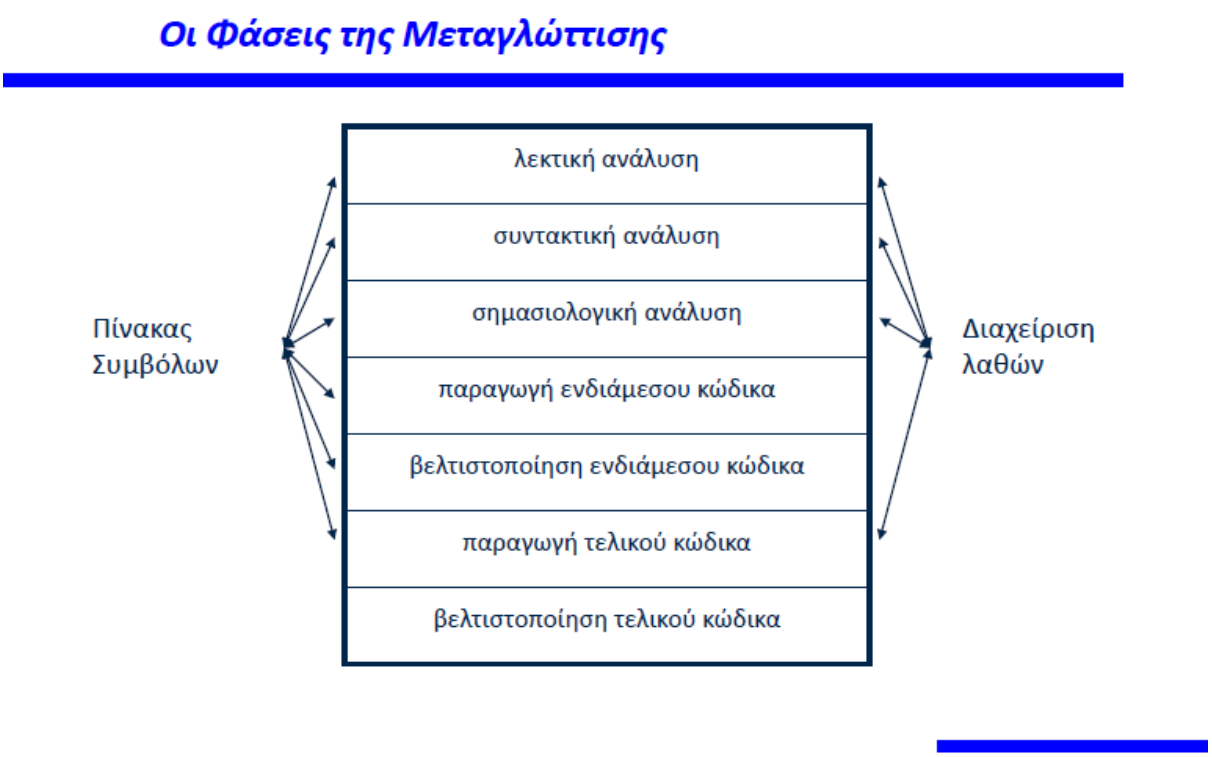
Φραγκαθούλας Χρήστος 4196
Παναγιώτης Κατσαντάς 3390

Εργασία Μεταφραστές 2021

Μεταγλωττιστής ή μεταφραστής (compiler) ονομάζεται ένα πρόγραμμα υπολογιστή που διαβάζει κώδικα γραμμένο σε μια γλώσσα προγραμματισμού (την πηγαία γλώσσα) και τον μεταφράζει σε ισοδύναμο κώδικα σε μια άλλη γλώσσα προγραμματισμού (τη γλώσσα στόχο). Το κείμενο της εισόδου ονομάζεται πηγαίος κώδικας (source code), ενώ η έξοδος του προγράμματος, η οποία συχνά έχει δυαδική μορφή, αντικειμενικός κώδικας (object code).Στη δική μας περίπτωση ξεκινάει από τη πηγαία γλώσσα ***CIMPLE*** μέχρι τη γλώσσα στόχο Assembly.

Η διαδικασία της μεταγλώττισης ακολουθεί τα εξής στάδια:



Στα πλαίσια της άσκησης αυτής ακολουθούνται τα βήματα: Λεκτική Ανάλυση,Συντακτική Ανάλυση, Σημασιολογική Ανάλυση, Παραγωγή Ενδιάμεσου Κώδικα, Παραγωγή Τελικού Κώδικα

Περιεχόμενα

Λεκτική Ανάλυση

Ο λεκτικός αναλυτής παρέχει τις παρακάτω λειτουργίες:

- Διαχωρίζει το εισερχόμενο κείμενο σε λεκτικές μονάδες (*tokens*) και το μεταφέρει με τον τρόπο αυτό στο συντακτικό αναλυτή.
- Αποθηκεύει τα σύμβολα που διαβάζει σε πίνακα συμβόλων.
- Αποθηκεύει άλλα στοιχεία όπως τις συμβολοσειρές σε δυναμική μνήμη.
- Αναγνωρίζει λεκτικά λάθη στην είσοδο (π.χ. σύμβολα που δεν επιτρέπονται στη γλώσσα).
- Αφαιρεί τα σχόλια
- Συσχετίζει αριθμούς γραμμών με στοιχεία της εισόδου του.
- Παράγει κατάλληλα διαγνωστικά μηνύματα.

Με τον τρόπο αυτό διαχωρίζονται οι εργασίες της λεκτικής και της συντακτικής ανάλυσης και κάθε μια υλοποιείται με τον πιο αποδοτικό τρόπο.
Ο λεκτικός αναλυτής αποτελεί μια συνάρτηση που καλείται από το συντακτικό αναλυτή και επιστρέφει σε αυτόν έναν ακέραιο ο οποίος χαρακτηρίζει την εκάστοτε λεκτική μονάδα, τη λεκτική μονάδα.

Μια **λεκτική μονάδα** (**token**) είναι μια ακολουθία από χαρακτήρες που έχουν καταχωρηθεί με βάση τους κανόνες της γλώσσας Cimple, με αλφάβητο:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,…,Z και a,…,z),
• τα αριθμητικά ψηφία (0,…,9) με τιμές από –(232 −1) έως 232 −1,
• τα σύμβολα των αριθμητικών πράξεων (+, -, *, /),
• τους τελεστές συσχέτισης (<, >, =, <=, >=, <>)
• το σύμβολο ανάθεσης (:=)
• τους διαχωριστές (;, “,”, :)
• τα σύμβολα ομαδοποίησης ([,], (,) , { , })
• του τερματισμού του προγράμματος (.)
• και διαχωρισμού σχολίων (#)

και δεσμευμένες λέξεις:
program, declare, if, else, while, switchcase, forcase, incase, case, default,
not, and, or, function, procedure, call, return, in, inout, input, print

καθώς και λευκούς χαρακτήρες:
tab, space, return

Ο Λεκτικός αναλυτής λειτουργεί ως ένα αυτόματο καταστάσεων.Περιλαμβάνει κωδικοποιημένη πληροφορία για όλες τις πιθανές ακολουθίες χαρακτήρων που μπορούν να βρίσκονται σε κάθε λεκτική μονάδα που αναγνωρίζει.

```
#        numbers letters  +-           */          {}()[]       ,;         :       <       >       =        #      " \n\t\r"  #.
states = [[dig,   idk,    addOperator, mulOperator, groupSymbol, delimiter, asgn,   smaller, larger, relOp,     rem,     start,   stop],  # start 0
          [dig,   number, number,      number,      number,      number,    number,  number,  number, number,    number,  number,  stop],  # dig 1
          [idk,   idk,    keyIden,     keyIden,     keyIden,     keyIden,   keyIden, keyIden, keyIden, keyIden,   keyIden, keyIden, stop],  # idk 2
          [error, idk,    error,       error,       groupSymbol, error,     error,   error,   error,  assignment, error,   asgnStop,stop],  # asgn 3
          [relOp, relOp,  relOp,       relOp,       relOp,       relOp,     relOp,   error,   relOp,  relOp,     error,   relOp,   stop],  # smaller 4
          [relOp, relOp,  relOp,       relOp,       relOp,       relOp,     relOp,   relOp,   relOp,  relOp,     error,   relOp,   stop],  # larger 5
          [rem,   rem,    rem,         rem,         rem,         rem,       rem,     rem,     rem,    rem,       start,   rem,     rem]    # rem 6
          ]
```

Η διαδικασία που ακολουθεί η λεκτική ανάλυση κάθε φορά που αυτή καλείται ονομάζεται “Tokenization” όπου ο λεκτικός αναλυτής ξεκινώντας από μια αρχική κατάσταση, με κάθε είσοδο ενός χαρακτήρα αλλάζει την κατάσταση μέχρι να φτάσει σε μια τελική. Στη συνέχεια γίνεται κατηγοριοποίηση των λεκτικών μονάδων και αυτές προωθούνται για περαιτέρω επεξεργασία στο συντακτικό αναλυτή. Σε περίπτωση που ο λεκτικός αναλυτής βρει κάποια μη επιτρεπτή λεκτική μονάδα, αναφέρει σφάλμα.

Σε πολλές καταστάσεις, όπως αυτές που φαίνονται παρακάτω, η αποδοχή μιας λεκτικής μονάδας γίνεται όταν ο επόμενος χαρακτήρας είναι “λευκός”, οπότε χρησιμοποιούμε τη τεχνική της οπισθοχώρησης (backtracking) σε σχέση με χαρακτήρες που έχουν ήδη διαβαστεί (τους λευκούς).

```
136         if((state == keyIden) or (state == number) or (state == asgnStop) or
137         ((state == relOp) and (last_word[-1] != '=') and (last_word != "<>")) ): #final states.
138             file_counters[0] -= 1
139             file_counters[2] -= 1
140             if(file_counters[2] < 0):
141                 file_counters[2] = 0
142             file_counters[4] -= 1
143             if(last_word[-1] == '\n'):
144                 file_counters[1] -= 1
145             if((not last_word[-1].isalpha()) and (not last_word[-1].isdigit()) or (state == relOp)):
146                 last_word = last_word[:-1]
```

όπου file_counters = [0, 1, 1, False, 1] # code_file_counter, row, col, ! δε χρησιμοποιείται, temp_col
code_file_counter = χαρακτήρας που μόλις διαβάστηκε από το αρχείο πηγαίου κώδικα.
temp_col = προσωρινή στήλη. Παίρνει την τιμή «1» όταν file_counters [2] = 1 (αυτό συμβαίνει όταν ο char = '\ n' και ο lexer διαβάζει τον επόμενο char)

Αυτό είναι το τελευταίο κομμάτι κώδικα όσον αφορά τον lexer που θα κληθεί από ένα σωστό πρόγραμμα. Επισημάνει πως εάν υπάρχει κώδικας ή σχόλια μετά την εμφάνιση της τελείας (η οποία καθορίζει και το τέλος του προγράμματος), εμφανίζει αντίστοιχο σφάλμα.

```
elif(char in "."):
    next_state = 12
    if(comment_counter != 1):
        txt = file.read(200)
        txt = re.sub(r"[\n\t\r ]*","",txt)
        print(txt)
        for line in txt:
            if( len(line) >= 1 ):
                file.close()
                sys.exit("Code / Comments after '.' character is/are not acceptable.
                ",column: " +(str(file_counters[4] -1)))

        print("Lexer exited successfully!")
        ##########Code exits#########
else:
    if(comment_counter != 1):
        file.close()
        sys.exit("Invalid character: "+ char +". Error at line: "+str(int((file_count
        ",column: " +(str(file_counters[4] -1)))
last_word += char
state = states[state][next_state]
```

Το κομμάτι του else αφορά μη επιτρεπτούς χαρακτήρες για τη γλώσσα για τους οποίους εμφανίζεται ανάλογο σφάλμα όταν αυτοί δε περιέχονται σε σχόλια.
Τέλος φαίνεται η αλλαγή κατάστασης στο αυτόματο

Συντακτική Ανάλυση

Είναι το τμήμα του μεταγλωττιστή που διαβάζει τον πηγαίο κώδικα με βάση μια αναλυτική γραμματική

Ο συντακτικός αναλυτής:

- υλοποιεί τη συντακτική ανάλυση μιας συγκεκριμένης γλώσσας,
- δέχεται ως είσοδο ένα πρόγραμμα με τη μορφή ακολουθίας λεκτικών μονάδων,
- ελέγχει αν το πρόγραμμα είναι σύμφωνο με τη γραμματική της γλώσσας που υλοποιεί (αν ανήκει στη συγκεκριμένη γλώσσα),
- σε περίπτωση συντακτικού λάθους ενημερώνει το χρήστη.
- Βασίζεται σε γραμματική LL(1) αναγνωρίζει από αριστερά στα δεξιά, την αριστερότερη δυνατή παραγωγή και όταν βρίσκεται σε δίλημμα ποιον κανόνα να ακολουθήσει της αρκεί να κοιτάξει το αμέσως επόμενο σύμβολο στην συμβολοσειρά εισόδου.

Η συντακτική ανάλυση αποτελείται από πολλά υποπρογράμματα (τα οποία συνδέονται μεταξύ τους είτε άμεσα είτε έμμεσα) τα οποία βασίζονται στη γραμματική της cimple.

