*Μπαντουβάκης Γεώργιος, ΑΜ:4119*
*Τσαπανίδης Μιχαήλ, ΑΜ:4186*

*26/5/2021*

# *ΜΕΤΑΦΡΑΣΤΕΣ*

## *Αναφορά*

Η αναφορά που ακολουθεί αφορά την κατασκευή του μεταγλωττιστή met.py για την γλώσσα Cimple που υλοποιήσαμε κατά την διάρκεια του εξαμήνου.

Η εργασία χωρίζεται σε 3 μέρη τα οποία θα αναλυθούν εκτενώς παρακάτω και είναι:

1. Λεκτικός και Συντακτικός αναλυτής
2. Ενδιάμεσος κώδικας
3. Πίνακας Συμβόλων και Τελικός κώδικας

### *1. Λεκτικός και Συντακτικός αναλυτής*

Σε αυτό το κομμάτι του project στόχος είναι να μπορέσουμε να ελέγξουμε αν τηρούνται όλοι οι κανόνες της γραμματικής της γλώσσας Cimple η οποία μας δόθηκε.

*Αρχικά θα ασχοληθούμε με τον Λεκτικό αναλυτή.*
Ο σκοπός του Λεκτικού αναλυτή είναι να μας επιστρέφει κάθε φορά που καλείται, ένα token ,το οποίο είναι μια δομή αποθήκευσης που κρατάει πληροφορίες για την επόμενη λεκτική μονάδα. Πιο αναλυτικά ο λεκτικός αναλυτής είναι μια συνάρτηση, στον κώδικά μας η lex() ( η οποία έχει υλοποιηθεί στις γραμμές 48-160), η οποία διαβάζει το αρχείο εισόδου ( το αρχείο .ci δηλαδή, που περιέχει τον κώδικά μας ) γράμμα-γράμμα και επιστρέφει την επόμενη λεκτική μονάδα που συναντά. Η λεκτική μονάδα μπορεί να είναι:

- Δεσμευμένη λέξη της γλώσσας (οι δεσμευμένες λέξεις της γλώσσας έχουν οριστεί αναλυτικά στον κώδικας ως bound_words στην γραμμή 14-15).
- Σύμβολο της γλώσσας (Τα αποδεκτά σύμβολα της γλώσσας έχουν οριστεί στις λίστες: symbols, operators και groupsymbols στις γραμμές 17-21).
- Αναγνωριστικό ή σταθερά (αναγνωριστικά που ξεκινούν από γράμμα και αποτελούνται από γράμματα ή ψηφία και φυσικούς αριθμούς ως αριθμητικές σταθερές).
- Λάθη ( μη επιτρεπτός χαρακτήρας και μη σωστή χρήση σχολίων).

Ο λεκτικός αναλυτής εφόσον αναγνωρίσει κάποια λεκτική μονάδα την αποθηκεύει σε ένα token το οποίο και επιστρέφει.

Το token είναι μια δομή που έχουμε κατασκευάσει ( γραμμές 9-12 ) στην οποία αποθηκεύονται 3 πράγματα, η λεκτική μονάδα ( tokenString ), ο τύπος της (tokenType ) και η γραμμή (lineNo) του αρχείου στην οποία βρέθηκε.

*Μπαντουβάκης Γεώργιος, ΑΜ:4119* *26/5/2021*
*Τσαπανίδης Μιχαήλ, ΑΜ:4186*

Ο τύπος της λεκτικής μονάδας μπορεί να είναι:

- Keyword(αναγνωριστικά που ξεκινούν από γράμμα και αποτελούνται από γράμματα ή ψηφία
- Number(ψηφία και φυσικοί αριθμοί ως αριθμητικές σταθερές)
- Add operator ( + , - )
- Mul operator ( * , / )
- Delimiter ( , ή ; )
- Assignment ( := )
- Real operator ( < , > , <> , <= , >= , = )
- Group symbol ( [ , ] , ( , ) , { , } )
- Terminal ( . )

Εφόσον αναλύσαμε τον σκοπό και την λειτουργία του Λεκτικού αναλυτή, ας μεταβούμε στον κώδικα που κάνει τα παραπάνω. Από δω και στο εξής θα αναφερόμαστε στον λεκτικό αναλυτή ως lex εφόσον είναι και το όνομα της συνάρτησης που τον υλοποιεί στον κώδικα μας.

