*Ακ. έτος: 2018-* 2019

*Καψιμάλης Αθανάσιος, ΑΜ 2265, cs12226*
*Μοναστηρλή Χρυσάνθη, ΑΜ 1716, cs091716*

# ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΣΚΗΣΗ ΜΕΤΑΦΡΑΣΤΕΣ

## Εισαγωγικά για Μεταγλωττιστές

Εμείς στην παρούσα εργασία ασχολούμαστε με την υλοποίηση ενός μεταγλωττιστή. Ένας μεταγλωττιστής γενικά αποτελείται από δύο μέρη: το ένα τμήμα είναι αφιερωμένο στην αναγνώριση και ανάλυση του προγράμματος στην γλώσσα υψηλού επιπέδου και το άλλο αναλαμβάνει την παραγωγή της συμβολής γλώσσας και της γλώσσας μηχανής. Το τμήμα που κάνει την αναγνώριση της γλώσσας προγραμματισμού δεν εξαρτάται από τον υπολογιστή στον οποίο θα εκτελεστεί αλλά μόνο από τη γλώσσα προγραμματισμού. Αντίστοιχα, το τμήμα που κάνει την παραγωγή της γλώσσας μηχανής δεν εξαρτάται από τη γλώσσα προγραμματισμού αλλά μόνο από το ρεπερτόριο εντελών μηχανής που παράγει.

Το πρώτο τμήμα του μεταγλωττιστή εκτελεί τρεις λειτουργίες: τη λεκτική, τη συντακτική και τη σημασιολογική ανάλυση του προγράμματος. Από τις λειτουργίες αυτές προκύπτει μία αναπαράσταση του προγράμματος, η οποία δεν εξαρτάται από τη γλώσσα προγραμματισμού. Την αναπαράσταση αυτή χρησιμοποιεί το δεύτερο μέρος του μεταγλωττιστή, που επιτελεί την παραγωγή κώδικα μηχανής ή συμβολικής γλώσσας και βελτιστοποίησής του.

## 1η Φάση : Λεκτική Ανάλυση

Στη λεκτική ανάλυση του προγράμματος, ο μεταγλωττιστής το χωρίζει στις βασικές μονάδες: τις λέξεις, τα σύμβολα, τους αριθμούς κλπ. Τα διαστήματα που διαχωρίζουν τις μονάδες του προγράμματος αφαιρούνται και το αποτέλεσμα είναι μία σειρά από μονάδες του προγράμματος, tokens.

## Λεκτικός Αναλυτής

Καταρχάς, στην πρώτη φάση της εργαστηριακής άσκησης αξιοποιήθηκε ως βάση το παράδειγμα του αυτόματου που περιλαμβάνεται στις διαφάνειες του μαθήματος (pdf Λεκτικός Αναλυτής) κάνοντας, βέβαια, τις κατάλληλες μετατροπές έτσι ώστε το πρόγραμμά μας να ανταπεξέρχεται στις απαιτήσεις της εκφώνησης. Ειδικότερα, λαμβάνουμε υπόψιν τις δοθείσες λεκτικές μονάδες. Για παράδειγμα, κάνουμε τους απαραίτητους ελέγχους για

ενδεχόμενα λάθη όπως τα σχόλια μέσα σε σχόλια, τις δεσμευμένες λέξεις ,καθώς και τα σύμβολα <> , >=, <= για να μπορούμε να μετακινούμε τον κένσορα ορθά και να εμφανίζονται τα σωστά αποτελέσματα στο τερματικό.

Παρόλα αυτά, είναι φρόνιμο ,αρχικά, να αναλύσουμε διεξοδικά το αυτόματο που σχεδιάσαμε ώστε να γίνει πλήρως κατανοητή η σχεδίασή του. Έτσι, διαβάζουμε προσεκτικά όσα μάς δόθηκαν και βάσει αυτών δημιουργούμε καταστάσεις και ανάλογες περιπτώσεις αυτών όπου αυτό είναι αναγκαίο. Σύμφωνα με αυτά, ξεκινάμε απο την κατάσταση 0. Εκεί , αν συναντήσουμε γράμμα μεταβαίνουμε στην κατάσταση 1 , αν στην κατάσταση 1 συναντήσουμε κάποιο γράμμα ή ψηφίο δημιουργείται βρόχος, αλλιώς μεταβαίνουμε στην τελικά κατάσταση anagnoristikotk, όπου με το tk υπονοούμε το token. Αν στην κατάσταση 0 συναντήσουμε ψηφίο μεταβαίνουμε στην κατάσταση 2, κι εκεί αν συναντήσουμε ψηφίο δημιουργείται βρόχος αλλιώς μεταβαίνουμε σε σταθερά, όπου ο λεκτικός αναλυτής , lektikos(), τερματίζει αφού επιστρέψει την αριθμητική σταθερά στον συναντικό αναλυτή, synt.py αρχείο .

Συνεχίζοντας, αν στην κατάσταση 0 συναντήσουμε +, μεταβαίνουμε στην κατάσταση +tk,δηλαδή αναγνωρίζουμε το σύμβολο +, αλλιώς αν συναντήσουμε -, μεταβαίνουμε στην κατάσταση -tk όπου αναγνωρίζουμε το σύμβολο -,αλλιώς αν συναντήσουμε *, μεταβαίνουμε στην κατάσταση *tk όπου αναγνωρίζουμε το σύμβολο *. Βέβαια, αν μετά συναντήσουμε =, μεταβαίνουμε στην κατάσταση =tk όπου αναγνωρίζουμε το σύμβολο =. Προσέχουμε , ιδιαίτερα, όταν στην κατάσταση 0 συναντήσουμε < που μας οδηγεί στην κατάσταση 3 όπου αυτή μας οδηγεί στις εξής περιπτώσεις : μετά το < να ακολουθεί = όπου μεταβαίνουμε στην κατάσταση <=tk και το αναγνωρίζουμε, να ακολουθεί > άρα να καταλήγουμε στην κατάσταση <>tk και το αναγνωρίζουμε ή οτιδήποτε άλλο ακολουθεί το αναγνωρίζουμε.

Ακόμα, στο αυτόματο απο την κατάσταση 0 με > μεταβαίνουμε στην κατάσταση 4 , η οποία διακαλίζεται σε >= αν μας δοθεί = μετά το > ή σε κάθε άλλη περίπτωση αναγνωρίζουμε ό,τι ακολουθεί. Ας μην ξεχάσουμε απο την αρχική κατάσταση μπορεί να συναντήσουμε : όπου αν ακολουθείται απο = μάς οδηγεί σε αναγνωριστική κατάσταση := , αλλιώς πάμε σε κατάσταση λάθους.

Απο την κατάσταση 0, αν συναντήσουμε / μεταβαίνουμε στην κατάσταση 6. Αν στην κατάσταση 6 συναντήσουμε / , μεταβαίνουμε στην κατάσταση 7 , όπου ήδη έχουμε συναντήσει // απο την αρχή του αυτόματου και με αλλαγή γραμμής μεταβαίνουμε σε τελική κατάσταση. Επίσης, αν στην κατάσταση 7 συναντήσουμε end of file τότε μεταβαίνουμε στην κατάσταση λάθους, error. Βέβαια, αν στην κατάσταση 7 συναντήσουμε ξανά /, τότε έχει ως εκεί δημιουργηθεί το /// και άρα μεταβαίνουμε στην κατάσταση 8, η οποία με την σειρά της αν συναντήσει τέλος αρχείου μεταβαίνει και εκείνη στην κατάσταση λάθους error, αν συναντήσει αλλαγή γραμμής μεταβαίνει σε τελική κατάσταση, αν συναντήσει * ή / μεταβαίνει σε κατάσταση λάθους error2 , γιατί δεν γίνεται να έχουμε σχόλια μέσα σε σχόλια και αν συναντήσει οτιδήποτε άλλο μεταβαίνει στην κατάσταση 7. Να προσθέσουμε εδώ ότι η κατάσταση 7 αν συναντήσει οτιδήποτε διαφορετικό από αυτά που αναφέραμε παραπάνω κάνει βρόχο, δηλαδή επιστρέφει στον εαυτό της.

Απο εκεί και πέρα , αν απο την κατάσταση 6 βρούμε * τότε πηγαίνουμε στην κατάσταση 9, /* που σηματοδοτεί άνοιγμα σχολίων, η οποία αν ακολουθείται απο end of file πηγαίνει σε τελική κατάσταση λάθους error, αν ακολουθείται απο / πηγαίνει στην

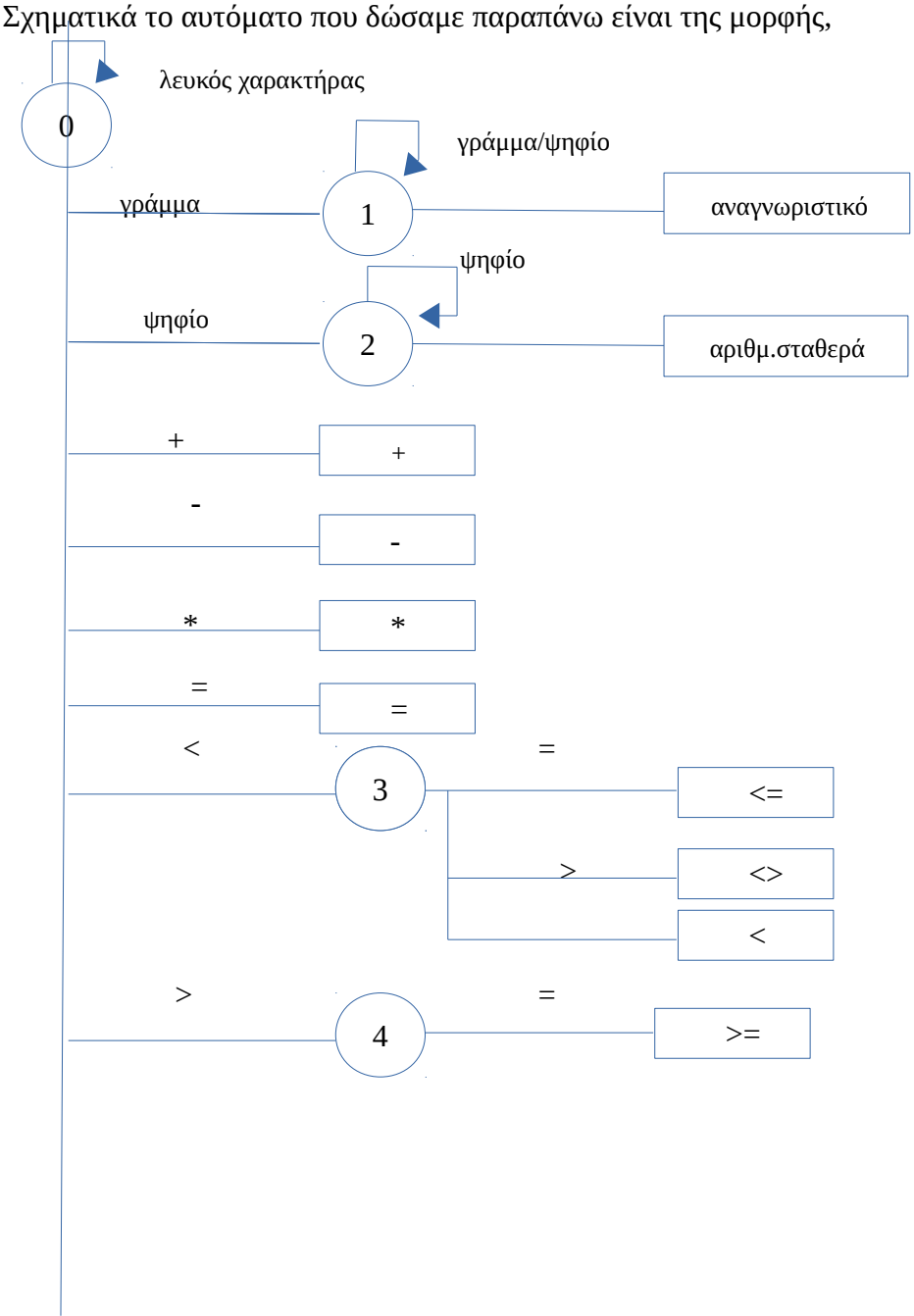
κατάσταση 11, αν ακολουθείται απο * πηγαίνει στην κατάσταση 10 και για οτιδήποτε άλλο δημιουργεί βρόχο.

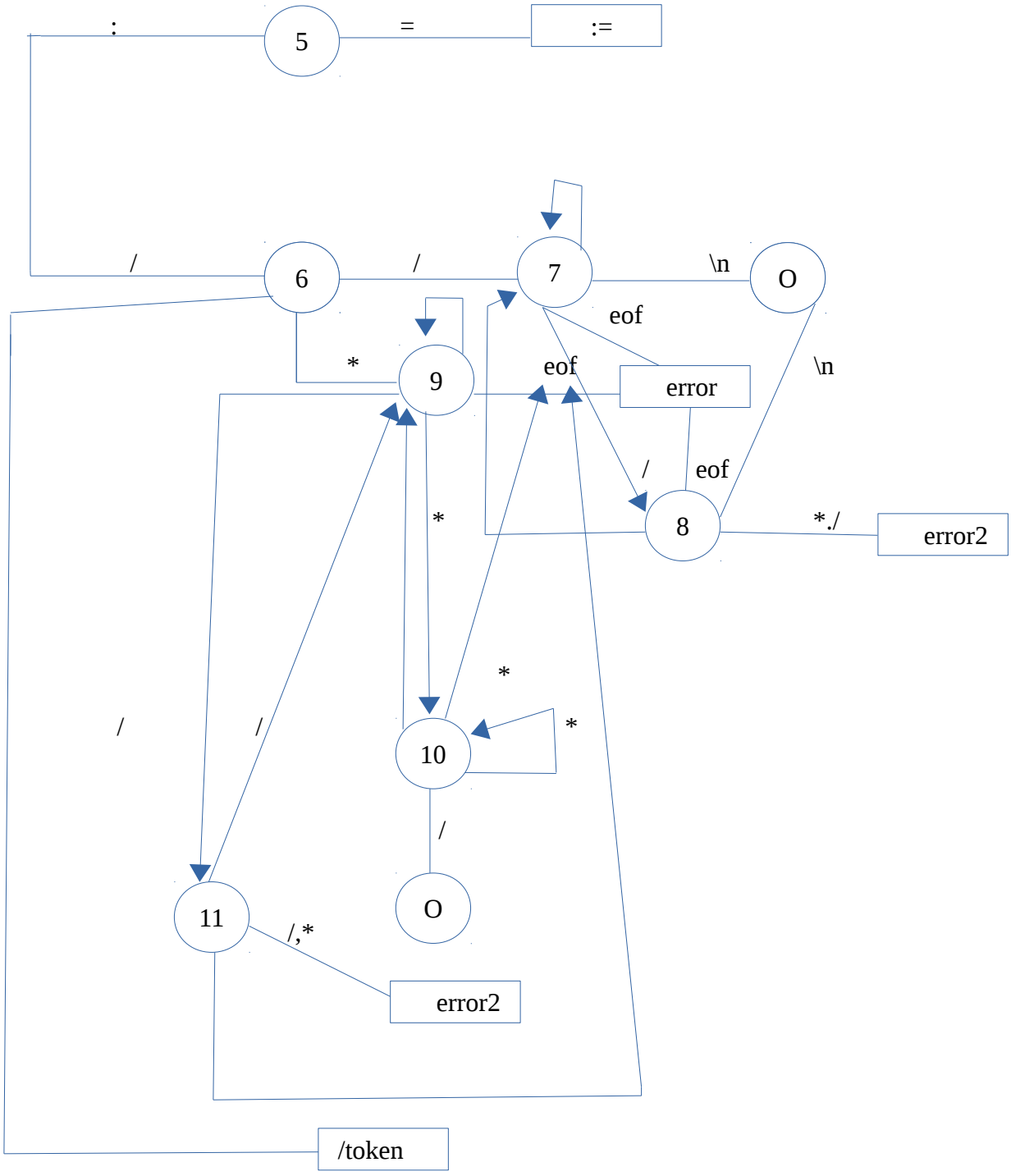
Η κατάσταση 10 με την σειρά της αν ακολουθείται απο * δημιουργεί βρόχο, αν ακολουθείται απο / πηγαίνει στην αρχική κατάσταση, αν βρει end of file πηγαίνει στην αναγνωριστική κατάσταση error και για οτιδήποτε άλλο μεταβαίνει στην κατάσταση 9. Τέλος, η κατάσταση 11 αν βρει end of file πηγαίνει στο error, αν μετά απο αυτή υπάρχει / ή * πηγαίνει στην αναγνωριστική κατάσταση λάθους error2 ,καθώς έχει δημιουργηθεί π.χ. /*// και για οποιαδήποτε άλλη περίπτωση μεταβαίνει στην κατάσταση 9.

Συμπερασματικά, το αυτόματό μας, για παράδειγμα, αποδέχεται το ///\n και απορρίπτει τα //// και //abc// .

Σημείωση : Στον λεκτικό μας αναλυτή κάνουμε τον έλεγχο για τα τριάντα πρώτα γράμματα και αν έρθουν περισσότερα τα αγνοούμε. Επίσης, στον λεκτικό αναλυτή κάνουμε τον έλεγχο για τιμές των ακεραίων έχοντας κατά νου ότι στην Starlet το πρόσημο είναι προαιρετικό.

Σχηματικά το αυτόματο που δώσαμε παραπάνω είναι της μορφής,




:
5
=
:=
/
6
/
7
\n
O
eof
*
9
eof
error
\n
/
eof
*
8
*./
error2
*
/
/
*
10
/
O
11
/,*
error2
/token

## Συντακτική Ανάλυση

Στη συντακτική ανάλυση του προγράμματος, ελέγχεται κατά πόσο αυτό ακολουθεί τους συντακτικούς κανόνες της γλώσσας προγραμματισμού, και μετασχηματίζεται σε κάποια ενδιάμεση αναπαράσταση, την ενδιάμεση γλώσσα, η οποία δεν εξαρτάται απο την γλώσσα προγραμματισμού.

## Συντακτικός Αναλυτής (Γραμματική – Κανόνες)

Καταρχάς, η φιλοσοφία τού δικού μας συντακτικού αναλυτή είναι ότι πηγαίνουμε και “κρυφοκοιτάμε” την επόμενη λεκτική μονάδα κάθε φορά. Έτσι, λοιπόν, ξεκινάμε ελέγχοντας όλους τους κανόνες της Starlet έναν έναν.

Πρώτα απο όλα,αν βρισκόμαστε στην program ελέγχουμε αν η λέξη που διαβάσαμε είναι η λέξη “program”, καλούμε λοιπόν τον λεκτικό αναλυτή για το επόμενο string και αν αυτό είναι το id, δηλαδή το anagnoristikotk μας τότε καλούμε την συνάρτηση block και έπειτα αν αυτός ο κανόνας που μας δόθηκε απο την γραμματική της Starlet δεν τελειώνει με endprogram τότε έχουμε λάθος και το τυπώνουμε κρατώντας τον αριθμό γραμμής και βγαίνουμε απο το πρόγραμμα. Επίσης, ελέγχουμε αν δεν έχουμε βρει αναγνωριστικό και αν συμβαίνει αυτό βγαίνουμε απο το πρόγραμμα έχοντας τυπώσει το αντίστοιχο μήνυμα λάθους και τέλος αν δεν είμαστε στην κατάσταση programtk τότε βγαίνουμε, επίσης, απο το πρόγραμμα γράφοντας μήνυμα λάθους ότι δεν βρέθηκε το η λέξη program.

Γίνεται, επομένως, αντιληπτό ότι και οι υπόλοιποι κανόνες που έχουν μετατραπεί στις κατάλληλες συναρτήσεις ακολουθούν το ίδιο πνεύμα του προαναφερθέντος κανόνα , με τους αναγκαίους πάντα χειρισμούς για την σωστή υλοποίηση της άσκησης. Άρα, ο επόμενος κανόνας που ακολουθεί είναι ο block , άρα φτιάχνουμε την συνάρτηση block και καλούμε τις συναρτήσεις declarations(),subprograms(), statements() με την αντίστοιχη σειρά , καθώς έτσι μάς δίνονται απο την εκφώνηση.

Αργότερα, έχουμε την συνάρτηση declarations όπου γίνεται ο έλεγχος αν υπάρχει η κατάσταση declaretk και εάν αυτό υφίσταται τότε καλούμε τον λεκτικό αναλυτή ώστε να πάρουμε την επόμενη λεκτική μονάδα , καλούμε την συνάρτηση varlist και μετά ελέγχουμε αν υπάρχει ερωτηματικό ύστερα απο την varlist και αν αυτό συμβαίνει τότε καλούμα ξανά τον λεκτικό αναλυτή για να πάρουμε την επόμενη λεκτική μονάδα αλλιώς δεν βρίσκουμε ερωτηματικό και βγαίνουμε απο το πρόγραμμα τυπώνοντας μήνυμα λάθους, καθώς και την γραμμή του συγκεκριμένου λάθους. Αυτό συμβαίνει, γιατί στην εκφώνηση έχουμε παρένθεση αστεράκι κι εμείς απο θεωρία υπολογισμού γνωρίζουμε ότι αυτό σημαίνει ότι το περιεχόμενο της παρένθεσης αστεράκι μπορεί να υπάρχει απο μία ως άπειρες φορές. Τυπώνουμε όπως και να έχει τα απαραίτητα μηνύματα λάθους

Ύστερα, έχουμε τον κανόνα subprograms. Η συνάρτηση, λοιπόν, που μετατρέπουμε τον κανόνα subprograms, όσο καταναλώνει λόγω του (“κάτι”)* το string function token καλούμε την συνάρτηση lektikos για να καταναλώσουμε την επόμενη λεκτική μονάδα και μετά την subprogram. Έτσι, έπειτα καλούμε την subprogram. Αν ξεκινάει με id δηλαδή το αναγνωριστικό μας τότε καλούμε τον λεκτικό αναλυτή για την επόμενη λεκτική μονάδα , καλούμε την funcbody συνάρτηση και έπειτα ελέγχουμε αν η επόμενη λεκτική μονάδα είναι η endfunction. Αν είναι αυτή τότε καλούμε λεκτικό αναλυτή για να δούμε την επόμενη λεκτική μονάδα ,και αν δεν είναι αυτή τότε τυπώνουμε μήνυμα λάθους και βγαίνουμε απο το πρόγραμμα και επίσης αν δεν έχουμε βρει

αναγνωριστικό τότε πάλι βγάζουμε μήνυμα λάθους και κρατάμε αριθμό γραμμής και βγαίνουμε απο το πρόγραμμα.

Έπειτα, ο κανόνας funcbody που τον μετατρέψαμε σε συνάρτηση καλεί τις συναρτήσεις formalpars και block. Αργότερα,ελέγχουμε την formalpars. Για να υπάρχει αυτή η συνάρτηση , πρέπει να ξεκινάει με ( token και αν αυτή υπάρχει τότε καλούμε λεκτικό αναλυτή για να δούμε την επόμενη λεκτική μονάδα,καλούμε την fromalparlist σύμφωνα με τον κανόνα και έπειτα βλέπουμε αν τελειώνει ο κανόνας με ) και καλούμε λεκτικό αναλυτή ξανά. Αν δε βρούμε βέβαια κλείσιμο παρένθεσης τότε τυπώνουμε μήνυμα λάθους κρατώντας την γραμμή και βγαίνουμε απο το πρόγραμα και ελέγχουμε και για την περίπτωση που δεν υπάρχει το άνοιγμα παρένθεσης ώστε να βγούμε απο απο το πρόγραμμα σε αυτή την περίπτωση.

Αφού κάνω αυτούς τους ελέγχους, μέσω της formalparitem ελέγχουμε αν υπάρχει η formalparlist ώστε να υπάρχει και η formalpars. Με άλλα λόγια ελέγχουμε την πρώτη λεκτική μονάδα απο την formalparitem : in , inout ή inandout καθόσον βρισκόμαστε στην formalparlist και αν κάποια απο αυτές υπάρχει τότε καλούμε πράγματι την συνάρτηση formalparitem και έπειτα όσο έχουμε “,tk” τότε καλούμε λεκτικό αναλυτή για την επόμενη λεκτική μονάδα κι έπειτα formalparitem. Εδώ δεν έχουμε if ούτε else ούτε μήνυμα λάθους καταταρχάς και λόγω του kleene star , αλλά και λόγω της περίπτωσης τού κενού που δεχόμαστε τα πάντα.

Τώρα, στην συνάρτηση formalparitem , αφού ήδη έχουμε ελέγξει ότι υπάρχει, ελέγχουμε σε ποια απο τις τρεις περιπτώσεις βρισκόμαστε μέσω των λεκτικών μονάδων που έχουμε δει πιο πάνω και ελέγχουμε αν μετά απο αυτές, έχοντας καλέσει τον λεκτικό αναλυτή πρώτα υπάρχει το αναγνωριστικό μετά ώστε να μεταβούμε στην κατάλληλη συνάρτηση και να ξανακαλέσουμε λεκτικό αναλυτή , αλλιώς να τυπώσουμε ένα περιγραφικό μήνυμα λάθους και να βγούμε απο το πρόγραμμα.

Συνεπώς, ακολουθώντας αυτή την φιλοσοφία, ένα απο τα κομμάτια του κώδικά μας που θα ήταν φρόνιμο να εξηγήσουμε περαιτέρω είναι το κομμάτι που αφορά τον κανόνα statement όπου εμείς για κάθε katastasth a.k.a. <assignment-stant> , <if-stat> κλπ που στον κώδικά μας μεταφράζουμε σε “anagnoristikotk”, “ifftk” κλπ καλούμε τον λεκτικό αναλυτή και διαβάζουμε την πρώτη λεκτική μονάδα. Έτσι, όταν καλούνται ως κανόνες οι <assignment-stant> , <if-stat> κλπ , έχουμε ήδη διαβάσει την πρώτη λεκτική μονάδα μέσω του statement και επομένως ελέγχουμε απο την δεύτερη λεκτική μονάδα και έπειτα, πάντα όμως, σκεπτόμενοι και τα ενδεχόμενα λάθη που μπορεί να προκύψουν , τα οποία τα τυπώνουμε με τα ανάλογα μηνύματα λάθους μέσω της print.

Διόλου ασήμαντο δε, δεν είναι το γεγονός ότι εσκεμμένα παραλείψαμε τον κανόνα <exit-stat> :: exit, αφού ο έλεγχός του για την ύπαρξη της exit γίνεται στον κανόνα <statement>. Επιπλέον , ένα άλλο σημείο που προσέξαμε στον συντακτικό αναλυτή ,είναι για παράδειγμα ,ο διαχωρισμός του πότε πρέπει να ελέγχουμε για άνοιγμα και κλείσιμο αγκύλης στους εκάστοτε κανόνες και πότε απλώς είναι μέρος της θεωρίας υπολογιμού για το πλήθος ,π.χ. το ( “κάτι” )* όπου εδώ δεν προσμετράμε στον έλεγχο την αγκύλη και το αστεράκι. Είναι ευνόητο , έτσι, ότι τα κόμματα και οι άνω κάτω τελείες αντιμετωπίστηκαν με την παραπάνω νοοτρόπια ( δηλαδή έλεγχο με τα απαραίτητα μηνύματα λάθους πάντα).