Η υλοποίησή της γίνεται ως εξής:

- εάν και ο λεκτικός αναλυτής επιστρέφει λεκτική μονάδα που αντιστοιχεί στο τερματικό αυτό σύμβολο έχουμε αναγνωρίσει επιτυχώς τη λεκτική μονάδα
- αντίθετα εάν ο λεκτικός αναλυτής δεν επιστρέψει τη λεκτική μονάδα που περιμένει ο συντακτικός αναλυτής, έχουμε λάθος και καλείται ο διαχειριστής σφαλμάτων
- Όταν αναγνωριστεί και η τελευταία λέξη του πηγαίου προγράμματος, τότε η συντακτική ανάλυση έχει στεφθεί με επιτυχία.

Για τη σωστή σύνταξη ενός προγράμματος πρέπει να ακολουθούνται τα εξής:

- Στην αρχή κάθε προγράμματος πρέπει να υπάρχει η δεσμευμένη λέξη program.
  program : program ID block .

- Η σωστή δομή προγράμματος είναι πρώτα οι δηλώσεις των μεταβλητών, έπειτα οι δηλώσεις και ο ορισμοί των συναρτήσεων και τέλος οι δηλώσεις.Αυτή η δομή ακολουθείται για κάθε εσωτερικό block
  block : declarations subprograms statements

- Δήλωση μεταβλητών (κανένα ή περισσότερα declare μπορούν να υπάρχουν σε ένα block)
  declarations : (declare varlist ;)∗

- ο τρόπος δήλωσης μεταβλητών. Ένα ID και πολλά στη συνέχεια χωρισμένα με “,” η κενή δήλωση.
  varlist : ID (, ID)∗
  | ε

- Καμία ή περισσότερες συναρτήσεις/διαδικασίες μπορούν να υπάρχουν σε ένα block
  subprograms : (subprogram)∗

- Ορισμός συνάρτησης/ διαδικασίας. Προϋποθέτει τη δεσμευμένη λέξη function ή procedure, έπειτα το όνομά της, και μέσα σε παρενθέσεις τοποθετούνται τα ορίσματα με βάση τη parlist και τέλος υπάρχει ένα νέο block με τα δικά του declare, subprograms,statements.
  subprogram : function ID (formalparlist) block
  | procedure ID (formalparlist) block

- Λίστα από τυπικές παραμέτρους οι οποίες χωρίζονται αυστηρά με “,”
  formalparlist : formalparitem (, formalparitem)*
  | ε

- Μια τυπική παράμετρος (“in” πέρασμα με τιμή, “inout” πέρασμα με αναφορά)
  formalparitem : in ID
  | inout ID

- Σε κάθε block επιτρέπονται ένα ή περισσότερα statement. Για μονό statement υπάρχει αμέσως μετά ένα “;” ενώ τα πολλαπλά statement πρέπει να περικλείονται από αγκύλες αλλά και με ερωτηματικά μεταξύ τους. Υπάρχει βέβαια η δυνατότητα μονού statement μεταξύ αγκύλων χωρίς τη χρήση ερωτηματικού μετά από αυτό.
  statements : statement ;
  | { statement (; statement)* }

- Τα επιτρεπτά statements είναι τα παρακάτω. Επιτρέπεται να έχουμε κενό statement.
  statement : assignStat
  | ifStat
  | whileStat
  | switchcaseStat
  | forcaseStat
  | incaseStat
  | callStat
  | returnStat
  | inputStat
  | printStat
  | ε

- Statement εκχώρησης τιμής σε μεταβλητή. Επιβάλλει όνομα μεταβλητής, έπειτα τα σύμβολα “:=” και τέλος ένα expression.
  assignStat : ID := expression

- Statement συνθήκης if. Ξεκινάει με τη δεσμευμένη λέξη “ if ” και μέσα σε παρενθέσεις περικλείονται τα condition της συνθήκης. Στη συνέχεια ακολουθούν τα statements και τέλος η συνθήκη elsepart.
  ifStat : if (condition) statements elsepart

- Statement συνθήκης else. Ξεκινάει με τη δεσμευμένη λέξη “else” ακολουθούν statements ή το else μπορεί και να παραλειφθεί
  elsepart : else statements
  | ε

- Statement whilestat. Ξεκινάει με τη δεσμευμένη λέξη while και μέσα σε παρενθέσεις ακολουθούν διάφορα condition. Έπειτα υπάρχουν τα statements.
  whileStat : while (condition) statements

- Statement switchCaseStat. Ξεκινάει με τη δεσμευμένη λέξη switchCase και ακολουθούν διάφορα case (condition) οπου το κάθε case έχει δικά του statements.Μπορεί βέβαια να μην υπάρχουν καθόλου τα case(condition) statements. Υποχρεωτικά όμως στο τέλος πρέπει να έχουμε το “default” ακολουθούμενο από τα δικά του statements.
  switchcaseStat: switchcase
  (case (condition) statements)*
  default statements

- Statement forcaseStat . Ξεκινάει με τη δεσμευμένη λέξη forCase και ακολουθούν διάφορα case (condition) οπου το κάθε case έχει δικά του statements.Μπορεί βέβαια να μην υπάρχουν καθόλου τα case(condition) statements. Υποχρεωτικά όμως στο τέλος πρέπει να έχουμε το “default” ακολουθούμενο από τα δικά του statements.
  forcaseStat : forcase
  (case (condition) statements)*
  default statements

- Statement forcaseStat . Ξεκινάει με τη δεσμευμένη λέξη forCase και ακολουθούν διάφορα case (condition) οπου το κάθε case έχει δικά του statements.Μπορεί βέβαια να μην υπάρχουν καθόλου τα case(condition) statements.
  incaseStat : incase
  (case (condition) statements)*

- Statement returnStat . Ξεκινάει με τη δεσμευμένη λέξη return και μέσα σε παρενθέσεις υπάρχει ένα expresion
  returnStat : return(expression)

- Statement callStat . Ξεκινάει με τη δεσμευμένη λέξη call, ακολουθεί το όνομα της διαδικασίας που καλείται και μέσα σε παρενθέσεις καλείται η acltualparlist.τα ορίσματα που της δίνονται δηλαδή
  callStat : call ID(actualparlist)

- Statement printStat . Ξεκινάει με τη δεσμευμένη λέξη print και μέσα σε παρενθέσεις ακολουθούν υπάρχει ένα expression (η έκφραση δηλαδή που πρόκειται να τυπωθεί).
  printStat : print(expression)

- Statement inputStat . Ξεκινάει με τη δεσμευμένη λέξη input και μέσα σε παρενθέσεις ακολουθούν υπάρχει ένα expression (η έκφραση δηλαδή που πρόκειται να τυπωθεί).
  inputStat : input(ID)

- Συνάρτηση actualparlist. Χρησιμεύει για να δέχεται τα ορίσματα των διαδικασιών τα οποία βρίσκονται μέσω της actualparitem και χωρίζονται μεταξύ τους με “,” . Είναι δυνατόν να υπάρχουν από 0, έως πολλά ορίσματα.
  actualparlist : actualparitem (, actualparitem)*
  | ε

- Συνάρτηση actualparitem. Χρησιμεύει για να λαμβάνει τα ορίσματα. Ένα όρισμα είναι η δεσμευμένη λέξη “in” και δίπλα ένα expression ή η δεσμευμένη λέξη inout και δίπλα ένα ID.
  actualparitem : in expression
  | inout ID

- Συνάρτηση condition. Είναι η συνάρτηση που καλείται όταν θέλουμε να έχουμε μία συνθήκη στη πρόγραμμά μας. Αποτελείται από τουλάχιστον μία κλήση της συνάρτησης boolterm και εάν θέλουμε να έχουμε συνθήκη με “ή” γράφουμε τη δεσμευμένη λέξη “or” και ανακαλούμε τη συνάρτηση boolterm.
  condition : boolterm (or boolterm)*

- Συνάρτηση boolterm. Είναι η συνάρτηση που καλείται από τη condition. Αποτελείται από τουλάχιστον μία κλήση της συνάρτησης boolfactor και εάν θέλουμε να έχουμε συνθήκη με “και” γράφουμε τη δεσμευμένη λέξη “and” και ανακαλούμε τη συνάρτηση boolfactor .
  boolterm : boolfactor (and boolfactor)*

- Συνάρτηση boolfactor . Είναι η συνάρτηση που καλείται από τη boolterm. Μπορεί να αναγνωρίσει τα εξής: Αν ξεκινάει με not, περιμένει να δει άνοιγμα και κλείσιμο “[ ]” και ενδιάμεσα τη κλήση της συνάρτησης condition. Αν δεν υπάρχει not περιμένει να δει άνοιγμα και κλείσιμο “[ ]” και ενδιάμεσα τη κλήση της συνάρτησης condition. Τέλος εάν δεν υπάρχει κάτι από τα παραπάνω, καλείται η συνάρτηση expression επειτα ψάχνει τελεστή από τη λίστα REL_OP και τέλος καλείται ξανά η συνάρτηση expression.
  boolfactor : not [condition]
  | [condition]
  | expression REL_OP expression

- Συνάρτηση expression. Σκοπός της είναι να ελέγχει την ορθότητα των εκφράσεων που περιέχονται στο πρόγραμμα. Καλεί αρχικά την optionlSign, έπειτα τη term, και τέλος καλεί από μία έως πολλές φορές τη λίστα με τους τελεστές ADD_OP η οποία ακολουθείτε από κλήση της συνάρτησης term.
  expression : optionalSign term (ADD_OP term)*

- Συνάρτηση term. Το πως θα είναι ο κάθε όρος μέσα στο κώδικα ορίζεται με αυστηρό τρόπο. Αρχικά καλείται η συνάρτηση factor και έπειτα καλείτε μία έως πολλές φορές τη λίστα με τους τελεστές MUL_OP η οποία ακολουθείτε από κλήση της συνάρτησης factor .
  term : factor (MUL_OP factor)*

- Συνάρτηση factor. Ένας παράγοντας μπορεί να είναι είτε ένας απλός ακέραιος, είτε μια έκφραση expression που θα περικλείεται από παρενθέσεις, είτε υπάρχει ένας συνδυασμός από τη λίστα ID και κλήση της συνάρτησης idtail το οποίο αφορά τη κλήση συνάρτησης.
  factor : INTEGER
  | (expression)
  | ID idtail
- Συνάρτηση idtail. Αυστηρά άνοιγμα και κλείσιμο παρενθέσεων και ενδιάμεσα τη κλήση της actualparlist αλλιώς μπορεί να είναι και κενή
  idtail : (actualparlist)
  | ε

- Συνάρτηση optionalSign. Περιέχει ένα τελεστή από τη λίστα ADD_OP ή τίποτα.
  optionalSign : ADD_OP
  | ε
- Λίστα REL_OP. Τελεστές σύγκρισης.
  REL_OP : = | <= | >= | > | < | <>

- Λίστα ADD_OP . Τελεστές πρόσθεσης, αφαίρεσης.
  ADD_OP : + | -

- Λίστα MUL_OP . Τελεστές πολλαπλασιασμού, διαίρεσης.
  MUL_OP : * | /

- Λίστα INTEGER . Λίστα επιτρεπτών ακεραίων.
  INTEGER : [0-9]+

- Λίστα ID . Λίστα επιτρεπτών αναγνωριστικών.
  ID : [a-zA-Z][a-zA-Z0-9]*

Σημασιολογική Ανάλυση

Εκτός από τη συντακτική ορθότητα ενός προγράμματος μας ενδιαφέρει και η εκτέλεση κάποιων άλλων ελέγχων, τα οποία δεν είναι δυνατό να περιγραφούν από μια γραμματική χωρίς συμφραζόμενα:

Αυτό το κομμάτι καλύπτει ερωτήσεις του τύπου:

- Λάθος αριθμός παραμέτρων στη συνάρτηση func()
- Λάθος τρόπος για το πέρασμα μιας παραμέτρου, in inout.
- Λάθος στη χρήση της μεταβλητής f (π.χ. να μη χρησιμοποιηθεί το όνομα μιας μεταβλητής ώς συνάρτηση).
- Δεν έχει δηλωθεί η μεταβλητή a.
- Η μεταβλητή, συνάρτηση ή διαδικασία που χρησιμοποιείται να μην έχει δηλωθεί πάνω από μια φορά στο ίδιο βάθος φωλιάσματος.
- Υπάρχει τουλάχιστον ένα return στον ορισμό μιας συνάρτησης.
- Να μην υπάρχει return στη διαδικασία.
- Να εκχωρείται η τιμή επιστροφής της συνάρτησης σε μία μεταβλητή.
- Να μην υπάρχει εκχώρηση για τη κλήση της call.
- Ποια από τις δηλώσεις της a χρησιμοποιείται όταν γίνεται αναφορά σε αυτή

Επειδή οι προαναφερόμενοι έλεγχοι βασίζονται σε πληροφορίες που γίνονται διαθέσιμες κατά τη μεταγλώττιση, λέμε ότι γίνεται στατική σημασιολογική ανάλυση του προγράμματος.
Τα αποτελέσματα των ελέγχων αυτών εξαρτώνται από:

- Συγκεκριμένες ιδιότητες των λεκτικών μονάδων όπως τύπος, τιμή
- Από που πληροφορίες που δε συνοδεύουν τη λεκτική μονάδα στο σημείο εμφάνισής της, όπως μερικά flags εντός του προγράμματος.
- Από υπολογισμούς που ίσως χρειάζεται να γίνουν.

```
counter_blocks = 0
return_counter = -1
```

1) Για κάθε κλήση της συνάρτησης block, στο εσωτερικό της αυξάνεται το counter_blocks κατά ένα πριν από κάθε άλλη ενέργεια, αλλά και το μειώνουμε κατά ένα πριν από την έξοδο της συνάρτησης.
   Με αυτό διασφαλίζουμε τη συνθήκη ότι αν το πρόγραμμα βρίσκεται στη main δε θα υπάρχει πουθενά return.
   Στο κώδικα αυτό διασφαλίζεται στη συνάρτηση statement κάθε φορά που ανιχνεύει ένα statement ως returnStat.

```
elif(token == "returntk"):
    if counter_blocks != 1:
        return_counter += 1
        word, token = lexer(file_counters)
        print(word+" "+token)
        returnStat(file_counters)
    else:
        file.close()
        sys.exit("Syntax Error. Program's main can not have a 'return' statement.
        ",column: " +(str(file_counters[4] - len(word))))
```

2) Σε κάθε κλήση της subprogram πριν από κάθε άλλη ενέργεια θέτουμε return_counter = 0 και λίγο πριν επιστρέψει η function, ελέγχει εάν

```
if(return_counter == 0):
    file.close()
    sys.exit("Syntax Error. 'Return' was expected in function '"+id+"'.
    ",column: " +(str(file_counters[4] - len(word))))
```

Έτσι ορίζουμε ότι εάν δεν υπάρχει κάποιο return ως statement πιο πάνω στη συνάρτηση, τότε θα πρέπει να υπάρχει στην έξοδο αυτής.