Ο lex() λοιπόν, όπως προαναφέραμε και παραπάνω βρίσκεται στις γραμμές 48-160, καλεί αρχικά την συνάρτηση getchar(), η οποία το μόνο που κάνει είναι όποτε καλείται να διαβάζει και να επιστρέφει τον αμέσως επόμενο χαρακτήρα από αυτόν που διάβασε τελευταία φορά, και αποθηκεύει την τιμή που επιστρέφει η getchar() στην μεταβλητή char. Έπειτα ακολουθεί μια σειρά από If στην οποία ο lex ελέγχει σε ποιο tokenType από τα παραπάνω ανήκει το char που έχει αποθηκευμένο προς το παρόν. Παρακάτω αναλύονται περεταίρω κάποιες γραμμές του κώδικα που χρειάζονται μια επιπλέον εξήγηση (τα νούμερα υποδηλώνουν τις γραμμές στον κώδικα):

- *54-58* : γίνεται ένας έλεγχος της τιμής της μεταβλητής temp_value, αυτό συμβαίνει ώστε να ελέγξουμε εάν σε κάποια προηγούμενη ανάγνωση ο Lex έχει καταναλώσει κάποιο γράμμα του αρχείου καθώς κάνει έλεγχο για να βρει την επόμενη λεκτική μονάδα και δεν το έχει επιστρέψει. Σε αυτήν την περίπτωση χρησιμοποιείτε πρώτα ο χαρακτήρας που είναι αποθηκευμένος στην μεταβλητή char και μετά κάνει ανάγνωση του επόμενου χαρακτήρα του αρχείου.
- *59-61 :* Σε περίπτωση που έχουμε αλλαγή γραμμής προσαυξάνουμε το lineNo του token.
- *62-63* : γίνεται έλεγχος για τον αν ο χαρακτήρας που έχουμε είναι κενό ή tab , σε αυτήν την περίπτωση ξανά καλούμε τον lex διότι αυτοί είναι 2 χαρακτήρες που δεν μας ενδιαφέρουν στην λεκτική ανάλυση.
- 64-74: Γίνεται έλεγχος σχολίων, εάν ο λεκτικός βρει το σύμβολο # αγνοεί το οτιδήποτε ακολουθεί μέχρι να ξανασυναντήσει το σύμβολο #, εάν δεν το βρει και φτάσει σε EOF τυπώνει μήνυμα λάθους και το πρόγραμμα τερματίζεται.

- 75-88: Εάν ο lex συναντήσει κάποιο γράμμα αποθηκεύει την ακολουθία που ακολουθεί είτε από γράμματα είτε από νούμερα ως keyword. Ωστόσο πριν την αποθήκευση του token κάνει έναν έλεγχο ( 82 - 84 ) εάν το αναγνωριστικό που διάβασε έχει μέγεθος μικρότερο από 30 χαρακτήρες όπως μας περιορίζουν οι κανόνες της Cimple, σε αντίθετη περίπτωση υπάρχει μήνυμα λάθους και το πρόγραμμα τερματίζει.
- 89-102: Σε αυτό το σημείο γίνεται ο έλεγχος για τους αριθμούς , εάν ο lex βρει κάποιον αριθμό πριν αποθηκεύσει το token ελέγχει ( 96 – 97 ) εάν ο αριθμός περιορίζεται αναμεσά $-2^{32}$ -1 και $2^{32}$ -1 ο οποίος είναι ένας ακόμα κανόνας της Cimple, σε αντίθετη περίπτωση υπάρχει μήνυμα λάθους και το πρόγραμμα τερματίζει ξανά.
- 104-111: Εδώ γίνεται ένας διαχωρισμός εάν βρεθεί κάποιος χαρακτήρας που ανήκει στα symbols σε «addOperator» και «mulOperator»
- 113-147: Εδώ γίνεται ξανά ένα διαχωρισμός των operators, αν βρεθεί χαρακτήρας ελέγχεται σε ποιες από τις υποκατηγορίες των operators ανήκει( delimiter, assignment, realOperator ). Σε περίπτωση που ο χαρακτήρας είναι ο « : » τότε εάν δεν ακολουθεί το σύμβολο « = » τότε το πρόγραμμα εμφανίζει μήνυμα λάθους και τερματίζει.
- 154-160: Εάν ο χαρακτήρας είναι « . » τότε γίνεται αναγνώριση ότι πρόκειται για τον τερματικό χαρακτήρα του προγράμματος(terminal), σε αντίθετη περίπτωση εάν δεν μπορέσει ο χαρακτήρας να αντιστοιχηθεί σε καμία από τις παραπάνω περιπτώσεις τότε τυπώνεται μήνυμα λάθους και το πρόγραμμα τερματίζει εφόσον πρόκειται για χαρακτήρα ο οποίος δεν είναι αναγνωρίσιμος από την γλώσσα.

