν καταχωρητή. Αφού έχει βρει η gnlvcode() πόσα επίπεδα πρέπει να ανέβει ακόμα , επαναλαμβάνει την ίδια διαδικασία. Μετά το πέρας όλων των μεταβάσεων ο καταχωρητής αυτός δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης το οποίο υπέδειξε ο πίνακας συμβόλων. Τέλος ο καταχωρητής κατεβαίνει κατά offset θέσεις ώστε να δείξει στη θέση μνήμης που βρίσκεται η ζητούμενη πληροφορία και το αποτέλεσμα που επιστρέφει η gnlvcode() είναι το περιεχόμενο του καταχωρητή. Όσον αφορά το πως ακριβώς χειριστήκαμε εμείς αυτή τη συνάρτηση εξηγείται παρακάτω.

Gnlvcode(v): Κρατάει σε μια μεταβλητή το επίπεδο στο οποίο καλέστηκε η gnlvcode() αν το topScope υπάρχει . Διατρέχει τον πίνακα με όλα τα Scopes και για τους προγόνους του κάθε στοιχείου του πίνακα, και για κάθε οντότητα που συναντάει ελέγχει αν το όνομα της είναι ίδιο με το όνομα της μεταβλητής v. Αν είναι τότε κάνει set ένα flag το οποίο αν δεν γίνει set θα οδηγηθούμε σε error . Κρατάει το επίπεδο στο οποίο βρέθηκε η μεταβλητή και κάνει την αφαίρεση levelDiff = StartingLevel- levelFound . Έπειτα γράφει μια φορά στο ASM αρχείο lw t0,-8(sp) και γράφει μετά lw t0,-8(t0) τόσες φορές όσες και η διαφορά που υπολογίστηκε προηγουμένως(για να ανεβεί τα υπόλοιπα n-1 επίπεδα) . Τέλος ελέγχει αν βρέθηκε κάποια οντότητα και αν δεν βρέθηκε επιστρέφει error και τερματίζει.

Η δεύτερη βοηθητική συνάρτηση την οποία θα σχολιάσουμε είναι η loadvr(). Η συνάρτηση αυτή παράγει τον κώδικα για να διαβαστεί η τιμή μιας μεταβλητής από τη μνημη. Παιρνει σαν ορισματα το ονομα της μεταβλητης της οποιας θελουμε να διαβασουμε τη τιμη και το ονομα του καταχωρητη στον οποιο θελουμε να τοποθετηθει. Η loadvr() βασίζεται σε πληροφορίες που παρεχει ο πινακας συμβολων και αναλογα με τη περιπτωση παραγει και τον αντιστοιχο κωδικα.

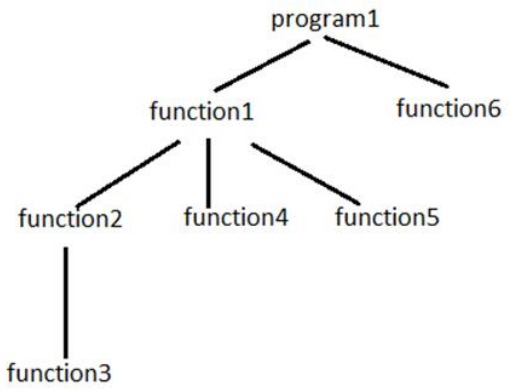
Οι περιπτώσεις που εμφανίζονται στην συνάρτηση loadvr() εξηγούνται παρακάτω αναλυτικά.