3) Σε κάθε κλήση της subprogram πριν από κάθε άλλη ενέργεια θέτουμε return_counter = 0 και λίγο πριν επιστρέψει η procedure, ελέγχει εάν

```
if(return_counter > 0):
    file.close()
    sys.exit("Syntax Error. 'Return' is not accepted in procedure '"+id+"'.")
```

Έτσι ορίζουμε πως η procedure δεν θα έχει κάποιο return ως statement μέσα της.

4) Με το παρακάτω τρόπο, σε κάθε κλήση της asignStat ελέγχουμε ότι δε θα γίνει ποτέ ανάθεση σε όνομα μεταβλητής που αντιστοιχεί σε διαδικασία:

```
def asgnStat(file_counters):
    global word, token,quadList,quads,procedure_names

    if(token == "keywordtk"):
        z = word

        word, token = lexer(file_counters)
        print(word+" "+token)

        if(token == "asignementtk"):
            word, token = lexer(file_counters)
            print(word+" "+token)

            if(word in procedure_names):
                file.close()
                sys.exit("Syntax Error. Can not assign procedure to variable.
                ",column: " +(str(file_counters[4] - len(word))))
```

procedure_names είναι ένας πίνακας που προσθέτουμε τα ονόματα κάθε φορά που ορίζεται μία διαδικασία.

5) Πιο κάτω στην asgnStat γίνεται και ο έλεγχος για το εάν υπάρχει εκχώρηση τιμής πριν τη κλήση οποιαδήποτε συνάρτησης:

```
elif (z not in procedure_names):
    file.close()
    if(z in function_names):
        sys.exit("Syntax Error. Keyword ':=' excpected before function call.
        ",column: " +(str(file_counters[4] - len(z))))
```

Ο έλεγχος για ανάθεση τιμής πρέπει να γίνεται μόνο όταν υπάρχει το “:=” στο πρόγραμμα, και όχι μέσα σε κάποιο condition π.χ., εφόσον εκεί η τιμή του χρησιμοποιείται για σύγκριση.

6) Σε αυτό το σημείο (στη τρίτη δυνατή περίπτωση κλήσης της factor) ελέγχουμε εάν ο χρήστης πρόκειται να χρησιμοποιήσει μία μεταβλητή ώς συνάρτηση (στο σημείο if(word== ”(“) :

```
elif(token == "keywordtk"):

    i = main_obj.Search(word,None)
    if i==-1:
        file.close()
        sys.exit("Syntax Error. Keyword '"+word+"' unidentified.
        ",column: " +(str(file_counters[4] - len(word))))


    ret = word
    word, token = lexer(file_counters)
    print(word+" "+token)

    # elegxei kai th ( wste na mh pianei kai tis aples metavlhtes
    if(word == "("):
        i = main_obj.Search(ret,"function",main_obj.scope)
        if i==-1:
            file.close()
            sys.exit("Keyword '"+ret+"' is not a Function. Error
            ",column: " +(str(file_counters[4] - len(ret))))
```

Οι υπόλοιπες περιπτώσεις ελέγχονται με τη βοήθεια του πίνακα συμβόλων. Ο πίνακας συμβόλων είναι απαραίτητος δίοτι σε γλώσσες με δομή block και στατική (λεκτική) εμβέλεια απαιτείται η επεξεργασία δηλώσεων να γίνεται όπως σε μία στοίβα.

- Με την είσοδο σε ένα block, γίνεται επεξεργασία όλων των δηλώσεων του block και προστίθενται στον πίνακα συμβόλων οι συνδέσεις τους.
- Με την έξοδο από το block, αφαιρούνται οι συνδέσεις των δηλώσεων και αποκαθίστανται οι προηγούμενες συνδέσεις που μπορεί να υπήρχαν.

Αυτή η διαδικασία λέγεται ανάλυση εμβέλειας.

Ο πίνακας συμβόλων είναι υλοποιημένος με αντικειμενοστραφή προγραμματισμό ως εξής:

Ορισμός κλάσης ps. Εντός αυτής υπάρχει ένας δισδιάστατος πίνακας που αντιστοιχεί στο πίνακα συμβόλων, το scope (δηλαδή σε πιο scope βρίσκεται τη κάθε στιγμή), offsets[] είναι ο πίνακας όπου κάθε γραμμή του δείχνει το offset του κάθε scope.

- Init: κάνει την αρχικοποίηση, βάζοντας την αρίθμηση στο πρώτο στοιχείο της κάθε γραμμής του πίνακα (για οπτικούς λόγους)
- Append: προσθέτει στο τέλος κάθε scope μία καινούρια εγγραφή
- Change_offset: Συνάρτηση η οποία καλείται πριν προστεθεί νέα εγγραφή
- Add_scope: Χρησιμοποιείται όταν ανοίγει νέο block
- Remove_scope: Χρησιμοποιείται όταν κλείνει ένα block

!!!! Τα offsets που επιλέγουμε να δημιουργηθεί ο πίνακας συμβόλων βασίζονται στην οργάνωση μνήμης του MIPS και εξηγούνται στη τελευταία ενότητα.!!!

- Print: Τύπωμα του πίνακα στην οθόνη, μέσω μια βοηθητικής συνάρτησης get για τη κάθε εγγραφή ξεχωριστά.

```
class ps:
    pinakas_symvolwn = [[]]
    scope = 0
    offset = 0
    offsets = []

    def __init__(self):
        self.pinakas_symvolwn[0].append("0")
        self.offsets.append(8)

    def append(self,record):
        self.pinakas_symvolwn[self.scope].append(record)

    def change_offset(self):
        self.offsets[self.scope] += 4

    def add_scope(self):
        self.scope += 1
        self.offsets.append(8)
        self.pinakas_symvolwn.append([str(self.scope)])

    def remove_scope(self):
        del self.pinakas_symvolwn[self.scope]
        del self.offsets[self.scope]
        self.scope -= 1

    def print(self):
        for row in self.pinakas_symvolwn:
            k = row[0]
            for j in range(1,len(row)):
                k += " ["+ row[j].get()+"] "
            print(k)
```

7) Επιπλέον στην κλήση της asgnStat ελέγχουμε αν το αριστερό μέλος της εκχώρησης είναι μεταβλητή (και δεν είναι όνομα συνάρτησης ή διαδικασίας), ώστε να μπορεί να πραγματοποιηθεί η εκχώρηση τιμης:

```
i = main_obj.Search(z,"asign")
print(i)
if "forp" in str(i):
    file.close()
    sys.exit("Syntax Error. Keyword '"+z+"' is already used as function/procedure name.
    ",column: " +(str(file_counters[4] - len(word))))
if i == -1:
    file.close()
    sys.exit("Syntax Error. Keyword '"+z+"' unidentified. Error at line: "+str(int((file
    ",column: " +(str(file_counters[4] - len(word))))
```

Σε αυτό το σημείο ελέγχουμε επίσης αν το αριστερό μέλος (η μεταβλητή) της εκχώρησης τιμής έχει δηλωθεί στο πρόγραμμα.

8) Σε αυτό το σημείο της varlist ελέγχουμε αν η κάθε μεταβλητή που δηλώνεται έχει δηλωθεί ξανά στο ίδιο βάθος.

```
if(token == "keywordtk"):

    i = main_obj.Search(word,None,main_obj.scope)
    if (i != -1):
        file.close()
        sys.exit("Variable name '"+word+"' already in use.
            ",column: " +(str(file_counters[4] - len(word))
```

9) Σε κάθε κλήση συνάρτησης ή διαδικασίας ελέγχουμε τις παραμέτρους που δίνει ο χρήστης ώστε να διασφαλίσουμε τον σωστό τύπο για κάθε όρισμα. Παράλληλα ελέγχουμε και τον αριθμό των παραμέτρων που έχουν δοθεί με τον αριθμό των παραμέτρων που “περιμένει” η συνάρτηση ή διαδικασία.

```
tmp_ar = []

for q in parameters:

    if q[2] == "CV" or q[2] == "REF":
        tmp_ar.append(q[2])

    nextquad()
    quadList.append(genquad(quads,q[0],q[1],q[2],""))

if(len(arguments) == len(tmp_ar)):
    t = 0
    for q in tmp_ar:
        if isinstance(arguments,(list,[])):
            if (arguments[t] == "in " and tmp_ar[t] != "CV") or (arguments[t] == "inout " and tmp_ar[t] != "REF"):
                file.close()
                sys.exit("Syntax Error. Missmatch arguments of funtion '"+function_name+"'. Error at line: "+str(int((file_count
        t += 1
elif(len(arguments) > len(tmp_ar)):
    file.close()
    sys.exit("Syntax Error. Arguments missing for function " + function_name + ". Excpected "+ str((len(arguments))) + " given "
else:
    file.close()
    sys.exit("Syntax Error. Too many arguments for function " + function_name + ". Excpected "+ str((len(arguments))) + " given
```

10) Όσον αφορά το πρόβλημα “ποια από τις δηλώσεις της a χρησιμοποιείται όταν γίνεται αναφορά σε αυτή” εκμεταλλευόμαστε τον πίνακα συμβόλων και βρίσκουμε την πιο πρόσφατη αναφορα στην εκάστοτε μεταβλητή μέσω της παρακάτω συνάρτησης:

```
def Search_Entity_backwards(self,name):
    for i in range(len(self.pinakas_symvolwn)-1,-1,-1):
        row = self.pinakas_symvolwn[i]
        for j in range(1,len(row)):
            if row[j].getName() == name:
                return row[j]
    else:
        return -1
```

Παραγωγή Ενδιάμεσου

Ο ενδιάμεσος κώδικας αποτελεί τη γλώσσα επικοινωνίας ανάμεσα στο εμπρόσθιο και οπίσθιο τμήμα ενός μεταγλωττιστή.
Ο χωρισμός σε εμπρόσθιο και ενδιάμεσο τμήμα επιτρέπει:

- Τη δημιουργία μεταγλωττιστών για νέες αρχιτεκτονικές με αλλαγή μόνο του οπίσθιου τμήματος.
- Τη βελτιστοποίηση στο επίπεδο μηχανής του ενδιάμεσου κώδικα.

Τυπικά ο ενδιάμεσος κώδικας παράγεται από μία μετάφραση κατευθυνόμενη προς τη σύνταξη.

Ο ενδιάμεσος κώδικας αποτελείται από ένα σύνολο αριθμημένων τετράδων με ένα τελεστή και τρια τελούμενα. Οι τετράδες εκτελούνται σειριακά με βάση το μοναδικό αριθμό που έχουν στη πρώτη τους θέση, εκτός αν μία τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό.