Ταυτόχρονα, στην γραμματική της Starlet παρατηρήσαμε ότι κάποιοι κανόνες μοιάζουν μεταξύ τους όπως η <formalparlist> με την <actualparlist> και η <forecase-stat> με την <incase-stat>, πράγμα που μάς βοήθησε να κατανοήσουμε καλύτερα αυτή την γλώσσα και να υλοποιήσουμε κάποια κομμάτια της εύκολα καθώς αλλάζουν σε κάποια μέρη του προγράμματος που μοιάζουν μεταξύ τους μόνο ελάχιστα πράγματα. Παράλληλα, ένα άλλο κομμάτι στον κώδικά μας που φαίνεται απλό είναι οι κλήσεις τις return_stat() και print_stat() όπου καλούμε μόνο την expression() πράγμα που υφίσταται επειδή μέσω της statement έχουμε ελέγξει ήδη την πρώτη λεκτική μονάδα απο return_stat() και print_stat() , δηλαδή return και print αντίστοιχα.

Τέλος, μερικά σημεία που απαιτούνε και αυτά προσοχή είναι τα εξής: Στον κανόνα expression ,αφού καλέσουμε optional_sign() και term() όπως δίνονται απο την γραμματική, έχουμε

την add-oper , απο την οποία καταναλώνουμε + ή – και έπειτα καλούμε λεκτικό αναλυτή και term(). Ύστερα, κατά τον ίδιο τρόπο στην term() αφού καλέσουμε την factor(), σύμφωνα με την γραμματική της Starlet ελέγχουμε την mul-oper , δηλαδή επί και διά, “κρυφοκοιτάμε” όπως καταλαβαίνουμε την επόμενη λεκτική μονάδα ώστε να μπορέσουμε την συνάρτηση lektikos() και την factor(). Επιπρόσθετα, στην idtail() δεν χρειάζεται να κάνουμε else γιατί έχουμε or κενό και έτσι αφού ελέγξουμε μέσω της actualpars αν υπάρχει άνοιγμα παρένθεσης, τότε καλώ στην ουσία την actualpars για να πάρω την idtail. Μετά απο αυτό, στην optional_sign() μέσω της add-oper ελέγχουμε για συν ή μείον και αν βρισκόμαστε σε μία απο τις δύο αυτές καταστάσεις τότε καλούμε τον λεκτικό αναλυτή. Αυτό επίσης είναι ένα απο τα σημεία που δεν ελέγχουμε για λάθη γιατί έχουμε και την άλλη περίπτωση που μπορεί να έχουμε οτιδήποτε.

Τελικά, καλούμε τον λεκτικό αναλυτή και την συνάρτηση program για να μπορεί να διαβαστεί όλο το αρχείο χωρίς κάποιο πρόβλημα.

## 2η Φάση : Ενδιάμεσος Κώδικας

### *Εισαγωγή*

Αντί να μεταφράζουν απευθείας το αρχικό πρόγραμμα στην τελική γλώσσα, οι περισσότεροι μεταγλωττιστές το μεταφράζουν πρώτα σε ένα ισοδύναμο πρόγραμμα γραμμένο σε κάποια ενδιάμεση γλώσσα, το οποίο στη συνέχεια μεταφράζουν στην τελική γλώσσα. Η μετάφραση δηλαδή γίνεται σε δύο διαδοχικά βήματα. Η ενδιάμεση γλώσσα είναι χαμηλότερου επιπέδου από την αρχική γλώσσα, αλλά υψηλότερου επιπέδου από την τελική. Το ισοδύναμο πρόγραμμα στην ενδιάμεση γλώσσα ονομάζεται ενδιάμεσος κώδικας.

### *Γενικά για την υλοποίησή μας*

Η γλώσσα μας έχει τετράδες. Κάθε τετράδα έχει πέντε πράγματα. Η μορφή είναι:

**label : op, x, y, z**

, όπου label είναι αύξον αριθμός (αριθμός τετράδας), op είναι **+, -, *, /** , op1 και op2 είναι σταθερές ή μεταβλητές και op3 είναι μεταβλητή.

Για παράδειγμα, αν θέλουμε να μεταφράσουμε το :

r = 4
p1 = 3.14
area = p1* r * r , όπου | σπάει και χρησιμοποιούμε το τ_1 που είναι μια προσωρινή μεταβλητή. Άρα, τυχαία στην γραμμή 100:

100: :=, 4, _ , r
110: :=, 3.14, _, pi
120: *, pi, r, t_1
130: *,t_1, πρόβλημα , όπου πρόβλημα: r, t_2

140: := , τ_2, _ , area . Χρησιμοποιούμε , δηλαδή στο πρόγραμμά μας, προσωρικές μεταβλητές t_counter , όπου counter = 1,2,3… κλπ.

Πέρα απο αυτό, με το op μπορούμε να κάνουμε **jump** , όπου η μορφή της jump είναι η εξής:

jump , _ , _ , (αριθμός τετράδας) .

Ο αριθμός της τετράδας στην jump μάς δείχνει εκεί που πρέπει να μεταβούμε (στην κατάλληλη ετικέτα). Ουσιαστικά είναι σαν να έχουμε goto . Δηλαδή με την εντολή jump , _ , _ , 100 είναι σαν να λέμε goto 100.

Επίσης, έχουμε σχεσιακούς τελεστές:

**relop** sol1, sol2 , z , όπου relop είναι <, > , <> , <= , >= , = , sol1 και sol2 είναι μεταβλητές ή αριθμοί και z είναι μία ετικέτα ή αλλιώς label. Για παράδειγμα,
η εντολή > , m, n , 20 είναι σαν να λέμε αν m>n τότε goto 20. Εάν, δεν ισχύει , βέβαια, η συνθήκη απλά πηγαίνουμε στην επόμενη τετράδα. Έτσι, για m > n έχουμε το εξής:

100: > , m, n , _ (true), όπου στην δικιά μας περίπτωση το _ είναι 20
110: jump, _, _ ,_ (false)

Είναι καλό ,εδώ , να τονιστεί ότι γενικά κάθε **συνθήκη** παράγει δύο “τρύπιες” τετράδες που τις συμπληρώνουμε στο μέλλον . Για να γνωρίζουμε ποιες είναι αυτές, δημιουργούμε δύο λίστες ετικετών τετράδων true και false. Για το προηγούμενο παράδειγμα, δηλαδή, θα έχουμε true=[100] και false=[110]. Ένα άλλο χαρακτηριστικό παράδειγμα συνθηκών είναι η <if-stat>.

Εκτός απο τους προηγούμενους τελεστές όμως , έχουμε και τους λογικούς τελεστές not, and και or. Πιο ειδικά, όταν έχουμε not , λόγου χάρη R → not (B) {p1} , όπου $\{p_i\}$ είναι σημείο στην γραμματική που βάζουμε κώδικα για την παραγωγή τετράδων, αντιστρέφουμε στην ουσία τις λίστες και έχουμε R.true = R.false και R.false = R.true. Απο την άλλη πλευρά , όταν έχουμε λόγου χάρη
το a > b and c < d γίνεται το εξής:

100: > , a b , _ ← true1 =[100] που διαγράφεται αυτή η λίστα & στο κενό έχει μπει το 120
110: jump, _, _ ,_ ← false1=[110]
120: > , c, d , _ ← true2 =[120]
130: jump, _, _ ,_ ← false2=[130] και συνενώνουμε τις false1 και false2 : false1=[110,130] και βάζουμε την true2 στην true1 : true1 =[120] .Με αντίστοιχο τρόπο λειτουργούμε και για την or. Δηλαδή για a > b or c < d έχουμε:

100: > , a b , _ ← true1 =[100]
110: jump, _, _ ,_ ← false1=[110]
120: > , c, d , _ ← true2 =[120]
130: jump, _, _ ,_ ← false2=[130] και συνενώνουμε τις true1 και true2 : false1=[100,120] και βάζουμε την false2 στην false1 : false1 =[130] .

Ακόμα, έχουμε την begin_block , name, _, _
end_block, name, _, _ που είναι κυρίως για ζωγραφιά για το πού θα βάλουμε τις εντολές του προγράμματος. Αυτό δεν σημαίνει ότι είναι φωλιασμένη.

Επίσης, έχουμε την **halt** , _, _ , _ που μπαίνει πριν το endblock του κυρίως προγράμματος και τερματίζει τα πάντα.

Επιπρόσθετα, έχουμε την **ανάθεση** που πραγματοποιείται με την εντολή :

:= , σταθερά ή μεταβλητή , μεταβλητή , _
Για παράδειγμα, το :=, m , _, n ισοδυναμεί με n := m .

Επιπλέον, έχουμε την **par** για παραμέτρους:
par, x , mode, _
όπου x είναι το όνομα της συνάρτησης και mode είναι ο τύπος τη. Οι τύποι μπορούν να είναι οι εξής:

**CV** είναι call by value , in
**REF** είναι call by reference , inout
**CP** είναι call by copy , inandout
**RET** είναι return value , επιστροφή τιμής σύμφωνα πάντα με την εκφώνηση.

Εδώ να σημειώσουμε ότι οι κλήσεις συναρτήσεων πραγματοποιούνται με **call** , name, _ , _ και η **ret** x,_, _ επιστρέφει το αποτέλεσμα της συνάρτησης. Για παράδειγμα, για την
x = foo(in a, inout b) έχουμε:

```
100:  par ,  a,   CV,   _
110:  par   ,b  , REF ,  _
120:  par  , t_3 ,  RET,  _      , για επιστροφή τιμής
125:  call , foo,  _ ,  _        ,το αποτέλεσμα πάει στην t_3
130:  := ,t_3,   _ ,  x
```

Γίνεται , λοιπόν, αντιληπτό ότι βολεύει να κάνουμε τις εξής συναρτήσεις:

- **nextquad() :** επιστρέφει γραμμή – ετικέτα σε μορφή string, δηλαδή αν έχουμε την γραμμή 120, αυτή επιστρέφει την γραμμή 130. Δηλαδή, επιστρέφει την ετικέτα της επόμενης τετράδας.
- **genquad( op, x, y, z)** : δημιουργούμε τετράδα κάτω απο την τελευταία τετράδα, δηλαδή γίνεται απλή αντιγραφή. Με απλά λόγια παράγει την επόμενη τετράδα.
- **newtemp() :** επιστρέφει το όνομα της επόμενης προσωρινής μεταβλητής σε μορφή string
- **emptylist()** : δημιουργεί μια άδεια λίστα δεικτών/ετικετών.
  ( Σε αυτό το σημείο να σημειώσουμε ότι έχουμε δύο λίστες. Η μία λίστα είναι του προγράμματος με τις τετράδες και η άλλη είναι η λίστα των ετικετών/δεικτών τετράδων. Στην ουσία με την δεύτερη λίστα βάζουμε σημαδάκια στην άλλη λίστα με τις τετράδες. )
- **makelist(x)** : φτιάχνουμε νέα λίστα με μόνο ένα σημαδάκι που δείχνει εκεί που θέλουμε.
  Για παράδειγμα, αν πούμε a = makelist είναι σαν να λέμε a = [] κι αν πούμε a = make(‘120’) είναι ίσο με a = [‘120’] . Δημιουργείται δηλαδή μια λίστα ετικετών τετράδων και περιέχει μόνο την ετικέτα x.
- **mergelist(x, y)** : κολλάει την λίστα y μετά την λίστα x. Τις συνενώνει στην λίστα x. Ύστερα, δεν υπάρχει y. Πρακτικά, μόνο με αυτόν τον τρόπο φτιάχνουμε μεγαλύτερες λίστες.
- **backpatch(x, z)** : πηγαίνει στις τετράδες των οποίων οι ετικέτες είναι στη λίστα x και συμπληρώνει το τελευταίο πεδίο τους με z. Δηλαδή, x είναι μία λίστα και το z είναι μία ετικέτα. Έχουμε την ανάγκη να φτιάξουμε λίστες που σημαδεύουν κάποιες τετράδες , (((πρακτικά έχουμε λίστα απο λίστες,))) και σε αυτό μάς εξυπηρετεί η backpatch.

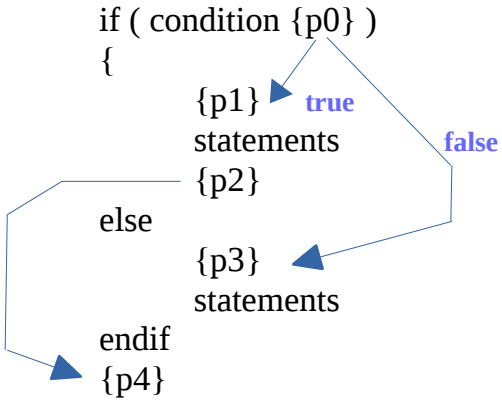
Τέλος, κάθε κανόνας επιστρέφει είτε ένα αποτέλεσμα **place** , όπου place είναι μια μεταβλητή είτε **δύο λίστες ετικετών τετράδων** true και false. Εδώ, ας δώσουμε ένα παράδειγμα για το place ώστε να γίνει κατανοητή η χρήση του. Ας υποθέσουμε ότι έχουμε τον κανόνα $E \rightarrow T^{(1)} ( + T^{(2)})^{*}$ . Για

διευκόλυνση προσθέτω την ετικέτα {p1} : E → $T^{(1)}$ ( + $T^{(2)}$ {p1} )* ,όπου το {p1} είναι πριν το τέλος του while. Επομένως, έχουμε
{p1} : ζητάμε να μας δώσει μια νέα προσωρινή μεταβλητή : w = newtemp() που είναι string
Τώρα έχουμε την πληροφορία για να φτιάξουμε πχ το 100: + , b, c , t_1
Θέλουμε να γυρίσει μια πρόσθεση ανάμεσα σε $T^{(1)}$, $T^{(2)}$ και w : genquad("+",T1.place, T2.place,w)
Όπως γυρίζει το while περνάει απο {p1} : T1.place = w γιατί θέλουμε και άλλη τετράδα άρα (((p2))) : E.place = T1.place που μάς γυρίζει αυτό που θέλουμε. Αν, βέβαια, δεν είχαμε
E → $T^{(1)}$ ( + $T^{(2)}$)* δε θα έμπαινε ποτέ στην {p1} και genquad. Τί βγαίνει λοιπόν ως αποτέλεσμα;
T1.place = "b" που γίνεται τ_1 και αυτό με την σειρά του γίνεται t_2 και το + το καταναλώνει ο λεκτικός αναλυτής. Στο T2.place="c" , το "c" γίνεται "d" και μπαίνουμε στο {p1} : w = "t_1" όπου γίνεται t_2 . Άρα, καταλαβαίνουμε ότι προχωράει πάει {p1} , παράγει νέα τετράδα, πάμε {p2} , δεν υπάρχει άλλο αστεράκι. Δηλαδή, 100: +, b, c, t_1
110: +,t_1, d, t_2 . Με απλά λόγια, περνάμε προς τα πάνω το αποτέλεσμα ή όσα δεν μπορούμε να διαχειριστούμε.

## *Χειρισμός Βασικών Δομών Της Starlet*

Για να προσαρμόσουμε τις δομές της Starlet στον ενδιάμεσο κώδικά μας κάνουμε τις κατάλληλες μετατροπές στις εκάστοτε δομές με την χρήση ετικετών ,βλέπε {$p_i$} όπου i=0, 1,2.. κλπ, που προαναφέραμε στην αναφορά μας. Συγκεκριμένα ,

### *A') if - else – endif :*



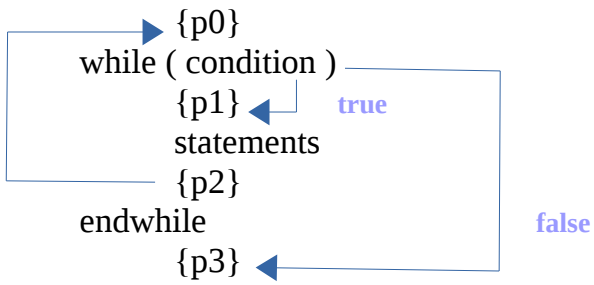

όπου αρχικά στην {p0} έχουμε δύο έτοιμες λίστες απο condition,η μία λίστα είναι η true που μάς οδηγεί στην {p1} και η άλλη είναι η false που μάς οδηγεί στην {p3}. Στην συνέχεια,η {p2} μάς οδηγεί στην {p4}. Απο το παρόν, δηλαδή, οδηγούμαστε στο μέλλον όπου κάνουμε backpatch. Ειδικότερα,
{p0}: _ δημιουργούνται οι δύο λίστες
{p1}: backpatch(condition.true,nextquad()) αν η συνθήκη είναι αληθής, στην λίστα true δημιουργούμε την επόμενη τετράδα πέντε στοιχείων. Πρακτικά, δείχνει στη πρώτη τετράδα των statements. Δεν παράγεται κώδικας.
{p2}: jump = makelist(nextquad())

genquad(“jump”, “ _ “ , “ _ “, “ _ “ ) αφού έχουν υλοποιηθεί τα statements πρέπει να γίνει jump μετά την endif για να συνεχίσει η υλοποίηση του προγράμματος. Δημιουργείται , λοιπόν, μία καινούρια λίστα μετά το τέλος του endif και γίνεται jump σε εκείνη την ετικέτα.

{p3}: backpatch(condition.false, nextquad()) αν η συνθήκη της condition είναι false τότε ελέγχονται τα statements του else , αλλά επειδή δεν έχουμε κάποια τετράδα εκεί, την δημιουργούμε εμείς. Πρακτικά κάνουμε μεταγλώττιση. Με τί σειρά παράγεται ο κώδικας; Πρακτικά πρέπει να κάνουμε κάτι ώστε αν εκτελεστεί το statements του if να μην εκτελεστεί το statements του else άρα συνεχίζουμε με backpatch.

{p4}: backpatch(jumplist, nextquad()) βρισκόμαστε μετά το endif όπου ήδη έχουμε δημιουργήσει το jumplist. Πηγαίνουμε, λοιπόν, στις τετράδες του jumplist και

***B’) while - endwhile :***




όπου όταν βρισκόμαστε μέσα στην παρένθεση της while μετά την condition δημιουργούνται δύο λίστες true και false. Η true που μάς οδηγεί πριν τα statements της while στην {p1} και η false μάς οδηγεί στην {p3} μετά την endwhile. Τέλος, η {p2} μάς οδηγεί στο {p0} ώστε να μπορεί με κάθε επανάληψη να δημιουργείται μια νέα τετράδα. Πρακτικά, σε αυτή την δομή πηγαίνουμε **απο** το παρόν με jump στην τετράδα ή backpatch(λίστα,τετράδα) **στο** παρελθόν όπου κρατάμε την ετικέτα της επόμενης τετράδας. Ειδικότερα,

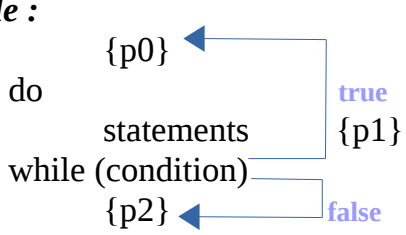
{p0}: startwhile = nextquad() δημιουργία νέας τετράδας λόγω της επανάληψης που θέλουμε να υπάρχει όταν η συνθήκη της while είναι αληθής και εκτελούνται επιτυχώς τα statements

{p1}: backpatch(condition.true,nextquad()) αν η συνθήκη είναι αληθής ,στην λίστα true δημιουργούμε μία νέα τετράδα για τα statements

{p2}: genquad(“jump”, “ _ “ , “ _ “, “ _ “ ) μετά τα statements , αφού έχουν υλοποιηθεί ορθά, τότε μεταβαίνουμε στην αρχή της while για να παραχθεί νέα τετράδα και να προχωρήσει η ομαλή λειτουργία

{p3}: backpatch(condition.false,nextquad()) αν η συνθήκη του while είναι ψευδής τότε μεταβαίνουμε μετά το endwhile , οπότε για να προχωρήσει το πρόγραμμα , δημιουργείται νέα τετράδα έξω απο την λούπα

***Γ') do – while :***



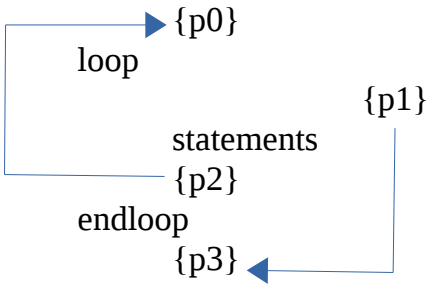

Στην do-while τα statements εκτελούνται μία φορά και αν η συνθήκη (condition) είναι αληθής τότε η λίστα true μάς οδηγεί πριν το do ώστε να δημιουργηθεί καινούρια τετράδα στην αρχή, αλλιώς σε διαφορετική περίπτωση η συνθήκη είναι ψευδής και μάς οδηγεί στην λίστα που κρατάμε τις false τετράδες, δηλαδή οδηγούμαστε εκτός της do-while. Ειδικότερα,

{p0}: startDo = nextquad() δημιουργούμε την πρώτη τετράδα του προγράμματός μας ώστε να εκτελεστούν τα statements

{p1}: backpatch(condition.true, startDo) αν η συνθήκη είναι αληθής, η λίστα τετράδων των true παίρνει την ετικέτα startDo και συμπληρώνει το πού θα πάμε

{p2}: backpatch(condition.false,nextquad()) αν η συνθήκη condition είναι ψευδής, τότε πηγαίνουμε στην λίστα false ετικετών τετράδων κι εκεί δημιουργούμε μία καινούρια τετράδα για να μπορέσει να συνεχίσει η υλοποίηση με τον υπόλοιπο κώδικα.

***Δ') loop - endloop :***




{p0}: startLoop = nextquad() στην αρχή του προγράμματος δημιουργούμε μία τετράδα που είναι η αρχική μας

exitList = emptyList() και μία κενή λίστα για όταν θα κάνουμε exit απο την λούπα γιατί δεν άρχουμε αρχικά κάποια λίστα

Όταν δούμε, λοιπόν exit, πηγαίνουμε στο {p1}

{p1}: elist = makelist(nextquad()) δημιουργούμε μία κενή λίστα με μια τετράδα. Επειδή αρχικά δεν ξέρουμε πόσες λίστες θα έχουμε γενικά, φτιάχνουμε μια makelist για να κάνουμε μετά backpatch

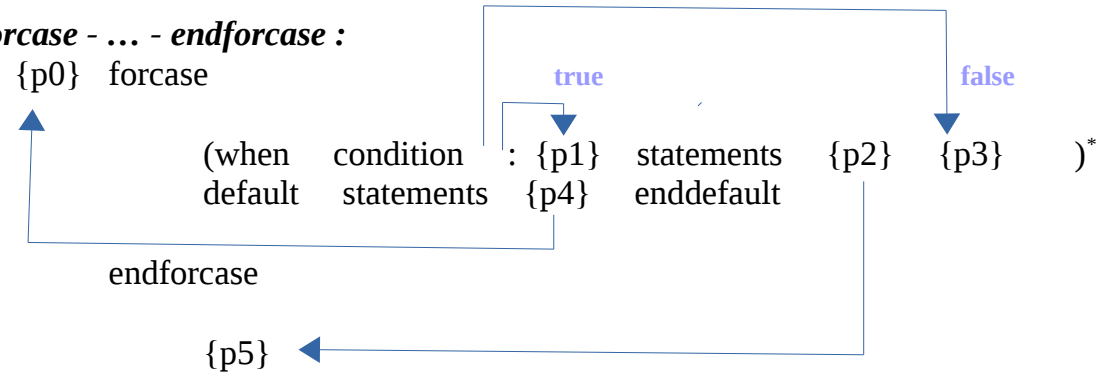
genquad("jump", " _ " , " _ ", " _ " ) βγαίνουμε έξω απο την λούπα με μια jump που δείχνει στο {p3} το οποίο βρίσκεται μετά το endloop για να συνεχίσει το πρόγραμμά μας να τρέχει κανονικά με τις υπόλοιπες εντολές

exitList = merge(exitList, elist) συνενώνουμε την exitList και την elist σε μία λίστα, αφού και οι δύο είναι κενές λίστες που δημιουργήθηκαν σε διαφορετικά σημεία της loop-endloop

{p2}: genquad("jump", " _ " , " _ ", " _ " ) αφού υλοποιηθούν τα statements και δεν υπάρχει exit , η loop συνεχίζει να τρέχει και άρα γίνεται jump στην ετικέτα startLoop για να ξαναρχίσει η υλοποίησή της

{p3}: backpatch(exitList, nextquad()) αφού έχουμε βγει απο το loop και είμαστε μετά το endloop , παίρνουμε την κενή λίστα exitList και συμπληρώνουμε την επόμενη τετράδα για να μπορεί το πρόγραμμα να πάει παρακάτω

***E') forcase - … - endforcase :***




Αρχικά, στην forcase στην θέση του {p0} δημιουργούμε την επόμενη τετράδα για να αρχίσει η μεταγλώττιση της δομής και επίσης φτιάχνουμε μία κενή λίστα όπου εκεί θα βάζουμε τις τετράδες στις οποίες μεταβαίνουμε αργότερα. Άρα,

{p0}: startforcase=nextquad()
jumplist=emptylist()

Έπειτα, αν το condition είναι αληθής, τότε δημιουργούμε την λίστα true με τις εντολές που τις αντιστοιχούν πηγαίνοντας στο {p1}. Άρα,

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

Τώρα, αν υλοποιηθεί κάποιο statements απο το {p2} μεταβαίνουμε στο {p5}, βγαίνουμε δηλαδή απο το forcase. Απο το {p2} έχουμε πολλά βέλη ουσιαστικά προς το {p5}. Σε αυτή την περίπτωση κάνουμε merge. Επειδή δεν έχουμε αρχικά λίστα , την δημιουργούμε εμείς με emptylist. Άρα,

{p2}: jlist=makelist(nextquad())
genquad("jump", " _ " , " _ ", " _ " )
jumpList=merge(jumpList,jlist)

Απο εκεί και πέρα, αν το condition του when που τρέχουμε είναι false , τότε μεταβαίνουμε μετά τα statements ,πριν το επόμενο when και δημιουργούμε στην λίστα false την επόμενη τετράδα για τις εντολές που θα ακολουθήσουν

{p3}: backpatch(condition.false, nextquad())

Αν βρεθούμε ,βέβαια, μετά τα statements του default και πριν απο το enddefault τότε καταλαβαίνουμε ότι πρέπει να ξανατρέξουμε την forcase για αυτό κάνουμε jump στην ετικέτα starforcase

{p4}: genquad("jump", " _ " , " _ ", startforcase )

Τέλος, μετά την endforcase , έχουμε το {p5} που μάς λέει ότι στην jumpList φτιάχνουμε την επόμενη τετράδα για να συνεχίσει η μεταγλώττιση με τις εντολές εκτός του forcase.