loadvr(v, reg): παίρνει ως όρισμα μια μεταβλητή και έναν καταχωρητή (τον αριθμό μόνο όχι και το γράμμα) . Αν η μεταβλητή είναι integer (άρα αριθμητική σταθερά) τότε γράφουμε στο αρχείο ASM την εντολή li reg, integer. Διαφορετικά έχουμε μια από τις άλλες 5 περιπτώσεις όμως όλες θα χρειαστούν κάποια νούμερα που θα τα βρούμε στον πίνακα συμβόλων. Γι’ αυτόν τον λόγο με μια for ελέγχουμε για όλα τα scopes που αποθηκεύσαμε προηγουμένως στην λίστα AllScopes ένα – ένα όλα τα entities αυτού του scope και όλων των προγόνων του. Αν κάποιο από αυτά τα scopes έχει ίδιο όνομα με το όρισμα v της loadvr τότε ξέρουμε ότι πρόκειται για τη σωστή μεταβλητή και κρατάμε σε διάφορες μεταβλητές ότι στοιχείο κρίνουμε χρήσιμο για αργότερα. Ενδεικτικά τον τύπο της οντότητας (variable ,parameter,temp) το level στο οποίο βρίσκεται, και αναλόγως με το αν είναι variable ,parameter ή temp το offset του και το parMode αν είναι παράμετρος. Η υλοποίηση αυτή κάνει σίγουρα περιττούς ελέγχους αφού η λίστα AllScopes περιέχει ήδη όλα τα scopes που

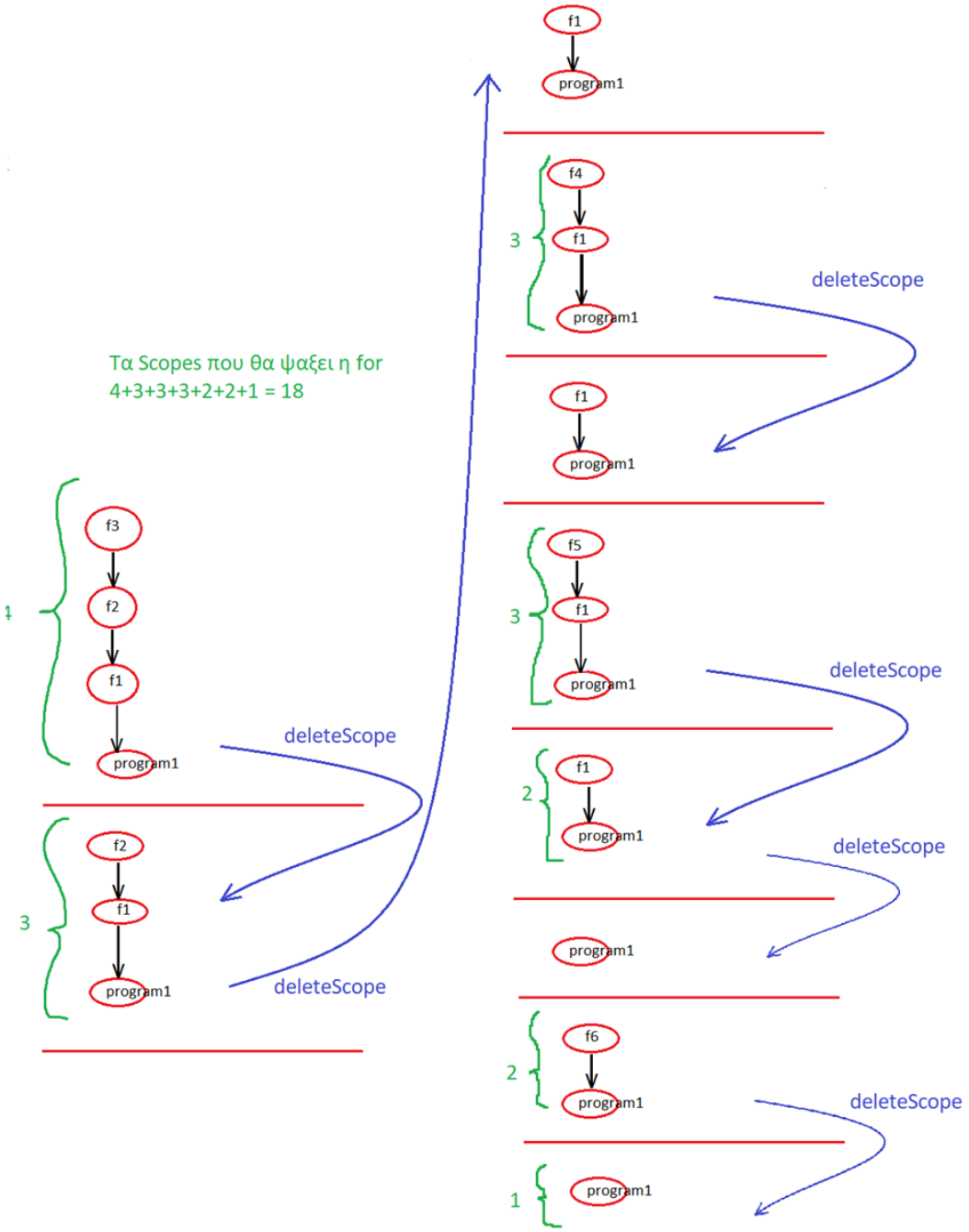
δημιουργήθηκαν κατά την εκτέλεση του προγράμματος αλλά ταυτόχρονα πηγαίνει και από τα παιδιά στους γονείς .

```
program1{
        function1{
                ...
                function2{
                        ...
                        function3 {
                            ...
                        }
                }
                function4{
                        ...
                }
                function 5{
                    ...
                }
        }
        function6 {
            ...
        }

}
```