Υλοποίηση Ενδιάμεσου Κώδικα:

quads = -1 (Ο αύξων αριθμός ο οποίος καθορίζει την μοναδική ετικέτα της τετράδας)
quadList = list() (Η λίστα που αποτελείται από τις τετράδες κάθε προγράμματος)

Βοηθητικές Συναρτήσεις:

```
def nextquad():
    global quads
    quads += 1
    return quads

def genquad(index,op,x,y,z):
    quad = [index,op,x,y,z]
    return quad

def newtemp():
    global var_counter
    var = "T_"
    var = var + str(var_counter)
    var_counter += 1
    return var

def emptylist():
    emptylist = list()
    return emptylist
```

```
def makelist(x):
    tmp_list = list()
    tmp_list.append(x)
    return tmp_list

def merge(list1, list2):
    for x in list2:
        list1.append(x)
    release_list(list2)
    return list1

# delete list from memory
def release_list(a):
    del a[:]
    del a

def backpatch(list, z):
    global quadList
    for ptr_quad in list:
        quadList[ptr_quad][4] = z
```

nextquad(): Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.

genquad(): Δημιουργεί την επόμενη τετράδα που θα τοποθετηθεί στην λίστα.

newtemp(): Δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή

empylist(): Δημιουργεί μία κενή λίστα ετικετών τετράδων.

makelist(): Δημιουργεί μια λίστα τετράδων που περιέχει μόνο το όρισμα

merge(): Δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών

backpatch(): Επιστρέφει μια μια τις τετράδες της λίστας με συμπληρωμένο το τελευταίο πεδίο

Δημιουργία Τετράδων:

Πριν τη δημιουργία τετράδων κρατάμε το όνομα του προγράμματος σε μία μεταβλητή με μία κάτω παύλα στο τέλος της.

Αρχή και τέλος κάθε block

[quads,begin_block,name,_,_]
[quads,halt,_,_,_] (Δημιουργείται μόνο στο block της main και το ξεχωρίζουμε με τη συνθήκη if('_' in name). Η κάτω παύλα μπήκε σε περίπτωση που το όνομα του προγράμματος είναι ίδιο με το όνομα κάποιας συνάρτησης.)
[quads,end_block,name,_,_]

Τελεστές αριθμητικών πράξεων

[quads,op,x,y,z]
op: αριθμητική πράξη. Ένα εκ των +,-,*,/
x,y: Μπορεί να είναι όνομα μεταβλητών ή αριθμητικές σταθερές
z: Μπορεί να είναι όνομα μεταβλητής.

π.χ. a+b είναι [quads,+,a,b,T_var_counter]

Τελεστής εκχώρησης τιμής

[quads,:=,x,_,z]
Το τελούμενο x μπορεί να είναι όνομα μεταβλητής ή αριθμητική σταθερά
Το τελούμενο z μπορεί να είναι όνομα μεταβλητής

π.χ. a := b είναι [quads,:=,b,_,a]

Τελεστής άλματος χωρίς συνθήκης

[quads,jump,_,_,z]
Μεταπήδηση χωρίς όρους στη θέση z.

Τελεστής άλματος με συνθήκη

[quads,relop,x,y,z]
relop είναι ένα εκ των =, >, <>, >=,<=

π.χ.
100: =,a,4,120
110: jump,_,_,140
120: :=,1,_b
130: jump 150
140: :=,2,_,b
150: …

Συναρτήσεις και Διαδικασίες

[quads,par,x,m,_]
όπου x το όνομα της παραμέτρου και m ο τρόπος περάσματος
Τρόποι περάσματος παραμέτρου:
CV: πέρασμα με τιμή
REF: πέρασμα με αναφορά
RET: επιστροφή τιμής συνάρτησης

[quads,call,name,_,_]
Κλήση της συνάρτησης/διαδικασίας με όνομα name

[quads,ret,x,_,_]
Επιστροφή τιμής συνάρτησης

Παράδειγμα κλήσης συνάρτησης:
x := foo(in a, inout b)

100: par , a , cv, _
110: par , b , ref , _
120: par , T_1 , ret , _
130:call , foo, _ , _
140: :=, T_1, _ , x

Παράδειγμα κλήσης διαδικασίας:
call foo(in a, inout b)

100: par , a , cv, _
110: par , b , ref , _
120:call , foo, _ , _

Αριθμητικές Παραστάσεις

E -> T1 (+ T2{P1})* {P2}

{P1}: w = newTemp()
nextquad()
quadList.append(genquad(quads,"+",T1.place,T2.place,w))
T1.place = w
{P2}: e.place = T1.place
Το P1 βρίσκεται μέσα στο while. Το T1.place κρατάει το αποτέλεσμα της κάθε επανάληψης. Αν σταματήσει το while το e.place είναι πάλι το t1.place οπότε δε συμπληρώσαμε κάτι από κάτω εφόσον γίνεται αμέσως return.

```
def expression(file_counters):
    global word,token, quadList, quads ,main_obj

    sign = optionalSign(file_counters)


    t1 = term(file_counters)
    if sign != None:
        t1 = sign + str(t1)

    while(token == "addOperatortk"):
        op = word

        word, token = lexer(file_counters)
        print(word+" "+token)

        t2 = term(file_counters)

        w = newtemp()
        nextquad()
        quadList.append(genquad(quads,op,t1,t2,w))       #endiamesou
        t1 = w


        main_obj.change_offset()
        temp_var_obj = Temp_Variable(w,main_obj.offsets[main_obj.scope])
        main_obj.append(temp_var_obj)

    return t1
```

Αριθμητικές Παραστάσεις

T -> F1 (x F2{P1})* {P2}

{P1}: w = newTemp()
nextquad()
quadList.append(genquad(quads,”x”,F1.place,F2.place,w))
F1.place = w

{P2}: T.place = F1.place

Με την ίδια λογική γίνεται και η υλοποίηση του term

```
def term(file_counters):
    global word,token, quadList, quads , main_obj

    f1 = factor(file_counters)

    while(token == "mulOperatortk"):
        mul = word

        word, token = lexer(file_counters)
        print(word+" "+token)

        f2 = factor(file_counters)
        w = newtemp()
        nextquad()
        quadList.append(genquad(quads,mul,f1,f2,w))
        f1 = w


        main_obj.change_offset()
        temp_var_obj = Temp_Variable(w,main_obj.offsets[main_obj.scope])
        main_obj.append(temp_var_obj)

    return f1
```

Αριθμητικές Παραστάσεις

F -> (E) {P1}

{P1}: F.place = E.place

F -> id {P1}

{P1}: F.place=id.place

Στην περίπτωση που δεν θα δούμε κάποιον τελεστή θα επιστρέψει στο F.place το E.place και το id.place αντίστοιχα.

Λογικές Παραστάσεις - OR

B -> Q1 {P1} (or {P2} Q2 {P3})*

{P1}: B.true= Q1.true
B.false= Q1.false
{P2}: backpatch(B.false, nextquad())
{P3}: B.true= merge(B.true, Q2.true)
B.false= Q2.false

P1 : Δημιουργούμε τοπικά ένα πίνακα. Στη πρώτη θέση του πίνακα κρατάμε τα αποτελέσματα true του παρακάτω επιπέδου και στη δεύτερη τα false.

P2 : Συμπληρώνουμε τη δεύτερη θέση του πίνακα (τα false αποτελέσματα) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.

P3 : Ενώνουμε τις δυο true λίστες σε μια και τοποθετούμε στην B.false την Q2.false
ένα παράδειγμα αυτής της υλοποίησης βρίσκεται στη condition

```
def condition(file_counters):
    global word,token,quads

    Q_list = boolterm(file_counters)
    Q_list_true = Q_list[0]
    Q_list_false = Q_list[1]

    while(token == "ortk"):
        word, token = lexer(file_counters)
        print(word+" "+token)

        backpatch(Q_list[1],quads + 1)
        Q_list = boolterm(file_counters)
        Q_list_true = merge(Q_list_true, Q_list[0])
        Q_list_false = Q_list[1]

    tmp = list()
    tmp.append(Q_list_true)
    tmp.append(Q_list_false)

    release_list(Q_list)
    return tmp
```

Στις περιπτώσεις των OR κάνουμε backpatch τη λίστα με τα false διότι πρέπει να εξετάσουμε όλες τις επιμέρους συνθήκες για να αποφασίσουμε αν η συνθήκη θα είναι αληθής. Για να πάρουμε τη τετράδα που θα κάνουν jump (για true) θα πρέπει να φτάσουμε στο τέλος όλων των επιμέρους συνθηκων.

Λογικές Παραστάσεις - AND

Q->R1{P1}(and {P2}R2 {P3})*

{P1}: Q.true= R1.true
Q.false= R1.false
{P2}: backpatch(Q.true, nextquad())
{P3}: Q.false= merge(Q.false, R2.false)
Q.true= R2.true

P1 : Δημιουργούμε τοπικά ένα πίνακα. Στη πρώτη θέση του πίνακα κρατάμε τα αποτελέσματα true του παρακάτω επιπέδου και στη δεύτερη τα false.

P2 : Συμπληρώνουμε τη δεύτερη θέση του πίνακα (τα true αποτελέσματα) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.

P3 : Ενώνουμε τις δυο false λίστες σε μια και τοποθετούμε στην Q.true την R2.true
ένα παράδειγμα αυτής της υλοποίησης βρίσκεται στη boolterm

```
def boolterm(file_counters):
    global word,token,quads

    Q_list = boolfactor(file_counters)
    Q_list_true = Q_list[0]
    Q_list_false = Q_list[1]


    while(token == "andtk"):
        word, token = lexer(file_counters)
        print(word+" "+token)

        backpatch(Q_list[0],quads + 1)
        Q_list = boolfactor(file_counters)
        Q_list_false = merge(Q_list_false, Q_list[1])
        Q_list_true = Q_list[0]

    tmp = list()
    tmp.append(Q_list_true)
    tmp.append(Q_list_false)

    release_list(Q_list)
    return tmp
```

Στις περιπτώσεις των AND κάνουμε backpatch τη λίστα με τα true διότι πρέπει να είναι όλες οι επιμέρους συνθήκες true ώστε να αποτιμηθεί αληθής η συνθήκη. Για να πάρουμε τη τετράδα που θα κάνουν jump (για true) θα πρέπει να φτάσουμε στο τέλος όλων των επιμέρους συνθηκων.

Λογικές Παραστάσεις

R ->E1 relop E2 {P1}

{P1}: R.true=makelist(nextquad())
quadList.append(genquad(quads,relop, E1.place, E2.place, “_”))
R.false=makelist(nextquad())
quadList.append(genquad(quads,“jump” , “_” , “_” , “_”))

Δημιουργία μίας μη συμπληρωμένης τετράδας και ανάθεσή της στη λίστα των true .
Δημιουργία κατάλληλης τετράδας.
Δημιουργία μη συμπληρωμένης τετράδας και εισαγωγή της στη λίστα των false.
Δημιουργία μη συμπληρωμένης τετράδας για το jump

```
e1 = expression(file_counters)


if(token == "relOperatortk"):
    op = word
    word, token = lexer(file_counters)
    print(word+" "+token)

    e2 = expression(file_counters)

    R_true = makelist(nextquad())
    quadList.append(genquad(quads,op,e1,e2,""))
    R_false = makelist(nextquad())
    quadList.append(genquad(quads,"jump","","",""))

    tmp = list()
    tmp.append(R_true)
    tmp.append(R_false)
    return tmp
```

Κλήση Διαδικασίας

call assign_v(in a, inout b)

[quads,par, a, CV, _]
[quads,par, b, REF, _]
[quads, call, assign_v, _, _]

Κλήση συνάρτησης:

error = assign_v(in a, inout b)

[quads,par, a, CV, _]
[quads,par, b, REF, _]
w = newTemp()
[quads,par, w, RET, _]
[quads,call, assign_v, _, _]

Εντολή Return

S -> return (E) {P1}

{P1} : quadList.append(genquad(quads,“retv”,E.place,”_”,”_”))

Εκχώρηση τιμής

S-> id := E {P1};

{P1} : quadList.append(genquad(quads,“:=“,E.place,”_”,id))

Δομή While

S -> while {P1}B do {P2}S1{P3}

{P1}: Bquad:=nextquad()
{P2}: backpatch(B.true,nextquad())
{P3}: quadList.append(genquad(quads,“jump”,”_”,”_”,Bquad))
backpatch(B.false,nextquad())

P1: Στο Bquad κραταμε το νούμερο της επόμενης τετράδας που αποτελεί τη συνθήκη
P2 : Συμπληρώνουμε την πρώτη θέση του πίνακα (τα true αποτελέσματα) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.
P3: Φτιάχνουμε τετράδα ώστε να γίνει jump στη τετράδα που αντιστοιχεί στον έλεγχο της συνθήκης.Συμπληρώνουμε τη δεύτερη θέση του πίνακα (τα false αποτελέσματα) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump έξω από τη δομή.

```
def whileStat(file_counters):
    global word, token, whileflag,quadList, quads

    if(word == '('):

        word, token = lexer(file_counters)
        print(word+" "+token)

        Bquad = quads+1
        B = condition(file_counters)

        Btrue = B[0]
        Bfalse = B[1]

        if(word == ')'):
            word, token = lexer(file_counters)
            print(word+" "+token)

            whileflag += 1

            backpatch(Btrue,quads + 1)
            statements(file_counters)
            nextquad()
            quadList.append(genquad(quads,"jump","","",Bquad))
            backpatch(Bfalse,quads + 1)
```

Δομή If

S -> if B then {P1} S1{P2}TAIL {P3}

{P1}: backpatch(B.true,nextquad())
{P2}: ifList=makelist(nextquad())
quadList.append(genquad(quads,“jump”,”_”,”_”,”_”))
backpatch(B.false,nextquad())
{P3}: backpatch(ifList,nextquad())