*Έπειτα θα ασχοληθούμε με τον Συντακτικό αναλυτή.*

Όπως προαναφέραμε, ο σκοπός αυτής της φάσης είναι ο έλεγχος της τήρησης των γραμματικών κανόνων της γλώσσας Cimple. Αυτός ακριβώς είναι ο ρόλος του Συντακτικού Αναλυτή (Σ.Α.). Σε αντίθεση με τον lex, ο Σ.Α. δεν αποτελείται από μόνο μια συνάρτηση, αλλά από ένα σύνολο συναρτήσεων οι οποίες καλώντας τον lex εκτελούν τον απαραίτητο έλεγχο και επιστρέφουν καταλληλά μηνύματα σε περίπτωση συντακτικού λάθους.

Αναλυτικότερα οι συναρτήσεις οι οποίες αποτελούν τον Σ.Α. είναι οι εξής:

- program()
- block()
- declarations()
- varlist()
- subprograms()
- subprogram()
- formalparlist()
- formalparitem()
- statements()
- statement()
- assignStat()
- ifStat()
- elsePart()
- whileStat()
- switchcaseStat()
- incaseStat()
- returnStat()
- callStat()
- printStat()
- inputStat()
- actualparlist()
- actualparitem()
- condition()
- boolterm()
- boolfactor()
- expression()
- term()
- factor()
- idtail()
- optionalSign()
- REL_OP()
- ADD_OP()
- MUL_OP()
- INTEGER()
- ID()

Ο Σ.Α. ουσιαστικά ξεκινάει όταν κληθεί η συνάρτηση program(). Η συνάρτηση αυτή ελέγχει αν το πρόγραμμα ξεκινάει με την λέξη «program», καλεί την συνάρτηση block() που καθορίζει την βασική δομή του προγράμματος και ελέγχει αν το πρόγραμμα τερματίζει με το τερματικό σύμβολο τελειά «.». Σε αντίθετη περίπτωση τυπώνει τα ανάλογα μηνύματα και τερματίζει το πρόγραμμα.

Η συνάρτηση block() όπως αναφέραμε καθορίζει την βασική δομή του προγράμματος, η οποία είναι:

1. Declarations
2. Subprograms
3. Statements

Ας δούμε αναλυτικά τι συμβαίνει στα τρία αυτά βασικά μέρη του προγράμματος.