Η allScopes εχει μεσα στον πινακα μια εγγραφη για το ποιο ειναι το topScope ακριβως πριν κληθει η deleteScope(). Αρα στο παραδειγμα μας εχει 7 εγγραφες/στοιχεια (μην ξεχναμε την τελευταια delete της "main").
AllScopes = {f3,f2,f4,f5,f1,f6,program1}
Συνεπως η loadvr για να βρει το σωστο entity θα ψαξει για καθε μια απο αυτες τις εγγραφες, καθε προγονο της και καθε entity του καθε προγονου της ενα ενα μεχρι να βρει καποιο του οποιου το ονομα να ειναι ιδιο με το entity που ψαχνουμε. Στο συγκεκριμενο παραδειγμα θα ψαξει συνολικα 18 scopes (καποια οπως το f1 παραπανω απο μια φορα).



Οι 5 περιπτώσεις που αναφέραμε πιο πριν είναι :

1) Αν έχουμε **Τοπική μεταβλητή** (type = 0 and level=topScope.nestingLevel) ή **παράμετρο που έχει περαστεί με τιμή** (type=2 and mode=0) ή **προσωρινή μεταβλητή** (type=3):
    - Γράφουμε στο ASM αρχείο lw reg,-offset(sp)
2) Αν έχουμε **Παράμετρο που έχει περαστεί με αναφορά**(type=2 and mode=1):
    - Γράφουμε στο ASM αρχείο lw t0,-offset(sp)
    - Γράφουμε στο ASM αρχείο lw reg,(t0)

3) Αν έχουμε **Τοπική μεταβλητή**(type=0 and level<topScope.nestingLevel) ή **παράμετρο που έχει περαστεί με τιμή η οποία ανήκει σε πρόγονο**(type=2 and mode=0 and level<topScope.nestingLevel)**:**
    - Καλούμε την gnlvcode με όρισμα την μεταβλητή v
    - Γράφουμε στο ASM αρχείο lw reg,(t0)
4) Αν έχουμε **Παράμετρο που έχει περαστεί με αναφορά, η οποία ανήκει σε πρόγονο**(type=2 and mode=1 and level<topScope.nestingLevel):
    - Καλούμε την gnlvcode για την μεταβλητή v
    - Γράφουμε στο αρχείο ASM lw t0, (t0)
    - Γράφουμε στο αρχείο ASM lw reg, (t0)
5) Αν έχουμε **καθολική μεταβλητή** (level=0) :
    - Γράφουμε στο ASM αρχείο lw reg -offset(gp).

Η τελευταία βοηθητική συνάρτηση είναι η storerv() , η οποία είναι παρόμοια ως προς την υλοποίηση με την loadvr() . Το μόνο διαφορετικό είναι ότι εδώ, αντί για εντολή ανάγνωσης lw , έχουμε εντολή αποθήκευσης sw.Παίρνει ίδια ορίσματα με την loadvr() .