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

Ενδεικτικό παράδειγμα της forcase για τον τρόπο με τον οποίο λειτουργεί είναι το εξής:

```
forcase
    when    m<n  :    k := p
```

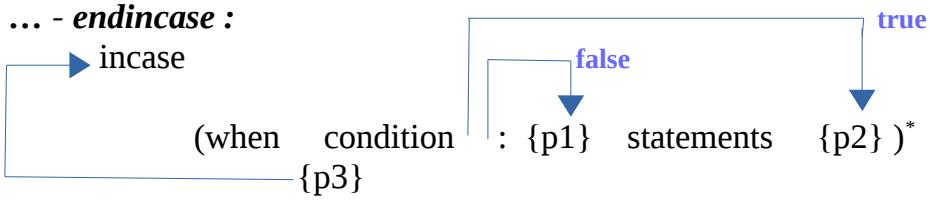
when n>m : p := k
default

με startforcase:100 έχουμε,

100: > , m, n , _ ← true που μετά διαγράφεται
110: jump, _, _ , 140 ← false που μετά διαγράφεται
120: := ,p , _ ,k
130: jump, _, _ ,_ ← jList , όπου όταν βάζουμε το jump μόνο στο jList έχουμε κενές τετράδες
140: > , n, m , _ ← true
150: jump, _, _ ,_
160: := ,k , _ ,p
170: jump, _, _ , jList

Να σημειώσουμε ότι μετά το emptylist και το merge έχουμε στην jumpList [130, 170].

***ΣΤ') incase - … - endincase :***




Καταρχάς, πριν την incase δημιουργούμε την επόμενη τετράδα ώστε να μπορεί να ξεκινήσει η δομή και επίσης δημιουργούμε μία μεταβλητή flag όπου κρατάμε αν είναι true ή false η συνθήκη condition της when για να γνωρίζουμε σε ποιο σημείο του κώδικα πρέπει να μεταβαίνουμε κάθε φορά.

Έτσι, αρχικά έχουμε

{p0}: startincase=nextquad() όπου δημιουργούμε την επόμενη τετράδας
flag = newtemp() το flag που προαναφέραμε για να μας διευκολύνει στον κώδικα
genquad(":=", " 0" , " _ ", flag ) ξεκινάμε με flag:=0 δηλαδή false

Αν η συνθήκη στο condition είναι true, τότε στην λίστα με τις τετράδες που είναι αληθείς δημιουργούμε την επόμενη τετράδα για την εντολή που ακολουθεί και κάνουμε το flag:=1 δηλαδή αληθές γιατί έχουμε μπει στην συνθήκη true. Άρα,η μετάβαση μετά το αληθές condition γίνεται στο {p1} που βρίσκεται πριν απο τα statements ώστε να δημιουργηθούν αυτά στην παρούσα ετικέτα και άρα ,έχουμε

{p1}: backpatch(condition.true, nextquad())
genquad(":=", " 1" , " _ ", flag )

Αν, βέβαια, ελεγχθεί το condition και είναι ψευδές τότε μεταβαίνουμε στην ετικέτα {p2} που βρίσκεται μετά τα statements και μάς οδηγεί το βελάκι που είναι false . Εκεί δημιουργείται η λίστα false που κρατάει τις τετράδες που αντιστοιχούν σε αυτή την περίπτωση. Τότε, έχουμε

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

Εάν τώρα έστω και μία απο τις statements εκτελείται, τότε το πρόγραμμά μας μεταβαίνει στην αρχή της incase , οπότε εκεί μάς επιστρέφει το flag στην ουσία που έχουμε

κρατήσει και είναι αληθές και επομένως για να βρεθούμε ξανά στο startincase χρησιμοποιούμε την ετικέτα {p3} και συνεπώς έχουμε

{p3}: genquad(“=”, flag, “ 1“, startincase)

***Η’) return :***

Η οποία είναι της μορφής S → return E {p1} και άρα αν το μεταγλωττίσουμε αυτό μάς δίνει {p1} : genquad(“retv”, _ , _, E.place ) . Δηλαδή, απλά σημειώνουμε ότι η επιστροφή τιμής της συνάρτησης γίνεται με το E.place. Δεν χρειάζεται κάτι άλλο.

Τελείως η ίδια λογική είναι και με το input και το print που ακολουθούν.

***Θ’) input:***

S → input id {p1} που όταν το μεταγλωττίζουμε γίνεται:

{p1} : genquad(“inp”, _ , _, id )

***Ι’) print :***

S → print E {p1} που όταν το μεταγλωττίζουμε γίνεται:

{p1} : genquad(“out”, _ , _, E.place )

**Ανάλυση Υλοποίησης Του Ενδιάμεσου Κώδικα Σε Python**

Για να λειτουργήσει ο ενδιάμεσος κώδικάς μάς, αρχικά , δημιουργήσαμε τις συναρτήσεις nextquad(), newtemp(), genquad(), emptylist(), makelist(), merge(), backpatch() και ektypwshEndiamesou() :

- Η nextquad() επιστρέφει την ετικέτα – label της επόμενης τετράδας
- Η newtemp() δημιουργεί μια καινούρια μεταβλητή ,η οποία στην ουσία είναι ένας μετρητής , ο οποίος κάθε φορά αυξάνεται. Άρα δημιουργούνται μεταβλητές τύπου T_1, T_2 κ.ο.κ.
- Η genquad() δημιουργεί στην ουσία λίστες με πέντε στοιχεία και τις ονομάζει τετράδες. Κάθε τετράδα αντιστοιχεί σε μία εντολή και επειδή κάθε εντολή θέλουμε να είναι σε διαφορετική γράμμη, έτσι το label το αυξάνουμε κάθε φορά κατά δέκα για να δείξουμε ότι κάθε τετράδα a.k.a πεντάδα στοιχείων a.k.a. εντολή είναι σε διαφορετική γραμμή. Θα μπορούσαμε να αυξάνουμε το label και κατά ένα , αλλά επιλέξαμε το κατά δέκα όπως κατά το πλείστον το κάναμε και κατά την διάρκεια των διαλέξεων.
- Η emptylist() επιστρέφει μία κενή λίστα
- Η makelist(x) επιστρέφει μία λίστα με ένα στοιχείο x μέσα.

- Η merge(λίστα1,λίστα2) επιστρέφει τις δύο λίστες ενωμένες με σειρά[λίστα1λίστα2]
- Η backpatch() για κάθε ετικέτα της λίστας που δέχεται ως παράμετρο , ελέγχει κάθε τετράδα. Όταν βρίσκει την τετράδα που αντιστοιχεί στην ετικέτα, τότε συμπληρώνει το τελευταίο στοιχείο με την δεύτερη παράμετρο της συνάρτησης, δηλαδή z. Έτσι, για κάθε ετικέτα στην λίστα list1 , την ελέγχω με κάθε τετράδα και όταν βρω την τετράδα που αντιστοιχεί στην ετικέτα την συμπληρώνω με z . Το tetrades[i][0] είναι η θέση της ετικέτας στην τετράδα και η θέση της τετράδας που πρέπει να συμπληρώσουμε είναι η τέταρτη αφού το z είναι το τέταρτο όρισμα (έχουμε και την ετικέτα για αυτό πάει στο τέσσερα).
- Στην συνάρτηση EktypwshEndiamesou() γίνεται η εκτύπωση του ενδιάμεσου κώδικα. Αρχικά εδώ, με την εντολή sys.argv[1][:len(sys.argv[1])-3] κόβουμε το .stl απο το αρχείο εξού και το μείον τρία και έπειτα κάνουμε την μετανομασία σε .int . Το αρχείο που θα γραφούν τα αποτελέσματα το ανοίγουμε μόνο για write για αυτό και η εντολή arxeioInt =open(onomaArxeiou, "w"). Κι έπειτα, για να εκτυπωθούν όλες οι τετράδες χρησιμοποιούμε μία for για να πάρουμε όλο το μήκος κάθε τετράδας και τις μεταβλητές, σταθερές ή σύμβολα που περιέχει τα χωρίζουμε με κόμμα μέσω της συνάρτησης join της python ώστε ό,τι εμφανίζεται να είναι της μορφής για παράδειγμα 100: > , m, n , _ .Όταν τελειώσουμε δε, την εργασία αυτή κλείνουμε το τρέχον αρχείο.

Από εκεί και πέρα, ξεκινώντας να μελετάμε τον κώδικα της πρώτης φάσης παρατηρήσαμε ότι πρέπει να κάνουμε μετατροπές και προσθήκες. Έτσι, στην συνάρτηση program προσθέσαμε την global μεταβλητή onomaProgrammatos όπου εκχωρούμε το όνομα του προγράμματος απο την αρχική μεταβλητή που είχαμε leksh μέσω μιας temporary μεταβλητής,onoma. Καθώς, τώρα στην γραμματική της Starlet η πρώτη λέξη που δέχεται η συνάρτηση block() είναι το onoma , τότε η block() παίρνει ως παράμετρο το onoma και καλούμε την **block(onoma)**.

Ύστερα, στον κώδικα της συνάρτησης block() που είχαμε κρατήσει απο τον συντακτικό αναλυτή, έπειτα απο την κλήση της subprograms() , προσθέσαμε το genquad("begin block", onoma, "_", "_") ώστε να ανοίξουμε **begin block** με όνομα το onoma και μέσα στο begin block εκτελούνται τα statements() , αλλά και ο έλεγχος αν το onoma είναι ίδιο με το onomaProgrammatos και αν είναι το πρόγραμμα τερματίζει επιτόπου μέσω της παραγωγής τετράδας genquad("halt", "_", "_", "_") και το block που ανοίξαμε κλείνει με το **end block** της εντολής genquad("end block", onoma, "_", "_").

Εδώ, να σημειώσουμε (επειδή μπορεί να το παραλείψαμε στην προηγούμενη σελίδα σχετική με συντακτικό αναλυτή) ότι στην συνάρτηση subprogram(), η συνάρτηση funcbody() που καλούμε εκεί, δέχεται και εκείνη την παράμετρο onoma , δηλαδή το id ή αναγνωριστικό μας, γιατί αν κοιτάξουμε “βαθιά” στην γραμματική βλέπουμε ότι η funcbody() οδηγεί στην formalpars() ,η formalpars() οδηγεί στην formalparlist(), η formalparlist() οδηγεί στην formalparitem() και η formalparitem() πέρα απο τα in, inout, inandout περιλάμβάνει το id. Χρειάζεται επομένως να περαστεί ως παράμετρος το όνομα του προγράμματος για να γνωρίζουμε σε ποιο πρόγραμμα βρισκόμαστε ακριβώς. Να σημειωθεί ότι η formalparlist() αφορά δήλωση.

Αργότερα, έχοντας φτάσει στην συνάρτηση statement() , εστιάζουμε στην περίπτωση που έχουμε **exit** , όπου εκεί πρέπει να δημιουργηθεί η επόμενη τετράδα και να κάνουμε jump στο σημείο εκείνο που βρίσκεται αμέσως μετά απο το σημείο που κάνουμε exit. Επομένως, χρησιμοποιούμε τις εντολές

```
elist=makelist(nextquad())
genquad("jump","_","_","_")
 exitList = merge(exitList, elist)
```

όπου στην ουσία δημιουργούμε μία νέα λίστα elist που περιλαμβάνει την επόμενη τετράδα, με την genquad κάνουμε jump, δηλαδή πηδάμε, στην τετράδα που μάς βγάζει απο το πρόγραμμα και δεν ξέρουμε προς το παρόν ποια είναι αυτή και τέλος με την merge , εκχωρούμε στην λίστα exitList την ένωση των λιστών exitList και elist καθώς στην ουσία το “ίδιος” λίστας.

Έπειτα, στην συνάρτηση **assignment_stat()** , αφού κάνουμε τον έλεγχο για τον λεκτικό αναλυτή αν η κατάσταση είναι ίση με “ :=tk “ , καλούμε την expression και εκχωρούμε το αποτέλεσμά της στην μεταβλητή eplace. Ύστερα, με την χρήση genquad , αφού έχουμε κρατήσει το αποτέλεσμα της expression στην μεταβλητή που προαναφέραμε , κάνουμε ανάθεση του eplace στο αναγνωριστικό μας anagnoristiko, το οποίο το έχουμε περάσει και ως παράμετρο στην assignment_stat() , δηλαδή έχουμε assignment_stat(anagnoristiko). Έτσι , έχει τιμή το anagnoristiko μας και μπορούμε να το αξιοποιήσουμε.

Ύστερα, έχοντας φτάσει στην συνάρτηση **if_stat()** κι έχοντας κάνει τους απαραίτητους ελέγχους για “ (tk “ δηλώνουμε δύο λίστες για το condition : η μία λίστα είναι η condTrue για τις εντολές που υλοποιούνται όταν η συνθήκη είναι αληθής και η άλλη λίστα είναι η condFalse όταν η συνθήκη είναι ψευδής. Αφού γίνει , μετά, ο έλεγχος και το token then συμπληρώσαμε τον κώδικα που περιγράψαμε αναλυτικά στο ***A’) if - else – endif***.

Μετά, έχοντας φτάσει στην συνάρτηση **while_stat()** παίρνουμε ως βάση την υλοποίηση που πραγματοποιήσαμε στο ***B’) while - endwhile*** και τοποθετούμε το startWhile = nextquad() πριν απο τον έλεγχο katastash == "(tk" , γιατί σύμφωνα με την γραμματική μας η condition-συνθήκη της συνάρτησης περικλείεται απο παρένθεση. Έτσι λοιπόν και οι λίστες condTrue, condFalse τοποθετούνται μετά τον έλεγχο του "(tk" και πριν απο τον έλεγχο του "}tk" για να έχουμε ήδη έτοιμες τις λίστες μετά όταν μεταβούμε στην συνθήκη if katastash == ")tk" όταν είναι αληθής για να ξεκινήσουμε την διαδικασία που προαναφέραμε στο *Χειρισμός Βασικών Δομών Της Starlet.*

Τώρα, ακολουθεί η συνάρτηση **do_while_stat()** , την οποία υλοποιήσαμε με βάση την ***Γ’) do – while*** που έχουμε προαναφέρει. Εδώ πάλι λαμβάνουμε υπόψιν τις παρενθέσεις που περικλείουν το condition , ώστε ο κώδικάς μας να εξυπηρετεί την μεταγλώττιση. Γίνεται , λοιπόν, κατανοητό ότι για αυτό τον λόγο η δημιουργία των true και false λιστών και το backpatch της λίστας true που οδηγεί στην αρχή της do_while_stat() γίνεται στο ενδιάμεσο τμήμα ελέγχου των if katastash == "(tk" και if katastash == ")tk".

Στο πρόγραμμά μάς, έπειτα, σειρά έχει η συνάρτηση loop_stat() που ακολουθεί την υλοποίηση του ***Δ’) loop - endloop*** που εξηγήσαμε αναλυτικά παραπάνω. Εδώ, να σημειώσουμε μόνο ότι το endloop ΄της Δ’) loop – endloop γίνεται στον κώδικά μας στο σημείο if katastash == "endlooptk": . Επίσης, να σημειώσουμε κάτι ακόμα που δεν αναφέραμε προηγουμένως και ισχύει γενικά για όλο το αρχείο μας : εμείς στην δεύτερη

φάση της εργασίας δημιουργούμε τον ενδιάμεσο κώδικα. Χτίζοντας πάνω στον κώδικα της πρώτης φάσης αντιληφθήκαμε ότι η θέση που καλούμε τον λεκτικό αναλυτή lektikos() δεν επηρεάζει το πού θα τοποθετήσουμε π.χ. κάποια genquad ή backpatch. Για παράδειγμα, το τμήμα κώδικα

```
if katastash == "endlooptk":
 lektikos()
genquad("jump","_","_",startLoop)
backpatch(exitList, nextquad())
```

είναι ισοδύναμο με το

```
if katastash == "endlooptk":
genquad("jump","_","_",startLoop)
backpatch(exitList, nextquad())
lektikos()
```

Επισπρόθετα, ακολουθεί η συνάρτηση **forcase_stat()** η οποία υλοποιήθηκε με βάση το ***Ε') forcase*** - **…** - ***endforcase*** . Απο εκεί και πέρα, όμως λάβαμε υπόψιν και την γραμματική της Starlet για να τοποθετήσουμε τις κατάλληλες εντολές στα κατάλληλα σημεία του αρχείου μας int.py . Κι έτσι , εντός των παρενθέσεων δημιουργήσαμε τις λίστες true και false λόγω της condition. Ο υπόλοιπος κώδικας της δομής αυτής δεν διαφέρει σε σχέση με την γραμματική που έχουμε, αν λάβουμε υπόψιν τους ελέγχους που υπάρχουν όπως τον if katastash == ":tk": και αν τον αντιστοιχίσουμε το σύμβολο της Startlet : που υπάρχει στον αντίστοιχο κανόνα, γίνεται εύκολα αντιληπτό η νοοτρόπια πρόσθεσης των εντολών που γράψαμε στον *Χειρισμό Βασικών Δομών Της Starlet*, αλλά τώρα προσαρμόζοντάς τους στο αρχείο μας int.py . Για παράδειγμα, οι εντολές

```
ifkatastash=="defaulttk"
lektikos()
if katastash == ":tk":
```

αντιστοιχούν στην γραμματική μας στο τμήμα default: .

Αργότερα, ακολουθεί η συνάρτηση incase_stat(), η οποία υλοποιήθηκε με βάση το ***ΣΤ') incase*** - **…** - ***endincase*** που περιγράψαμε παραπάνω και προσαρμόστηκε στο int.py αρχείο μας. Επίσης, στις συναρτήσεις return_stat() , print_stat() και input_stat() έγιναν οι κατάλληλες τροποποιήσεις στο αρχικό μας πρόγραμμά σύμφωνα με την ανάλυση που κάναμε στα κομμάτια ***Η') return*** , ***Θ') input*** και ***Θ') input*** αντίστοιχα.

Οι τροποποιήσεις που ακολουθούν τώρα αφορούν την **actualparitem()** συνάρτηση. Εδώ στην ουσία γίνεται το πέρασμα των παραμέτρων και όπως έχουμε προαναφέρει, υπάρχει η εξής αντιστοιχία :

```
CV    -> in
REF   -> inout
CP    -> inandout    .
```

Συνεπώς, κάνοντας τον κατάλληλο έλεγχο της katastash ,ο οποίος μάς δείχνει σε ποια περίπτωση βρισκόμαστε, δημιουργούμε και την κατάλληλη τετράδα αναλόγως. Με άλλα λόγια στην περίπτωση in παράγουμε το genquad("par", eplace, "CV", "_") , στην περίπτωση inout παράγουμε την genquad("par", leksh, "REF", "_") και στην περίπτωση inandout παράγουμε την genquad("par", leksh, "CP", "_") . Απο εκεί και πέρα αν θέλουμε να εξηγήσουμε κάτι περαιτέρω για κάθε περίπτωση στην call by value έχουμε την μεταβλητή eplace που επιστρέφει το αποτέλεσμα, στην

call by reference έχουμε την leksh την λεκτική μονάδα δηλαδή και στην περίπτωση call by copy έχουμε πάλι την λεκτική μονάδα leksh.

Μετά , έχουμε την συνάρτηση **condition()** στον κώδικά μας.Σε αυτό το σημείο χειριστήκαμε την συνάρτηση με βάση τις ετικέτες πάλι και υλοποιήσαμε το εξής λαμβάνοντας υπόψιν τον κανόνα του boolterm:

$C \rightarrow BT^{(1)}$ {p1}( or {p2} $BT^{(2)}$ {p3})* , όπου C σημαίνει condition και BT σημαίνει boolterm.

Εδώ στο {p1} οι τετράδες που θα αποτύχουν εδώ είναι οι πρώτες τετράδες $BT^{(2)}$ άρα nextquad() , δηλαδή πρώτη τετράδα του $BT^{(2)}$. Έχουμε, όπως γίνεται αντιληπτό, δύο λίστες true και false όπου περνάμε πληροφορία σε C και ό,τι δε μπορεί να χειριστεί το περνάμε επάνω. Έπειτα, συμπληρώνει ό,τι μπορεί και ό,τι δε μπορεί περνάει C.true και C.false.

{p0}: C.true = $BT^{(1)}$.true στο {p1} δεν μπορούμε να συμπληρώσουμε, περνάμε προς τα επάνω. Εδώ έχουμε ένα BT στην θέση ενός C

C.false = $BT^{(2)}$.false

{p1}: backpatch(C.false, nextquad()) βάλαμε C για να είναι αισθητικά πιο ωραίο

{p2}: $BT^{(1)}$.false = merge($BT^{(1)}$.false, $BT^{(2)}$.false)

$BT^{(1)}$.true = $BT^{(2)}$.true , απλά περνάμε την $BT^{(1)}$ Απο $BT^{(1)}$ διαχειριστήκαμε την C.false .Δεν ξέρουμε πού κάνουμε jump C.true και C.false. Τα έξω jump σς μπορούμε να τα διαχειριστούμε.Εμείς διαχειριστήκαμε τα μέσα jump.

{p3}: C.false = $BT^{(2)}$.false όπου C.false έχει γίνει backpatch άρα είναι άδεια

C .false = merge($BT^{(1)}$.true, $BT^{(2)}$.true) , όμως το $BT^{(1)}$.true γίνεται C .true , δηλαδή κάνουμε τις τρεις λίστες δύο (όπου για τις λίστες έχουμε τρύπιες τετράδες στο τελευταίο τελούμενο συμπληρώνουμε τις λίστες). Δεν υπάρχει C.false . Πρέπει την BT .false να την βάλουμε στην C .false. Ακόμα, πρέπει να βάλουμε το τρίτο true στην λίστα.

Στην συνέχεια έχουμε την συνάρτηση **boolterm()** στο πρόγραμμά μας. Σε αυτό το σημείο χειριστήκαμε την συνάρτηση με βάση τις ετικέτες πάλι και υλοποιήσαμε το εξής λαμβάνοντας υπόψιν τον κανόνα του boolfactor:

$BT \rightarrow BF^{(1)}$ ( and {p1} $BF^{(2)}$ {p2})* {p0} , όπου BT σημαίνει boolterm και BF σημαίνει boolfactor.

Το BF δίνει δύο λίστες true και false που πρέπει να συμπληρωθούνε και κάνουμε εκεί jump. Γυρνάμε στις λίστες που έχουν σημαδάκια στις τετράδες και έχουν μείνει τρύπιες.

{p0}: BT.true = $BF^{(1)}$.true

BT.false = $BF^{(1)}$.false Για να συνεχιστεί η εκτέλεση, πρέπει να είναι true. Η BT.true συμπληρώνει την επόμενη τετράδα με $BF^{(2)}$. Άρα συμπληρώνει:

{p1}: backpatch($BF^{(1)}$.true, nextquad())

Δεν υπάρχει BT λίστα. Απο εδώ παίρνουμε τρεις λίστες BT.false, $BF^{(2)}$.true και $BF^{(2)}$.false. Ενώνουμε τις δύο false γιατί δε μας νοιάζει ποια δεν ίσχυσε, θα μας πάνε στο ίδιο σημείο.

{p2}: $BF^{(1)}$.false = merge($BF^{(1)}$.false, $BF^{(2)}$.false) όπου $BF^{(1)}$.false είναι BT.false αλλά το έχουμε έτσι γιατί σε διαφορετική περίπτωση θα μας δημιουργούσε πρόβλημα

$BF^{(1)}$.true = $BF^{(2)}$.true , απλά περνάμε την $BF^{(1)}$.

Παρακάτω, υπάρχει η συνάρτηση **boolfactor()**, η οποία υλοποιήθηκε με βάση το εξής: $BF \rightarrow E^{(1)}$ relop $E^{(2)}$ {p0} , δηλαδή έχουμε

{p0}: BF.true = makelist(nextquad()) δημιουργία λίστας για true

genquad(relop, $E^{(1)}$.place, $E^{(2)}$.place,”_”) αν ισχύει η σχέση relop, $E^{(1)}$.place, $E^{(2)}$.place κάνει jump στο ”_”

BF.false = makelist(nextquad()) δημιουργία λίστας για false

genquad("jump","_","_","_”) αν δεν ισχύει το relop του {p0} κάνει jump εδώ.

Καταλαβαίνουμε, λοιπόν, ότι εδώ έχουμε μία σύγκριση που πρέπει να διαχειριστούμε. Τα $E^{(1)}$.place και $E^{(2)}$.place είναι δύο μεταβλητές. Ό,τι δε μπορούμε να το συμπληρώσουμε, όποια τετράδα, το περνάμε επάνω true και false. Επίσης , έχουμε και την περίπτωση στην γραμματική που το condition είναι ανάμεσα σε αγκύλες , δηλαδή έχουμε κανόνα της μορφής R → ( B ) {p1} όπου R είναι ο boolfactor και B το condition στην γραμματική. Το B δουλεύει. Απο κάτω έρχονται B.true και B.false. Οι αγκύλες είναι για την προτεραιότητα. Δηλαδή έχουμε , {p1}: R.true = B.true

R.false = B.false για αυτό και στον κώδικα δημιουργούμε και επιστρέφουμε δύο λίστες. Όμως , έχουμε και άλλη μία περίπτωση, αυτή του not. Ο κανόνας της γραμματικής έχει την μορφή R → not ( B ) {p1} , και καταλαβαίνουμε ότι πρόκειται για μια απλή αντιστροφή των λιστών , δηλαδή έχουμε

R.true = B.false (στο πρόγραμμά μας bfTrue = condTrue)

R.false = B.true (στο πρόγραμμά μας bfFalse = condFalse) .

Ακόμα, το αποτέλεσμα της συνάρτησης expression() το εκχωρούμε σε μια μεταβλητή e1place, το αποτέλεσμα της συνάρτησης relational_oper() το εκχωρούμε στην μεταβλητή relop και ξανακαλούμε την συνάρτηση expression() εκχωρώντας αυτή την φορά το αποτέλεσμα στην μεταβλητή e2place και αυτό λόγω του κανόνα <expression>< relational_oper><expression>.

Μετά, ακολουθεί η συνάρτηση **expression().** Εδώ μπορούμε να έχουμε είτε πρόσθεση είτε αφαίρεση, για αυτό χρησιμοποιούμε μία μεταβλητή op , η οποία μπορεί να πάρει είτε συν είτε μείον και αναλόγως την περίπτωση εκτελείται και η κατάλληλη πράξη.

$E \rightarrow T^{(1)}$ ( op $T^{2)}$ {p1})* p{2} ,όπου op είναι + ή – και το {p1} είναι πριν το τέλος του while (λόγω kleene star) και στο {p1} ζητάμε να μάς δώσει μια νέα προσωρινή μεταβλητή w = newtemp() που θα είναι string. Έχουμε τώρα την πληροφορία για να φτάξουμε π.χ. a+b . Θέλουμε όμως να γυρίσει ,για παράδειγμα, μια πρόσθεση ανάμεσα σε $T^{(1)}$ , $T^{(2)}$ και w για αυτό γράφουμε genquad(op, t1place, t2place, w) . Όπως γυρίζει τώρα το while περνάει απο το {p1} για αυτό θέλουμε άλλη μια τετράδα και γράφουμε t1place = w στο πρόγραμμά μας. Άρα, {p2} : eplace = t1place στον κώδικά μας για να μάς γυρίζει αυτά που θέλουμε. Να σημειώσουμε ότι αν δεν έχουμε kleene star , δε θα μπει ποτέ στις εντολές w=newtemp(), genquad(op,t1place,t2place,w) και t1place = w. Εμείς, όμως, θέλουμε κώδικα.