P1: Συμπληρώνουμε την πρώτη θέση του πίνακα (τα true αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.
P2: Δημιουργία λίστας με την ετικέτα του jump. Δημιουργία κενής jump. Συμπληρώνουμε την δεύτερη θέση του πίνακα (τα false αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.
P3: Backpatch στη τετράδα που αναφέρεται η ifList για να διασφαλίσουμε ότι δεν θα μεταφερθούμε στις εντολες του else αν εχουν εκτελεστεί οι εντολές της if.

```
def ifStat(file_counters):
    global word, token,quadList,quads

    if(word == '('):

        word, token = lexer(file_counters)
        print(word+" "+token)

        B = condition(file_counters)

        Btrue = B[0]
        Bfalse = B[1]

        if(word == ')'):

            word, token = lexer(file_counters)
            print(word+" "+token)

            if(token == "declaretk" or token == "functiontk" or token == "proceduretk"):
                file.close()
                sys.exit("Syntax Error. Wrong program structure! The valid structure for
2. functions (if any) and 3. statements (if any). Error at line: "+str(int((file_counters
            ",column: " +(str(file_counters[4] - len(word))))

            backpatch(Btrue,quads + 1)
            statements(file_counters)
            ifList = makelist(nextquad())
            quadList.append(genquad(quads,"jump","","",""))
            backpatch(Bfalse,quads + 1)

            print(word+" "+token)
            elsepart(file_counters)
            backpatch(ifList,quads + 1)
```

Δομή forcase

S -> forcase {P1}
(when (condition) do {P2}
sequence {P3}
end do) *
endforcase

{P1}: p1Quad=nextquad()
{P2}: backpatch(cond.true,nextquad())
{P3}: quadList.append(genquad(quads,“jump”, ”_”, ”_”,p1quad))
backpatch(cond.false,nextquad())

P1: p1Quad είναι ένα σημάδι στη τετράδα που έχει το πρώτο condition
P2: Συμπληρώνουμε την πρώτη θέση του πίνακα (τα true αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.
P3: Αν μπει στα statement του condition συμπληρώνει μία τετράδα jump για το πρώτο condition. Συμπληρώνουμε την δεύτερη θέση του πίνακα (τα false αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.

```
def forcaseStat(file_counters):
    global word, token, quadList, quads

    if(token != "casetk"):
        file.close()
        sys.exit("Syntax Error. Keyword 'case' excpected here in order to start
        ",column: " +(str(file_counters[4] - len(word))))

    p1Quad = quads + 1

    while(token == "casetk"):
        word, token = lexer(file_counters)
        print(word+" "+token)

        if(word == '('):
            word, token = lexer(file_counters)
            print(word+" "+token)

            C = condition(file_counters)

            if(word == ')'):
                word, token = lexer(file_counters)
                print(word+" "+token)

                if(token == "declaretk" or token == "functiontk" or token == "pr
                    file.close()
                    sys.exit("Syntax Error. Wrong program structure! The valid s
2. functions (if any) and 3. statements (if any). Error at line: "+str(int((file_
            ",column: " +(str(file_counters[4] - len(word))))

                CondTrue = C[0]
                CondFalse = C[1]

                backpatch(CondTrue,quads + 1)

                statements(file_counters)

                nextquad()
                quadList.append(genquad(quads,"jump","","",p1Quad))
                backpatch(CondFalse,quads + 1)
```

Δομή Incase

S -> incase {P1}
(when (condition) do {P2}
sequence {P3}
end do) *
endincase {P4}

{P1}: w=newTemp()
p1Quad=nextquad()
quadList.append(genquad(quads,“:=“,1,”_”,w))
{P2}: backpatch(cond.true,nextquad())
quadList.append(genquad(quads,“:=“,0,”_”,w))
{P3}: backpatch(cond.false,nextquad())
{P4}: quadList.append(genquad(quads,“=”, w,0,p1quad))

```
def incaseStat(file_counters):
    global word, token, quadList, quads, main_obj

    if(token != "casetk"):
        file.close()
        sys.exit("Syntax Error. Keyword 'case' excpected here in order to star
        ",column: " +(str(file_counters[4] - len(word))))

    w = newtemp()
    p1Quad = nextquad()
    quadList.append(genquad(quads,":=","1","",w))


    main_obj.change_offset()
    temp_var_obj = Temp_Variable(w,main_obj.offsets[main_obj.scope])
    main_obj.append(temp_var_obj)

    while(token == "casetk"):
        word, token = lexer(file_counters)
        print(word+" "+token)

        if(word == '('):

            word, token = lexer(file_counters)
            print(word+" "+token)

            C = condition(file_counters)

            print(C)

            if(word == ')'):
                word, token = lexer(file_counters)
                print(word+" "+token)

                if(token == "declaretk" or token == "functiontk" or token == '
                    file.close()
                    sys.exit("Syntax Error. Wrong program structure! The valid
2. functions (if any) and 3. statements (if any). Error at line: "+str(int((fi
            ",column: " +(str(file_counters[4] - len(word))))

                CondTrue = C[0]
                CondFalse = C[1]

                backpatch(CondTrue,quads + 1)
                nextquad()
                quadList.append(genquad(quads,":=","0","",w))

                statements(file_counters)

                backpatch(CondFalse,quads + 1)
```

P1: Το w λειτουργεί ως ένα flag.Το p1Quad είναι ένα σημάδι στη τετράδα που έχει το πρώτο condition. Δημιουργία τετράδας για ανάθεση του flag σε μία τιμή.
P2: Συμπληρώνουμε την πρώτη θέση του πίνακα (τα true αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump. Δημιουργία τετράδας για ανάθεση του flag σε μία τιμή.
P3: Συμπληρώνουμε την δεύτερη θέση του πίνακα (τα false αποτελέσματα που έχει επιστρέψει η condition) για τα οποία γνωρίζουμε εκείνη τη στιγμή που θα πρέπει κάνουν jump.
P4: Με τη τετράδα αυτή εξασφαλίζει ότι εάν μπει στο statement κάποιας συνθήκης τότε ο έλεγχος θα μεταβεί ξανά στη P1.

Δομή Input

S -> input (id) {P1}

{P1}: quadList.append(genquad(quads,“inp”,id.place,”_”,”_”))

Δομή Print

S -> print (E) {P2}

{P2}: quadList.append(genquad(quads,“out”,E.place,”_”,”_”))

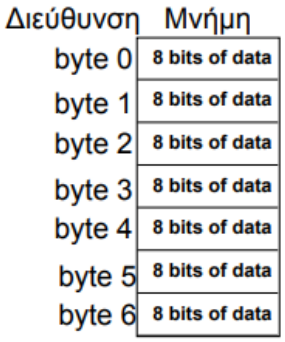
Παραγωγή Τελικού Κώδικα

Η τελική φάση στο μοντέλο του μεταγλωττιστή μας είναι η παραγωγή του τελικού κώδικα. Δέχεται ως είσοδο την ενδιάμεση αναπαράσταση (η οποία παράγεται από το εμπρόσθιο τμήμα του μεταγλωττιστή)και μαζί με τις σχετικές πληροφορίες που λαμβάνει από το πίνακα συμβόλων, παράγει ένα σημασιολογικά ισοδύναμο τελικό κώδικα.

Σημαντικό είναι γίνεται αποτελεσματική χρήση των πόρων της εκάστοτε μηχανής στόχου (συγκεκριμένα στον MIPS). Σημαντικό ρόλο πέρα από τη σωστή ανάθεση των πόρων είναι η επιλογή των κατάλληλων εντολών αλλά και η σωστή διάταξη αυτών μεταξύ τους.

Λίγα λόγια για τον Mips:

- Η μνήμη θεωρείται ως ένας μεγάλος, μονοδιάστατος πίνακας λέξεων/bytes/bits με συγκεκριμένη αρχική διεύθυνση (base address).
- Για να έχει πρόσβαση στη μνήμη μια εντολή πρέπει να μπορεί να παρέχει τη «διεύθυνση» των ζητούμενων δεδομένων.
- Μια διεύθυνση μνήμης (memory address) καθορίζεται με ένα δείκτη στον πίνακα (0, 1, 2,…).
- «Διευθυνσιοδότηση με Byte» (Byte addressing), δηλαδή:

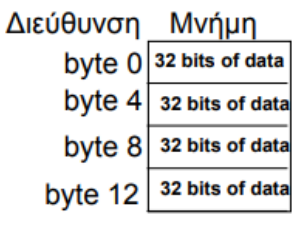


π.χ. Θεωρήστε ένα πίνακα A με 100 λέξεις (όπου 1 λέξη = 1 byte), με g -> $s1, h -> $s2, και ο καταχωρητής αρχικής διεύθυνσης (base register) -> $s3

```
g = h + A[5];
lw $t0, 5($s3)          #lw = load word
add $s1, $s2, $t0
```

base register: έχει την αρχική διεύθυνση του πίνακα, δηλ. τη διεύθυνση του 1ου στοιχείου (λέξης) του πίνακα.
offset: ο δείκτης στον πίνακα, σταθερός στην εντολή
διεύθυνση = base register + offset

- Συνήθως μια «λέξη» (word) είναι μεγαλύτερη από 1 byte.
- Στη MIPS, οι καταχωρητές μπορούν να κρατήσουν μέχρι και 32 bits, άρα, μια λέξη έχει 32 bits ή 4 bytes.



Byte address της 1ης λέξης = 4 (= 1 x 4)
Byte address της 2ης λέξης = 8 (= 2 x 4)
…
Byte address της ηης λέξης = n x 4

Για το παραπάνω παράδειγμα (g = h + A[5];)

```
lw $t0, 20($s3)          # byte address του offset είναι 5x4=20S
add $s1, $s2, $t0
```

Οι καταχωρητές που έχει ο Mips είναι:

καταχωρητές προσωρινών τιμών:	$t0…$t7
καταχωρητές οι τιμές των οποίων διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων:	$s0…$s7
καταχωρητές ορισμάτων:	$a0…$a3
καταχωρητές τιμών:	$v0,$v1
stack pointer	$sp
frame pointer	$fp
return address	$ra

Οι εντολές που χρησιμοποιούμε για τις αριθμητικές πράξεις είναι:

Πρόσθεση	add $t0,$t1,$t2	t0=t1+t2
Αφαίρεση	sub $t0,$t1,$t2	t0=t1-t2
Πολλαπλασιασμός:	mul $t0,$t1,$t2	t0=t1*t2
Διαίρεση:	div $t0,$t1,$t2	t0=t1/t2

Οι εντολές που χρησιμοποιούμε για μετακίνηση δεδομένων:

move	$t0,$t1	t0=t1 μεταφορά ανάμεσα σε καταχωρητές
li	$t0, value	t0=value σταθερά σε καταχωρητή
lw	$t1,mem	t1=[mem] περιεχόμενο μνήμης σε καταχωρητή
sw	$t1,mem	[mem]=t1 περιεχόμενο καταχωρητή σε μνήμη
lw	$t1,($t0)	t1=[t0] έμμεση αναφορά με καταχωρητή
sw	$t1,-4($sp)	t1=[$sp-4] έμμεση αναφορά με βάση τον $sp

move:	μετακίνηση τιμής ενός καταχωρητή σε έναν άλλο
li:	φόρτωσε τη τιμή value σε ένα καταχωρητή (είναι μόνο για ακέραιους)
lw:	φόρτωσε στον καταχωρητή τιμή από την μνήμη
	φόρτωσε στον καταχωρητή την τιμή που βρίσκεται σε έναν άλλο καταχωρητή
sw:	φόρτωσε στην μνήμη την τιμή του καταχωρητή
	φόρτωσε την τιμή του καταχωρητή σε θέση βασισμένη στον $sp

Οι εντολές που χρησιμοποιούμε για άλματα:

b label	branch to label	
beq $t1,$t2,label	jump to label if $t1=$t2	(branch if equal)
blt $t1,$t2,label	jump to label if $t1<$t2	(branch if less than)

bgt $t1,$t2,label	jump to label if $t1>$t2	(branch if greater than)
ble $t1,$t2,label	jump to label if $t1<=$t2	(branch if less/equal than)
bge $t1,$t2,label	jump to label if $t1>=$t2	(branch if greater/equal than)
bne $t1,$t2,label	jump to label if $t1<>$t2	(branch if not equal)

Οι εντολές που χρησιμοποιούμε στην κλήση συναρτήσεων:

j label	jump to label	
jal label	κλήση συνάρτησης	(jump and link)

- Η έννοια του αποθηκευμένου προγράμματος υπονοεί ότι η διεύθυνση της εντολής που εκτελείται είναι πάντα γνωστή
- Φυλάγεται σε ένα καταχωρητή που ονομάζεται Μετρητής Προγράμματος (Program Counter -- PC)
- jal αποθηκεύει PC + 4 στον $ra.

> Το jal αποθηκεύει τη διεύθυνση επιστροφής (η διεύθυνση της επόμενης εντολής) στον ειδικό καταχωρητή $ ra, πριν μεταβεί στη συνάρτηση.
> Το jal είναι η μόνη εντολή MIPS που μπορεί να έχει πρόσβαση στην τιμή του μετρητή προγράμματος, ώστε να μπορεί να αποθηκεύσει τη διεύθυνση επιστροφής PC + 4 στον $ ra.