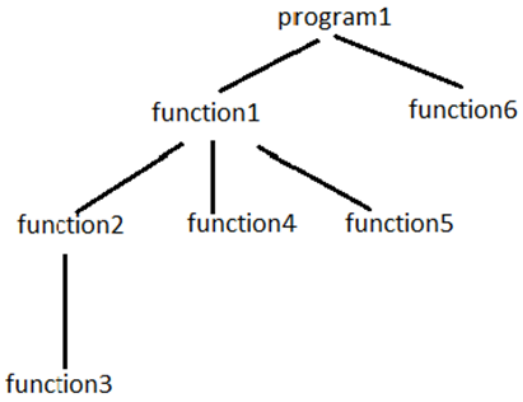
Οι περιπτώσεις που εμφανίζονται στην συνάρτηση loadvr() εξηγούνται παρακάτω αναλυτικά.

Storerv(reg, v): παίρνει σαν όρισμα το όνομα της μεταβλητής της οποίας θέλουμε να διαβάσουμε και τον καταχωρητή στον οποίο θέλουμε να την αποθηκεύσουμε . Αν η μεταβλητή αυτή είναι integer τότε φορτώνουμε την τιμή της αριθμητικής σταθεράς σε ένα καταχωρητή χρησιμοποιώντας την loadvr() και στη συνέχεια τοποθετούμε την μεταβλητή στην κατάλληλη θέση στη στοίβα . Αν η μεταβλητή αυτή δεν είναι integer τότε έχουμε άλλες 5 περιπτώσεις όπως και παραπάνω που για αυτές χρειαζόμαστε πληροφορίες που μας παρέχει ο πίνακας συμβόλων. Γι’ αυτό το λόγο κάνουμε και εδώ ένα for που ελέγχει για όλα τα scopes και τους προγόνους τους ,που αποθηκεύσαμε στην λίστα AllScopes, τα entities του εκάστοτε scope. Εάν κάποια οντότητα από αυτά τα scopes έχει ίδιο όνομα με την μεταβλητή v τότε καταλαβαίνουμε ότι βρήκαμε τη σωστή μεταβλητή και κρατάμε όποια πληροφορία μας είναι απαραίτητη για την συνέχεια . Συγκεκριμένα κρατάμε τον τύπο της οντότητας δηλαδή αν είναι (variable , parameter ή temporary variable) , το level στο οποίο βρίσκεται , το

