ΟΜΑΔΑ:

Δαφνάκης Γεώργιος

Τσαπικούνη Γεωργία

Ανάλυση μεταφραστή γλώσσας Cimple δομημένος σε Python

Η αναφορά χωρίζεται σε τρείς φάσεις.Η πρώτη φάση αναφέρεται σε:

- Λεκτικό αναλυτή
- Συντακτικού αναλυτή

Η δεύτερη φάση διακρίνεται σε:

- Ενδίαμεσο κώδικα
- Πίνακα συμβόλων

Και τέλος η τρίτη φάση αναφέρεται σε:

- Τελικό κώδικα

Πρώτη Φάση

Λεκτικός αναλυτής

Στη συγκεκριμένη φάση δημιουργούμε τον λεκτικό αναλυτή.Ο λεκτικός αναλυτής υλοποιείται από μια συνάρτηση η οποία καλείται από τον συντακτικό αναλυτή και κάθε φορά που καλείται επιστρέφει την επόμενη διαθέσιμη στην είσοδο λεκτική μονάδα.

Προτού ορίσουμε την κύρια συνάρτηση def lex() αναθέτουμε τιμές σε κάθε πιθανό token που θέλουμε να διαβάσει και να αναγνωρίσει ο λεκτικός αναλυτής. Για παράδειγμα, για τον πολλαπλασιαστικό αριθμητικό τελεστή

(*) ορίσαμε μια σταθερά tokens_mult=105. Αντίστοιχα για όλες τις κατηγορίες (number,keyword,id,addOperator,mulOperator,relOperator,assignment,delimiter, groupSymbol) παίρνουμε τις αντίστοιχες υποκατηγορίες και τις αναθέτουμε στα αντίστοιχα tokens. Έπειτα, δημιουργούμε έναν πίνακα καταστάσεων ο οποίος καλύπτει όλες τις πιθανές περιπτώσεις του αυτόματου. Δηλαδή, ο πίνακας αποτελείται από γραμμές οι οποίες αναπαριστούν τις έξι πιθανές καταστάσεις του αυτόματου και από στήλες οι οποίες αναπαριστούν την αντίστοιχη εκχώρηση με βάση το τρέχον token (σύμφωνα πάντοτε με το αυτόματο που μας έχει δοθεί).

Στη συνέχεια, υλοποιούμε την συνάρτηση lex(). Η κύρια λειτουργία της είναι να διαβάζει ένα αρχείο Cimple (χαρακτήρα προς χαρακτήρα),να αντιστοιχεί το κάθε σύμβολο στο αντίστοιχο token, και με βάση το αυτόματο να καταλήγει είται σε ERROR είτε σε επιτυχής καταχώρηση.

Αρχικά, ορίζουμε έναν πίνακα Arraylex[].Χρησιμοποιούμε μια while-loop η οποία λειτουργεί όσο βρισκόμαστε σε μια από τις έξι καταστάσεις του αυτόματου.

Αντιστοιχούμε κάθε χαρακτήρα του αρχείου σε ένα token ,μεταβάλλοντας έτσι το current_state. Τέλος, τίθεται σε λειτουργία το αυτόματο . Επιπρόσθετα, το αυτόματο είναι ικανό να αναγνωρίσει ενδεχόμενα λάθη. Συγκεκριμένα:

- **UNEXPECTED_INPUT_ERROR:** Τυπώνεται στην οθόνη εάν το πρόγραμμα διαβάσει κάποιον χαρακτήρα ο οποίος δεν ανήκει στο letter_list ή στο digit_list.
- **UNEXPECTED_LETTER_ERROR:** Τυπώνεται στην οθόνη εάν το αυτόματο βρίσκεται στην κατάσταση state_dig και ο επόμενος χαρακτήρας που θα διαβάσει είναι γράμμα.
- **ASSIGNEMENT_ERROR:** Τυπώνεται στην οθόνη εάν το αυτόματο βρίσκεται στην κατάσταση state_asgn και το πρόγραμμα διαβάσει οτιδήποτε άλλο πέραν από τον χαρακτήρα “=”.
- **EOF_ERROR:** Τυπώνεται στην οθόνη εάν το αυτόματο βρίσκεται στην κατάσταση state_rem και το πρόγραμμα δεν ξαναδιαβάσει τον χαρακτήρα “#”.
- **OUT_OF_BOUNDS_ERROR:** Τυπώνεται στην οθόνη εάν το πρόγραμμα διαβάσει ως είσοδο ένα string παραπάνω από 30 χαρακτήρες ή έναν αριθμό εκτός του διαστήματος $[-(2^{32}-1),(2^{32}-1)]$.

Ολοκληρώνοντας, με την παραπάνω διαδικασία που αναφέραμε καταφέραμε να γεμίσουμε τον πίνακα Arraylex.Πιο αναλυτικά ,κάθε γραμμή του αποτελείται από το current_state, το tokens_string και το linecounter του κάθε χαρακτήρα.