1. <u>Declarations</u>
   Σε αυτό το μέρος, γίνεται ο έλεγχος για την ορθή σύνταξη των δηλώσεων των μεταβλητών. Για να συμβεί αυτό, καλείται η συνάρτηση declarations() η οποία ελέγχει αν όλες οι δηλώσεις ξεκινάνε με την λέξη «declare» και χωρίζονται μεταξύ τους με κόμμα «,». Τέλος, ελέγχει αν τελειώνουν με το σύμβολο «;»

2. <u>Subprograms</u>
   Σε αυτό το μέρος γίνεται ο έλεγχος για την ορθή σύνταξη των υποπρογραμμάτων που ορίζονται μέσα στο κυρίως πρόγραμμα. Αναλυτικότερα, καλείται η συνάρτηση subprograms() η οποία ελέγχει αν υπάρχουν υποπρογράμματα και σε περίπτωση που υπάρχουν καλεί την συνάρτηση subprogram(). Η συνάρτηση αυτή αρχικά ελέγχει αν οι δηλώσεις των υποπρογραμμάτων ξεκινάνε με τις λέξεις-κλειδιά « function» ή «procedure». Σε περίπτωση λάθους, τυπώνονται τα καταλληλά μηνύματα και τερματίζει το πρόγραμμα. Εάν δεν υπάρχουν συντακτικά λάθη στην

δήλωση των υποπρογραμμάτων, καλούνται οι συναρτήσεις formalparlist() και formalparitem() οι οποίες ελέγχουν για τις μεταβλητές εισόδου και εξόδου (πχ. in x, inout y) εάν ορίζονται σωστά με την χρήση των λέξεων-κλειδιών «in» και «inout». Τέλος, καλείται ξανά η συνάρτηση block() διότι τα υποπρογράμματα που ορίζονται εντός του κυρίου προγράμματος έχουν την ίδια δομή.

3. Statements

   Η συνάρτηση αυτή ( statements() ) ελέγχει αρχικά αν υπάρχουν ένα ή περισσότερα statements. Στην περίπτωση ενός statement, πρέπει κατά την λήξη της δήλωσης του να υπάρχει το σύμβολο «;», ενώ σε περίπτωση που υπάρχουν από δυο και πάνω statements, πρέπει αυτά να δηλώνονται μέσα σε αγκύλες και το καθένα να λήγει με το σύμβολο «;». Εφόσον εξασφαλιστούν όλες αυτές οι προϋποθέσεις, καλεί την συνάρτηση statement() η οποία αναλόγως τον τύπο του statement καλεί την αντίστοιχη συνάρτηση για να γίνει ο συντακτικός έλεγχος της δήλωσης του statement. Τα αποδεκτά statements είναι οι δομές:

   - if – ifStat()
   - else – elsepart()
   - while – whileStat()
   - switchcase – switchcaseStat()
   - forcase – forcaseStat()
   - incase – incaseStat()
   - call – callStat()
   - return – returnStat()
   - input – inputStat()
   - print – printStat()
   - assign – assignStat()

   Θα αναλύσουμε επιγραμματικά την λειτουργία των παραπάνω συναρτήσεων.

   - *ifStat():*

     Αφού γίνει έλεγχος για την ύπαρξη της λέξης κλειδί if , καλείται η συνάρτηση condition που θα αναλυθεί παρακάτω για να γίνει έλεγχος για την σωστή σύνταξη της συνθήκης που θέτει η If. Έπειτα ξανακαλείται η συνάρτηση statements() και γίνεται μια αναδρομή προς τα πάνω γιατί μέσα στην If ορίζονται μόνο statements. Τέλος καλείται η συνάρτηση elsepart() η οποία το μόνο που κάνει είναι να ελέγξει εάν μετά το if ακολουθεί κάποιο else, σε αυτήν την περίπτωση ξανακαλεί την συνάρτηση statements(), αλλιώς δεν κάνει τίποτα.