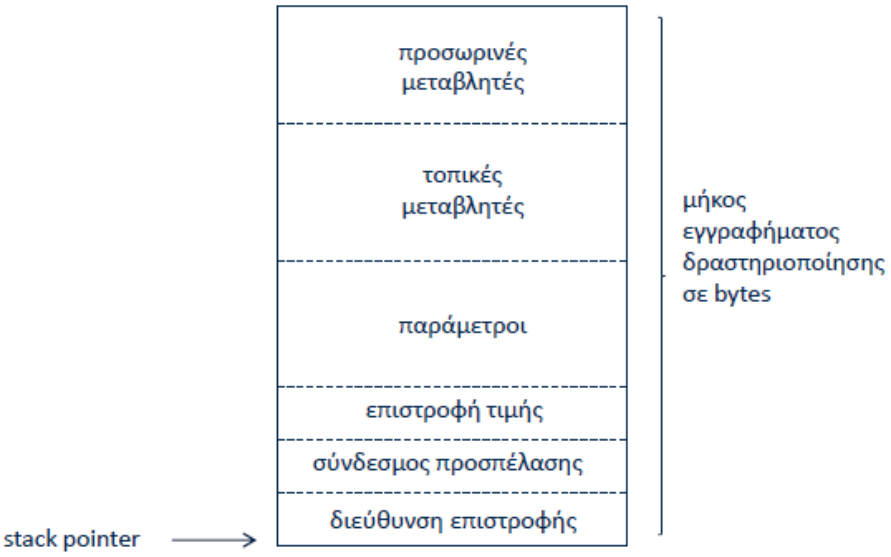
Πως επιστρέφουμε στο κύριο πρόγραμμα από μια συνάρτηση?

jr $ra	(return address) jump register (άλμα χωρίς συνθήκη) στη διεύθυνση που έχει ο καταχωρητής, στο παράδειγμα είναι ο $ra που έχει την διεύθυνση επιστροφής συνάρτησης

Στην επιστήμη της πληροφορικής, η **στοίβα κλήσεων** (**call stack**) είναι μια δομή δεδομένων στοίβας που κρατά πληροφορίες σχετικά με τις ενεργές υπορουτίνες ενός προγράμματος. Αυτός ο τύπος στοίβας είναι γνωστός και σαν **στοίβα εκτέλεσης** (**execution stack**), **στοίβα ελέγχου** (**control stack**), **στοίβα συναρτήσεων** (**function stack**), ή **στοίβα χρόνου εκτέλεσης** (**run-time stack**), και συχνά αναφέρεται απλά σαν «η στοίβα». Αν και η συντήρηση της στοίβας κλήσεων είναι σημαντική για τη σωστή λειτουργία των πιο πολλών προγραμμάτων, οι λεπτομέρειες συνήθως είναι αόρατες στις γλώσσες υψηλού επιπέδου.

> Μια στοίβα κλήσεων αποτελείται από **πλαίσια στοίβας** (**stack frames**), γνωστά και ως **εγγραφές δραστηριοποίησης** (**activation records**) ή **εγγραφές δραστηριοποίησης** (**activation frames**). Αυτά είναι δομές δεδομένων ανεξάρτητες από την αρχιτεκτονική του υπολογιστή που περιλαμβάνουν πληροφορίες σχετικά με την κατάσταση της κάθε υπορουτίνας. Κάθε πλαίσιο στοίβας αντιστοιχεί σε μια κλήση σε μια υπορουτίνα που δεν έχει ακόμα επιστρέψει. Για παράδειγμα, αν μια υπορουτίνα `DrawLine` εκτελείται, έχοντας μόλις κληθεί από μια υπορουτίνα `DrawSquare`, η κορυφή της στοίβας κλήσεων θα έχει την εξής διάταξη (όπου η στοίβα μεγαλώνει προς την κορυφή):

Το πλαίσιο στοίβας (εγγράφημα δραστηριοποίησης) στην κορυφή της στοίβας αντιστοιχεί στη ρουτίνα που εκτελείται αυτήν τη στιγμή. Ένα γράφημα δραστηριοποίησης περιέχει:



- Προσωρινές Μεταβλητές: όπως αυτές που προκύπτουν από τον υπολογισμό εκφράσεων, σε περιπτώσεις που οι προσωρινές τιμές δε μπορούν να αποθηκευτούν σε καταχωρητές.
- Τοπικές Μεταβλητές: μεταβλητών δηλαδή που είναι γνωστές μόνο μέσα στα όρια της ενεργής υπορουτίνας και δεν κρατούν τις τιμές τους μετά την επιστροφή από αυτή.
- Παράμετροι: οι τρέχουσες παράμετροι που χρησιμοποιούνται από τη καλούσα διαδικασία.
- Επιστροφή Τιμής: χώρος για τη τιμή επιστροφής της καλούμενης συνάρτησης.
- Σύνδεσμος Προσπέλασης: για να εντοπίσει δεδομένα που καλούνται από την κληθείσα διαδικασία αλλά βρίσκονται αλλού π.χ. σε μία άλλη εγγραφή δραστηριοποίησης.
- Διεύθυνση επιστροφής: Όταν καλείται μια υπορουτίνα, η θέση της εντολής στην οποία θα επιστρέψει πρέπει να αποθηκευτεί κάπου.

Στην αρχιτεκτονική το offset είναι ένας ακέραιος αριθμός ο οποίος υποδεικνύει την απόσταση μεταξύ ενός αντικειμένου και ενός σημείου ή στοιχείου που έχει δοθεί. Η έννοια της απόστασης είναι έγκυρη μόνο αν όλα τα στοιχεία έχουν το ίδιο μέγεθος (bytes στην περίπτωση μας).
Στη αρχιτεκτονική υπολογιστών και στον προγραμματισμό χαμηλού επιπέδου (όπως η γλώσσα assembly), ένα offset υποδηλώνει συνήθως τον αριθμό των θέσεων διευθύνσεων που προστίθενται σε μια βασική διεύθυνση για να φτάσουμε σε μια συγκεκριμένη απόλυτη διεύθυνση. Σε αυτήν την έννοια του offset, χρησιμοποιείται μόνο η βασική μονάδα διευθύνσεων, συνήθως το 8-bit byte, για τον καθορισμό του μεγέθους του offset. Σε αυτό το πλαίσιο, ένα offset ονομάζεται μερικές φορές μια σχετική διεύθυνση.

Η πρόσβαση στη στοίβα συχνά γίνεται μέσω ενός καταχωρητή που ονομάζεται **δείκτης στοίβας** (**stack pointer**) και ενός offset, ο οποίος και επίσης χρησιμοποιείται για να δείχνει την κορυφή

της στοίβας. Εναλλακτικά, η μνήμη μέσα στο πλαίσιο μπορεί να προσπελαστεί μέσω ενός ξεχωριστού καταχωρητή, που συχνά ονομάζεται **δείκτης πλαισίου** (**frame pointer**), ο οποίος συνήθως δείχνει σε μια σταθερή θέση στη δομή του πλαισίου, όπως η θέση της διεύθυνσης επιστροφής.

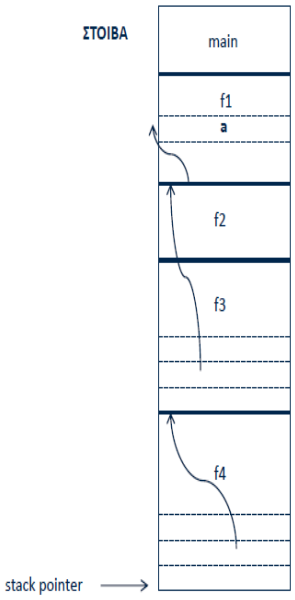
Τα πλαίσια στοίβας δεν έχουν πάντα το ίδιο μέγεθος. Διαφορετικές υπορουτίνες έχουν διαφορετικό αριθμό παραμέτρων, επομένως μέρος του πλαισίου στοίβας θα είναι διαφορετικό για διαφορετικές υπορουτίνες, αν και συνήθως σταθερό για όλες τις ενεργοποιήσεις μιας συγκεκριμένης υπορουτίνας.

Βοηθητικές Συναρτήσεις

- gnvlcode

- μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής
- από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα πάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει

```
lw $t0,-4($sp)              στοίβα του γονέα
όσες φορές χρειαστεί:
        lw $t0,-4($t0)      στοίβα του προγόνου που έχει τη μεταβλητή
addi $t0,$t0,-offset        διεύθυνση της μη τοπικής μεταβλητής
```



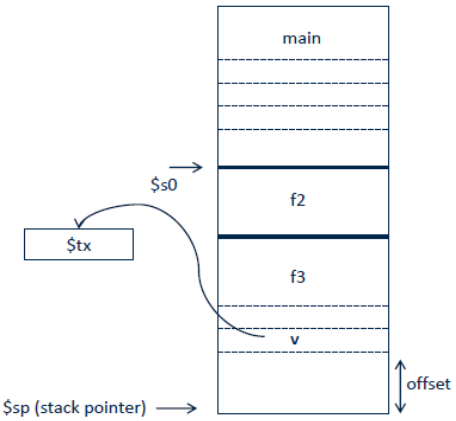

```
def gnvlcode(entity):
    global asm_file, main_obj

    temp_scope = main_obj.scope - 1

    asm_file.write("\tlw $t0, -4($sp)\n")

    while ((main_obj.Search(entity.getName(),None,temp_scope) == -1) and (temp_scope >= 0)):
        print(main_obj.Search(entity.getName(),None,temp_scope))
        asm_file.write("\tlw $t0, -4($t0)\n")
        temp_scope -= 1
    asm_file.write("\taddi $t0, $t0, -%s\n" %entity.offset)
```

- Loadvr

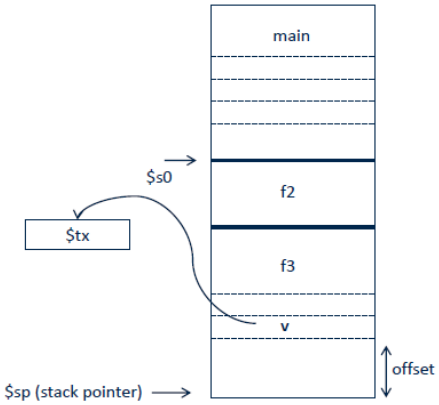
- μεταφορά δεδομένων στον καταχωρητή r
- η μεταφορά μπορεί να γίνει από τη μνήμη (στοίβα)
- ή να εκχωρηθεί στο r μία σταθερά
- η σύνταξη της είναι loadvr(v,r)

Διακρίνουμε περιπτώσεις

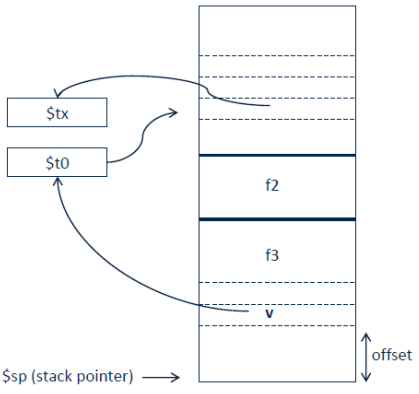
- αν v είναι σταθερά
  li $tr,v
- αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα
  lw $tr,-offset($s0)



- αν η v έχει δηλωθεί στη συνάρτηση που αυτή τη στιγμή εκτελείται και είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή, ή προσωρινή μεταβλητή
  lw $tr,-offset($sp)



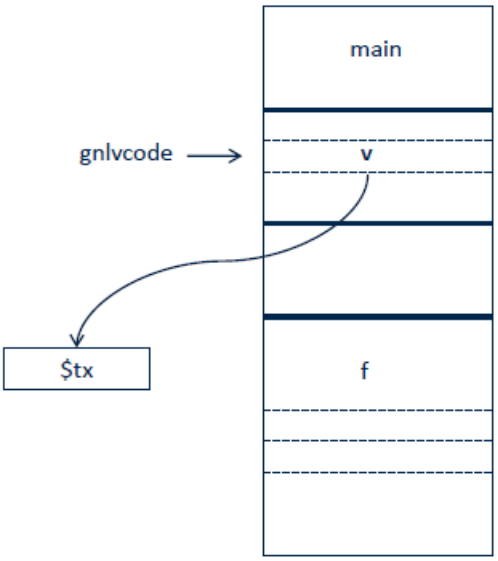
- αν η v έχει δηλωθεί στη συνάρτηση που αυτή τη στιγμή εκτελείται και είναι τυπική παράμετρος που περνάει με αναφορά
  lw $t0,-offset($sp)
  lw $tr,($t0)



$sp (stack pointer)