Σειρά, τώρα, έχει η συνάρτηση **term()**. Σε αυτό το σημείο του προγράμματός μας διαχειριζόμαστε τον πολλαπλασιασμό και την διαίρεση και η φιλοσοφία αυτής της συνάρτησης είναι ίδια με την φιλοσοφία της υλοποίησης τής πρόσθεσης και της αφαίρεσης που αναλύσαμε μόλις πριν. Με άλλα λόγια έχουμε μία μεταβλητή op , η οποία μπορεί να πάρει είτε επί είτε διά και αναλόγως την περίπτωση εκτελείται και η κατάλληλη πράξη:

$T \rightarrow F^{(1)}$ ( op $F^{2)}$ {p1})* p{2},όπου op είναι * ή / και το {p1} είναι πριν το τέλος του while (λόγω kleene star) και στο {p1} ζητάμε να μάς δώσει μια νέα προσωρινή μεταβλητή w = newtemp() που θα είναι string. Έχουμε τώρα την πληροφορία για να φτάξουμε π.χ. a*b . Θέλουμε όμως να γυρίσει ,για παράδειγμα, έναν πολλαπλασιασμό

ανάμεσα σε $F^{(1)}$ , $F^{(2)}$ και w για αυτό γράφουμε genquad(op, f1place, f2place, w) . Όπως γυρίζει τώρα το while περνάει απο το {p1} για αυτό θέλουμε άλλη μια τετράδα και γράφουμε f1place = w στο πρόγραμμά μας. Άρα, {p2} : tplace = fplace στον κώδικά μας για να μάς γυρίζει αυτά που θέλουμε. Να σημειώσουμε ότι αν δεν έχουμε kleene star , δε θα μπει ποτέ στις εντολές w = newtemp(), genquad(op, f1place, f2place, w) και f1place = w.

Εν συνεχεία, υπάρχει η συνάρτηση **factor()** που αποτελείται απο τρεις περιπτώσεις. Στην περίπτωση που έχουμε σταθερά κρατάμε την λεκτική μονάδα σε μια μεταβλητή place και την επιστρέφουμε , αν είμαστε στην περίπτωση της expression, απλώς ελέγχουμε για παρενθέσεις και επιστρέφουμε αυτό που επιστρέφει η συνάρτηση expression() μέσω μιας μεταβλητής place και τέλος αν βρισκόμαστε στην περίπτωση του id ,δηλαδή του αναγνωριστικού μας, τότε κατανοούμε ότι έχουμε την περίπτωση :

F → id {p0} | (E) {p1} και άρα έχουμε

{p0}: F.place = id

{p1}: F.place = E.place . Βασικά στον κώδικά μας το idtail εάν δεν είναι κενό σημαίνει ότι έχουμε κλήση συνάρτησης και το αποτέλεσμα είναι η παράμετρος RET που θα βρει. Περνάμε την σταθερά που έχουμε ώστε και να την βάλουμε στο call και να την επιστρέψουμε αν το idtail είναι το κενό(e/E).

Ύστερα, ακολουθεί η συνάρτηση **idtail(place)** , η οποία παίρνει ως παράμετρο το place γιατί το id θα το χρειαστούμε στο actualparitem(). Αυτό συμβαίνει, επειδή καλείται η actualpars() που καλεί το actualparlist() που καλεί την actualparitem() που καταναλώνει το id. Είναι η μοναδική περίπτωση που περνάμε κάτι απο κάτω προς τα πάνω. Κατά τα άλλα , δημιουργούμε μια νέα μεταβλητή w και παράγουμε μια νέα παράμετρο w την οποία και θα επιστρέψουμε αφού γίνει κλήση. Να σημειωθεί εδώ ότι η actualpars() αφορά κλήση.

Έπειτα, στην συνάρτηση **relational_oper()** γίνεται ο έλεγχος για να δούμε σε ποια περίπτωση σύγκρισης βρισκόμαστε. Αν έχουμε ισότητα, μικρότερο ίσο, μεγαλύτερο ίσο, μεγαλύτερο, μικρότερο και διάφορο. Την συνάρτηση αυτή σύμφωνα με την γραμματική την καλούμε στην συνάρτηση boolfactor().

**Εκτύπωση Ενδιάμεσου Κώδικα σε C**

Την εκτύπωση του ενδιάμεσου κώδικα σε γλώσσα C την υλοποιούμε στην συνάρτηση **def ektypwshC()**. Εκεί κρατάμε το όνομα του αρχείου εισόδου χωρίς την κατάληξη .stl και βάζουμε την κατάληξη .int για την C. Το αρχείο arxeioC το ανοίγουμε για write γιατί θέλουμε να γράψουμε στο αρχείο, δηλαδή για να εμφανίζεται ο κώδικας μορφοποιημένος όπως μάς ζητείται.

Πιο ειδικά, αρχικά στο αρχείο που θα είναι γραμμένο σε γλώσσα προγραμματισμού C συμπεριλαμβάνουμε την βιβλιοθήκη #include <stdio.h> και επίσης χρειαζόμαστε μια int main() , αφού στο τέλος έχουμε το return 0; . Έπειτα, στο αρχείο μας για όλες τις τετράδες που έχουμε δημιουργήσει δηλώνουμε όλες τις μεταβλητές , τύπου int. Κρατάμε ξεχωριστά αυτές που είναι προσωρινές για αυτό και έχουμε και arxeioC.write("int "+metavlhtes[i]

+";\n") και arxeioC.write("int "+metavlhtesTemp[i] +";\n") . Στις δύο τελευταίες εντολές βάζουμε και το ερωτηματικά backslash n διότι οι εντολές στην c θέλουν ερωτηματικό στο τέλος για να εκτελεστούν.

Έπειτα, για να ξέρουμε σε ποια τετράδα βρισκόμαστε κρατάμε την ετικέτα στην οποία βρισκόμαστε σε σχόλια. Για παράδειγμα, μέσω της εντολής arxeioC.write("//"+",".join(tetrades[i]) +"\n") αναλόγως σε ποιο σύμβολο βρισκόμαστε μπορεί να εμφανιστεί λόγου χάρη το //50,+,k,l,T_3 ως σχόλιο αν βρισκόμαστε στο σημείο Label50: T_3=k+l; στο πρόγραμμά μας που είναι πια σε C.

Δηλαδή, εμείς έχουμε πάρει όλες τις πιθανές περιπτώσεις που μπορεί να προκύψουν μέσω μιας if-elif-else. Πιο συγκεκριμένα, για κάθε τετράδα παίρνουμε το Label και για τον αριθμό της ετικέτας χρησιμοποιούμε το στοιχείο που βρίσκεται στην μηδενική θέση της τετράδας αφού αυτό μάς υποδηλώνει σε ποια τετράδα βρισκόμαστε και έτσι χειριζόμαστε κατάλληλα την υλοποίηση αν έχουμε πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, εκχώρηση, jump, τα σύμβολα μικρότερο, μικρότερο ίσο, μεγαλύτερο, διάφορο, ισότητα, exit ή return. Βέβαια, εμείς μετά το Label που το έχουμε ως string και τον αριθμό στον οποίο αντιστοιχεί κάθε φορά , έχουμε και άνω κάτω τελεία πιο πολύ για οπτικούς λόγους, να είναι ωραία η εκτύπωση τού κώδικά μας.

Αν έχουμε κάποια πράξη +, - , * ή / τότε παίρνουμε το στοιχείο που βρίσκεται στην τέταρτη θέση , γιατί εκεί βρίσκεται η μεταβλητή στην οποία θέλουμε να αποθηκεύσουμε το αποτέλεσμα, γράφουμε ίσον γιατί στην C έτσι έχουμε την εκχώρηση και έπειτα παίρνουμε ό,τι βρίσκεται στην δεύτερη θέση που μπορεί π.χ. να είναι μια μεταβλητή ή ένας αριθμό , συνεχίζουμε προσθέτοντας το κατάλληλο string +, - , * και / αναλόγως την πράξη που έχουμε να υλοποιήσουμε και έπειτα προσθέτουμε και το τρίτο στοιχείο της τετράδας που μπορεί να είναι μεταβλητή ή αριθμός και βάζουμε στο τέλος ερωτηματικό για να είναι ορθή η μορφή τής εντολής στην C. Για παράδειγμα, με τον παραπάνω τρόπο δημιουργούμε την εντολή Label360: T_13=a+1; ,η οποία μέσω των σχολίων που έχουμε προσθέσει //360,+,a,1,T_13 μάς δείχνει ξεκάθαρα σε ποιο σημείο του προγράμματός μας βρισκόμαστε.

Ύστερα, έχουμε την περίπτωση της εκχώρησης. Στην γραμματική μας η εκχώρηση γίνεται με το σύμβολο := . Παρόλα αυτά, στην γλώσσα C η εκχώρηση γίνεται με το σύμβολο =. Επίσης, όπως έχουμε δημιουργήσει εμείς τις τετράδες, εκχωρούμε το στοιχείο που βρίσκεται στην δεύτερη θέση τής τετράδας στην τέταρτη θέση της τετράδας μας και εννοείται προσθέτουμε το ερωτηματικό μάς στο τέλος. Για παράδειγμα, η τετράδας που βρίσκεται σε σχόλια //470,:=,1,_,k που στην γραμματική μας αυτό σημαίνει Label 470: -k := 1 και είναι μια ανάθεση , στην C αντιστοιχεί στο Label470: k=1; .

Μετά, ακολουθεί η εντολή jump που στην γλώσσα C αντιστοιχεί στο goto και συνεχίζουμε με την ετικέτα και την τέταρτη θέση στην τετράδα που έχουμε δημιουργήσει αφού μάς δείχνει σε ποιο σημείο τού προγράμματός μας μεταβαίνουμε και φυσικά στο τέλος ακολουθεί ερωτηματικό. Για παράδειγμα, έχουμε σε σχόλιο την τετράδα //460,jump,_,_,490 ,δηλαδή βρισκόμαστε το Label 460 και θέλουμε να μεταβούμε στο Label 490. Έτσι, λοιπόν, δημιουργούμε την εντολή Label460: goto Label490; .

Πλέον, έχουμε φτάσει στο σημείο των συγκρίσεων μέσα στις περιπτώσεις μας και οι περιπτώσεις των <, > , <> , <= , >= , = μοιάζουν. Εμείς γνωρίζουμε ότι για να έχουμε μία σύγκριση στην C πρέπει να έχουμε κάποια if και μέσα στην παρένθεση να γίνεται η σύγκριση. Σύμφωνα, λοιπόν, με το πρόγραμμα που υλοποιούμε και όλα τα παραπάνω , η

σύγκριση στις τετράδες μας γίνεται μεταξύ του δεύτερου και του τρίτου στοιχείου τους και η κάθε σύγκριση μάς οδηγεί σε ένα label ,δηλαδή με goto στο τέταρτο στοιχείο της εκάστοτε τετράδας και στο τέλος έχουμε κλασσικά το ερωτηματικό μας. Για παράδειγμα, η τετράδα που βρίσκεται μέσα σε σχόλια //450,<,a,b,470 αντιστοιχεί στο Label450: if(a<b) goto Label470; . Σε αυτό το σημείο βέβαια να τονίσουμε ότι έχουμε κάνει και την εξής αντιστοιχία μεταξύ τής γραμματικής μας και της γλώσσας προγραμματισμού C : το σύμβολο της γλώσσας Starlet <> στην C αντιστοιχεί στο σύμβολο != και το σύμβολο της γλώσσας Starlet = στην C αντιστοιχεί στο σύμβολο ==. Συνεπώς, αυτά που αναφέραμε σε αυτή την παράγραφο λαμβάνουν υπόψιν αυτές τις αντιστοιχίες ώστε το πρόγραμμά μας να εκτελεί ορθά την παρούσα λειτουργία.

Επίσης, μέσα στο if-elif-else τής προηγούμενης παραγράφου υπάρχει και ο έλεγχος elif tetrades[i][1] == "out" όπου στην ουσία το print της Starlet το κάναμε “out” στον ενδιάμεσο κώδικα και τώρα στην γλώσσα C το μετατρέπουμε σε string printf συν το στοιχείο συν τέταρτη θέση τής τετράδας για να εκτυπώνονται τα σωστά αποτελέσματα στο αρχείο που έχουμε ως test. Για παράδειγμα, η γραμμή //160,out,_,_,b δηλαδή η ετικέτα 160 με out την μεταβλητή b αντιστοιχεί στην γλώσσα C στο Label160: printf("%d\n",b); .

Βέβαια, με παρόμοιο τρόπο εργαστήκαμε και για την εμφάνιση τού scanf στην γλώσσα C, στην οποία θέλουμε να γίνεται η εκτύπωση αρχείου. Η μόνη διαφορά με την printf στην γλώσσα C είναι ότι στο scanf έχουμε και το σύμβολο & στις μεταβλητές. Πιο συγκεκριμένα, μέσα στην ίδια if-elif-else που πριν λίγο αναφέραμε, υπάρχει και ο έλεγχος elif tetrades[i][1] == "inp" όπου στην ουσία το input της Starlet το κάναμε “inp” στον ενδιάμεσο κώδικα και τώρα στην γλώσσα C το μετατρέπουμε σε string scanf συν το στοιχείο συν τέταρτη θέση τής τετράδας για να εκτυπώνονται τα σωστά αποτελέσματα στο αρχείο που έχουμε ως test. Για παράδειγμα, η γραμμή //480,inp,_,_,a δηλαδή η ετικέτα 480 με inp την μεταβλητή a αντιστοιχεί στην γλώσσα C στο Label480: scanf("%d",&a); . Έτσι, λοιπόν, διαμορφώνονται σωστές εντολές στην γλώσσα C για την printf και την scanf.

Ακόμα, για οποιαδήποτε άλλη περίπτωση εκτυπώνουμε απλά το ερωτηματικό. Για παράδειγμα, όταν έχουμε ξαφνική έξοδο από το block στο οποίο βρισκόμαστε μέσω της εντολής halt. Δηλαδή το η τετράδα 690,halt,_,_,_ που έχουμε εμείς σε σχόλια αντιστοιχεί σε γλώσσα C στο Label690: ; .

Τέλος, αφού έχουμε τελειώσει με όλες τις παραπάνω περιπτώσεις για να ολοκληρωθεί ένα πρόγραμμα σε C πρέπει πριν το κλείσουμε να γράφουμε return 0; και στην επόμενη σειρά να κλείνει η αγκύλη. Για αυτό τον λόγο έχουμε γράψει και τις δύο εντολές arxeioC.write("return 0;\n") και arxeioC.write("}\n") και φυσικά κάνουμε close() για να κλείσει το αρχείο C μας.

Εν κατακλείδι, αφού έχουμε ολοκληρώσει και την εκτύπωση σε γλώσσα C κάτω κάτω στο πρόγραμμά μας να σημειώσουμε ότι έχουμε τα
lektikos()
program()
ektypwshC()
ektypwshEndiamesou()
ώστε να καλούνται οι συναρτήσεις μας για όλο το μέγεθος του αρχείου και να εκτελούνται ομαλά όλες οι λειτουργίες.

**<u>Σημασιολογική Ανάλυση</u>**

Παράλληλα με την συντακτική ανάλυση γίνεται και η σημασιολογική ανάλυση. Ο σημασιολογικός αναλυτής διαπιστώνει τα σφάλματα και ενημερώνει τον προγραμματιστή για αυτό.

***Πίνακας Συμβόλων***

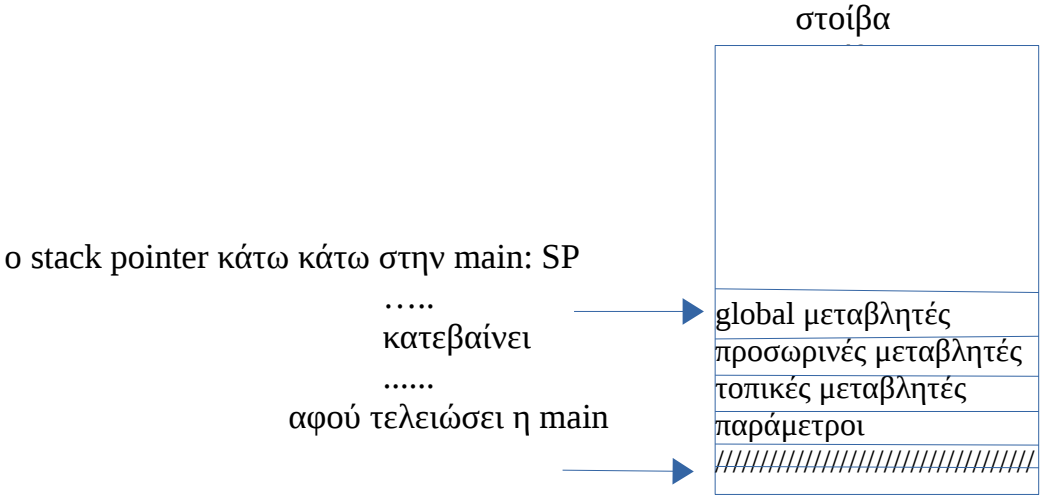
**Εισαγωγικά**

Πέρα απο την λεκτική και την συντακτική ανάλυση , όμως, υπάρχει και η σημασιολογική ανάλυση, δηλαδή η μετάφραση καθοδηγούμενη απο την σύνταξη. Σε αυτό το κομμάτι του project μας μάς βοηθάει ο πίνακας συμβόλων. Ο πίνακας συμβόλων είναι μία δυναμική δομή όπου αποθηκεύεται πληροφορία σχετικά με τα σύμβολα (μεταβλητές και συναρτήσεις) του προγράμματος. Έτσι, με τον πίνακα συμβόλων γνωρίζουμε ποιος μπορεί να τρέξει τί, ποιος μπορεί να δει τί κλπ. Ειδικότερα, πίνακας συμβόλων προσπελαύνεται κάθε φορά που ο compiler συναντά κάποιο σύμβολο,γιατί ο compiler μάς πρέπει να ξεχωρίζει τον τύπο κάθε συμβόλου όταν το συναντά. Επίσης, για τον πίνακα συμβόλων χρησιμοποιούμε τρεις διαφορετικές οντότητες: τετράγωνα, τρίγωνα και κύκλους.

Κάθε τετράγωνο ονομάζεται **entity** και αφορά μεταβλητές, παραμέτρους και συναρτήσεις. Κάθε entity έχει όνομα και τύπο. Κάθε τύπος μπορεί να είναι είτε μεταβλητή που περιέχει το offset ( όπου offset είναι η απόσταση της μεταβλητής από την αρχή του εγγραφήματος) είτε παράμετρος που περιέχει το offset και τον τύπο της παραμέτρου που ενδέχεται να είναι CV, REF ή CP είτε συνάρτηση που περιέχει την startquad (όπου startquad είναι το πρώτο quad δηλαδή τετράδα της συνάρτησης δηλαδή begin block), το framelength (όπου framelength είναι το μήκος του εγγραφήματος σε bytes δηλαδή πόσο χώρο χρειάζεται στην μνήμη) και την λίστα arguments (όπου arguments είναι τα δικά μας τρίγωνα δηλαδή παράμετροι). Οι συναρτήσεις στον πίνακα συμβόλων γράφονται σαν entity γιατί τις χρειαζόμαστε για τον τελικό κώδικα. Δεν μετράνε καθόλου στο framelength.

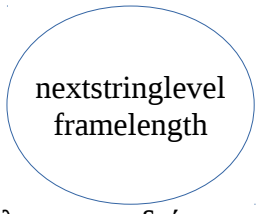
Εδώ είναι φρόνιμο να αναφέρουμε ότι μάς γεννούνται τα ερωτήματα “ Πού είναι ο δείκτης ο στοίβας;” , “Πού είναι η μεταβλητή μου;” για αυτό και χρησιμοποιούμε offset. Το offset μάς δείχνει πού είναι η μεταβλητή, πόσες θέσεις ή αλλιώς bytes επάνω. Επίσης το framelength αφορά τον χώρο στην μνήμη. Ακόμα, ο δείκτης στα ορίσματα που χρησιμοποιούμε είναι για να κρατάμε πολλά πράγματα π.χ. τύπο , ορίσματα και άλλα. Ακόμα, ας συνεχίσουμε ότι τις παραμέτρους τις χρησιμοποιούμε δύο φορές : μία σε μορφή entity και μία σε δείκτη παραμέτρων. Γενικά τις παραμέτρους πρέπει να τις γράψουμε, να τις διαβάσουμε, να τις βρούμε στην μνήμη και να ξέρουμε τί δικαιώματα έχουμε.

Επιπρόσθετα, στις σταθερές δεν κρατάμε offset γιατί η ανάθεση γίνεται σε χρόνο εκτέλεσης.
Παρακάτω, παραδίδουμε ένα απλό σχήμα για το τί συμβαίνει στην στοίβα:




π.χ. έχουμε την σταθερά const int A=3 . Τί πληροφορίες πρέπει να κρατήσουμε για το A;

Επιπλέον, έχουμε τους κύκλους που ονομάζονται **scopes**. Τα scopes αφορούν τις συναρτήσεις και το κυρίως πρόγραμμα. Κάθε scope έχει όνομα, βάθος φωλιάσματος δηλαδή πόσο μέσα είναι η επόμενη συνάρτηση, enclosing scope που δείχνει στο scope μέσα στο οποίο είναι δηλωμένο και λίστα απο entities.Βολεύει μέσα στον κύκλο να έχουμε και το framelength, γιατί το framelength είναι βοηθητικό αφού ανανεώνεται διαρκώς και μάς εξυπηρετεί ώστε να γνωρίζουμε το “μέγεθος”. Δηλαδή, σχηματικά θα μπορούσε να υφίσταται το εξής:




struct με όρους C

,όπου nextstringlevel και framelength είναι δύο ακέραιοι

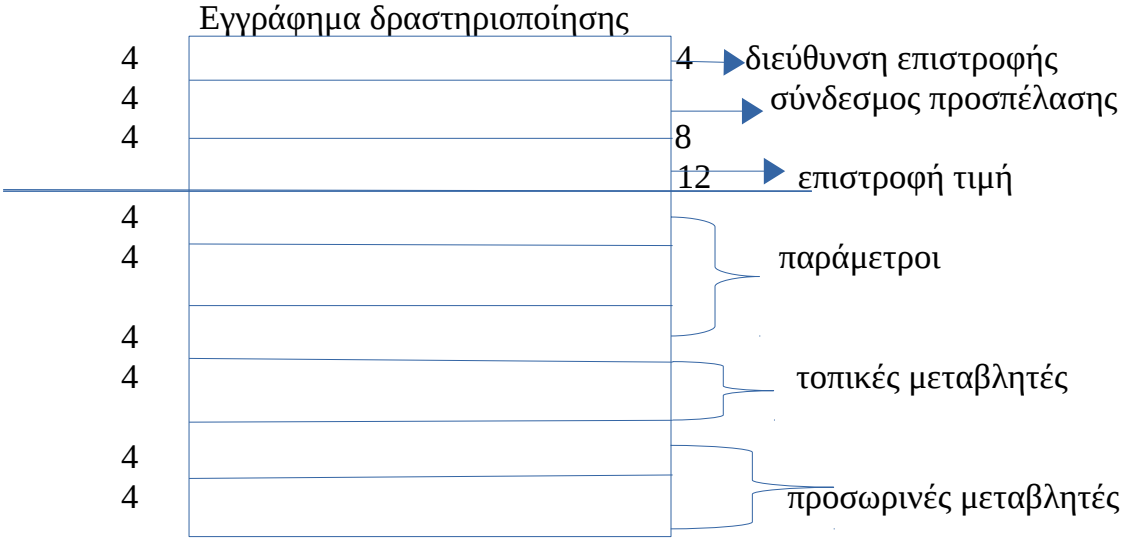
Παράλληλα, για δείκτη στα ορίσματα έχουμε τα τρίγωνα που ονομάζονται **arguments**. Τα arguments περιέχουν μόνο τον τύπο. Να σημειώσουμε ότι στα τρίγωνα μπορούμε να κρατάμε και τον τύπο περάσματος και τον τύπο παραμέτρου. Αν κάναμε μία λίστα απο τρίγωνα θα ήμασταν εντάξει. Στην λίστα με τα τρίγωνα παίζει ρόλο και η σειρά που έχουν τα τρίγωνα και το πλήθος που έχουν τα τρίγωνα.

Έχοντας αναφέρει όλα αυτά στην αρχή, είναι πρέπον να αναφέρουμε τώρα και το τί απαίτειται να υλοποιήσουμε στο πρόγραμμά μας. Αυτά που πρέπει να υλοποιήσουμε είναι:

- Εισαγωγή entity
- Εισαγωγή argument
- Εισαγωγή scope
- Διαγραφή scope
- Αναζήτηση
- και φυσικά να τυπώνεται μήνυμα λάθους αν δεν βρίσκουμε κάτι.

Ταυτόχρονα, για να εξηγήσουμε καλύτερα παρακάτω τί συμβαίνει στον κώδικα που προσθέσαμε για τον πίνακα συμβόλων και την σημασιολογική ανάλυση πρέπει να

εξηγήσουμε κάποια θεωρητικά πράγματα ακόμα που σχετίζονται με αυτό το στάδιο της εργασίας. Έτσι, λοιπόν, παραθέτουμε σχηματικά το εγγράφημα δραστηριοποίησης για να εξηγήσουμε τον λόγο που ξεκινάμε απο τα 12 bytes σε κάθε entity:




όπου η διεύθυνση επιστροφής είναι το σημείο που κάνει jump η συνάρτηση. Ξέρουμε πού θα ξεκινήσουμε, αλλά δεν ξέρουμε πού θα επιστρέψουμε. Ο σύνδεσμος προσπέλασης είναι ένας δείκτης στο πού θέλουμε να δούμε μια μεταβλητή (πού είναι στο δείκτη στοίβας) απο πού θα ξεκινήσουμε να ψάχνουμε και η επιστροφή τιμής είναι πρακτικά η διεύθυνση που δημιουργούμε π.χ. 11: par, t_2 RET , εκεί θα πάει να γράψει. Αυτά τα 12 bytes δεν τα χρησιμοποιούμε πάντα όλα, αλλά αυτά απλά υπάρχουν ακόμα και τότε. Να υπενθυμίσουμε ότι στο σχήμα το κάθε ορθογώνιο είναι 4 bytes.

Ορισμένες έννοιες που , επιπλέον, είναι καλό να παραθέσουμε είναι οι παρακάτω:

- Διεύθυνση επιστροφής : είναι η εντολή την οποία θα εκτελέσουμε αφού τελειώσει η συνάρτηση. Πού θα κάνει η συνάρτηση jump όταν ολοκληρώσει την εκτέλεσή της. Εμείς θα της πούμε πού θα κάνει jump.
- Σύνδεσμος προσπέλασης : δείχνει στο εγγράφημα στο οποίο ψάχνουμε μεταβλητές, παραμέτρους που δεν έχουμε αλλά χρησιμοποιούμε. Με άλλα λόγια είναι ένας δείκτης σε κάποιο στοιχείο στη στοίβα που μας βοηθάει να ψάχνουμε μεταβλητές που δε μπορούμε να δούμε , π.χ. προπάππου. Δε σημαίνει ότι θα το βρούμε αμέσως. Μπορούμε να δούμε τις μεταβλητές του πατέρα , του παππού, της main. Μεταβλητές που δε μπορούμε να δούμε είναι π.χ. του αδερφού (απο τον πίνακα συμβόλων καταλαβαίνουμε πόσα επίπεδα επάνω ανεβαίνουμε π.χ. 1, 2 κ.ο.κ.).
- Επιστροφή τιμής : η διεύθυνση της προσωρινής μεταβλητής στην οποία επιστρέφουμε ή πιο απλά αποθηκεύουμε το αποτέλεσμα της συνάρτησης.