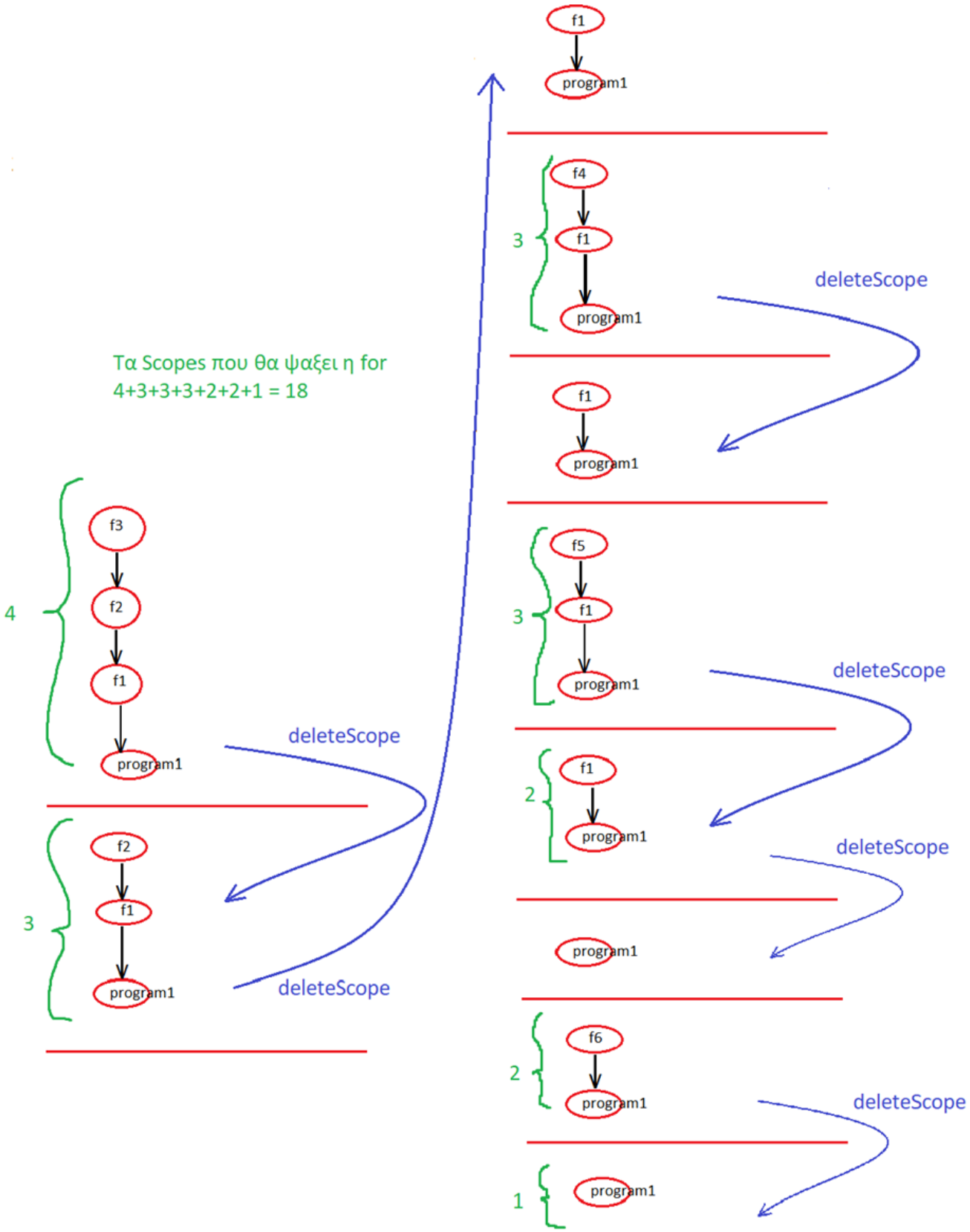
offset που έχει ανάλογα με το τι τύπος οντότητας είναι και το parMode (πέρασμα με τιμή ή με αναφορά) αν είναι parameter .

```
program1{
        function1{
                ...
                function2{
                        ...
                        function3 {
                                ...
                        }
                }
                function4{
                        ...
                }
                function 5{
                        ...
                }
        }
        function6 {
                ...
        }

}
```



Η allScopes εχει μεσα στον πινακα μια εγγραφη για το ποιο ειναι το topScope ακριβως πριν κληθει η deleteScope(). Αρα στο παραδειγμα μας εχει 7 εγγραφες/στοιχεια (μην ξεχναμε την τελευταια delete της "main").
AllScopes = {f3,f2,f4,f5,f1,f6,program1}
Συνεπως η storerv για να βρει το σωστο entity θα ψαξει για καθε μια απο αυτες τις εγγραφες, καθε προγονο της και καθε entity του καθε προγονου της ενα ενα μεχρι να βρει καποιο του οποιου το ονομα να ειναι ιδιο με το entity που ψαχνουμε. Στο συγκεκριμενο παραδειγμα θα ψαξει συνολικα 18 scopes (καποια οπως το f1 παραπανω απο μια φορα).