Συντακτικός αναλυτής

Στη συνέχεια ο συντακτικός αναλυτής καλεί την lex() και ελέγχει εάν το πηγαίο πρόγραμμα ανήκει ή όχι στη γλώσσα Cimple.

- **def program()**: Το πρόγραμμα ξεκινάει με τη λέξη program. Σε κάθε άλλη περίπτωση εμφανίζεται στην οθόνη μήνυμα σφάλματος. Έπειτα καλείται ο λεκτικός αναλυτής που περιμένει ένα tokens_id. Σε περίπτωση σφάλματος τυπώνεται στην οθόνη μήνυμα σφάλματος. Αν διαβάσει το tokens_id καλεί τον lex() και έπειτα την block(). Τέλος, για να ολοκληρωθεί το πρόγραμμα ο lex() αναμένει ένα tokens_fullStop. Σε κάθε άλλη περίπτωση εμφανίζεται μήνυμα λάθους .

- **def block()**: Το block αυτό αποτελείται από declarations, subprograms, blockstatements . Εάν η λεκτική μονάδα είναι tokens_leftBlock τότε καλείται ο lex() και αμέσως δημιουργείται ένα scope στον πίνακα συμβόλων που θα αναλύσουμε παρακάτω. Εάν δε βρισκόμαστε στο κύριο πρόγραμμα καλούμε την add_parameters() και συμπληρώνουμε τις αντίστοιχες παραμέτρους που διαβάζουμε από τον πηγαίο κώδικα. Στη συνέχεια καλούνται οι συναρτήσεις declarations() , subprograms(), η genquad() που θα εξηγήσουμε στον ενδιάμεσο κώδικα και η blockstatements(). Σε περίπτωση που η λεκτική μονάδα είναι διαφορετική από το tokens_left τυπώνεται μήνυμα σφάλματος .Έπειτα, το πρόγραμμα αναμένει το tokens_rightBlock και έπειτα καλείται η lex(). Εάν βρισκόματσε στο κύριο πρόγραμμα καλούμε την genquad αλλιώς υπολογίζεται το Framelength. Τέλος, καλούνται οι συναρτήσεις που θα επεξηγήσουμε στη συνέχεια SymbolBoardPrint() , lastCheck() και deleteScope() .

- **def declarations()**: Η συνάρτηση αυτή μπαίνει σε μια while-loop. Όσο διαβάζει tokens_declare καλεί τον λεκτικό αναλυτή , τη συνάρτηση varlist() και μετέπειτα εάν η λεκτική μονάδα είναι το tokens_Qmark καλείται πάλι ο

lex() . Σε περίπτωση σφάλματος του tokens_Qmark , τυπώνεται μήνυμα λάθους.

- **def varlist()**: Εάν η λεκτική μονάδα είναι tokens_id καλείται ο λεκτικός αναλυτής και έπειτα μπαίνει σε μια while-loop. Οσο διαβάζει tokens_coma καλείται ο lex().Εάν η λεκτική μονάδα είναι tokens_id καλείται ο λεκτικός αναλυτής αλλιώς εμφανίζεται στην οθόνη μήνυμα λάθους.

- **def subrograms()**: Στη συνάρτηση αυτή όσο η λεκτική μονάδα αντιστοιχεί στο tokens_function() καλείται η συνάρτηση subprogram() (η οποία θα αναλυθεί παρακάτω)

- **def subprogram():** Στη συνάρτηση αυτή εάν η λεκτική μονάδα αντιστοιχεί σε tokens_function,τότε καλείται ο λεκτικός αναλυτής.Έπειτα εξετάζεται η περίπτωση εάν η λεκτική μονάδα αντιστοιχεί σε tokens_id και εάν ναι,ενημερώνεται ο πίνακας συμβόλων (ο οποίος θα εξηγηθεί παρακάτω) και καλείται ο λεκτικός αναλυτής. Στη περίπτωση που η προηγούμενη συνθήκη είναι αληθής , το πρόγραμμα εξετάζει εάν η λεκτική μονάδα αντιστοιχεί σε tokens_leftParenthesis.Εφόσον ισχύει,το πρόγραμμα θα εξετάσει και την ύπαρξη της δεξιάς παρένθεσης (tokens_rightParenthesis) και εάν είναι αληθής καλείται η lex().Σε άλλη περίπτωση το πρόγραμμα θα τυπώσει μύνημα λάθους (είτε εάν μετά απο αριστερή παρένθεση δεν διαβάσει δεξιά παρένθεση είτε εάν η λεκτική μονάδα είναι tokens_function και η επόμενη λεκτική μονάδα δεν είναι αριστερή παρένθεση). Έπειτα το πρόγραμμα εξετάζει εάν η λεκτική μονάδα ειναι tokens_procedure, εάν ναι τότε καλείται για άλλη μια φορά ο λεκτικός αναλυτής και ακολουθεί πάλι ο προηγούμενος έλεγχος σχετικά με τις παρενθέσεις.

- **def formalparlist():** Στη συνάρτηση αυτή καλείται η formalparitem() (η οποία θα αναλυθεί παρακάτω) και για όσο η λεκτική μονάδα αντιστοιχεί σε

tokens_coma εξακολουθεί να καλείται ο λεκτικός αναλυτής και η formalparitem().