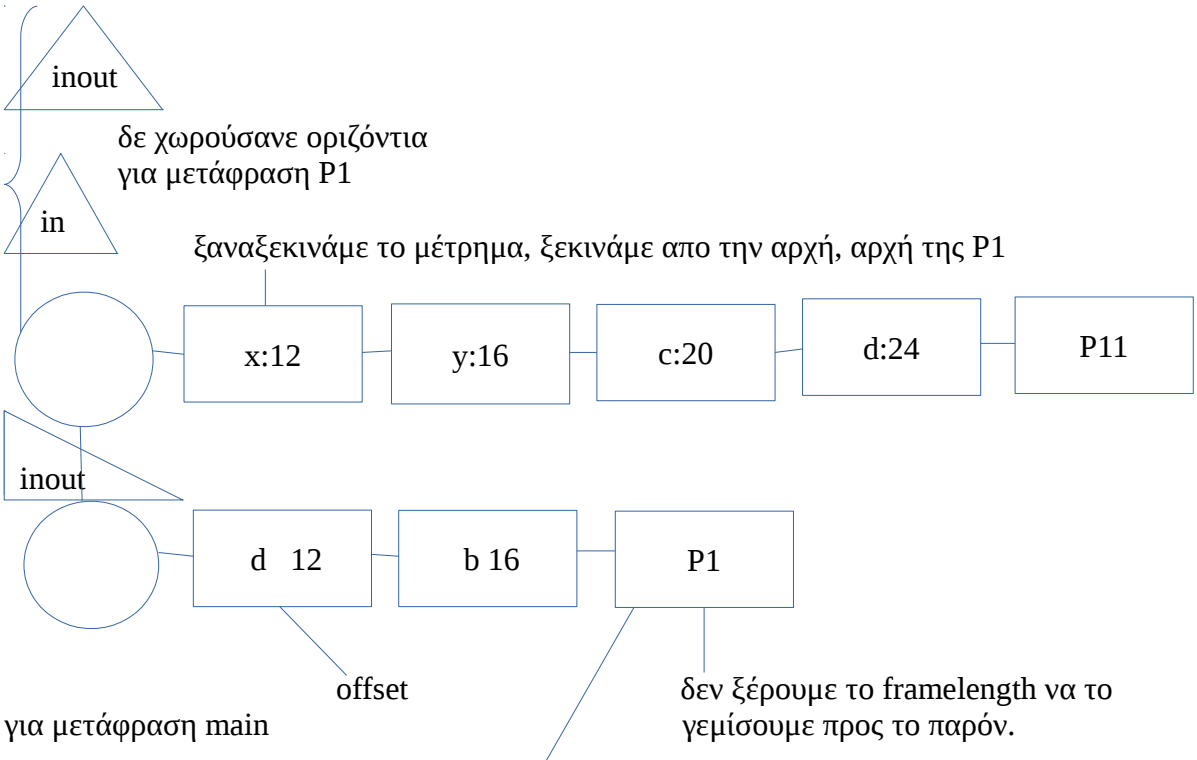
Παρακάτω, δίνουμε ένα παράδειγμα κώδικα και το πώς μετατρέπεται σε πίνακα συμβόλων. Ο πίνακας ανεβαίνει απο κάτω προς τα πάνω (στο πρότζεκτ μας, πέρα απο το παράδειγμα, κάθε συνάρτηση έχει 12 bytes και κάθε μεταβλητή έχει 4 bytes) :

```
program  exams
        var   a   b
        function  P1(in x, inout y)
                var   c   d
                function  P11(in x, inout z)
```

```
var    e
function  P21(in x)
{
        e = x
        z = x
        e = P21(in a)
        ret(e)
}
```

και πάμε τώρα να κάνουμε τον πίνακα...

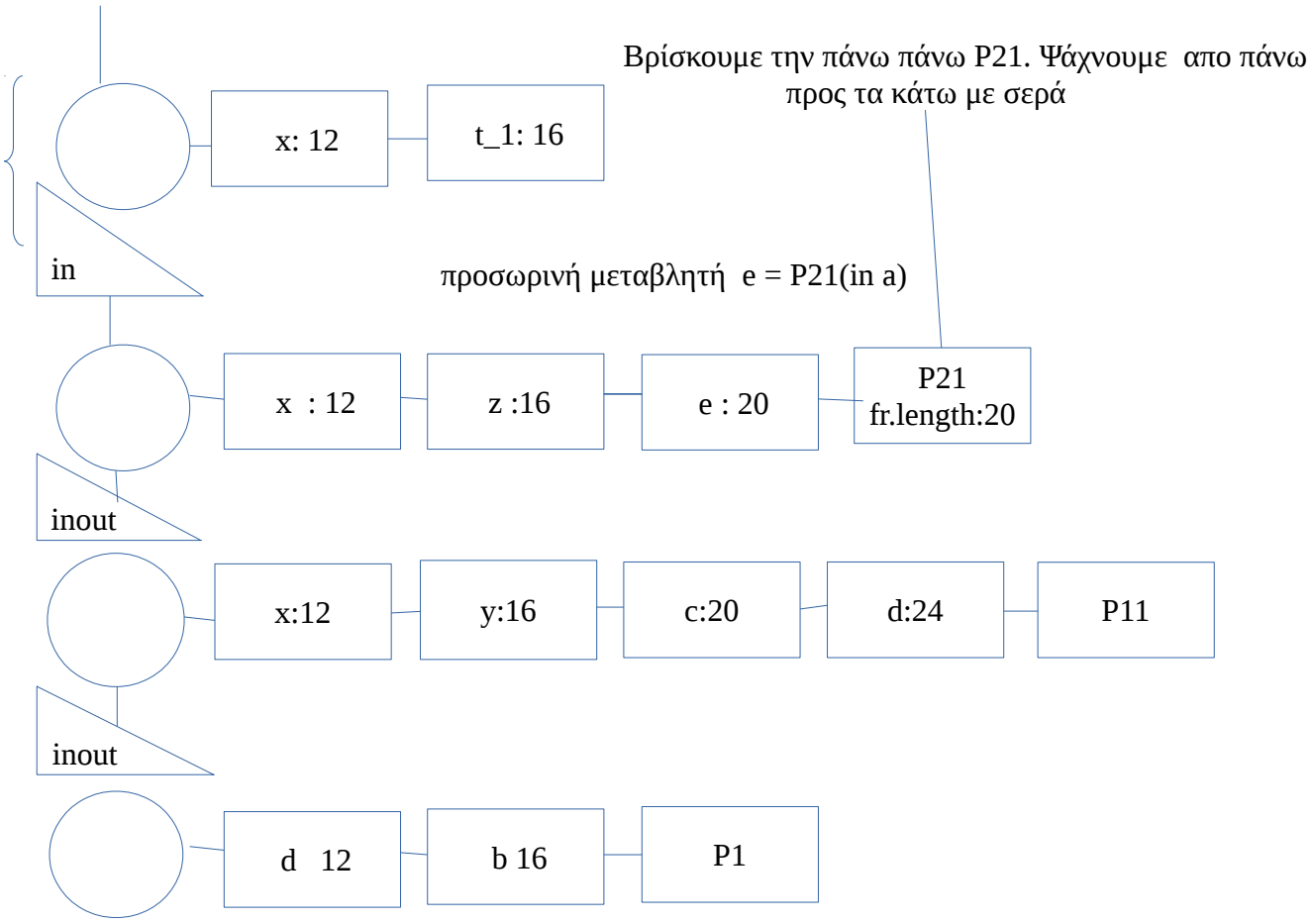
...για αρχή...




θα ζήσει όσο χρειάζεται η P1, εδώ είμαστε στη στην main άρα θα ζήσει για πάντα

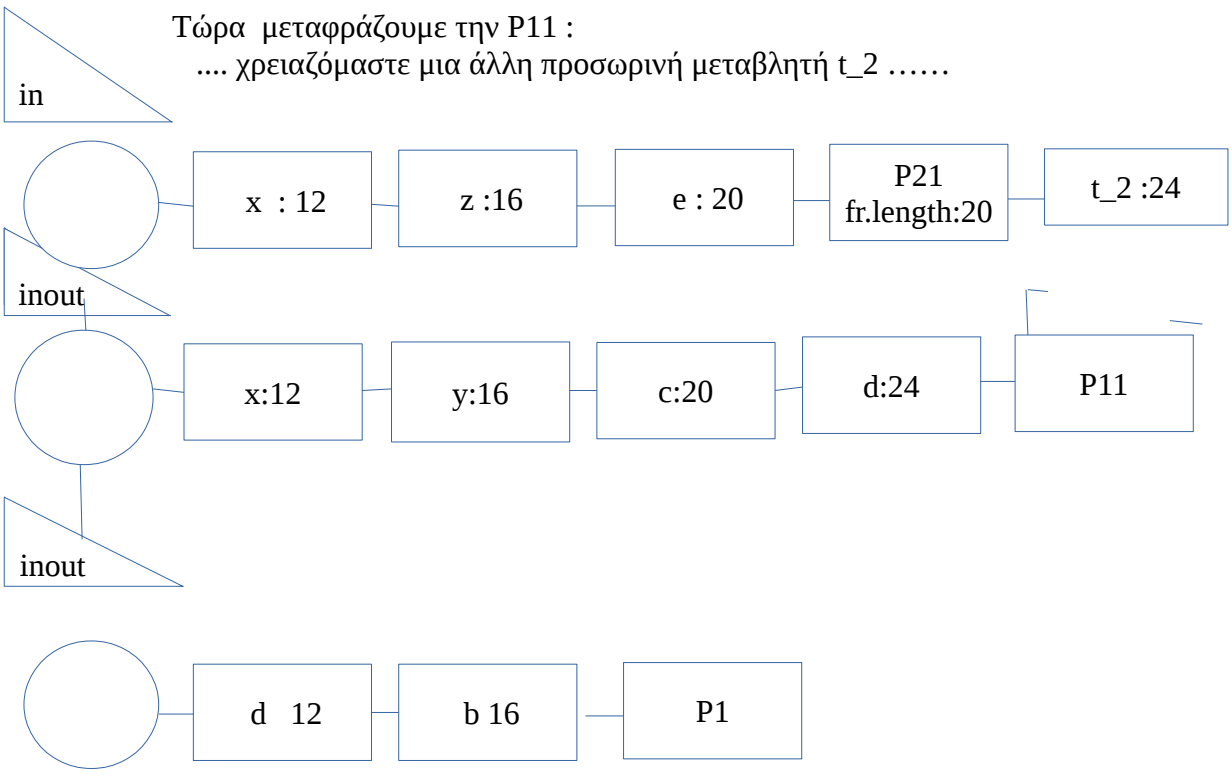
Ύστερα, ο πίνακας ανεβαίνει προς τα επάνω....

όταν φτάνουμε εδώ, σβήνονται τα σημεία

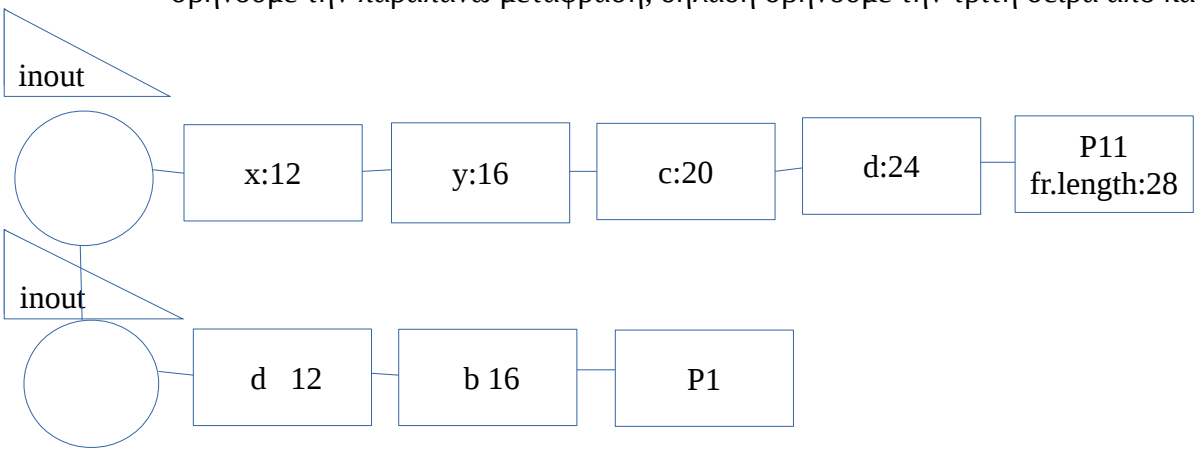



Τώρα μεταφράζουμε την P11 :

.... χρειαζόμαστε μια άλλη προσωρινή μεταβλητή t_2 ……

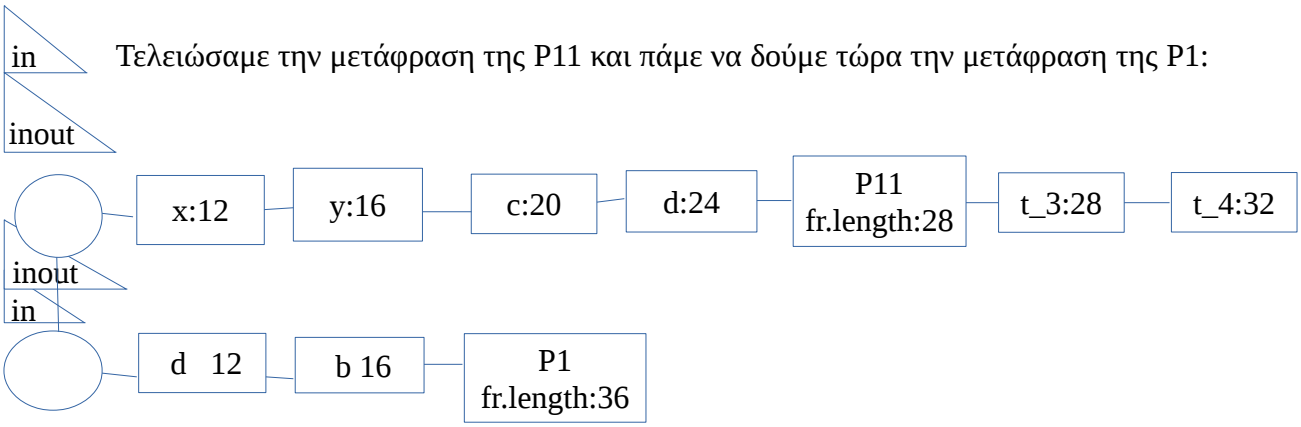


..αφού ολοκληρώσαμε την μετάφραση, ξέρουμε το framelength της P11. Το συμπληρώνουμε και σβήνουμε την παραπάνω μετάφραση, δηλαδή σβήνουμε την τρίτη σειρά απο κάτω....



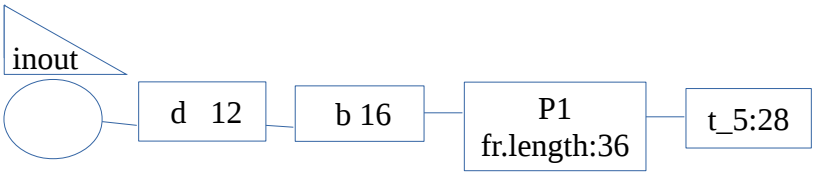

Με τον ίδιο τρόπο εργαζόμαστε και για την P21 , αν και το προσπεράσαμε ...

Ερώτηση: Πότε θα ξέρουμε εμείς πότε είναι η πρώτη εκτελέσιμη τετράδα μετά την P11;
Απάντηση: Μόλις φτάσουμε στο begin block, μόλις ξεκινάει η συνάρτηση.




Τελειώσαμε την μετάφραση της P11 και πάμε να δούμε τώρα την μετάφραση της P1:

Μετά η δεύτερη σειρά απο το τέλος φεύγει και μένει μόνο η πρώτη γραμμή :




το framelength της main είναι 24 και δεν μπαίνει στον πίνακα συμβόλων.

Αλλά, πιο πάνω αναφέραμε το begin block & end block και απαιτείται για να γίνει κατανοητή η χρήση του στην παρούσα φάση να δούμε ένα παράδειγμα:
Υποθέτουμε ότι έχουμε το παρακάτω πρόγραμμα,

```
program      sandyThanasis declare    x, y
        function p1(in s, inout t)
                declare z
                function p1(in k)
```

```
            endfunction
        endfunction
    endprogram
```

το οποίο μας δίνει...

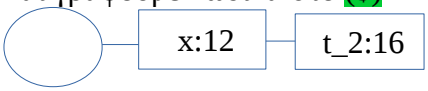
0. begin block, p1, _, _
1. end block, p1, _, _ διαγράφουμε το scope της p1
2. begin block, p, _, _
3. end block, p, _, _ διαγραφή του p

Ταυτόχρονα, πριν προχωρήσουμε στην θεματική ενότητα Ανάλυση Υλοποίησης Κώδικα Με Βάση Των Πίνακα Συμβόλων είναι σοφό να παραθέσουμε ένα παράδειγμα σύμφωνα με το οποίο κινηθήκαμε προγραμματιστικά, ώστε η ανάλυση που ακολουθεί να είναι περισσότερο εύληπτη. Ας υποθέσουμε , συνεπώς, ότι έχουμε τον εξής κώδικα :

```
program symbol

    const A=1;
    int a, b, c;

    proc P1(in x, inout y)
        int a;

        func int F11(in x)
            int a;
            b = a;
            a = x;
            c = F11(in x);
            return c;

    proc P2(inout y)
        int x;
        y = x;
        call  P1(in x, inout y);

    call P1(in a, inout b);
    call P2(in c);
```

ξεκινάμε λοιπόν τώρα τον πίνακα συμβόλων....

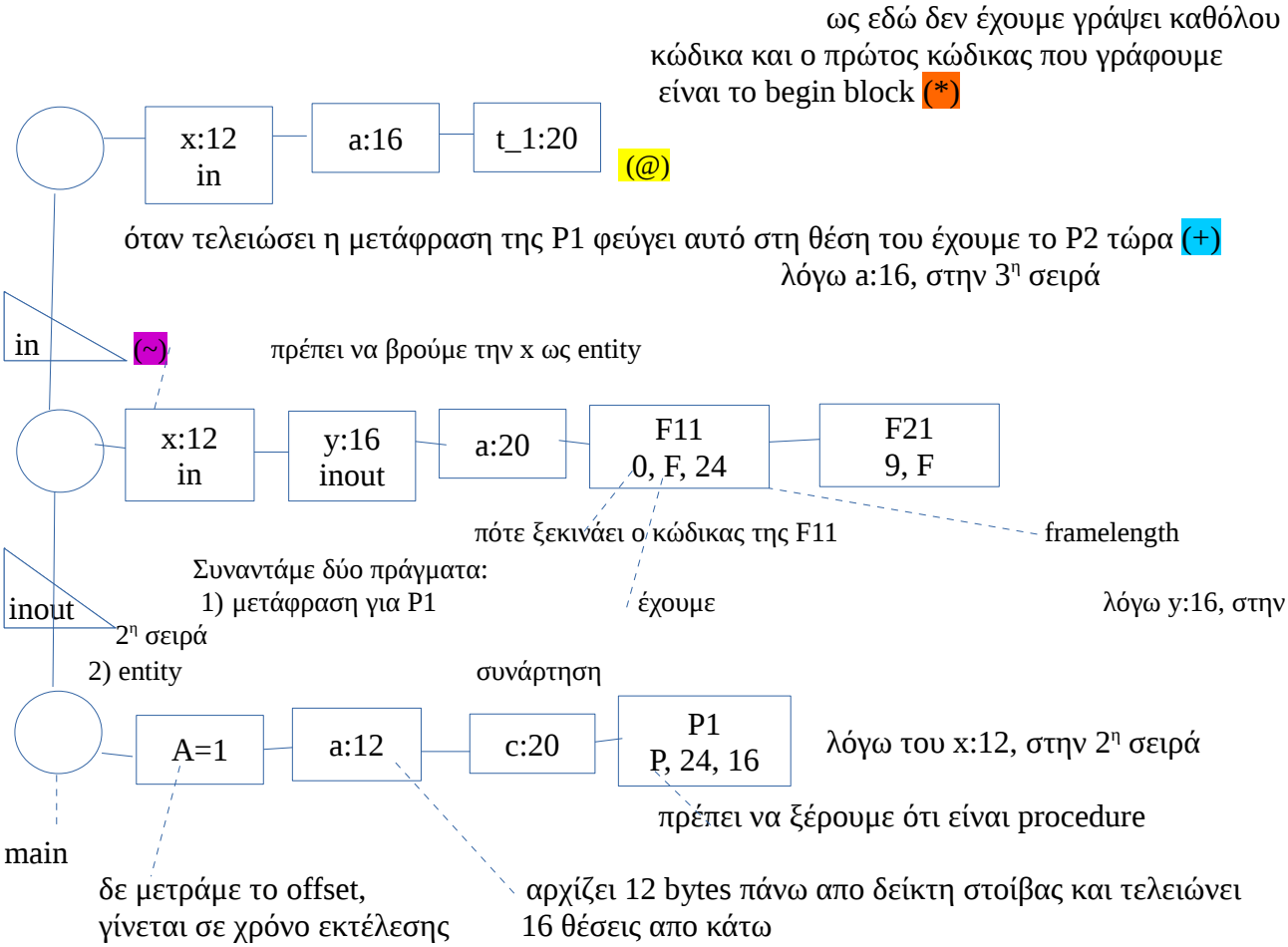
(!)το αδερφάκι του :
και γράφουμε κώδικα στο ($)




και όταν τελειώσουμε το σβήνουμε

αυτό μετά τον κώδικα που γράφουμε στο (*), φεύγει αυτή η σειρά και άρα φεύγει η μετάφραση απο

τη μνήμη , όμως η πληροφορία το πώς θα κληθεί παραμένει. Ξεκινάμε να μεταφράζουμε ένα “αδερφάκι” του και το αδερφάκι του θα πάρει την θέση του (!)




Στο τέλος φεύγει και η πρώτη σειρά συμβόλων που έχουμε...

(*) 0: begin_block F11
1: := a _ b
2: := x _ a
3: par CV x _
4: par RET t_1 _ ← με το που γράφουμε, πάμε και την σημειώνουμε στον πίνακα συμβόλων (@)
5: call F11
6: := t_1 _ c
7: retv c
8: endblock F21 ← γνωρίζουμε πια το framelength της F11 είναι 24 bytes
($) 9: begin_block F11
10: par x CV
11: par t_2 RET
12: call F11
13: :+ t_2 _ c
14: retv c
15: endblock F21

16: begin_block P1
17: endblock P1 αρχίζει ο κώδικας για το P2
18: begin_block P2
19: := A _ y βρίσκουμε call άρα ξεκινάμε τις παραμέτρους
20: par x CV
21: par y REF
22: call P1
23: endblock P2 <- μπορούμε να υπολογίσουμε το framelength της που είναι 30 και την πετάμε
24: begin_block symbol απο την μνήμη και μένει η main να τρέξουμε μόνο (~)
25: par a CV
26: par b REF
27: call P1
28: call c CV
29: call P2
30: halt
31:endblock symbol

## Ανάλυση Υλοποίησης Κώδικα Με Βάση Των Πίνακα Συμβόλων

Κατανοούμε ότι ο πίνακας συμβόλων έχει ανάλογη λογική με τον λεκτικό αναλυτή. Επομένως, σκεφτόμαστε πού θα βάλουμε κώδικα για τον πίνακα συμβόλων. Έτσι,έχοντας διατηρήσει όλα τα προηγούμενα στον κώδικά μας, κάναμε τις κατάλληλες μετατροπές και προσθήκες σε αυτό το στάδιο (γενικά η φιλοσοφία στην παρούσα υλοποίηση είναι ότι έχουμε λίστα απο λίστες που έχουν μέσα λίστες) :

- ***def newtemp()*** : είναι παλιά συνάρτηση στην οποία προσθέσαμε την μεταβλητή offset, η οποία μάς χρειάζεται στον πίνακα συμβόλων, και την μεταβλητή entity η οποία για προσωρινή μεταβλητή είναι της μορφής [τύπος, όνομα, offset]. Εδώ γίνεται κλήση της συνάρτησης eisagwghEntity(entity).
- ***def eisagwghEntity(entity)*** : είναι καινούρια συνάρτηση με την οποία εισάγουμε το entity στη λίστα με τα entities του scope στην αρχή του πίνακα συμβόλων. Η λίστα με τα entities είναι στη θέση δύο του scope για αυτό κάνουμε append το entity στον pinakasSymvolwn[0][2]. Επίσης, στη θέση μηδέν του entity έχουμε τον τύπο του. Έτσι, αν δεν είναι συνάρτηση, δηλαδή ‘function’ , αυξάνουμε το framelength της συνάρτησης. Αυτό γίνεται στην θέση τρία του scope εξού και το pinakasSymvolwn[0][3] += 4 . Η αύξηση, όπως βλέπουμε, γίνεται κατά τέσσερα σύμφωνα με την θεωρία που αναφέραμε πιο πάνω
- ***def eisagwghScope(onoma)*** : είναι καινούρια συνάρτηση με την οποία εισάγουμε scope. Το onoma που έχουμε ως παράμετρο στην συνάρτηση είναι το όνομα του scope που εισάγουμε. Για κάθε scope κρατάμε σε μία λίστα το βάθος φωλιάσματος, το όνομα, την λίστα των entities και το framelength, κι έπειτα προσθέτουμε το νέα scope στον πίνακα συμβόλων που τον έχουμε ονομάσει pinakasSymvolwn. Ακόμα, σε αυτή την συνάρτηση γίνεται η

αύξηση του βάθους φωλιάσματος, δηλαδή το βάθος φωλιάσματος αυξάνεται κάθε φορά κατά ένα.