Οι 5 περιπτώσεις που αναφέραμε προηγουμένως είναι οι εξής :

1) Αν έχουμε **τοπική μεταβλητή** (type==0 and level=currentScope.nestingLevel) ή **παράμετρος που έχει περαστεί με τιμή** (type ==2 and mode==0)) ή **προσωρινή μεταβλητή** (type==3) :
    - Γράφουμε στο ASM αρχείο sw reg, -offset(sp)
2) Αν έχουμε **παράμετρο που έχει περαστεί με αναφορά** (type==2 and mode==1) :
    - Γράφουμε στο ASM αρχείο lw t0, -offset(sp)
    - Γράφουμε στο ASM αρχείο sw reg,(t0)
3) Αν έχουμε **τοπική μεταβλητή** (type ==0 and level<currentScope.nestingLevel) ή **παράμετρο που έχει περαστεί με τιμή**(type==2 and mode==0 and level<currentScope.nestingLevel) **η οποία ανήκει σε πρόγονο** :
    - Καλούμε την gnlvcode()
    - Γράφουμε sw reg,(t0)
4) Αν έχουμε **παράμετρο που έχει περαστεί με αναφορά η οποία ανήκει σε πρόγονο** (type==2 and mode==1 and level<currentScope.nestingLevel) :
    - Καλούμε την gnlvcode()
    - Γράφουμε lw t0,(t0) στο ASM αρχείο
    - Γράφουμε sw reg,(t0) στο ASM αρχείο
    - Γράφουμε sw reg,(t0) στο ASM αρχείο
5) Αν έχουμε **καθολική μεταβλητή** (level==0):
    - Γράφουμε στο ASM αρχείο sw reg, -offset(gp)

Οι βοηθητικες συναρτησεις που φτιαξαμε παραπανω μας βοηθουν ωστε να παραγουμε τελικο κωδικα για τις εκχωρησεις και για τις αριθμητικες πραξεις. Για την εκχωρηση μιας μεταβλητης το μονο που εχουμε να κανουμε ειναι να φορτωσουμε μια μεταβλητη σε εναν καταχωρητη και απο εκει να τον μεταφερουμε στη θεση μνημης που βρισκεται η μεταβλητη στην οποια θα γινει η εκχωρηση. Οπότε σε αυτή τη περίπτωση , για να παραχθεί ο ζητούμενος κώδικας πρέπει αρχικά να καλέσουμε την loadvr() και έπειτα την storerv() με τις κατάλληλες παραμετρους. Τωρα στη περιπτωση που εχουμε εκχωρηση αριθμητικης σταθερας απλως αντικαθιστουμε την μεταβλητη που φορτωναμε με την αριθμητικη σταθερα. Για καλύτερη κατανόηση ας εξηγήσουμε ότι για κάθε μια από τις τέσσερις αριθμητικές πράξεις που υποστηρίζει η Cimple , οι δυο μεταβλητές που συμμετέχουν στην πράξη μεταφέρονται αντίστοιχα σε δυο

προσωρινούς συνήθως καταχωρητές μέσω της συνάρτησης loadvr(). Στη συνεχεια γινεται η πραξη με την χρηση της βοηθητικης συναρτησης produce με τους καταχωρητες και τελος μεταφερεται το αποτελεσμα σε μια μεταβλητη αποθηκευσης.

Με τις διακλαδωσεις ασχοληθηκαμε κατα την παραγωγη του ενδιαμεσου κωδικα , οποτε απο εκεινες τις εντολες πρεπει εδω να παραγεται ενας σχετικος κωδικας που θα εκτελει απλο αλμα στην περιπτωση jump του ενδιαμεσου κωδικα και λογικο αλμα για τη συγκριση δυο μεταβλητων.

Οταν ξεκιναει ενα προγραμμα , αυτο που πρεπει να εκτελεστει ειναι η πρωτη εντολη του κυριως προγραμματος.Αυτη η σειρα ομως δεν συμπιπτει με την πρωτη εντολη του τελικου κωδικα που εχει παραχθει μιας και κατα τη μεταφραση ακολουθειται η σειρα εμφανισης των εντολων στο αρχικο προγραμμα.Γι'αυτο το λογο οι εντολες του κυριως προγραμματος θα μεταφραστουν μετα τη μεταφραση ολων των συναρτησεων και των διαδικασιων. Ο κωδικας αρχικα πρεπει να εκτελεσει ενα αλμα στην πρωτη εντολη του κυριως προγραμματος , οποτε θα παραχθει η εντολη jump και θα κανουμε αλμα στην ετικετα main. Όταν μετάβουμε λοιπόν στην main θα πρέπει να αρχικοποιούμε τον δείκτη στοίβας sp και τον καταχωρητή για τις καθολικές μεταβλητές gp. Μετα, θα ακολουθησει η ετικετα της πρωτης εκτελεσης εντολης του κυριως προγραμματος L , η οποια δεν μας ενδιαφερει ποια θα ειναι . Οταν ολοκληρωθει η παραγωγη κωδικα για ολες τις εντολες του προγραμματος , φτανουμε και στην τελευταια εντολη , τη halt , με την οποια θα επιστραφει ο ελεγχος στο λειτουργικο συστημα.