- *whileStat():*
  Εδώ δεν υπάρχει κάτι αξιόλογο να αναλυθεί, λειτουργεί με τον ίδιο ακριβώς τρόπο με την ifStat() με την διαφορά ότι η λέξη-κλειδί τώρα είναι το while και στο τέλος δεν καλείται η elsepart().
- *switchcaseStat() - forcaseStat() - incaseStat():*
  Αυτές οι συναρτήσεις ανήκουν στην ίδια οικογένεια εφόσον ο κώδικας για τον έλεγχο τους είναι ίδιος με την μόνη διαφορά η κάθε μια έχει διαφορετική λέξη-κλειδί που ελέγχεται πρώτα. Πιο αναλυτικά ελέγχεται η λέξη-κλειδί για την κάθε μια (όπως υποδηλώνεται από το όνομα της), και έπειτα γίνεται έλεγχος εάν υπάρχουν οι λέξεις κλειδιά «case» και ακολουθούν τα conditions μέσα στα cases. Αφού δηλωθούν τα conditions όπως εξηγήθηκε και στην whileStat και την ifStat καλείται η συνάρτηση statements().Τέλος γίνεται έλεγχος για το αν υπάρχει η λέξη «default». Σε περίπτωση που δεν υπάρχει, πρέπει να υπάρχει το σύμβολο «;;».Εάν υπάρχει όμως η λέξη default ξανακαλείται η συνάρτηση condition διότι μετά το default ακολουθεί ένα condition σε περίπτωση που δεν γίνει match σε κάποιο case και μετά ξανά γίνεται έλεγχος εάν η default λήγει με το σύμβολο «;»
- *callStat() - returnStat() - inputStat() - printStat():*
  Αυτή είναι ακόμα μια οικογένεια συναρτήσεων που ακολουθούν τον ίδιο τρόπο ελέγχου. Δηλαδή αρχικά ελέγχεται η λέξη κλειδί με την οποία ξεκινάνε(όπως και οι προηγούμενες και έπειτα καλείται η συνάρτηση expression για να ελεγχθεί η έκφραση που υπάρχει αναμεσά στις παρενθέσεις που ακολουθούν την κάθε λέξη κλειδί. Εδώ να σημειώσουμε ότι επειδή κάθε συνάρτηση μπορεί να έχει τον ίδιο κώδικα ελέγχου αλλά επιτελεί διαφορετικές λειτουργίες , η συνάρτηση inputStat() δεν καλεί την expression αλλά την ID() ώστε να ελέγξει εάν η μεταβλητή εισόδου είναι αλφαριθμητικό , και η συνάρτηση callStat() δεν καλεί ούτε αυτή την expression(), αλλά την actualparlist(), εφόσον καλεί υποπρογράμματα του κυρίως προγράμματος ώστε να ελέγξει αν καλούνται σωστά.
- *assignStat():*
  Τέλος η συνάρτηση αυτή το μόνο που κάνει είναι , εάν συναντήσει σύμβολο ανάθεσης ( := ) να ελέγχει αν το αριστερό μέρος πρόκειται για αλφαριθμητικό (ώστε να έχει νόημα η ανάθεση) και έπειτα καλεί την expression() για να ελέγξει εάν η παράσταση – αριθμός που αναθέτουμε στην μεταβλητή είναι συντακτικά σωστή.

Στην ανάλυση των παραπάνω συναρτήσεων αναφερθήκαμε σε κάποιες συναρτήσεις οι οποίες εν μέρη έχουμε πει ότι είναι τμήμα του συντακτικού αναλυτή, αλλά δεν τις έχουμε αναλύσει ακόμα. Θα τις χωρίσουμε σε 2 κατηγορίες , η οικογένεια για τα condition( condition(), boolterm(), boolfactor() ) και η οικογένεια για τα expression( expression(), term(), factor(), idtail()).

*Οικογένεια condition*

Η συνάρτηση condition() καλείται από συναρτήσεις όπως για παράδειγμα η if για να ελέγξει κάποια συνθήκη. Πρόκειται για μια ακολουθία κλήσεων συναρτήσεων, η condition καλεί την boolterm() η οποία καλεί την boolfactor(). Η boolfactor ελέγχει το είδος της συνθήκης( άρνηση, απλή συνθήκη, περιέχεται mul_operator) και επιστρέφει στην booterm(), h boolterm ελέγχει αν υπάρχουν και άλλη συνθήκη σε σύζευξη(γι' αυτό η if Που υπάρχει στον κώδικα ελέγχει για την ύπαρξη του «and») και αν υπάρχει ξανακαλεί τον εαυτό της σε αντίθετη περίπτωση επιστρέφει στην condition η οποία ελέγχει αν υπάρχει και άλλη συνθήκη αλλά αυτήν την φορά με διάζευξη (εδώ η if ελέγχει για «or»).