- ***def diagrafhScope()*** : είναι καινούρια συνάρτηση με την οποία διαγράφουμε ένα scope. Μειώνουμε, λοιπόν, κάθε φορά το βάθος φωλιάσματος κατά ένα , εκτυπώνουμε τον πίνακα συμβόλων και διαγράφουμε την πρώτη του γραμμή.
- ***def formalparitem()*** : είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που κάναμε ορισμένες προσθήκες. Εδώ, κάθε φορά αφού πραγματοποιείται ο έλεγχος για να δούμε σε ποια κατάσταση βρισκόμαστε, δηλαδή αν είμαστε σε in ,inout ή inandout , παίρνουμε την μεταβλητή entity. Η entity για παράμετρο είναι της μορφής [τύπος, όνομα, offset] για αυτό αν έχουμε if katastash == "intk": το entity γίνεται entity = ["CV", leksh, offset] , αν έχουμε elif katastash == "inouttk": το entity γίνεται entity = ["REF", leksh, offset] και αν έχουμε elif katastash == "inandouttk": το entity γίνεται entity = ["CP", leksh, offset] . Όπως μπορείτε να διακρίνεται η λεκτική μονάδα δηλαδή η leksh μάς δίνει το όνομα του entity. Βέβαια, να σημειώσουμε όπως φαίνεται σε κάθε περίπτωση ορίζουμε και το offset στην τρίτη θέση του πίνακα συμβόλων , δηλαδή offset = pinakasSymvolwn[0][3] για αυτό μετά μπορούμε να χρησιμοποιούμε το offset στο entity. Έπειτα, σε όποια περίπτωση και αν βρισκόμαστε καλούμε μετά την συνάρτηση eisagwghEntity(entity) για να γίνει η προσθήκη της οντότητας αυτής.
- ***def varlist()*** : είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που τώρα κάναμε ορισμένες προσθήκες. Εδώ, επειδή έχουμε το id δηλαδή το αναγνωριστικό, κινούμαστε παρόμοια με ό,τι κάναμε στην συνάρτηση *def formalparitem()*. Με πιο απλά λόγια ορίζουμε πάλι το offset να είναι στην τρίτη θέση του πίνακα συμβόλων , ορίζουμε το entity να είναι της μορφής [τύπος, όνομα, offset] μόνο που εδώ έχουμε μεταβλητή άρα γίνεται entity = ['metavlith', leksh, offset] και καλούμε την συνάρτηση eisagwghEntity(entity) και αυτό γίνεται με βάση την γραμματική της Startlet , δηλαδή λαμβάνουμε υπόψιν και το kleene star και επομένως και μέσα στο while katastash == ",tk" , αφού γίνει append η leksh εφόσον έχει γίνει ο έλεγχος if katastash == "anagnoristikotk" ορίζουμε αντίστοιχα το offset, το entity και καλούμε την συνάρτηση eisagwghEntity(entity) γιατί κάθε φορά θα δημιουργείται και ένα καινούριο entity.
- ***def eyreshEntity(onoma)*** : είναι καινούρια συνάρτηση με την οποία κάνουμε αναζήτηση entity. Το onoma που έχουμε ως παράμετρο είναι το entity , το οποίο αναζητάμε. Διατρέχουμε, λοιπόν, όλο των πίνακα συμβόλων και αν δεν βρούμε το συγκεκριμένο entity , εκτυπώνουμε μήνυμα λάθους.
- ***def subprogram()*** : είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που τώρα κάναμε ορισμένες προσθήκες. Εισάγουμε το ‘function’ και σαν scope και σαν entity επειδή δεν ξέρουμε ακόμα το startquad και το framelength.Επομένως, εδώ αφού καλέσουμε λεκτικό αναλυτή ,καλούμε τις συναρτήσεις eisagwghEntity(entity) , eisagwghScope(onoma) και diagrafhScope(). Βέβαια, διαγραφή scope μπορεί να πραγματοποιηθεί μόνο αφού έχουμε καλέσει την συνάρτηση funcbody(onoma) ώστε να μπορεί να γίνει η κατάλληλη διαγραφή.

- ***def block(onoma)*** : είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που τώρα κάναμε ορισμένες προσθήκες. Αφού έχουμε καλέσει τις συναρτήσεις declarations() και subprograms() , πηγαίνουμε στο entity που αφορά την συνάρτηση που πάμε να ξεκινήσουμε και βάζουμε σαν startquad (θέση δύο στο entity) την ετικέτα του begin block. Το entity είναι πάντα τελευταίο στην λίστα με τα entities του κάτω scope. Σε διαφορετική όμως περίπτωση ( όταν βρισκόμαστε στο else δηλαδή) για τις συναρτήσεις βάζουμε στην τρίτη θέση το framelength στο entity με την ίδια φιλοσοφία όπως με το startquad πριν. Το framelength βρίσκεται στην τρίτη θέση του scope για την συνάρτηση. Επίσης, σε αυτή την συνάρτηση έχουμε προσθέσει έναν returnCounter , αν δεν υπάρχει δηλαδή return , εμφανίζεται μήνυμα λάθους. Από την άλλη πλευρά, αν στο κυρίως πρόγραμμα υπάρχει return τότε εμφανίζουμε και εκεί μήνυμα λάθους.
- Ως global έχουμε προσθέσει και αρχικοποιήσει δύο μεταβλητές, την loopCounter και την returnCounter, ώστε να ελέγχουμε και την ορθή σημασιολογική ανάλυση του προγράμματός μας. Πιο συγκεκριμένα, την μεταβλητή loopCounter, την χρειαζόμαστε για να ελέγχουμε την σωστή λειτουργία με τα exits και την returnCounter την χρειαζόμαστε για να ελέγχουμε την σωστή λειτουργία των returns. Αυτές τις μεταβλητές τις αξιοποιούμε στις συναρτήσεις def statement(),
- ***def statement()*** :είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που τώρα κάναμε ορισμένες προσθήκες. Ελέγχουμε το γεγονός ότι για να υπάρχει exit , πρέπει να υπάρχει ανοιχτό loop αλλιώς εμφανίζουμε σημασιολογικό μήνυμα λάθους. Ακόμα στην συνάρτηση statement() κρατάμε και το πλήθος των return.
- ***def loop_stat()*** :είναι η συνάρτηση που είχαμε απο προηγούμενα στάδια, μόνο που τώρα κάναμε ορισμένες προσθήκες. Εδώ αυξάνουμε το πλήθος των loop όταν ανοίγει ένα loop και μειώνουμε το πλήθος των loop κατά ένα ώστε να πραγματοποιείται η σωστή λειτουργία με τα exits της συνάρτηση def statement() που μόλις προαναφέραμε στην προηγούμενη κουκκίδα.

Να πούμε βέβαια ότι των πίνακα συμβόλων pinakasSymvolwn και το βάθος φωλιάσματος vathosFwliasmatos τα έχουμε κάνει global και τα αρχικοποιούμε στην αρχή.

Για να δώσουμε μία αίσθηση για το τί συμβαίνει στην υλοποίηση του πίνακα συμβόλων μας παραθέτουμε τα παρακάτω:

- scope → [ βάθος, όνομα, λίστα entities, framelength]
  δηλαδή έχουμε την μορφή [[vathos, n,[, , , ],y]
  [ … ] κλπ.

Έχουμε για παράδειγμα, [[par→ 3],
[function L → 4]
[var → 3]]

δηλαδή έχουμε pinakasSymvolwn[3][2][0],
για παράδειγμα για τις παραμέτρους έχουμε [CV,x,12].

**Εκτέλεση**

Τρέχουμε python pinakas.py test.stl :

```
cs091716@opti7020ws09:~/Desktop/ergasiaMetafrastes/t

0-test-64
        ['metavlith', 'x', 12]
        ['metavlith', 'y', 16]
        ['metavlith', 'a', 20]
        ['metavlith', 'b', 24]
        ['metavlith', 'c', 28]
        ['proswrinh', 'T_1', 32]
        ['proswrinh', 'T_2', 36]
        ['proswrinh', 'T_3', 40]
        ['proswrinh', 'T_4', 44]
        ['proswrinh', 'T_5', 48]
        ['proswrinh', 'T_6', 52]
        ['proswrinh', 'T_7', 56]
        ['proswrinh', 'T_8', 60]
['0', 'begin block', 'test', '_', '_']
['10', 'inp', '_', '_', 'a']
['20', 'inp', '_', '_', 'b']
['30', '>', 'a', 'b', '50']
['40', 'jump', '_', '_', '250']
['50', '>', 'a', 'b', '70']
['60', 'jump', '_', '_', '240']
['70', 'out', '_', '_', 'a']
['80', '-', 'a', '1', 'T_1']
['90', ':=', 'T_1', '_', 'a']
['100', '+', 'a', '3', 'T_2']
['110', '<', 'T_2', 'b', '150']
['120', 'jump', '_', '_', '130']
['130', '>', 'b', '10', '150']
['140', 'jump', '_', '_', '210']
['150', '/', 'a', '3', 'T_3']
['160', ':=', 'T_3', '_', 'a']
['170', 'out', '_', '_', 'a']
['180', '>', 'a', 'b', '150']
['190', 'jump', '_', '_', '200']
['200', 'jump', '_', '_', '230']
['210', ':=', 'b', '_', 'a']
['220', 'out', '_', '_', 'a']
['230', 'jump', '_', '_', '50']
['240', 'jump', '_', '_', '380']
['250', '<', 'a', 'b', '270']
['270', '+', 'a', 'b', 'T_4']
['280', 'out', '_', '_', 'T_4']
['290', 'jump', '_', '_', '380']
['300', '=', 'a', 'b', '320']
['310', 'jump', '_', '_', '350']
['320', '*', 'a', 'b', 'T_5']
['330', 'out', '_', '_', 'T_5']
['340', 'jump', '_', '_', '380']
['350', '-', 'a', 'b', 'T_6']
['360', 'out', '_', '_', 'T_6']
['370', 'jump', '_', '_', '250']
['380', ':=', '30', '_', 'a']
['390', ':=', '20', '_', 'b']
['400', '-', 'a', '1', 'T_7']
['410', ':=', 'T_7', '_', 'a']
['420', '=', 'a', 'b', '440']
['430', 'jump', '_', '_', '460']
['440', 'jump', '_', '_', '480']
['450', 'jump', '_', '_', '460']
['460', 'out', '_', '_', 'a']
['470', 'jump', '_', '_', '400']
['480', 'inp', '_', '_', 'a']
['490', 'inp', '_', '_', 'b']
['500', ':=', '0', '_', 'T_8']
['510', '>', 'a', 'b', '530']
['520', 'jump', '_', '_', '560']
['530', ':=', '1', '_', 'T_8']
['540', 'out', '_', '_', '4']
['550', ':=', 'b', '_', 'a']
['560', '<', 'a', 'b', '580']
['570', 'jump', '_', '_', '610']
['580', ':=', '1', '_', 'T_8']
['590', 'out', '_', '_', '5']
['600', ':=', 'b', '_', 'a']
['610', '<>', 'a', 'b', '630']
['620', 'jump', '_', '_', '660']
['630', ':=', '1', '_', 'T_8']
['640', 'out', '_', '_', '6']
['650', ':=', 'b', '_', 'a']
['660', '=', 'T_8', '1', '500']
['670', ':=', '1', '_', 'c']
['680', 'halt', '_', '_', '_']
['690', 'end block', 'test', '_', '_']
cs091716@opti7020ws09:~/Desktop/ergasia
```

Τρέχοντας python pinakas.py test2.stl :

```
cs091716@opti7020ws09:~/Desktop/ergasiaMetafrastes/turnPinaka$ python pinakas.py test2.stl

2-f2-44
        ['CV', 'x', 12]
        ['REF', 'y', 16]
        ['CP', 'z', 20]
        ['metavlith', 'a', 24]
        ['proswrinh', 'T_1', 28]
        ['proswrinh', 'T_2', 32]
        ['proswrinh', 'T_3', 36]
        ['proswrinh', 'T_4', 40]
1-f1-16
        ['metavlith', 'a', 12]
        ['function', 'f2', '0', 44]
0-x-24
        ['metavlith', 'a', 12]
        ['metavlith', 'b', 16]
        ['metavlith', 'd', 20]
        ['function', 'f1', -1, -1]

1-f1-16
        ['metavlith', 'a', 12]
        ['function', 'f2', '0', 44]
0-x-24
        ['proswrinh', 'T_6', 28]
        ['proswrinh', 'T_7', 32]
['0', 'begin block', 'f2', '_', '_']
['10', 'retv', '_', '_', 'x']
['20', '+', 'b', '2', 'T_1']
['30', '+', 'T_1', 'b', 'T_2']
['40', 'par', 'x', 'CV', '_']
['50', 'par', 'x', 'REF', '_']
['60', 'par', 'z', 'CV', '_']
['70', 'par', 'T_3', 'RET', '_']
['80', 'call', 'f2', '_', '_']
['90', '+', 'T_2', 'T_3', 'T_4']
['100', ':=', 'T_4', '_', 'a']
['110', 'inp', '_', '_', 'z']
['120', 'end block', 'f2', '_', '_']
['130', 'begin block', 'f1', '_', '_']
['140', 'retv', '_', '_', 'a']
['150', 'end block', 'f1', '_', '_']
['160', 'begin block', 'f3', '_', '_']
['170', 'retv', '_', '_', 'a']
['180', 'end block', 'f3', '_', '_']
['190', 'begin block', 'x', '_', '_']
['200', 'par', 'T_5', 'RET', '_']
['210', 'call', 'f1', '_', '_']
['220', ':=', 'T_5', '_', 'a']
['230', ':=', '30', '_', 'a']
['240', ':=', '10', '_', 'b']
['250', '>', 'b', '0', '270']
['260', 'jump', '_', '_', '420']
['270', 'out', '_', '_', 'b']
['280', '>', 'a', 'b', '300']
['290', 'jump', '_', '_', '320']
['300', 'jump', '_', '_', '390']
['310', 'jump', '_', '_', '320']
['320', '-', 'b', '1', 'T_6']
['330', ':=', 'T_6', '_', 'b']
['340', '>', 'a', '10', '360']
['350', 'jump', '_', '_', '380']
['360', 'jump', '_', '_', '390']
['370', 'jump', '_', '_', '380']
['380', 'jump', '_', '_', '270']
['390', '-', 'b', '1', 'T_7']
['400', ':=', 'T_7', '_', 'b']
['410', 'jump', '_', '_', '250']
['420', 'jump', '_', '_', '200']
['430', 'halt', '_', '_', '_']
['440', 'end block', 'x', '_', '_']
```

## Εκτύπωση Tests Σε Γλώσσα Προγραμματισμού C

### test.c

```c
#include <stdio.h>
int main(){
int x;
int y;
int a;
int b;
int c;
int T_1;
int T_2;
Label110: if(T_2<b) goto Label150;
//120,jump,_,_,130
Label120: goto Label130;
//130,>,b,10,150
Label130: if(b>10) goto Label150;
//140,jump,_,_,210
Label140: goto Label210;
//150,/,a,3,T_3
Label150: T_3=a/3;
//160,:=,T_3,_,a
Label160: a=T_3;
//170,out,_,_,a
Label170: printf("%d\n",a);
//180,>,a,b,150
Label180: if(a>b) goto Label150;
//190,jump,_,_,200
Label190: goto Label200;
//200,jump,_,_,230
Label200: goto Label230;
//210,:=,b,_,a
Label210: a=b;
//220,out,_,_,a
Label220: printf("%d\n",a);
//230,jump,_,_,50
Label230: goto Label50;
//240,jump,_,_,380
Label240: goto Label380;
//250,<,a,b,270
Label250: if(a<b) goto Label270;
//260,jump,_,_,300
Label260: goto Label300;
//270,+,a,b,T_4
Label270: T_4=a+b;
//280,out,_,_,T_4
Label280: printf("%d\n",T_4);
//290,jump,_,_,380
Label290: goto Label380;
//300,=,a,b,320
Label300: if(a==b) goto Label320;
//310,jump,_,_,350
Label310: goto Label350;
//320,*,a,b,T_5
Label320: T_5=a*b;
//330,out,_,_,T_5
Label330: printf("%d\n",T_5);
//340,jump,_,_,380
Label340: goto Label380;
//490,inp,_,_,b
Label490: scanf("%d",&b);
//500,:=,0,_,T_8
Label500: T_8=0;
//510,>,a,b,530
Label510: if(a>b) goto Label530;
//520,jump,_,_,560
Label520: goto Label560;
//530,:=,1,_,T_8
Label530: T_8=1;
//540,out,_,_,4
Label540: printf("%d\n",4);
//550,:=,b,_,a
Label550: a=b;
//560,<,a,b,580
Label560: if(a<b) goto Label580;
//570,jump,_,_,610
Label570: goto Label610;
//580,:=,1,_,T_8
Label580: T_8=1;
//590,out,_,_,5
Label590: printf("%d\n",5);
//600,:=,b,_,a
Label600: a=b;
//610,<>,a,b,630
Label610: if(a!=b) goto Label630;
//670,:=,1,_,c
Label670: c=1;
//680,halt,_,_,_
Label680: ;
//690,end block,test,_,_ 5);
Label690: ;
return 0;
}
 Label500;
```

## test2.c

```
1   #include <stdio.h>
2   int main(){
3     int a;
39    //120,end block,f2,_,_
40    Label120: ;
41    //130,begin block,f1,_,_
42    Label130: ;
43    //140,retv,_,_,a
44    Label140: ;
45    //150,end block,f1,_,_
46    Label150: ;
47    //160,begin block,f3,_,_
48    Label160: ;
49    //170,retv,_,_,a
50    Label170: ;
51    //180,end block,f3,_,_
52    Label180: ;
53    //190,begin block,x,_,_
54    Label190: ;
55    //200,par,T_5,RET,_
56    Label200: ;
57    //210,call,f1,_,_
58    Label210: ;
59    //220,:=,T_5,_,a
60    Label220: a=T_5;
61    //230,:=,30,_,a
62    Label230: a=30;
63    //240,:=,10,_,b
64    Label240: b=10;
65    //250,>,b,0,270
66    Label250: if(b>0) goto Label270
67    //260,jump,_,_,420
68    Label260: goto Label420;
69    //270,out,_,_,b
70    Label270: printf("%d\n",b);
71    //280,>,a,b,300
72    Label280: if(a>b) goto Label300
73    //290,jump,_,_,320
74    Label290: goto Label320;
75    //300,jump,_,_,390
76    Label300: goto Label390;
77    //310,jump,_,_,320
78    Label310: goto Label320;
79    //320,-,b,1,T_6
80    Label320: T_6=b-1;
81    //330,:=,T_6,_,b
82    Label330: b=T_6;
83    //340,>,a,10,360
84    Label340: if(a>10) goto Label360;
85    //350,jump,_,_,380
86    Label350: goto Label380;
87    //360,jump,_,_,390
88    Label360: goto Label390;
89    //370,jump,_,_,380
90    Label370: goto Label380;
91    //380,jump,_,_,270
92    Label380: goto Label270;
93    //390,-,b,1,T_7
94    Label390: T_7=b-1;
95    //400,:=,T_7,_,b
96    Label400: b=T_7;
97    //410,jump,_,_,250
98    Label410: goto Label250;
99    //420,jump,_,_,200
100   Label420: goto Label200;
101   //430,halt,_,_,_
102   Label430: ;
103   //440,end block,x,_,_
104   Label440: ;
105   return 0;
106   }
```

**Ανάλυση Αποτελεσμάτων Τερματικού Και Αρχείων**

- Αντιστοιχία κώδικα που εκτυπώνεται στο τερματικό με την εντολή python pinakas.py test.stl και του κώδικα που περιέχεται στο αρχείο test.stl :

| | |
|---|---|
| 0-test-64 | //framelength program test |
| ['metavlith', 'x', 12] | //declare x; |
| ['metavlith', 'y', 16] | //declare y; |
| ['metavlith', 'a', 20] | //declare a; |
| ['metavlith', 'b', 24] | //declare b; |
| ['metavlith', 'c', 28] | //declare c; |
| ['proswrinh', 'T_1', 32] | //προσωρινές μεταβλητές για τις πράξεις |
| ['proswrinh', 'T_2', 36] | |
| ['proswrinh', 'T_3', 40] | |
| ['proswrinh', 'T_4', 44] | |
| ['proswrinh', 'T_5', 48] | |
| ['proswrinh', 'T_6', 52] | |
| ['proswrinh', 'T_7', 56] | |
| ['proswrinh', 'T_8', 60] | |
| ['0', 'begin block', 'test', '_', '_'] | //program test |
| ['10', 'inp', '_', '_', 'a'] | //input a; |
| ['20', 'inp', '_', '_', 'b'] | //input b; |
| ['30', '>', 'a', 'b', '50'] | //if(a>b) then |

```
['40', 'jump', '_', '_', '250']                    //forcase
['50', '>', 'a', 'b', '70']                              //while (a>b)
['60', 'jump', '_', '_', '240']                    //else , γραμμή 24
['70', 'out', '_', '_', 'a']                       // print a;
['80', '-', 'a', '1', 'T_1']                       // a:=a-1;
['90', ':=', 'T_1', '_', 'a']
['100', '+', 'a', '3', 'T_2']                      //a + 3
['110', '<', 'T_2', 'b', '150']                    // if( a + 3 <b ) then
['120', 'jump', '_', '_', '130']                   //else , γραμμή 19
['130', '>', 'b', '10', '150']                     //if(b >10) then
['140', 'jump', '_', '_', '210']                   //else , γραμμή 19
['150', '/', 'a', '3', 'T_3']                      //a:= a/3;
['160', ':=', 'T_3', '_', 'a']
['170', 'out', '_', '_', 'a']                      //print a
['180', '>', 'a', 'b', '150']                      //enddowhile(a>b)
['190', 'jump', '_', '_', '200']                   // endif
['200', 'jump', '_', '_', '230']                   //endwhile
['210', ':=', 'b', '_', 'a']                       //a:=b;
['220', 'out', '_', '_', 'a']                      //print a
['230', 'jump', '_', '_', '50']                    //endif, γραμμή 22
['240', 'jump', '_', '_', '380']                   //endif, γραμμή 30
['250', '<', 'a', 'b', '270']                      // when (a<b):
['260', 'jump', '_', '_', '300']          //για μη αληθή συνθήκη πάμε στο when (a=b) :
['270', '+', 'a', 'b', 'T_4']                      //a+b
['280', 'out', '_', '_', 'T_4']                    //print a+b;
['290', 'jump', '_', '_', '380']                   //πάμε μετά το  endif; ,γραμμή 30
['300', '=', 'a', 'b', '320']                      // when (a=b) :
['310', 'jump', '_', '_', '350']                   //για μη αληθή συνθήκη πάμε  στο   default :
['320', '*', 'a', 'b', 'T_5']                      // a*b
['330', 'out', '_', '_', 'T_5']                    //print a*b;
['340', 'jump', '_', '_', '380']                 //για μη αληθή συνθήκη πάμε μετά το  endif;
['350', '-', 'a', 'b', 'T_6']                      // a-b
['360', 'out', '_', '_', 'T_6']                    //default :print a-b;
['370', 'jump', '_', '_', '250']                   //enddefault ,  endforcase
['380', ':=', '30', '_', 'a']                      //a:=30;
['390', ':=', '20', '_', 'b']                      //b:=20;
['400', '-', 'a', '1', 'T_7']                      //a:=a-1;
['410', ':=', 'T_7', '_', 'a']
['420', '=', 'a', 'b', '440']                      //if(a=b) then
['430', 'jump', '_', '_', '460']                   //για μη αληθή if(a=b)
['440', 'jump', '_', '_', '480']                   //για μετά το endloop;
['450', 'jump', '_', '_', '460']                   // για μετά το print a;
['460', 'out', '_', '_', 'a']                      //print a;
['470', 'jump', '_', '_', '400']                   //για επιστροφή στο loop
['480', 'inp', '_', '_', 'a']                      //input a;
```

```
['490', 'inp', '_', '_', 'b']                    // input b;
['500', ':=', '0', '_', 'T_8']          //χρησιμοποιούμε flag 0 ή 1  για έλεγχο όπως
                                        //έχουμε αναλύσει σε προηγούμενη  θεματική ενότητα
['510', '>', 'a', 'b', '530']                    //incase  when(a>b) :
['520', 'jump', '_', '_', '560']                 //πάμε στο   when(a<b) : (απο εκφώνηση)
['530', ':=', '1', '_', 'T_8']
['540', 'out', '_', '_', '4']                    //print 4;
['550', ':=', 'b', '_', 'a']                     //a:=b;
['560', '<', 'a', 'b', '580']                    //when(a<b) :
['570', 'jump', '_', '_', '610']                 //πάμε στο   when(a<>b) : (απο εκφώνηση)
['580', ':=', '1', '_', 'T_8']
['590', 'out', '_', '_', '5']                    //print 5;
['600', ':=', 'b', '_', 'a']                     //a:=b;
['610', '<>', 'a', 'b', '630']                   //when(a<>b) :
['620', 'jump', '_', '_', '660']              //για να δούμε αν θα πάμε στην αρχή της incase
['630', ':=', '1', '_', 'T_8']
['640', 'out', '_', '_', '6']                    //print 6;
['650', ':=', 'b', '_', 'a']                     //a:=b;
['660', '=', 'T_8', '1', '500']
['670', ':=', '1', '_', 'c']                     //c:=1
['680', 'halt', '_', '_', '_']
['690', 'end block', 'test', '_', '_']           //endprogram
```

- Αντιστοιχία κώδικα που εκτυπώνεται στο τερματικό με την εντολή python pinakas.py test2.stl και του κώδικα που περιέχεται στο αρχείο  test2.stl :

```
2-f2-44                                  // function f2  με  framelngth 44
    ['CV', 'x', 12]                      //in x
    ['REF', 'y', 16]                     // inout y
    ['CP', 'z', 20]                      //inandout z
    ['metavlith', 'a', 24]               //declare a; ,γραμμή 7
    ['proswrinh', 'T_1', 28]             //προσωρινές μεταβλητές για τις πράξεις
    ['proswrinh', 'T_2', 32]
    ['proswrinh', 'T_3', 36]
    ['proswrinh', 'T_4', 40]
1-f1-16                                  //function f1() με  framelngth 16
    ['metavlith', 'a', 12]               //declare a; ,γραμμή 5
    ['function', 'f2', '0', 44]
0-x-24                                   //program x με  framelngth 24
    ['metavlith', 'a', 12]               //declare a;
    ['metavlith', 'b', 16]               //declare b;
```

```
        ['metavlith', 'd', 20]              //declare d;
        ['function', 'f1', -1, -1]          //έχουμε την  function f1() μέσα στην program x

1-f1-16                                     //function f1() με  framelngth 16
        ['metavlith', 'a', 12]              //declare a; ,γραμμή 5
        ['function', 'f2', '0', 44]         // έχουμε την  function f2() μέσα στην function f1()
0-x-24
        ['metavlith', 'a', 12]
        ['metavlith', 'b', 16]
        ['metavlith', 'd', 20]
        ['function', 'f1', '130', 16]

1-f3-12                                     //function f3()
0-x-24                                      //program x με  framelngth 24
        ['metavlith', 'a', 12]              //declare a;
        ['metavlith', 'b', 16]              //declare b;
        ['metavlith', 'd', 20]              //declare d;
        ['function', 'f1', '130', 16]       // έχουμε την  function f2() μέσα στην function f1()
        ['function', 'f3', '160', 12]       //έχουμε την  function f3() μέσα στην program x

0-x-36                                      //ίδια λογική
        ['metavlith', 'a', 12]
        ['metavlith', 'b', 16]
        ['metavlith', 'd', 20]
        ['function', 'f1', '130', 16]
        ['function', 'f3', '160', 12]
        ['proswrinh', 'T_5', 24]
        ['proswrinh', 'T_6', 28]
        ['proswrinh', 'T_7', 32]
['0', 'begin block', 'f2', '_', '_']       //function f2
['10', 'retv', '_', '_', 'x']              //return x;
['20', '+', 'b', '2', 'T_1']               //b+2
['30', '+', 'T_1', 'b', 'T_2']             //b+2+b
['40', 'par', 'x', 'CV', '_']              //in x
['50', 'par', 'x', 'REF', '_']             //inout x
['60', 'par', 'z', 'CV', '_']              //in z
['70', 'par', 'T_3', 'RET', '_']           //   απο το return a; που επιστρέφει η f2()
['80', 'call', 'f2', '_', '_']             //f2(in x,  inout x, in z)
['90', '+', 'T_2', 'T_3', 'T_4']           // b+2+b + f2(in x,  inout x, in z);
['100', ':=', 'T_4', '_', 'a']             // a:=b+2+b + f2(in x,  inout x, in z);
['110', 'inp', '_', '_', 'z']              //input z;
['120', 'end block', 'f2', '_', '_']       // endfunction, γραμμή 12
['130', 'begin block', 'f1', '_', '_']     //function f1()
['140', 'retv', '_', '_', 'a']             // return a;
['150', 'end block', 'f1', '_', '_']       // endfunction , γραμμή 14
```

```
['160', 'begin block', 'f3', '_', '_']      //function f3()
['170', 'retv', '_', '_', 'a']              //return a;, γραμμή 17
['180', 'end block', 'f3', '_', '_']        //endfunction
['190', 'begin block', 'x', '_', '_']       //program x
['200', 'par', 'T_5', 'RET', '_']
['210', 'call', 'f1', '_', '_']             //f1()
['220', ':=', 'T_5', '_', 'a']              //a:=f1();
['230', ':=', '30', '_', 'a']               // a:=30;
['240', ':=', '10', '_', 'b']               //b:=10;
['250', '>', 'b', '0', '270']               //while(b>0)
['260', 'jump', '_', '_', '420']            //για  while πάμε
['270', 'out', '_', '_', 'b']               //print b;
['280', '>', 'a', 'b', '300']               //if( a> b) then
['290', 'jump', '_', '_', '320']            //αν δεν ισχύει η if( a> b) then  πάμε στο  endif
['300', 'jump', '_', '_', '390']            //για να βγούμε απο το loop, γραμμή 39
['310', 'jump', '_', '_', '320']            //για να βγούμε απο το loop,γραμμή 36
['320', '-', 'b', '1', 'T_6']               //b-1
['330', ':=', 'T_6', '_', 'b']              // b:=b-1;
['340', '>', 'a', '10', '360']              // if(a>10) then
['350', 'jump', '_', '_', '380']            // αν δεν ισχύει η  if
['360', 'jump', '_', '_', '390']            // exit;
['370', 'jump', '_', '_', '380']            //  για να προχωρήσει απλώς
['380', 'jump', '_', '_', '270']            //για επανάληψη του εσωτερικού loop
['390', '-', 'b', '1', 'T_7']               //b-1
['400', ':=', 'T_7', '_', 'b']              //b:=b-1
['410', 'jump', '_', '_', '250']            //για να συνεχίσει να τρέχει η  while
['420', 'jump', '_', '_', '200']            //για επανάληψη του loop, γραμμή 20
['430', 'halt', '_', '_', '_']
['440', 'end block', 'x', '_', '_']         //endprogram
```

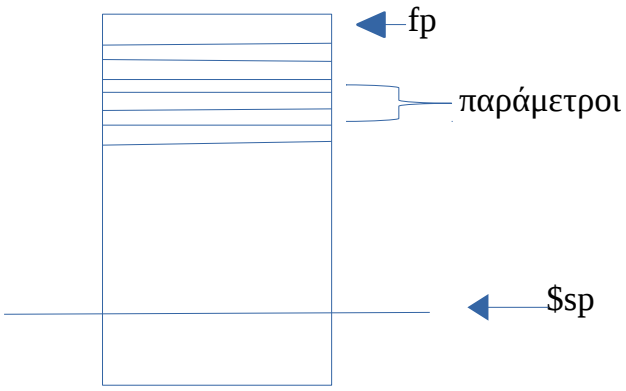
**<u>3η Φάση :  Τελικός Κώδικας</u>**

**Εισαγωγή MIPS**

***Καταχωρητές***

$t0-$t7  : καταχωρητές προσωρινών μεταβλητών
$s0-$s7 : καταχωρητές οι τιμές των οποίων διατηρούνται μεταξύ κλήσεων συναρτήσεων.
Για global μεταβλητές.