Οσον αφορα τις παραμετρους μιας συναρτησης , η σειρα τοποθετησης τους στο εγγραφημα δραστηριοποιησης αμεσως μετα απο τα 12 δεσμευμενα bytes , ειναι και η σειρα εμφανισης τους. Στο νέο εγγράφημα δραστηριοποίησης θα χρησιμοποιήσουμε τον καταχωρητή fp ως δείκτη και θα τον τοποθετήσουμε στην αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης ή της διαδικασίας εκεί που θα τοποθετηθεί ο δείκτης sp όταν ξεκινήσει η εκτέλεση της. Αρά, τοποθετούμε τον fp στη θέση του και έπειτα προσθέτουμε στον sp τόσα bytes όσα αποτελείται το εγγράφημα δραστηριοποίησης της καλούσας συνάρτησης. Ετσι υπολογιζουμε τη διευθυνση του τελους του εγγραφηματος δραστηριοποιησης της καλουσας και το σημειο στο οποιο θα τοποθετηθει το εγγραφημα δραστηριοποιησης της κληθεισας. Μετά την τοποθέτηση του fp , για κάθε παράμετρο που συναντάμε (par) και ανάλογα με το αν αυτή περνά με τιμή ή με αναφορά , κάνουμε τις ανάλογες ενέργειες.

Κατά το πέρασμα μιας παραμέτρου με τιμή , η τιμή της παραμέτρου αντιγράφεται στη θέση που έχει δεσμευτεί στο εγγράφημα δραστηριοποίησης για την παράμετρο αυτή. Η τιμή της παραμέτρου θα αναζητηθεί και θα τοποθετηθεί προσωρινά σε έναν καταχωρητή με χρήση της βοηθητικής συνάρτησης loadvr(). Στη συνέχεια , το επόμενο βήμα είναι η αντιγραφή της τιμής της παραμέτρου από τον καταχωρητή στην κατάλληλη θέση στη στοίβα που γίνεται με την εντολή sw και ο χώρος που θα δεσμεύσει εκεί, εξαρτάται από τον διαθέσιμο χώρο που έχουν αφήσει οι παράμετροι που υπάρχουν ήδη. Αφού όπως έχουμε ξαναφέρει στη Cimple κάθε παράμετρος καταλαμβάνει 4 bytes , τότε η i-οστη παράμετρος θα έχει offset : d = 12 + (i-1)*4 . Μια μεταβλητη μπορει να βρεθει στο εγγραφημα δραστηριοποιησης της συναρτησης που εκτελειται αν προκειται για τοπικη μεταβλητη , παραμετρο που εχει περαστει με τιμη ή αν αποτελει προσωρινη μεταβλητη . Μια μεταβλητη μπορει να βρεθει στο εγγραφημα δραστηριοποιησης μιας συναρτησης προγονου αν αποτελει τοπικη μεταβλητη ή παραμετρο που εχει περαστει με τιμη στον προγονο και τελος μια μεταβλητη ειναι καθολικη οταν εχει δηλωθει στο κυριως προγραμμα.