*Οικογένεια expression*

Η συνάρτηση expression() όπως λέει και το όνομα της ελέγχει για εκφράσεις, δηλαδή για παραστάσεις. Εδώ συμβαίνει κάτι παρόμοιο με παραπάνω, δηλαδή η expression() καλεί την term(), η term() την factor και η factor() την idtail(). Η factor αρχικά ελέγχει για το αν πρόκειται για ένα ή για πολλά expression , αν πρόκειται για ένα καλεί ξανά την συνάρτηση expression, αν πρόκειται για πολλά αφού ελέγξει ότι αν πρόκειται για υποπρογράμματα που έχει οριστεί προηγουμένως, ελέγχει αν καλείται σωστά, σε αντίθετη περίπτωση καλεί την idtail() η οποία με την χρήση της actualparlist() διαχωρίζει τα inputs. Αφού οι συναρτήσεις τελειώσουν επιστρέφουν το αποτέλεσμα στην term η οποία ελέγχει αν υπάρχει ελέγχει αν η παράσταση συνεχίζει με κάποιον Mul_operator, αν βρει Mul_operator ξανακαλεί τον εαυτό της για να ξαναγίνει η ίδια διαδικασία, σε αντίθετη περίπτωση επιστρέφει στην expression η οποία κάνει έλεγχο για ύπαρξη Add_operator, αν υπάρχει ξανακαλεί τον εαυτό της, εάν όχι τερματίζει και επιστρέφει στην συνάρτηση από την οποία κλήθηκε.

Κλείνοντας, θα αναφερθούμε και στην τελευταία ομάδα συναρτήσεων που ανήκουν στον συντακτικό τις οποίες αναφέραμε προηγουμένως. Αναλυτικότερα:

- *REL_OP:* Ελέγχει το διαθέσιμο token εκείνη την στιγμή που καλείται αν είναι ένα από: =, <=, >= , > ,< , <>
- *ADD_OP:* Ελέγχει το διαθέσιμο token εκείνη την στιγμή που καλείται αν είναι ένα από: + , -
- *MUL_OP:* Ελέγχει το διαθέσιμο token εκείνη την στιγμή που καλείται αν είναι ένα από: * , /
- *INTEGER:* Ελέγχει το διαθέσιμο token εκείνη την στιγμή που καλείται αν έχει tokenType=number.

- *ID:* Ελέγχει το διαθέσιμο token εκείνη την στιγμή που καλείται έχει tokenType=keyword.
- *OptionalSign:* καλεί την ADD_OP() για να ελέγξει αν υπάρχει το αντίστοιχο σύμβολο.

Σε περίπτωση που δεν τηρούνται οι παραπάνω συνθήκες αυτών των συναρτήσεων, τυπώνεται μήνυμα λάθους και το πρόγραμμα τερματίζει.

## *2. Ενδιάμεσος κώδικας*

Όπως είναι προφανές και από το όνομα, αυτό είναι το ενδιάμεσο στάδιο της μεταγλώττισης, όπου αφού το πρόγραμμα Cimple περάσει την λεκτική και συντακτική ανάλυση , δημιουργείται ένα αρχείο .int στο οποίο μετατρέπουμε τις εντολές που βρίσκονται στο πρόγραμμα Cimple σε τετράδες οι οποίες έχουν την μορφή:

- Ένας τελεστής
- Τρία τελούμενα

Π.χ. +,a,b,t_1 όπου αυτό συνεπάγεται με t_1=a+b

Σε περίπτωση που ο κώδικας μας δεν περιέχει υποπρογράμματα η συναρτήσεις όπως (forcase, incase, switchcase) τότε αυτόματα παράγεται ο κώδικας σε γλώσσα C που δεν είναι τίποτα παρά η μετατροπή του αρχείου .int σε αρχείο .c με τις κατάλληλες εντολές που θα δούμε και παρακάτω. Εάν το πρόγραμμα περιέχει όμως κάτι από τα παραπάνω τότε για την παραγωγή του κώδικα C είναι απαραίτητη η δημιουργία του πίνακα συμβόλων που θα δούμε στην επόμενη φάση.