$a0-$a3 : καταχωρητές ορισμάτων. Μπορούμε να δουλέψουμε και χωρίς αυτούς. Μέχρι τέσσερα ορίσματα.
$v0-$v1 : καταχωρητές τιμές. Έξοδος στο τέλος.
$sp : stack pointer
$fp : για πιο μικρό κώδικα συναρτήσεων. Δείχνει στη συνάρτηση που χτίζουμε. Δείκτης πλαισίου : frame length. Το $fp όσο “δημιουργούμε” είναι χρήσιμο.
$ra : return address, πού θα κάνουμε jump όταν τελειώσει μία συνάρτηση. Η εντολή στην οποία επιστρέφουμε μετά απο μια συνάρτηση.



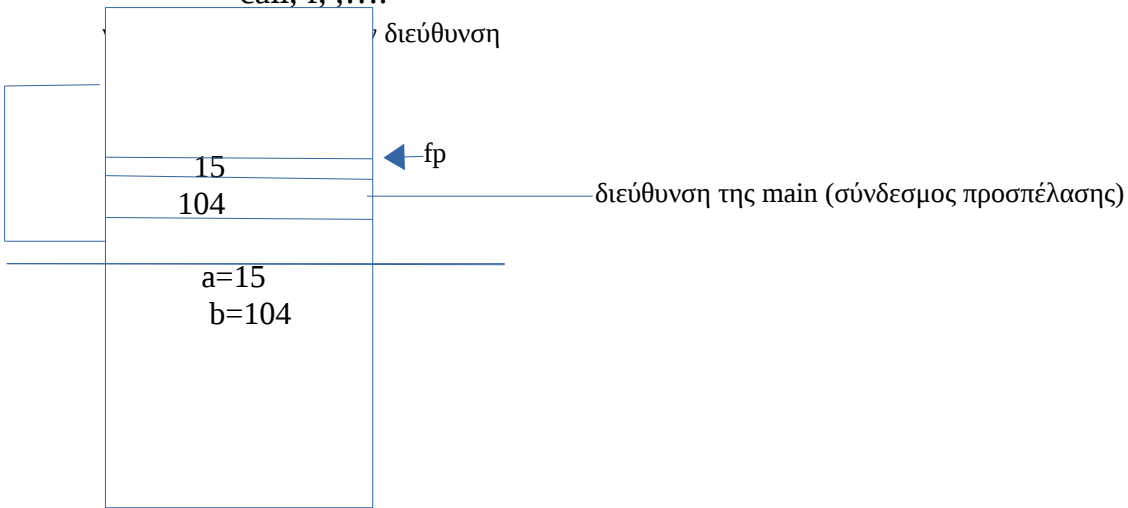

***Παράμετροι που βάζουμε***

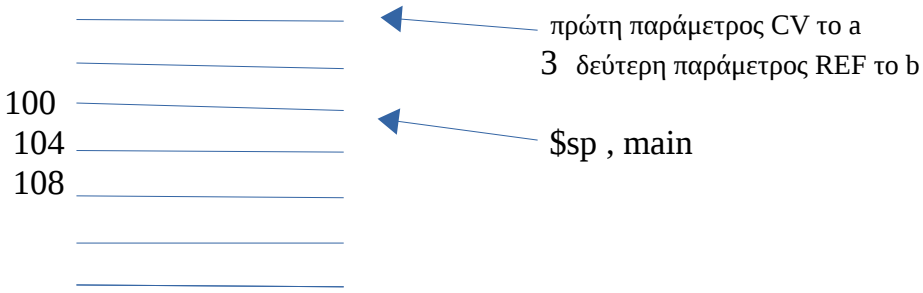
- Αν είναι CV , δηλαδή με τιμή, την τιμή της μεταβλητής.
- Αν είναι REF , βάζουμε την διεύθυνση της μεταβλητής.

Για παράδειγμα, αν έχουμε το

```
program x
        declare a, b
        function  f(in x, inout y)
        endfunction
        b := f(in a, inout b)
endprogram
```

δηλαδή έχουμε par , a, CV
par, b , REF
call, f, ,….






***Χρήσιμες Εντολές Assembly***

- add, sub, mul, div οι οποίες είναι της μορφής :

  add $t0, $t1, $t2 δηλαδή $t0 := $t1 + $t2
  sub $t0, $t1, $t2 δηλαδή $t0 := $t1 - $t2
  mul $t0, $t1, $t2 δηλαδή $t0 := $t1 * $t2
  div $t0, $t1, $t2 δηλαδή $t0 := $t1 / $t2

- υπάρχει η move : move $t0, $t1
- υπάρχει η load : li $t0, value
  lw $t1, mem
  sw $t1, mem
  lw $t1,($t0)
  $t1 = [$t0]
  sw $t1, -4($sp) : κατευθείαν πάμε στην μνήμη που θέλουμε
- άλματα :
  1. j label , πηγαίνουμε στο label
  2. conditional jump :
     - beq $t0, $t1, label δηλαδή if($t0==$t1) j label
     - bne <>
     - blt <
     - bge >=
     - ble <=
  3. jal label : κλήση συνάρτησης σε αυτό το label
  4. jr $ra : παίρνει έναν καταχωρητή και κάνει jump στην διεύθυνση καταχωρητή. Επιστροφή απο συνάρτηση.

*Παραδείγματα για κατανόηση*

- b := a

Είπαμε ότι $s0 έχει global , άρα a είναι στο $s0+12 και b είναι στο $s0+16. Ή φορτώνουμε στον $t0 το a άρα lw $t0,12($s0) ή βάζουμε στον b : sw $t0, 16($s0).

- b :=3
  li $t0,3
  sw $t0, 16($s0) .

***Εισαγωγή Τελικός Κώδικας***

(Ο πίνακας συμβόλων μάς δίνει π.χ. τα offset, τα πάντα απο εκεί τα παίρνουμε)

- *gnlvcode(a)* : μεταφέρει στον $t0 την διεύθυνση μίας μη τοπικής μεταβλητής

lw $t0, -4($sp) : το εγγράφημα δραστηριοποίησης του πατέρα μου, πού ξεκινάει.

βάλαμε τον σύνδεσμο προσπέλασης

Γράφουμε ( βάθος συνάρτησης – βάθος μεταβλητής – 1) φορές:

```
lw   $t0,-4($t0)
add  $t0, $t0, -offset
```

Δεν έχουμε φτάσει. Έχουμε άλλο ένα, γιατί ήδη κάναμε ένα. Απο πίνακα συμβόλων ξέρουμε ότι έχουμε δύο βήματα. Άρα με το δεύτερο βήμα βρίσκουμε τον παππού.

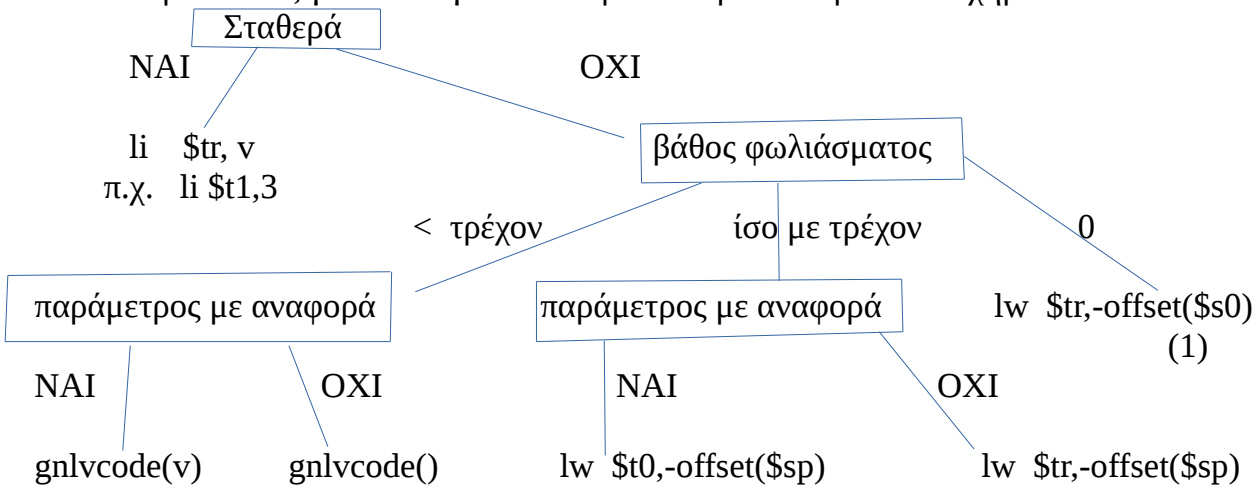
```
lw   $t0, -4($t0)
```

Ο $t0 δείχνει πια στην αρχή του παππού μας. Δεν βγάζουμε offset στον τελικό κώδικα. Βάζουμε απλά το νούμερο. Εκεί θέλουμε να καταλήξει η διεύθυνση που ψάχνουμε :

add $t0,$t0 , -offset για να πάρουμε την σωστή διεύθυνση.

Αν βάζαμε lw $t0, -offset θα παίρναμε την τιμή. Εμείς θέλουμε αυτό που κάνει το gnlvcode(a) που αναφέραμε πριν (τα αδέρφια έχουν φύγει απο την μέση).

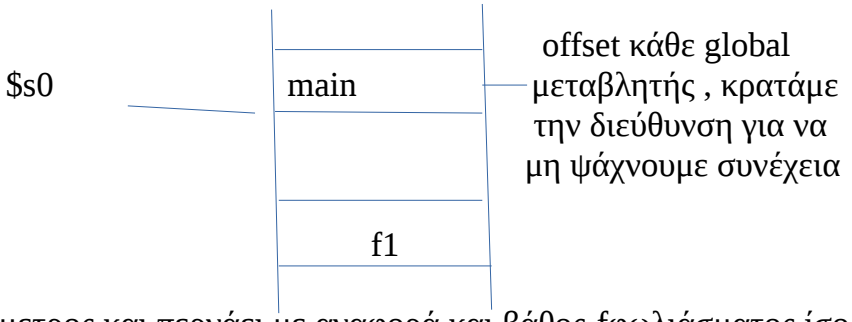
- loadvr(v,r) : φορτώνει στον καταχωρητή r την τιμή της v. Εδώ έχουμε διάφορες περιπτώσεις για τον λόγο αυτό παραθέτουμε το παρακάτω σχήμα:



lw $t0,($t0) lw $t,($t0) lw $t0,($t0) (2)
lw $tr,($t0) (4) (3)
(5)

(1) : εδώ η v είναι καθολική μεταβλητή, δηλαδή ανήκει στο κύριο πρόγραμμα. Είναι συνηθισμένο να τι σπροσπελαύνουμε και θέλει πολλά βήματα,χαλάμε προγραμματιστική ισχύ, κάνουμε λοιπόν εξαίρεση και την υλοποιούμε αλλιώς.

(2) : εδώ αν η v είναι τοπική μεταβλητή ή τοπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή.Με άλλα λόγια η τιμή της μεταβλητής βρίσκεται στην στοίβα μου.

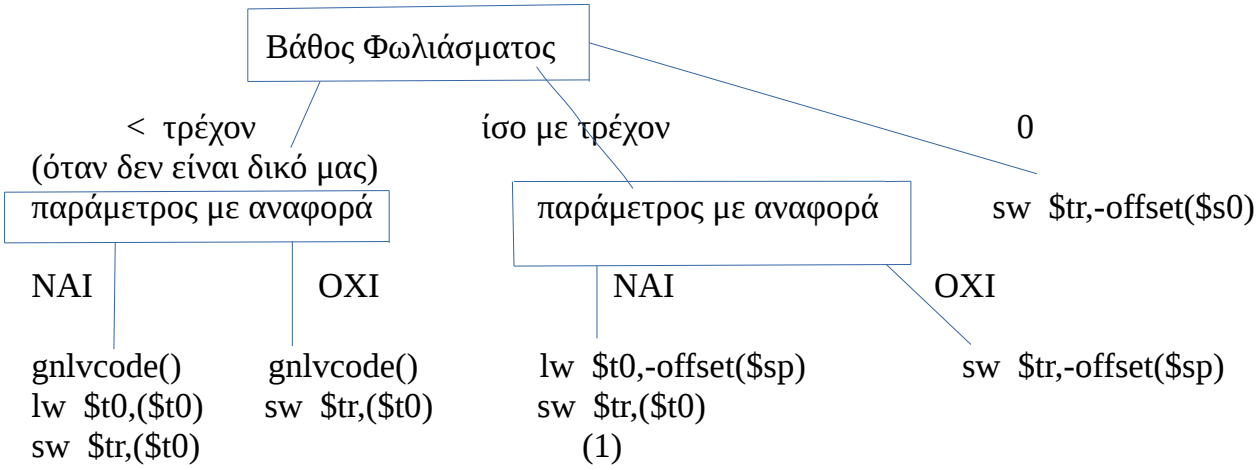



(3) : εδώ η v είναι τοπική παράμετρος και περνάει με αναφορά και βάθος fφωλιάσματος ίσο με το τρέχον.Με άλλα λόγια η διεύθυνση είναι στην στοίβα μου.

(4) : ΤΙΜΗ. Το v δε μου ανήκει, δεν είναι στη στοίβα μου, πρέπει να το βρούμε.Στο gnlvcode(v) το v είναι τοπική μεταβλητή και ο $t0 δείχνει στην v. Θέλουμε τώρα να πάρουμε το περιεχόμενο. Θα χρησιμοποιοήσουμε τον $t0.

(5) : βάζουμε στον $t0 το περιεχόμενο της διεύθυνσης που έδειχνε ο $t0 δηλαδή την διεύθυνση του v. Έχουμε ένα παραπάνω βήμα απο το προηγούμενο.

- *storerv(r,v) :* αποθηκεύει στην μεταβλητή v την τιμή του r. Εδώ έχουμε διάφορες περιπτώσεις για τον λόγο αυτό παραθέτουμε το παρακάτω σχήμα :




(1) : στη στοίβα μας κάτω έχουμε την διεύθυνση. Πρέπει να πάμε να την διαβάσουμε.

- *jump ,”_”,”_”,label*
  j label

- *relop x,y,z* όπου relop: =,<>,<,>,<=,>=
  loadvr(x,1)
  loadvr(y,2)
  branch $t1,$t2,z όπου branch: beq,bne,bgt,blt,bge,ble

- *:=,x,”_”, z*
  loadvr(x,1)
  storerv(1,z)

- *op,x,y,z* όπου op: +,-,*,/
  loadvr(x,1)
  loadvr(y,2)
  fop $t1,$t1,$t2 όπου fop: add,sub,mul,div
  storerv(1,z)

- *out,”_”,”_”,x*
  li $v0,1
  loadvr(x,0)
  move $a0,$t0
  syscall //καλούμε syscall

- *in,”_”,”_”,x* //input
  li $v0,5 το αποτέλεσμα στον $v0
  syscall
  move $t0,$r0 //πρέπει $t0 t→ $r0
  storerv(0,x)

- *retv,”_”,”_”,x* //παίρνουμε την τιμή του x , όλα τα άλλα πχ. Τί είναι η x
  loadvr(x,1) // δε μας απασχολεί
  lw $t0,-8($sp)
  sw $t1,($t0) σε αυτή που σε κάλεσε απάντησε ότι το αποτέλεσμα είναι στο x

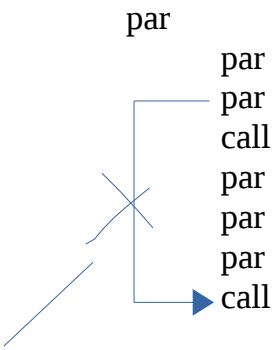
θα γραφεί

την μεταβλητή που επιστρέφει

Γενικά, όταν είμαστε σε μια συνάρτηση F και καλείται μια άλλη συνάρτηση f, φτιάχνουμε (απο παλιά τα έχουμε) μια προσωρινή μεταβλητή τ_i.

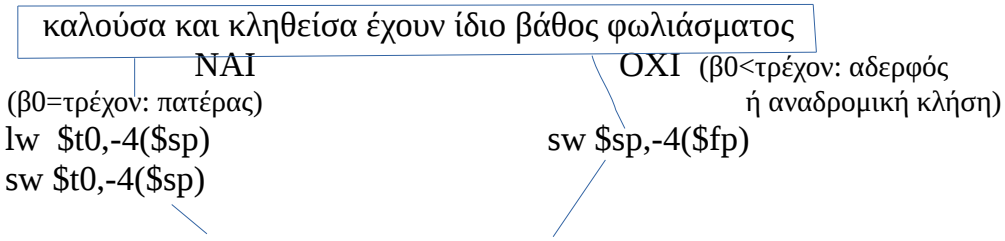
- *par,x,type,_*
  Στο 1ο par
  add $fp,$sp,framelength , όπου framelength είναι αριθμός που θα αντιστοιχεί

Γενικά, γνωρίζουμε ότι όσο βλέπουμε par ξέρουμε ότι διαβάζουμε παραμέτρους μέχρι να δούμε call. Για παράδειγμα,




δε μπορεί να είναι της κάτω call.

- *call,f,"_","_",*




add $sp, $sp , framelength #πρέπει να κατεβάσουμε τον $sp
jal label startquad της f #τρέξε για Minix (1)
add $sp,$sp,-framelength #την δ/νση αυτή την έχει η προηγούμενη εντολή

(1) : Η jal πάει και γεμίζει τον $ra με την διεύθυνση επιστροφής.Γεμίζει με το σημείο που πρέπει να κάνω jump όταν τελειώσω.
Γενικά πριν βρούμε την call, βρίσκουμε τα par. Επίσης, για την call γεμίζουμε τον σύνδεσμο προσπέλασης.

- *begin block,x,_,_*
  (όχι κυρίως πρόγραμμα)
  sw $ra,($sp) #περίπτωση να χαλάσει ο $ra είναι η f να καλέσει κάποιο άλλο κι έτσι να έχουμε καινούριο $ra για αυτό διασώζουμε τον $ra και πάμε και τον βάζουμε στην πρώτη θέση της στοίβας

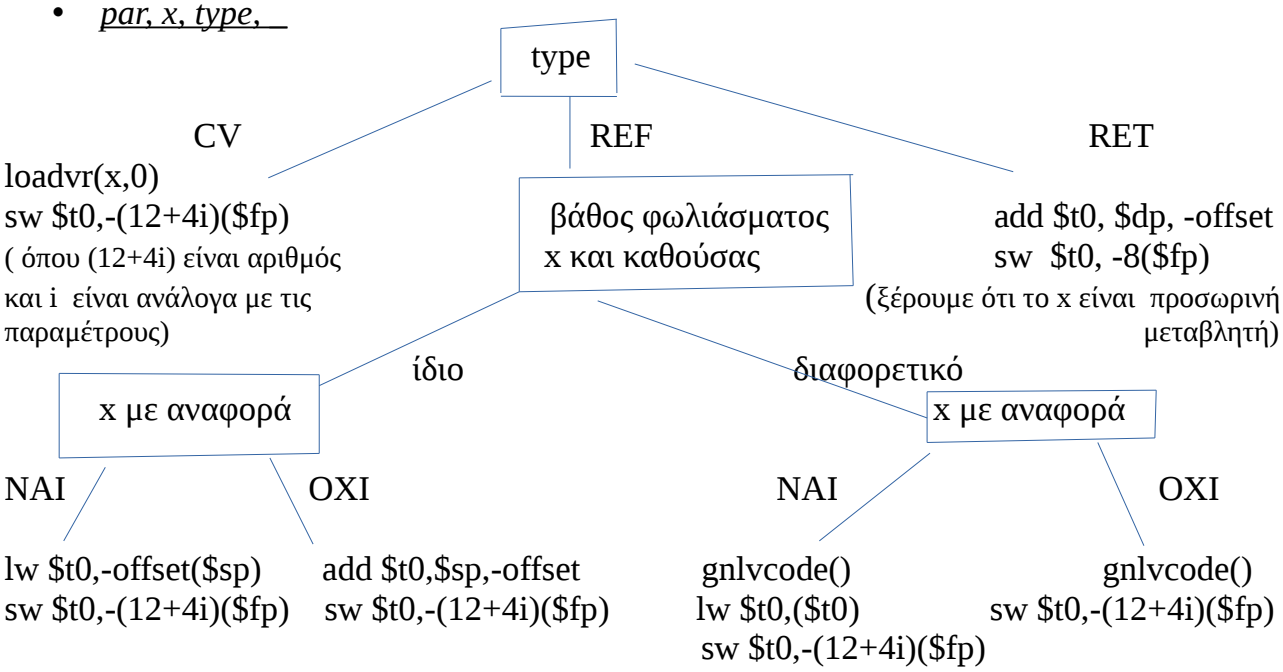
- *end block,x,_,_*
  lw $ra,($sp) #διαβάζουμε και επιστρέφουμε την τιμή πίσω στον $ra
  jr $ra

Σημείωση: αν κάνουμε δέκα κλήσεις αναδρομικές, δημιουργούνται δέκα εγγραφήματα το ένα κάτω απο το άλλο.