Ας εξετασουμε και τι συμβαινει κατα το περασμα παραμετρων με αναφορα.Η διευθυνση της μεταβλητης αντιγραφεται στο εγγραφημα δραστηριοποιησης της υπο δημιουργιας συναρτησης. Η παραγωγη κωδικα για το περασμα παραμετρου με αναφορα ειναι αρκετα πιο περιπλοκη απο οτι στο περασμα παραμετρου με τιμη , καθως τωρα δεν υπαρχει καποια ετοιμη συναρτηση που να μας τοποθετει τη διευθυνση της μεταβλητης σε καποιον καταχωρητη. Ετσι , θα πρεπει να υλοποιηθει ολοκληρη η παραγωγη κωδικα και να διαχωριστουν περιπτωσεις με βαση οτι μας επιστρεφει ο πινακας συμβολων. Διαχωριζουμε δυο βασικες περιπτωσεις κατα τις οποιες , στην πρωτη θεση της στοιβας που μας πηγαινει ο πινακας συμβολων βρισκεται η τιμη της μεταβλητης που θελουμε να περασουμε σαν παραμετρο και στη δευτερη θεση βρισκεται η διευθυνση της. Και σε αυτή τη περίπτωση ισχύει το ίδιο με το πέρασμα με τιμή όσον αφορά την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης οπότε και εδώ το offset είναι d = 12 + (i-1)*4 .

Πρωτα εχουμε τη παραμετρο με αναφορα , οταν στη στοιβα υπαρχει αποθηκευμενη η τιμη της παραμετρου . Πρόκειται για περιπτώσεις που η μεταβλητή είναι είτε τοπική ή προσωρινή ή παράμετρος που έχει περαστεί με τιμή στη συνάρτηση που εκτελείται (τοποθετούμε το offset sp στη θέση d bytes πάνω από τον fp), είτε τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή στη συνάρτηση πρόγονο (καλούμε την gnlvcode() και έπειτα

αντιγράφουμε τον καταχωρητή αυτόν στη θέση d bytes πάνω από το fp) είτε καθολική μεταβλητή (τοποθετούμε το offset gp στη θέση d bytes πάνω από τον fp). Για ολες αυτες τις περιπτωσεις δημιουργουμε διαφορετικο κωδικα παντα ομως , θελουμε να φτασουμε σε κοινο αποτελεσμα.

Επειτα εχουμε τη παραμετρο με αναφορα , οταν στη στοιβα υπαρχει αποθηκευμενη η διευθυνση της παραμετρου. Για να συμβαινει αυτο σημαινει ειτε οτι η μεταβλητη που θελουμε να περασουμε ως παραμετρο με αναφορα ειναι παραμετρος που εχει περαστει με αναφορα στην καλουσα συναρτηση ή διαδικασια ειτε οτι η παραμετρος εχει περαστει με αναφορα σε καποιον προγονο. Στη πρωτη περιπτωση , στη θεση offset πανω απο τον stack pointer υπαρχει τοποθετημενη η διευθυνση αυτης της μεταβλητης και απο εκει πρεπει να αντιγραφει στη θεση d που εχουμε δεσμευσει για την παραμετρο αυτη στη κληθησα συναρτηση , την οποια θα προσπελασουμε μεσω του frame pointer. Στη δεύτερη περίπτωση , η πρόσβαση σε θέση της στοίβας ενός προγόνου γίνεται με την gnlvcode() , οπότε αντιγράφουμε στη θέση της μνήμης τη διεύθυνση της μεταβλητής που μας ενδιαφέρει.

FinalCode(): Παίρνει την λίστα με όλες τις τετράδες που δημιουργήθηκαν στον ενδιάμεσο κώδικα και για κάθε μια από αυτές ελέγχει το όρισμα που βρίσκεται στην θέση 1 (η θέση 0 είναι ο αριθμός της τετράδας). Αναλόγως με το τι είναι το όρισμα στη θέση 1 εκτελεί τις κατάλληλες εντολές RISC-V. Σε αυτές τις εντολές πέρα από κατευθείαν εγγραφές στο αρχείο ASM έχουμε και κλήσεις των συναρτήσεων gnlvcode , loadvr, storerv που δημιουργήθηκαν παραπάνω.

Σας ευχαριστούμε !