Στην φάση του ενδιάμεσου κώδικα οι συναρτήσεις που θα χρειαστούμε είναι οι εξής:

- *nextquad*(): Η συνάρτηση αυτή οπότε καλείται δίνει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.
- *genquad*(): Δημιουργεί την επόμενη τετράδα, παίρνει 4 ορίσματα ( την τετράδα μας ) και τα προσθέτει σε μια λίστα, την programList η οποία είναι μια λίστα από τετράδες την οποία την τυπώνουμε στο τέλος στο αρχείο .int
- *newtemp*(): Δημιουργεί και επιστρέφει μια καινούρια προσωρινή μεταβλητή της μορφής T_1,T_2,T_3…
- *emptylist*(): Δημιουργεί μια κενή λίστα τετράδων
- *makelist*(x): Δημιουργεί μια λίστα τετράδων που περιέχει μόνο το χ
- *merge*(L1,L2): Συγχωνεύει τις λίστες L1 και L2
- *backpatch*(list, z) : Η list είναι μια λίστα από τετράδες όπου λείπει το τελευταίο τελούμενο. Η μέθοδος αυτή επισκέπτεται μια μια αυτές τις τετράδες και συμπληρώνει το z όπου είναι εφικτό.

Τις συναρτήσεις αυτές τις καλούμε μέσα στον συντακτικό αναλυτή, δηλαδή στις συναρτήσεις που χρησιμοποιεί ο συντακτικός αναλυτής, ώστε μόλις ένα token ελεγχθεί και επιβεβαιωθεί η σωστή χρήση του παράλληλα γράφεται και η τετράδα στην οποία ανήκει στην programlist.

Στην συνάρτηση program() (170-171) δημιουργείται η τετράδα που δηλώνει το τέλος του προγράμματος όπως και στην συνάρτηση block() (184 ) δημιουργείται η τετράδα που δηλώνει την έναρξη της main αλλά και οποιουδήποτε υποπρογράμματος ορίζεται μέσα στο κυρίως πρόγραμμα.

Στην συνάρτηση formalparitem() ξαναγίνεται κλήση της genquad για να δηλωθεί αν τα ορίσματα του υποπρογραμματος που κάλεσε την formalparitem() είναι in ή inout ( για In περνιέται ως CV ενώ για inout ως REF).

Σε όλο τον υπόλοιπο κώδικα η λογική με την οποία φτιάχνονται οι τετράδες είναι, ότι γράφονται 2 τετράδες η μια με την αληθή έκβαση της συνθήκης η οποία ελέγχεται και μια με την ψευδή, και αμέσως μετά γράφονται δυο jump με τον αριθμό της γραμμής στην οποία πρέπει να συνεχίσει το πρόγραμμα για να ακολουθήσει την κάθε περίπτωση. Αυτό αφορά την δομή όλων των συναρτήσεων που κάνουν κάποιο έλεγχο σε κάποιο condition( forcase, switchcase ,incase , if, while). Αφού τελειώσουμε με την δομή, όλες οι συναρτήσεις έχουμε πει ότι καλούν τις συναρτήσεις condition() ή expression(). Μέσα σε αυτές ξανακαλούμε την genquad για να υλοποιήσουμε τις τετράδες των συνθηκών και παραστάσεων.

*Εδώ να σημειωθεί ότι τα jump που κάνουν κλήση της backpatch δεν υλοποιήθηκαν* διότι, όταν φτιάχτηκαν λειτουργούσαν λάθος με αποτέλεσμα να χαλάνε και το υπόλοιπο αποτέλεσμα, έτσι αποφασίσαμε να τα αφαιρέσουμε και να έχουμε ένα καλύτερο αποτέλεσμα αλλά λίγο πιο λειψό.