- *αρχή μετάφραση*

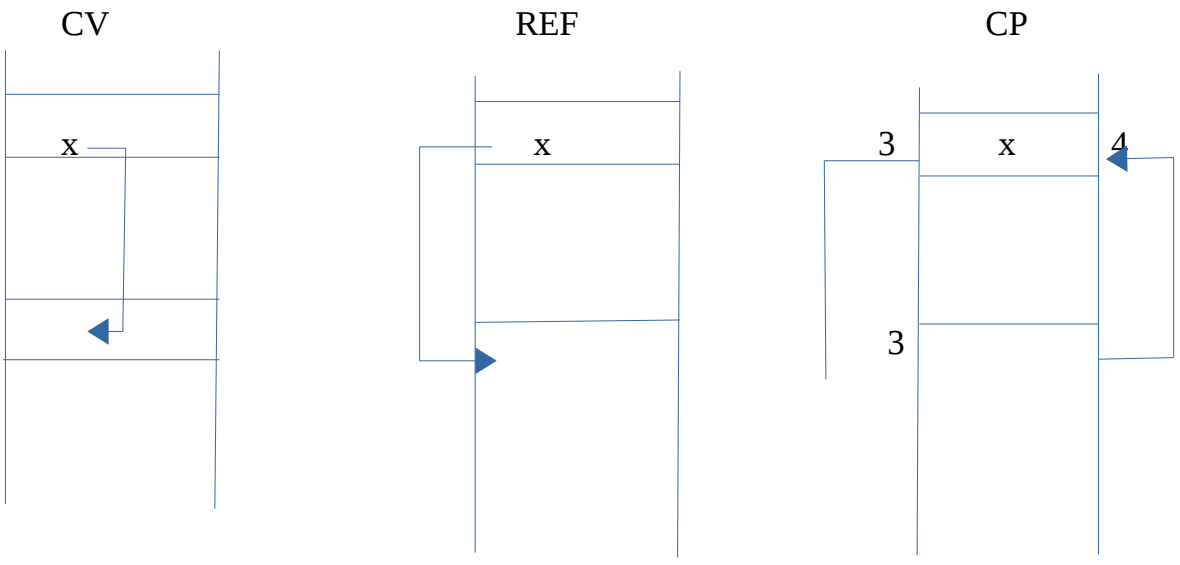
  j Lmain  #για να εκτελεστεί πρώτα η main, την Lmain την γράφουμε κάπου μετά
  add $sp,$sp,framelength  #τοποθετούμε εκεί που πρέπει τον $sp .Πρέπει να τον κατεβάσουμε  τον $sp κατά framelength της main
  move $s0,$sp  #είναι καλό να κρατήσουμε στον $s0 πού είναι το εγγράφημα της main για αυτό τον κρατάμε τον $sp

- *par, x, type, _*




*Σημείωση : Ποτέ δεν έχουμε δείκτη σε δείκτη. Πάντα έχουμε δείκτη σε μεταβλητή. Μόνο στην συνάρτηση είχαμε δείκτη σε δείκτη.*

*Παράδειγμα,*



*Παράδειγμα*

```
program example
        declare a,b enddeclare

        procedure p1(in x, inout y)
                declare c,d enddeclare

                function p11(in w,inout z)
                        declare e enddeclare

                        function p21(in x)
                                e:=x;
                                z:= w;
                                e:=p21(in a);
                                return(e)
                        endfunction

                        e:=x=z;
                        z:= w;
                        e:=p21(in c);
                        return(c)
                endfunction

                b:=100
                c:=p11(in b,inout c)
                y:=b+c
        end procedure

        call p1(in a, inout b)
endprogram
```

όπου έχουμε

P0:  bb p21
P1: := x _ e

```
P2: := w _ z
P3: par a CV
P4: par T_1 RET
P5: call p21
P6: := T_1 _ e
P7: retv e _ _
P8: endblock p21
P9: bb P1
P10: := z _ e
P11:  := w _ z
P12: par c CV
P13: par T_2  RET
P14: call  P21
P15: := T_2  _ e
P16: retv _ _ e
P17: end block P11
P18: bb P1
P19: := 100 _ b
P20: par b CV
P21: par c REF
P22: par T_3  RET
P23: call P11
P24: := T_3 _ c
P25: + b c T_4
P26: := T_4 _ y
P27: endblock
P28: bb main
P29: par a CV
P30: par b REF
P31: call P1
P32: halt                    // μάς βγάζει έξω
P33: endblock
```

όπου μόλις παραχθεί αυτό
και πίνακας συμβόλων :

όπου στον πίνακα συμβόλων το T_1:16 είναι προσωρινή μεταβλητή με offset 16 και εμείς θέλουμε την διεύθυνση της  L4: add $t0, $sp, -16

και e:20 όπου το e είναι στο πατέρα μας offset 20, τοπική μεταβλητή . Αυτό που μας δείχνει είναι ο σύνδεσμος προσπέλασης

τότε έχουμε παραγωγή τελικού κώδικα:

```
L:      j Lmain #για ομορφιά
L0:     sw $ra,($sp)
```

```
L1:     lw $t1,-12($sp)
                    gnlvcode() για να μας ανεβάσει ένα επίπεδο
        lw $t0,-4($sp)
        add $t0,$t0,-20 #στον $t0 (αντιγράφουμε διεύθυνση) τον χρησιμοποιούμε ως δείκτη
                            #για να δείξει στον $t0
        sw $t1,($t0)
                    η gnlvcode πάλι μάς ανεβάζει ένα επίπεδο
L2:     lw $t0,-4($sp)    #μάς έχει πάει στην αρχή εγγραφήματος τού πατέρα
        add $t0,$t0,-12
        lw $t1,($t0)  #βάλαμε στον $t1 τον w
        lw $t0,-4($sp)    #για να ανέβει στον πατέρα
        add $t0,$t0,-16 #προσθέτει το offset που χρειάζεται δηλαδή 16
        lw $t0,($t0)  #ο $t0 έχει την διεύθυνση μεταβλητής
        sw $t1,($t0)
                    Σε αυτή την φάση θα χτυπήσει.Εδώ ήξερε. Το “α” είναι η πρώτη global
                    μεταβλητή.
L3:     add $fp,$sp,20   #πρώτη παράμετρος, βάζουμε $fp στον $sp
        lw $t0,-12($s0)  #στο 12 μπαίνει η πρώτη παράμετρος
        sw $t0,-12($fp)
                    Πάμε στην άλλη παράμετρο που είναι RET.
L4:     add $t0,$sp,-16
        sw $t0,-8($fp)    #θέλουμε να γράψουμε τον $t0 στη θέση που έχουμε για αυτή την
                            #δουλειά, δηλαδή -8, επιστροφή τιμής συνάρτησης
L5:     lw $t0,-4($sp)   #να τη η call.Πρώτο πράγμα που φτιάχνει είναι σύνδεσμο
                            #προσπέλασης.Αφού είμαστε “οι ίδιοι” , έχουμε τον ίδιο πατέρα.
                #Δεν έχει αλλάξει `ακόμα ποιος τρέχει. Παίρνουμε τον πατέρα της παλιάς.
        sw $t0,-4($fp)    #αυτής που τώρα δημιουργούμε
        add $sp,$sp,20  #μεταφέρουμε τον $sp στον $fp
        jal L0 #θα φροντίσουμε να κάνει οτιδήποτε χρειάζεται. Η jal μάς στέλνει πίσω.
        add $sp,$sp,-20 #επιστροφή του stack pointer
                            Ψάχνει τί είναι το e. Δεν αλλάζει η εξήγηση που είχαμε
                            προηγουμένως.
L7:     lw $t0,-4($sp)
        add $t0,$t0,-20
        lw $t1,($t0)        #ο $t1 έχει τώρα την τιμή που θέλουμε
        lw $t0,-8($sp)   #βάλαμε στο -8($sp) τη τιμή του
        $t0sw $t1,($t0)  #καταφέραμε μέσω δείκτη να βάλουμε την τιμή που θέλαμε στο $t1
L8:     lw $ra,($sp)  #επαναφέρει στον $ra αυτό που είχαμε κρατήσει
        jr $ra
L9:     sw  $ra, ($sp)
L10:    lw   $t0, -16($sp)
        lw   $t1, ($t0)
        sw  $t1,-20 ($sp)
L11:    lw   $t1, -12($sp)
        lw   $t0, -16($sp)
```

```
L12:  add $fp,$sp,20        #το κατεβάζω κατά 20 θέσεις
      lw   $t0, -4($sp)     #πάμε στοίβα του πατέρα
       add $t0,$t0,-20      #ο $t0  να δείχνει το c
      lw   $t0, ($t0)
      sw  $t0,-12 ($fp)     #είμαστε στο P12
            (  πρέπει   να πάρουμε την διεύθυνση του T_2   τώρα    )
L13:   add $t0,$sp,-24
      sw  $t0,-8($fp)
L14:  sw  $sp,-4($fp)       # πρώτα φτιάχνουμε τον σύνδεσμο προσπέλασης.Ο πατέρας
                            #είναι 11. Έχουμε κλήση πατέρα σε παιδί
            ( πρέπει να κατεβάσουμε τον $sp απο την θέση 11 στην 21 )
       add $sp,$sp,20
      jal  L0          #είναι έτοιμη να τρέξει αρκεί να κάνουμε jal
      add $sp,$sp,-20       #γυρνάμε πίσω & παίρνουμε τον $sp
                                  ( τελείωσε η call )
L15:  lw $t1,-24($sp)
       sw $t1,-24($sp)      #τοπική
L16:  lw $t1,-20($sp)       #επιστρέφει το e
      lw $t0,-8($sp)
      sw $t1,($t0)
L17:   lw $ra,($sp)
      jr $ra
L18:   sw $ra,($sp)
L19:  li $t1,100
       sw $t1,-16($s0)
L20:   add $fp,$sp,28       #framelength P11
      lw $t1,-16($sp)
       sw $t1,-12($fp)      #στην 1η θέση παραμέτρου
                            (δική μου τοπική  είναι η c)
L21:  add $t0,$sp,20        #στον $sp το άθροισμα του $t0+20
                            (πρέπει να βρω πρώτα την T_4)
L22:   add $t0,$sp,-28
      sw $t1,-8($fp)
L23:  sw $sp,-4($fp)        #δείχνουμε πού είναι η στοίβα του πατέρα
            (πρέπει να δώσουμε το εγγράφημα δραστηριοποίησης)
       add $sp,$sp,28
      jal  L9
       add $sp,$sp,-28
L24:  lw $t1,-28($sp)
      sw $t1,-20($sp)   #είναι δικό μας
L25:  lw $t1,-16($sp)       #-16 γιατί είναι global
      lw $t2,-20($sp)       #-20 γιατί είναι τοπικό
      add $t1,$t1,$t2
      sw $t1,-32($sp)
L26:  lw $t1,-32($sp)
```

```
        lw $t0,-32($sp)
        sw $t1,($t0)          #πήγαμε πραγματικά στο y και όχι στην διεύθυνσή του
L27:    lw $ra,($sp)
        jr $ra
Lmain:  add $sp,$sp,20
          move $s0, $sp
L28:                          #δεν κάνουμε κάτι απλά προχωράμε
L29:    add $t0,$sp,36
        lw $t1,-12($sp)       #πάμε να πάρουμε την a
        sw $t1,-12($sp)       #και να την αποθηκεύσουμε
L30:    add $t0,$sp,-20
        sw $t0,-16($φp)
              (η main παιδί της θα καλέσει. Πρώτα κάνω σύνδεσμο προσπέλασης.)
L31:    add $sp,$fp,36        #κατεβάζουμε δείκτη στοίβας κατά 36
        jal  L18
        add $sp,$sp,-36
```

## Ανάλυση Υλοποίησης Τελικού Κώδικα

Για την ορθή υλοποίηση, σκεφτόμαστε πού θα βάλουμε κώδικα έχοντας διατηρήσει όλα τα προηγούμενα στο πρόγραμμά μας και πού θα κάνουμε τις κατάλληλες τροποποιήσεις.
Έτσι,

- Για να εκτυπωθεί αρχείο σε assembly προσθέσαμε το εξής κομμάτι κώδικα:
  onomaArxeiou=onomaArxeiou+"asm"
  arxeioASM=open(onomaArxeiou,"w")
  grammi=1 όπου το grammi=1 είναι για τα λάθη σε ποια γραμμή του .stl αρχείου είμαστε.
- ***def eyresh(onoma):*** είναι μια νέα συνάρτηση. Βρίσκουμε το entity και το βάθος φωλιάσματος της μεταβλητής. Διαφέρει απο την συνάρτηση eyreshEntity() γιατί η συνάρτηση eyresh() μάς βοηθάει στην παραγωγή της assembly, ενώ η συνάρτηση eyreshEntity() είναι καθαρά για την σημασιολογική ανάλυση και το προηγούμενο στάδιο ώστε να γίνονται οι έλεγχοι για τυχόν λάθη. Επίσης, στην συνάρτηση αυτή βλέπουμε αν το onoma που έχουμε δώσει ως παράμετρο είναι συνάρτηση ή π.χ. παράμετρος.
- ***def gnlvcode(x):*** είναι μία καινούρια συνάρτηση. Εδώ κάνουμε εκτύπωση της φόρτωσης του περιεχομένου του καταχωρητή $sp μειωμένου κατά τέσσερα στον καταχωρητή $t0 . Επίσης, βρίσκουμε το entity και το βάθος φωλιάσματος της μεταβλητής που δέχεται η gnlvcode() ως παράμετρο με την βοήθεια της συνάρτησης

eyresh(x). Έπειτα, έχοντας διατρέξει το κατάλληλο βάθος φωλιάσματος, for i in range(len(vathosFwliasmatos - vathosFwliasmatosMetavlhths -1)) όπου μείον ένα γιατί θέλουμε μέχρι και ένα επίπεδο απο κάτω , εκτυπώνουμε στο αρχείο την φόρτωση του περιεχομένου της διεύθυνσης -4($t0) στον καταχωρητή $t0 με load word κι έτσι τελικά είμαστε σε θέση να αυξήσουμε το offset της μεταβλητής και να βρούμε το σωστό offset και άρα πέρα απο την εντολή add της assembly που χρησιμοποιούμε και τον καταχωρητή $t0 που αξιοποιήσαμε πριν , βρίσκουμε και το offset στην θέση 3 των entity δηλαδή entity[2] κι αφού κάνουμε την μετατροπή του σε string γίνεται κανονικά η εκτύπωση που θέλουμε.

- ***def loadvr(v, r):*** η συνάρτηση αυτή είναι καινούρια. Εδώ όπως αναφέραμε και στην εισαγωγή προηγουμένως έχουμε διάφορες περιπτώσεις. Για να δούμε σε ποια περίπτωση είμαστε , λοιπόν, κάνουμε τους ελέγχους που απαντούνται για την παράμετρο v . Αν είναι σταθερά τότε πρέπει η συνάρτηση isdigit() να μάς επιστρέψει νούμερο (την isdigit() την έχουμε ήδη απο προηγούμενο στάδιο) και άρα η εκτύπωση στην assembly γίνεται με li και επίσης στην εκτύπωση του arxeioASM.write χρησιμοποιούμε το r για το πλήθος των καταχωρητών δηλαδή για παράδειγμα $t0 και μετά την τιμή, δηλαδή v. Σε διαφορετική περίπτωση ελέγχουμε αν η παράμετρος v είναι global κι αν συμβαίνει αυτό το κατάλληλο offset για την εκτύπωση του αρχείου σε assembly γίνεται με την βοήθεια και πάλι του str(entity[2]). Αλλιώς υπάρχει η περίπτωση να είναι η παράμετρος με αναφορά και άρα ελέγχουμε στην θέση entity[0] που υπάρχει το είδος αν είναι γραμμένο REF και αν είναι χειριζόμαστε τον κώδικα όπως αναφέραμε και στην εισαγωγή. Διαφορετικά, αν υπάρχει ίδιο βάθος φωλιάσματος η εκτύπωση σε assembly γίνεται με βάσει πάλι τους κατάλληλους τύπους και πάλι με χρήση μεταβλητής r για το νούμερο του καταχωρητή και str(entity[2]) για το offset.
- ***def storerv(r, v):*** είναι νέα συνάρτηση. Την λογική της την αναλύσαμε στο θεωρητικό κομμάτι πιο πάνω. Εδώ πάλι γίνονται οι κατάλληλοι έλεγχοι για το βάθος φωλιάσματος και για το αν η παράμετρος είναι με αναφορά ώστε να ελέγξουμε τις περιπτώσεις. Σε θέμα κώδικα και εντολών έχουμε εξηγήσει πιο πάνω ό,τι θα μπορούσε να χρειαστεί.
- ***def ektypwshASM():*** αποτελεί νέα συνάρτηση, η οποία πραγματοποιεί την εκτύπωση σε assembly. Εδώ κρατάμε μία μεταβλητή flag, επειδή θέλουμε να κρατάμε κάθε φορά την πρώτη παράμετρο απο κάθε συνάρτηση για να τοποθετούμε κατάλληλα τον $fp και μία παράμετρο parI για να μπορούμε να υπολογίζουμε κάθε φορά τον τύπο που μάς έχει δοθεί -(12+4*i) ,ο οποίος όπως είναι κατανοητό γίνεται -(12+4*parI).

    Ύστερα, ξεκινάμε και διατρέχουμε όλες τις τετράδες και δημιουργούμε σχόλια για την κάθε τετράδα με το σύμβολο # ,όπως επίσης γράφουμε και το αντίστοιχο Label με τον αντίστοιχο φυσικά αριθμό της τετράδας. Έτσι, λοιπόν, έπειτα παίρνουμε περιπτώσεις για τον κάθε τελεστή και “λέξη” που συναντάμε για να κάνουμε τις κατάλληλες μετατροπές σε Assembly.

Ο πρώτος έλεγχος που κάνουμε είναι αν στην πρώτη θέση της τετράδας έχουμε συν και επομένως πρέπει να γίνει πρόσθεση του στοιχείου στην τρίτη και στην τέταρτη θέση των τετράδων ώστε να γίνεται σωστά add $t1, $t1, $t2. Αυτό γίνεται αντίστοιχα για το μείον που αντιστοιχεί στο sub, για το το επί που αντιστοιχεί στο mul και για το div που

αντιστοιχεί στην διαίρεση. Μετά έχουμε την ανάθεση , δηλαδή :=, δηλαδή την εκχώρηση μιας τιμής σε μια μεταβλητή ή σταθερά.

Ακόμα, έχουμε το jump στις τετράδες το οποίο στην assembly αντιστοιχεί στο j Label και τον αντίστοιχο αριθμό ετικέτας που βρίσκεται στην πέμπτη θέση. Τώρα ακολουθούν όλες οι συγκρίσεις που έχουμε αναφέρει στην εισαγωγή του τελικού κώδικα και την ετικέτα στην οποία γίνεται η μετάβαση πάλι την παίρνουμε απο την πέμπτη θέση των τετράδων. Ακολουθούν , τώρα, η περίπτωση του out που στην ουσία κάνουμε μόνο του arxeioASM.write την εγγραφή των εντολών που έχουμε αναφέρει στην εισαγωγή , το ίδιο συμβαίνει και την περίπτωση του inp , δηλαδή έχουμε αναφέρει πιο πάνω τις εντολές και απλά πηγαίνουμε στις τετράδες για να δούμε αν είναι inp , ret κλπ για να δούμε τί θα εκτυπώσουμε στο αρχείο της assembly.

Απο εκεί και πέρα αν στις τετράδες συναντήσουμε par μέσω του flag που έχουμε δηλώσει για κάθε συνάρτηση βρίσκουμε το $fp όπως προείπαμε και ψάχνουμε να βρούμε ποια συνάρτηση θα καλέσουμε και αυτό γίνεται όταν στις τετράδες βρούμε το call. Πηγαίνουμε λοιπόν και απο την απο πάνω σειρά στον πίνακα συμβόλων, έχοντας φτάσει δηλαδή στην συνάρτηση που θέλουμε παίρνουμε entity και το βάθος φωλιάσματος της συνάρτησης με την βοήθεια της βοηθητικής συνάρτησης που έχουμε κατασκευάσει f. Επίσης, παίρνουμε το framelength της συνάρτησης απο τον την λίστα entity και αυτό βρίσκεται στην τέταρτη θέση. Έπειτα, στο αρχείο της assembly εκτυπώνουμε τις εντολές που αντιστοιχούν εδώ και είπαμε στην εισαγωγή αφού βέβαια έχουμε μετατρέψει το framelength σε string και αφού γίνει αυτό φυσικά κάνουμε το flag πια false.

Στη συνέχεια , κάνουμε τους ελέγχους στις τετράδες για να δούμε τί είδος παραμέτρου έχουμε αν είναι δηλαδή CV, REF και κινούμαστε ανάλογα. Τώρα για το call που ακολουθεί, θέλουμε ο $fp να μετακινηθεί και άρα πάλι ορίζουμε ένα flag και βλέπουμε πότε είμαστε στο κυρίως πρόγραμμα και πότε όχι.

**Εκτυπώσεις Αρχείου test3.stl σε test3.int, test3.c, test3.asm**

Εκτελώντας στο τερματικό python final.py test3.stl δημιουργούνται τρία αρχεία: test3.int, test3.c, test3.asm και στο τερματικό εμφανίζονται τα εξής :

```
cs091716@opti7020ws11:~/Desktop/ergasiaMetafrastes
st3.stl

1-f-24
        ['CV', 'x', 12]
        ['REF', 'y', 16]
        ['proswrinh', 'T_1', 20]
0-test-24
        ['metavlith', 'a', 12]
        ['metavlith', 'b', 16]
        ['metavlith', 'c', 20]
        ['function', 'f', '0', 24]

0-test-28
        ['metavlith', 'a', 12]
        ['metavlith', 'b', 16]
        ['metavlith', 'c', 20]
        ['function', 'f', '0', 24]
        ['proswrinh', 'T_2', 24]
['0', 'begin block', 'f', '_', '_']
['10', ':=', '100', '_', 'x']
['20', ':=', '200', '_', 'y']
['30', '+', 'x', 'y', 'T_1']
['40', 'retv', '_', '_', 'T_1']
['50', 'end block', 'f', '_', '_']
['60', 'begin block', 'test', '_', '_']
['70', ':=', '1', '_', 'a']
['80', ':=', '2', '_', 'b']
['90', 'par', 'a', 'CV', '_']
['100', 'par', 'b', 'REF', '_']
['110', 'par', 'T_2', 'RET', '_']
['120', 'call', 'f', '_', '_']
['130', ':=', 'T_2', '_', 'c']
['140', 'out', '_', '_', 'a']
['150', 'out', '_', '_', 'b']
['160', 'out', '_', '_', 'c']
['170', 'halt', '_', '_', '_']
['180', 'end block', 'test', '_', '_']
cs091716@opti7020ws11:~/Desktop/ergasiaMetafrastes
```

**Ανάλυση Πίνακα Συμβόλου αρχείου test3 που εμφανίζεται στο τερματικό**

Αρχικά,έχουμε λίστες για τα entities όπου βλέπουμε τί τύπου είναι, το όνομα της μεταβλητής και το offset της. Για παράδειγμα [‘CV’,’x’,12]. Επίσης, για τα scopes κρατάμε το βάθος φωλιάσματος, το όνομα, λίστα με τα entities και το framelength. Η μορφή των scopes μας είναι , για παράδειγμα, 1-f-24 δηλαδή βάθος φωλιάσματος 1 για την συνάρτηση f και το framelength της είναι 24. Απο εκεί και πέρα, κάτω απο κάθε scope και ένα tab πιο δεξιά εμφανίζονται τα entities του κάθε scope.

**Ανάλυση αρχείου test3.int**

```
0,beginblock,f,_,_   //function      f(in     x,inout          y)
10,:=,100,_,x        //x:=100;
20,:=,200,_,y        //y:=200;
30,+,x,y,T_1         //x+y
40,retv,_,_,T_1      //return        x+y;
50,end        block,f,_,_    //endfunction
60,begin      block,test,_,_
70,:=,1,_,a   //a:=1;
80,:=,2,_,b   //b:=2;
90,par,a,CV,_        //in     a
100,par,b,REF,_      //inout b
110,par,T_2,RET,_  //η T_2         είναι   τύπου return
120,call,f,_,  //f(in   a,      inout   b);
130,:=,T_2,_,c       //c:=f(in a, inout      b);
140,out,_,_,a        //print  a;
150,out,_,_,b        //print  b;
160,out,_,_,c        //print  c;
170,halt,_,_,_
180,end block,test,_,_       //endprogram
```

**Ανάλυση αρχείου test3.c**

Συμβατικά, μόνο εδώ στην ανάλυση του τρέχοντος αρχείου συμβολίζουμε τα σχόλιά μας(τα σχόλια δηλαδή των συγγραφέων του report) με # :

```
#include	<stdio.h>
int	main(){
int	a;		#declare	a,b,c;
int	b;
int	c;
int	T_1;
int	T_2;
//0,begin	block,f,_,_			#function	f(in	x,	inout	y)
Label0:	;
//10,:=,100,_,x				#x:=100;
Label10:	x=100;
//20,:=,200,_,y				# y:= 200;
Label20:	y=200;
//30,+,x,y,T_1				#return	x+y;
Label30:	T_1=x+y;
//40,retv,_,_,T_1
Label40:	;
//50,end	block,f,_,_			#endfunction
Label50:;
//60,begin	block,test,_,_
Label60:	;
//70,:=,1,_,a				#a :=	1;
Label70:	a=1;
//80,:=,2,_,b				#b	:=	2;
Label80:	b=2;
//90,par,a,CV,_				#c:=f(in	a,inout	b);
Label90:	;
//100,par,b,REF,_
Label100:	;
//110,par,T_2,RET,_
Label110:	;
//120,call,f,_,_
Label120:	;
//130,:=,T_2,_,c
Label130:	c=T_2;
//140,out,_,_,a				#print	a;
Label140:	printf("%d\n",a);
//150,out,_,_,b				#print	b;
```

```
Label150:     printf("%d\n",b);
//160,out,_,_,c                         #print c;
Label160:     printf("%d\n",c);
//170,halt,_,_,_
Label170:;
//180,end       block,test,_,_
Label180:     ;
return0;
}
```

**Ανάλυση αρχείου test3.asm**

Συμβατικά, μόνο εδώ στην ανάλυση του τρέχοντος αρχείου συμβολίζουμε τα σχόλιά μας(τα σχόλια δηλαδή των συγγραφέων του report) με // :

```
j         LabelMain
#0,begin      block,f,_,_    //function      f(in     x,inout          y)
Label0:
sw      $ra,($sp)
#10,:=,100,_,x                          //x:=100;
Label10:
li        $t1,100
sw      $t1,-12($sp)
#20,:=,200,_,y                          //y:=200;
Label20:
li        $t1,200
lw      $t0,-16($sp)
sw      $t1,($t0)
#30,+,x,y,T_1                           //return          x+y;
Label30:
lw      $t1,-12($sp)
lw      $t0,-16($sp)
lw      $t2,($t0)
add $t1, $t1,  $t2
sw      $t1,-20($sp)
#40,retv,_,_,T_1
Label40:
lw      $t1,-20($sp)
```

```
lw      $t0,-8($sp)
sw      $t1,($t0)
#50,end         block,f,_,_                     //endfunction
Label50:
lw      $ra,($sp)
jr      $ra
#60,begin       block,test,_,_
Label60:
LabelMain:
add     $sp,$sp,28
move  $s0,    $sp
#70,:=,1,_,a                                    //a:=1;
Label70:
li       $t1,    1
sw      $t1,-12($sp)
#80,:=,2,_,b                                    //b:=2;
Label80:
li       $t1,    2
sw      $t1,-16($sp)
#90,par,a,CV,_
Label90:
add      $fp, $sp,       24
lw      $t0,    -12($sp)
sw      $t0,-12($fp)
#100,par,b,REF,_
Label100:
add     $t0, $sp,       -16
sw      $t0,-16($fp)
#110,par,T_2,RET,_
Label110:
add     $t0,$sp,-24
sw       $t0,    -8($fp)
#120,call,f,_,_                                 //c:=f(in         a,inout         b);
Label120:
sw      $sp,    -4($fp)
add $sp, $sp, 24
jal      Label0
add $sp, $sp, -24
#130,:=,T_2,_,c
Label130:
lw      $t1,    -24($sp)
sw      $t1,    -20($sp)
#140,out,_,_,a                                  //print a;
Label140:
li       $v0,   1
```

```
lw      $t0,    -12($sp)
move  $a0,    $t0
syscall
#150,out,_,_,b                          //print b;
Label150:
li         $v0,   1
lw      $t0,-16($sp)
move  $a0,    $t0
syscall
#160,out,_,_,c                          //print c;
Label160:
li$v0,1
lw      $t0,    -20($sp)
move  $a0,    $t0
syscall
#170,halt,_,_,_
Label170:
#180,end      block,test,_,_            //endprogram
Label180:
```

**Επίλογος για Μεταγλωττιστές**

Ο κώδικάς μας τρέχει κανονικά και εμφανίζονται τα ανάλογα αποτελέσματα. Παρόλα αυτά, υπάρχουν ορισμένες αδυναμίες που αν και προσπαθήσαμε να υλοποιήσουμε τα εκάστοτε “κομμάτια” των ζητουμένων , τελικά δεν τα καταφέραμε. Οι αδυναμίες στις οποίες αναφερόμαστε είναι η απουσία υλοποίησης των arguments στον πίνακα συμβόλων, όπως και η περίπτωση CP στον τελικό κώδικα.