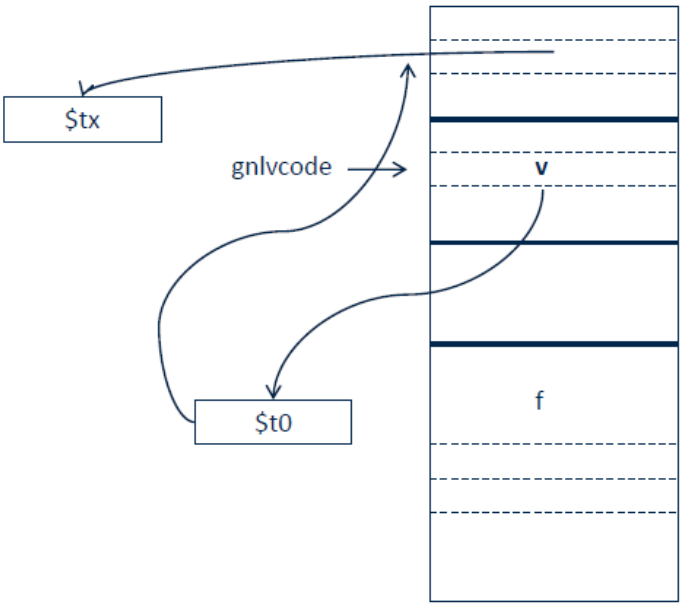
- αν η v έχει δηλωθεί σε κάποιο πρόγονο και εκεί είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή

  gnvlcode()
  lw $tr,($t0)



- αν η v έχει δηλωθεί σε κάποιο πρόγονο και εκεί είναι τυπική παράμετρος που περνάει με αναφορά

  gnvlcode()
  lw $t0,($t0)
  lw $tr,($t0)



```
def loadvr(v,r):
    global asm_file, main_obj

    if '-' in v:
        print(v)
        v = v.replace('-',"")
        print(v)

    if(v.isdigit()):
        asm_file.write('\tli $%s, %s\n' %(r, v))
        return

    scope = main_obj.Search_scope(v)
    if scope == -1:
        sys.exit("In loadvr quad, something unexpected happened. Program exits...")
    of = main_obj.Search_offset(v,scope)
    entity = main_obj.Search_Entity_backwards(v)


    if (scope == 0 and isinstance(entity,Variable)):
        asm_file.write("\tlw $%s, -%s($s0)\n" %(r,of))

    elif ( (main_obj.scope == scope and isinstance(entity,Variable)) or
        (main_obj.scope == scope and isinstance(entity,Parameter) and (entity.parMode == 'CV') ) or
        (main_obj.scope == scope and isinstance(entity,Temp_Variable) )):
        asm_file.write("\tlw $%s, -%s($sp)\n" %(r,of))

    elif (main_obj.scope == scope and isinstance(entity,Parameter) and (entity.parMode == 'REF')):
        asm_file.write("\tlw $t0, -%s($sp)\n" %of)
        asm_file.write("\tlw $%s, ($t0)\n" %r)

    elif ( main_obj.scope != scope and scope != 0 ):
        if ( (isinstance(entity,Variable)) or (isinstance(entity,Parameter) and (entity.parMode == 'CV'))) :
            gnvlcode(entity)
            asm_file.write("\tlw $%s, ($t0)\n" %r)

        elif isinstance(entity,Parameter) and (entity.parMode == 'REF'):
            gnvlcode(entity)
            asm_file.write("\tlw $t0, ($t0)\n")
            asm_file.write("\tlw $%s, ($t0)\n" %r)
```

- Storerv

  - μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v)
  - η σύνταξη της είναι storerv(r,v)

  Διακρίνουμε περιπτώσεις

  - αν v είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα
    sw $tr,-offset($s0)
  - αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή
    sw $tr,-offset($sp)
  - αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον
    lw $t0,-offset($sp)
    sw $tr,($t0)

- αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον
  gnvlcode(v)
  sw $tr,($t0)
- αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον
  gnvlcode(v)
  lw $t0,($t0)
  sw $tr,($t0)

```
def storerv(r,v):       # r = kataxwrhths, v = sth mnhmh
    global asm_file, main_obj

    scope = main_obj.Search_scope(v)
    if scope == -1:
        sys.exit("6.Something unexpected happened. Program exits...")
    of = main_obj.Search_offset(v,scope)
    entity = main_obj.Search_Entity_backwards(v)

    if (scope == 0 and isinstance(entity, Variable)):
        asm_file.write("\tsw $%s, -%s($s0)\n" %(r,of))
    elif ( (main_obj.scope == scope and isinstance(entity,Variable)) or
        (main_obj.scope == scope and isinstance(entity,Parameter) and (entity.parMode == 'CV') ) or
        (main_obj.scope == scope and isinstance(entity,Temp_Variable)) ):
        asm_file.write("\tsw $%s, -%s($sp)\n" %(r,of))
    elif (main_obj.scope == scope and isinstance(entity,Parameter) and (entity.parMode == 'REF')):
        asm_file.write("\tlw $t0, -%s($sp)\n" %of)
        asm_file.write("\tsw $%s, ($t0)\n" %r)
    elif ( main_obj.scope != scope and scope != 0 ):
        if ( (isinstance(entity,Variable)) or (isinstance(entity,Parameter) and (entity.parMode == 'CV'))) :
            gnvlcode(entity)
            asm_file.write("\tsw $%s, ($t0)\n" %r)
        elif isinstance(entity,Parameter) and (entity.parMode == 'REF'):
            gnvlcode(entity)
            asm_file.write("\tlw $t0, ($t0)\n")
            asm_file.write("\tsw $%s, ($t0)\n" %r)
```

Εντολές Αλμάτων

- jump, “_”, “_”, label
  b label

```
if quad[1] == "jump":
    asm_file.write("L_%s: b L_%s\n" %((quad[0]+1),(quad[4]+1)) )
    parameters = 0
```

  Δημιουργία εντολής branch χωρίς συνθήκη

- relop(?),x,y,z
  loadvr(x,$t1)
  loadvr(y, $t2)
  branch(?),$t1,$t2,z branch(?) : beq,bne,bgt,blt,bge,ble

```
elif quad[1] in "<=>=<>":
    asm_file.write("L_%s: \n" %(quad[0]+1) )
    loadvr(quad[2],"t1")
    loadvr(quad[3],"t2")
    if quad[1] == "=":
        asm_file.write("\tbeq $t1, $t2, L_%s\n" %(quad[4]+1) )
    elif quad[1] == "<":
        asm_file.write("\tblt $t1, $t2, L_%s\n" %(quad[4]+1) )
    elif quad[1] == ">":
        asm_file.write("\tbgt $t1, $t2, L_%s\n" %(quad[4]+1) )
    elif quad[1] == "<=":
        asm_file.write("\tble $t1, $t2, L_%s\n" %(quad[4]+1) )
    elif quad[1] == ">=":
        asm_file.write("\tbge $t1, $t2, L_%s\n" %(quad[4]+1) )
    elif quad[1] == "<>":
        asm_file.write("\tbne $t1, $t2, L_%s\n" %(quad[4]+1) )
    parameters = 0
```

Χρήση της loadvr για τις δύο μεταβλητές και δημιουργία κατάλληλης εντολής branch με συνθήκη

Εντολή Εκχώρησης

- :=, x, “_”, z

  loadvr(x, $t1)
  storerv($t1, z)

```
elif quad[1] == ":=":
    asm_file.write("L_%s: \n" %(quad[0]+1) )
    loadvr(quad[2],"t1")
    storerv("t1",quad[4])
    parameters = 0
```

Οι δύο παραπάνω βοηθητικές συναρτήσεις καλούνται κάθε φορά που έχουμε το σύμβολο της εκχώρησης.

Εντολές Αριθμητικών Πράξεων

op x,y,z

loadvr(x, $t1)
loadvr(y, $t2)
op $t1,$t1,$t2 op: add,sub,mul,div
storerv($t1,z)

```
elif quad[1] in "+-*/":
    asm_file.write('L_%s: \n' %(quad[0]+1) )

    loadvr(quad[2],"t1")
    loadvr(quad[3],"t2")
    if quad[1] == "+":
        asm_file.write("\tadd $t1,$t1,$t2\n")
    if quad[1] == "-":
        asm_file.write("\tsub $t1,$t1,$t2\n")
    if quad[1] == "*":
        asm_file.write("\tmul $t1,$t1,$t2\n")
    if quad[1] == "/":
        asm_file.write("\tdiv $t1,$t1,$t2\n")
    storerv("t1",quad[4])
    parameters = 0
```

Οι δύο παραπάνω βοηθητικές συναρτήσεις καλούνται κάθε φορά που έχουμε εντολή αριθμητικών πράξεων καθώς πρέπει να αναζητήσουμε το που βρίσκονται αυτές οι μεταβλητές.

Εντολές Εισόδου- Εξόδου

out “_”, “_”, x

li $v0,1
loadvr(x,$a0)
syscall

```
elif quad[1] == "out":
    asm_file.write('L_%s:\n' %(quad[0]+1) )
    asm_file.write("\tli $v0, 1\n")
    loadvr(quad[2],"a0")
    asm_file.write("\tsyscall\n")

    # this code is for printing newline
    asm_file.write("\tli $v0, 4\n")
    asm_file.write("\tla $a0, newline\n")
    asm_file.write("\tsyscall\n")
    parameters = 0
    #---------
```

in “_”, “_”, x

li $v0,5
syscall
storerv($v0,x)

```
elif quad[1] == "inp":
    asm_file.write('L_%s: \n' %(quad[0]+1) )
    asm_file.write("\tli $v0, 5\n")
    asm_file.write("\tsyscall\n")
    storerv("v0",quad[2])
    parameters = 0
```

Εντολή Επιστροφής Τιμής Συνάρτησης

retv “_”, “_”, x

loadvr(x, $t1)
lw $t0,-8($sp)
sw $t1,($t0)
lw $ra, -0($sp)
jr $ra
αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3η θέση του εγγραφήματος δραστηριοποίησης

```
elif quad[1] == "retv":
    asm_file.write('L_%s: \n' %(quad[0]+1) )
    loadvr(quad[2],"t1")
    asm_file.write("\tlw $t0, -8($sp)\n")
    asm_file.write("\tsw $t1, ($t0)\n")
    asm_file.write("\tlw $ra, -0($sp)\n")
    asm_file.write("\tjr $ra\n")

    parameters = 0
```

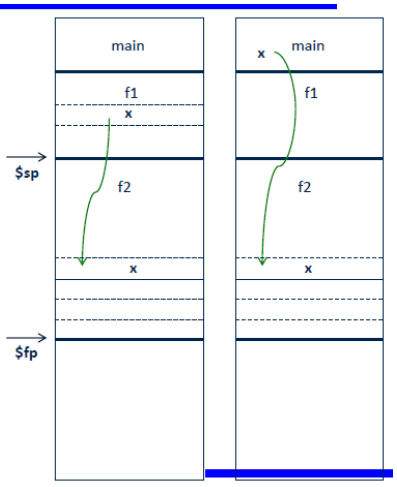
Παράμετροι Συνάρτησης

Πριν κάνουμε τις ενέργειες ώστε να γίνει το πέρασμα της πρώτης παραμέτρου, τοποθετούμε τον $fp να δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί addi $fp,$sp,framelength

```
elif quad[1] == "par":
    asm_file.write('L_%s: \n' %(quad[0]+1) )

    temp_fr = 0
    ofs = (12 + (4*parameters))
    function_name = ""
    for i in range(quad[0] + 1,len(quadList)):
        if quadList[i][1] == "call":
            function_name = quadList[i][2]
            break
    fr =  main_obj.Search_Entity_backwards(function_name).framelength
    temp_fr = fr

    if parameters == 0:
        asm_file.write("\taddi $fp, $sp, %s\n" %fr)
```

στη συνέχεια, για κάθε παράμετρο και ανάλογα με το αν περνά με τιμή ή αναφορά κάνουμε τις εξής ενέργειες:

- par,x,CV, _
  loadvr(x, $t0)
  sw $t0, -(12+4i)($fp)
  όπου i ο αύξων αριθμός της παραμέτρου



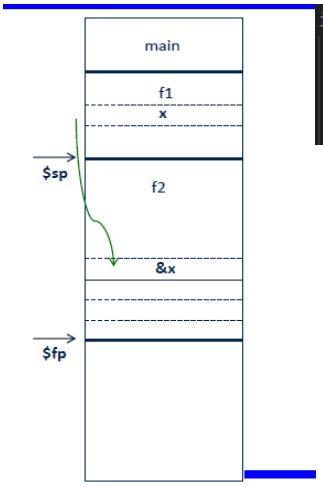

```
if quad[3] == "CV":
    loadvr(quad[2],"t0")
    asm_file.write("\tsw $t0, -%s($fp)\n" %ofs)
```

- par,x,REF, _

```
elif quad[3] == "REF":

    functions_Scope = main_obj.Search_scope(quadList[x1][2])  ######

    scope = main_obj.Search_scope(quad[2])
    if scope == -1:
        sys.exit("In ref parameter, something unexpected happened. Program exits...")
    of = main_obj.Search_offset(quad[2],scope)
    entity = main_obj.Get_Entity(scope,of)      #       !!!!!!!!!!!!!!    #
```

1) Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή

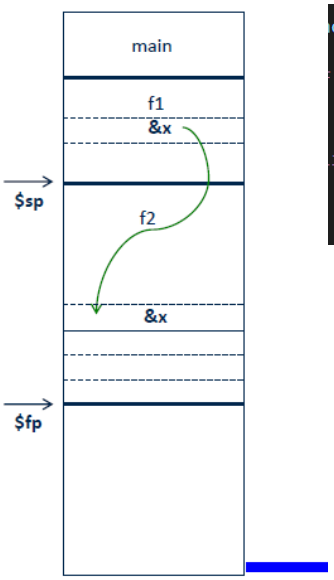
addi $t0,$sp,-offset
sw $t0,-(12+4i)($fp)




```
if functions_Scope == scope:       # scope synarthshs = scope metavlhths

    if isinstance(entity,Variable) or (isinstance(entity,Parameter) and (entity.parMode == 'CV')):
        asm_file.write("\taddi $t0, $sp,-%s\n" %of)
        asm_file.write("\tsw $t0, -%s($fp)\n" %ofs)
```

2) Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά

lw $t0,-offset($sp)
sw $t0,-(12+4i)($fp)

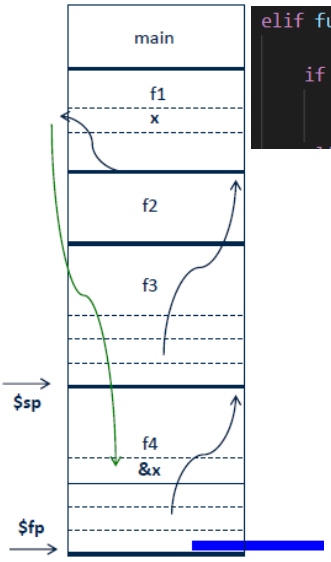



```
ctions_Scope == scope:       # scope synarthshs = scope metavlhths

 isinstance(entity,Variable) or (isinstance(entity,Parameter) and (entity.parMode == 'CV')):
  asm_file.write("\taddi $t0, $sp,-%s\n" %of)
  asm_file.write("\tsw $t0, -%s($fp)\n" %ofs)
if (isinstance(entity,Parameter) and (entity.parMode == 'REF')):
  asm_file.write("\tlw $t0, -%s($sp)\n" %of)
  asm_file.write("\tsw $t0, -%s($fp)\n" %ofs)
```

Η δεύτερη περίπτωση αναφέρεται στο elif.

3) Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή

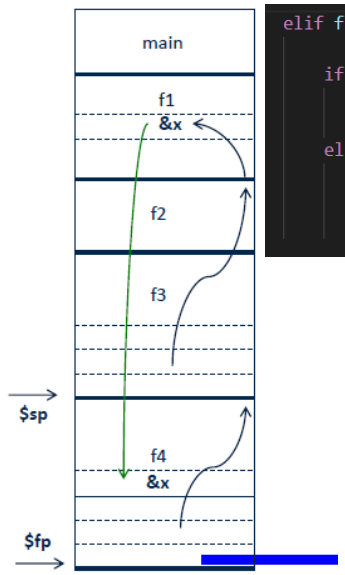
gnvlcode(x)
sw $t0,-(12+4i)($fp)




```
elif functions_Scope != scope:

    if isinstance(entity,Variable) or (isinstance(entity,Parameter) and (entity.parMode == 'CV')):
        gnvlcode(entity)
        asm_file.write("\tsw, $t0, -%s($fp)\n" %ofs)
```

4) Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά

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



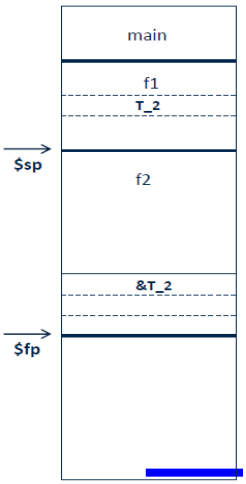

```
elif functions_Scope != scope:

    if isinstance(entity,Variable) or (isinstance(entity,Parameter) and (entity.parMode == 'CV')):
        gnvlcode(entity)
        asm_file.write("\tsw, $t0, -%s($fp)\n" %ofs)
    elif (isinstance(entity,Parameter) and (entity.parMode == 'REF')):
        gnvlcode(entity)
        asm_file.write("\tlw $t0, ($t0)\n")
        asm_file.write("\tsw $t0, -%s($fp)\n" %ofs)
```

Η τέταρτη περίπτωση αναφέρεται στο elif

- par,x,RET, _

  Γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή

  ```
  addi $t0,$sp,-offset
  sw $t0,-8($fp)
  ```




```
elif quad[3] == "RET":
    scope = main_obj.Search_scope(quad[2])
    if scope == -1:
        sys.exit("In ret parameter, something unexpected happened. Program exits...")
    of = main_obj.Search_offset(quad[2],scope)

    asm_file.write("\taddi $t0, $sp, -%s\n" %of)
    asm_file.write("\tsw $t0,-8($fp)\n")
```

Κλήση Συνάρτησης

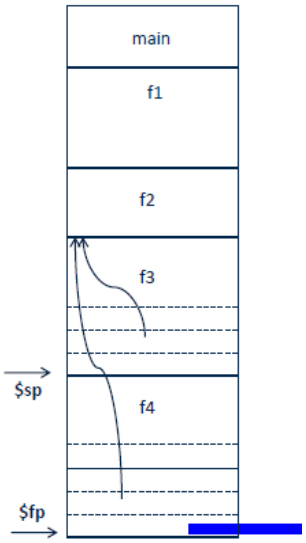
- call,_,_,f

```
elif quad[1] == "call":
    asm_file.write('L_%s: \n' %(quad[0]+1) )

    if parameters == 0:
        fr =  main_obj.Search_Entity_backwards(quad[2]).framelength
        asm_file.write("\taddi $fp, $sp, %s\n" %fr)

    t = quadList[x1]
    kalousa = t[2]
    klhtheisa = quad[2]

    kalousa_scope = main_obj.Search_scope(kalousa)
    if kalousa_scope == -1:
            sys.exit("In call quad, something unexpected happened. Program exits...")
    klhtheisa_scope = main_obj.Search_scope(klhtheisa)
    klhtheisa_scope += 1   # +1 giati th vriskei sto scope ths kalousa. Ara afth einai +1 panw.
    if klhtheisa_scope == -1:
            sys.exit("In call quad, something unexpected happened. Program exits...")
```

Αρχικά γεμίζουμε το 2ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης, τον σύνδεσμο προσπέλασης, με την διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της, ώστε η κληθείσα να γνωρίζει που να κοιτάξει αν χρειαστεί να προσπελάσει μία μεταβλητή την οποία έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει.

1) Αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα:

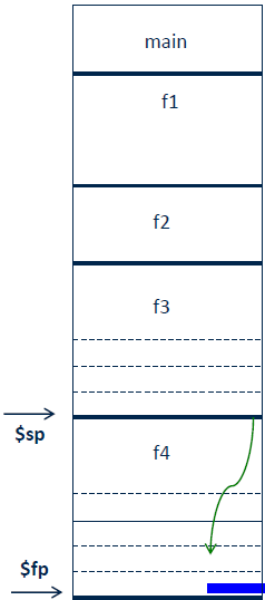
lw $t0,-4($sp)
sw $t0,-4($fp)




```
if kalousa_scope == klhtheisa_scope:
    asm_file.write("\tlw $t0, -4($sp)\n")
    asm_file.write("\tsw $t0, -4($fp)\n")
```

2) Αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας

sw $sp,-4($fp)




```
elif kalousa_scope != klhtheisa_scope:
    asm_file.write("\tsw $sp, -4($fp)\n")
```

Στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα
addi $sp,$sp,framelength
Καλούμε τη συνάρτηση: jal f
Και όταν επιστρέψουμε παίρνουμε πίσω
τον δείκτη στοίβας στην καλούσα:
addi $sp,$sp,-framelength

```
ent = main_obj.Search_Entity_backwards(quad[2])
fr =  ent.framelength
begin_quad = ent.start_quad
asm_file.write("\taddi $sp, $sp, %s\n" %fr)
asm_file.write("\tjal L_%s\n" %(begin_quad))
asm_file.write("\taddi $sp, $sp, -%s\n" %fr)

parameters = 0
```

Μέσα στην κληθείσα

- Στην αρχή κάθε συνάρτησης (begin_block) αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον $ra η jal

    sw $ra,($sp)

```
elif quad[1] == "begin_block":
    if ((quad[2]+"_" == program_name) and (quad[0]+1 == main_start_quad)):
        asm_file.write('L_%s: \n' %quad[2] )
        asm_file.write('L_%s: \n' %(quad[0]+1) )
        asm_file.write("\taddi $sp, $sp, %s\n" %main_framelenght)
        asm_file.write("\tmove $s0,$sp\n")
    else:
        asm_file.write('L_%s: \n' %(quad[0]+1) )
        asm_file.write("\tsw $ra, -0($sp)\n")
    parameters = 0
```

Ο κωδικας του if αφορά μόνο τη περίπτωση όπου βρισκόμαστε στη main. Πρέπει να κατεβάσουμε τον $sp κατά framelength της main

    addi $sp,$sp,framelength

Σημειώνουμε στον $s0 το εγγράφημα δραστηριοποίησης της main ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές

    move $s0,$sp

- Στο τέλος κάθε συνάρτησης κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον $ra. Μέσω του $ra επιστρέφουμε στην καλούσα

    lw $ra,($sp)
    jr $ra

```
elif quad[1] == "end_block":
    asm_file.write('L_%s: \n' %(quad[0]+1) )
    if ((quad[2]+"_" != program_name) or (quadList[x1][0]+1 != main_start_quad)):
        asm_file.write("\tlw $ra, -0($sp)\n")
        asm_file.write("\tjr $ra\n")

    parameters = 0
```

Όσον αφορά την αρχή του προγράμματος

Το κυρίως πρόγραμμα δεν είναι το πρώτο πράγμα που μεταφράζεται, οπότε στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος

    b Lmain

```
def intialize_asm_file(program_name):
    global quadList,asm_file, main_framelenght

    t_counter = 0
    #--------- this code is for printing newline
    asm_file.write('.data\n')
    asm_file.write('newline: .asciiz "\\n"\n')
    asm_file.write('.text\n\n\n')
    #---------

    asm_file.write('L0: b L_%s\n' %program_name)
```

Με τα write που περιέχονται στην initialize έχουμε τη δυνατότητα τα print που υπάρχουν στον υπόλοιπο τελικό κώδικα να καλούνται με ένα απλό newline.

Κάθε φορά που καλείται η συνάρτηση που δημιουργεί το τελικό κώδικα γίνονται τα εξής:

```
def paragwgh_telikou_kwdika(x1, x2):
    global quadList,asm_file,main_start_quad

    parameters = 0


    for k in range(x1,x2+1):
        quad = quadList[k]
```

Τα ορίσματα της συνάρτησης αυτής (χ1,χ2) είναι η ετικέτα της πρώτης και της τελευταίας τετράδας ενός block. Αυτό γίνεται καλώντας τη συνάρτηση αυτή στο κλείσιμο κάθε block. Το quad αντιστοιχεί σε μία τετράδα και οι παραπάνω έλεγχοι του τελικού κώδικα γίνονται με βάση τα περιεχόμενα του quad[1].

Σημειώσεις

Για να δει κάποιος τα αποτελέσματα αυτού του μεταγλωττιστή πρέπει:

Ώντας μεταγλωττιστής, αναγνωρίζει μόνο μία πηγαία γλώσσα (τη cimple), οπότε πρώτο μέλημα είναι η δημιουργία πηγαίου κώδικα σε cimple.

Χρησιμοποιώντας το τερματικό θα πρέπει να “δώσει” την εντολή:

```
python  ./cimple_4196_3390.py ./test_file.ci        (τα αρχεία της cimple
                                                    αποθηκεύονται με
                                                    κατάληξη .ci)
```

όπου cimple_4196_3390 το όνομα του μεταγλωττιστή και test_file το όνομα του αρχείου στον πηγαίο κώδικα

Αποτέλεσμα της παραπάνω εντολής είναι:

1) Η δημιουργία ενός αρχείου txt το οποίο περιέχει το πίνακα συμβόλων.
2) Η δημιουργία ενός αρχείο με κατάληξη .asm. Είναι αρχείο που μπορεί κανείς να το διαβάσει με ένα απλό text editor αλλά και να τρέξει τις εντολές αυτές μέσω ενός simulator για τον mips.
3) Έπειτα δημιουργείται ένα αρχείο με κατάληξη .int το οποίο περιέχει όλες τις τετράδες που δημιούργησε ο ενδιάμεσος κώδικας. Εάν το πρόγραμμα εμφανίσει κάποιο σφάλμα σε ένα από τα παραπάνω στάδια που αναφέρουμε ότι χρειάζονται για τη μεταγλώττιση, το αρχείο είτε δε θα δημιουργηθεί είτε δε θα ανανεωθεί.
4) Στη συγκεκριμένη περίπτωση γίνεται έλεγχος εάν στο πρόγραμμα .ci που δόθηκε στο μεταγλωττιστή δεν υπάρχουν συναρτήσεις και διαδικασίες, τότε δημιουργείται επίσης ένα τελευταίο αρχείο με κατάληξη .c. Πρόκειται για ένα αρχείο σε γλώσσα C το οποίο μπορεί να εκτελεστεί με τις παρακάτω εντολές:

```
gcc -o test_file ./test_file.c      (για τη μετάφραση)
./test_file                         (για το εκτελέσιμο)
```