Τέλος στον μεταγλωττιστή μας υπάρχει και η συνάρτηση create_c η οποία με δεδομένο ότι είναι συμπληρωμένη η λίστα programlist η οποία περιέχει τις τετράδες, τις τυπώνει σε ένα αρχειο .c σε κατάλληλο format ώστε να είναι εκτελέσιμο από την γλώσσα C. Κάτω από την κλήση αυτής της συνάρτησης (γραμμές 917-1026) υπάρχει μια απλή for η οποία τυπώνει την λίστα programlist στο αρχείο .int χωρίς βέβαια να ακολουθεί κάποιο συγκεκριμένο format

*Μπαντουβάκης Γεώργιος, ΑΜ:4119* *26/5/2021*
*Τσαπανίδης Μιχαήλ, ΑΜ:4186*

## 3. ***Πίνακας Συμβόλων και Τελικός κώδικας***

*Αρχικά να σημειώσουμε ότι δεν έχουμε υλοποιήσει τον πίνακα συμβόλων*

Για τον λόγο αυτό θα μεταβούμε κατευθείαν στον τελικό κώδικα και στην επεξήγησή του.

Σε αυτήν την φάση, η οποία είναι και η τελική της μεταγλώττισης σκοπός ήταν κάθε εντολή του ενδιάμεσου κώδικα να την μετατρέψουμε σε εντολή τελικού κώδικα, δηλαδή σε assembly για τον επεξεργαστή Mips. Οι κύριες ενέργειες σε αυτήν την φάση ήταν η απεικόνιση των μεταβλητών στην στοίβα, το πέρασμα των παραμέτρων και η κλήση των συναρτήσεων(929-1026).

Σε αυτό το μέρος η διαδικασία είναι πιο απλή από τα προηγούμενα μέρη εφόσον αυτό που γίνεται είναι να φορτώσουμε το αρχείο int και να αντιστοιχίσουμε μια μια τις εντολές σε γλώσσα assembly. Η διαδικασία που ακολουθήθηκε είναι αρκετά προφανής κοιτάζοντας μόνο τον κώδικα εφόσον αποτελείτε από αλλεπάλληλα if τα οποία ανάλογα την εντολή εάν πρόκειται για jump την μετατρέπουν σε «b και την γραμμή που θα γίνει το jump» και αν πρόκειται για πράξεις δεν μεριμνούμε πρώτα να φορτώσουμε τις μεταβλητές στην στοίβα και έπειτα να κάνουμε οποιαδήποτε πράξη μας επιτρέπει η assembly.

Σε αυτό το σημείο λόγω της απουσίας του πίνακα συμβόλων στην μετατροπή του κώδικα για κλήση συνάρτησης δεν μπορούσαμε να ψάξουμε σε τι βάθος φωλιάσματος βρίσκεται η καλούσα και η κληθείσα συνάρτηση οπότε στην λύση μας θεωρήσαμε ότι όλες βρίσκονται στο ίδιο επίπεδο ώστε να μπορεί να παραχθεί κάποιο αποτέλεσμα

*Παρατηρήσεις:*

Σχεδόν όλα τα προβλήματα που παρουσιάζει και διαπιστώσαμε έχουν ήδη αναφερθεί. Κάτι που δεν αναφέραμε είναι ότι ο συντακτικός αναλυτής ενώ στις περισσότερες περιπτώσεις λειτουργεί σωστά, όπως φαίνεται και από τα test τα οποία παραδίδαμε σε κάθε φάση, παρατηρήσαμε ότι όταν γινόντουσαν ορισμοί πολλών υποπρογραμμάτων παρουσιάζονταν σφάλματα τα οποία δεν θα έπρεπε, όπως και όταν σε συναρτήσεις όπως η print και η return χρησιμοποιούσαμε πολλές παρενθέσεις, μπορεί μερικές φορές το πρόγραμμα να κολλούσε και να έβγαζε ανύπαρκτα μηνύματα λάθους.