- **def formalparitem():** Στη συνάρτηση αυτη εάν η λεκτική μονάδα αντιστοιχεί σε tokens_in είτε σε tokens_inout τότε αντίστοιχα καλείται η lex() και εάν όχι τότε το πρόγραμμα τυπώνει αντίστοιχα λάθη. Σε κάθε περίπτωση πραγματοποιείται ένας εσωτερικός έλεγχος για το έαν η λεκτική μονάδα αντιστοιχεί σε tokens_id και εάν είναι αληθής τότε ενημερώνεται ο πίνακας συμβόλων.

- **def statements():** Στη συνάρτηση αυτή εάν η λεκτική μονάδα αντιστοιχεί σε tokens_leftBlock καλείται η lex() και η statement() (η οποία θα αναλυθεί αργότερα) και για όσον η λεκτική μονάδα είναι ισοδύναμη με tokens_Qmark καλείται συνεχώς ο λεκτικός αναλυτής και η statement().Επίσης, το πρόγραμμα αναμένει να αντιστοιχίσει τη λεκτική μονάδα με tokens_rightBlock και να καλέσει την lex(), σε αντίθετη περίπτωση τυπώνεται μύνημα λάθους.Εάν η αρχική συνθήκη δεν είναι αληθής,τότε καλείται η statement(),εξετάζεται εαν η λεκτική μονάδα ισούται με tokens_Qmark αλλίως τυπώνεται μύνημα λάθους.

- **def blockstatements():** Στη συνάρτηση αυτή αρχικά καλείται η statement() και για όσον η λεκτική μονάδα αντιστοιχεί σε tokens_Qmark τότε καλείται η lex() και η statement()

- **def statement():** Στη συνάρτηση αυτή εξετάζεται εάν η λεκτική μονάδα είναι ισοδύναμημε ένα απο τα παρακάτω:

tokens_id,tokens_if,tokens_while,_tokens_switchcase,_tokens_forcase,_tokens_incase,tokens_call,tokens_return,tokens_input,tokens_print και αντίστοιχα θα καλέσει μία απο τις assignStat(), ifStat(), whileStat(), switchcaseStat(),forcaseStat(), incaseStat(), callStat(), returnStat(), inputStat(),printStat() οι οποίες θα εξηγηθούν παρακάτω.

- **def assignStat():** Στη συνάρτηση αυτή εξετάζεται εάν η λεκτική μονάδα αρχικά είναι ισοδύναμη με tokens_id.Στη περίπτωση που είναι τότε καλείται η lex(), γίνεται έλεγχος για το εάν αντιστοιχεί σε tokens_asgn και σε αυτή τη περίπτωση καλείται ο λεκτικός αναλυτής και ενημερώνεται ο πίνακας συμβόλων . Σε οποιοδήποτε έλεγχο που δεν είναι αληθής τυπώνεται το αντίστοιχο μύνημα λάθους.

- **def ifStat():** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_if.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis και για tokens_rightParenthesis . Εάν και οι δύο έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις condition(),statements(),backpatch(),makelist(),genquad(),elsepart() (οι οποίες θα εξηγηθούν αργότερα) . Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def elsepart():** Στη συνάρτηση αυτή εξετάζεται εάν η λεκτική μονάδα είναι ίση με tokens_else.Εάν ισχύει αυτή η συνθήκη τότε καλέιται η lex() και η statements().

- **def whileStat():**Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_while.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis και για tokens_rightParenthesis . Εάν και οι δύο έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις condition(),statements(),backpatch(),makelist(),genquad(). Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def switchcaseStat:** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_switchcase.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis και για tokens_rightParenthesis για όσον η λεκτική μονάδα είναι ίση με tokens_case. Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις condition(),statements(),backpatch(),makelist(),genquad() και γίνεται ένας ακόμα έλεγχος για το έαν η λεκτική μονάδα είναι ίσοδύναμη με tokens_default ώστε να καλεστεί η lex(),statements(), και backpatch() για άλλη μια φορά. Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def forcaseStat():** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_forcase.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis

και για tokens_rightParenthesis για όσον η λεκτική μονάδα είναι ίση με tokens_case. Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις condition(),statements(),backpatch(),makelist(),genquad() και γίνεται ένας ακόμα έλεγχος για το έαν η λεκτική μονάδα είναι ίσοδύναμη με tokens_default ώστε να καλεστεί η lex(),statements(), και backpatch() για άλλη μια φορά. Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def incaseStat:** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_incase.Στη περίπτωση που είναι τότε καλείται η lex(), και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis και για tokens_rightParenthesis για όσον η λεκτική μονάδα είναι ίση με tokens_case. Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις condition(),statements(),backpatch(),makelist(),genquad() . Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def returnStat():**Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_return.Στη περίπτωση που είναι τότε καλείται η lex(), και γίνεται ο αντίστοιχος έλεγχος για

tokens_leftParenthesis και για tokens_rightParenthesis . Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλείται η συνάρτηση ,genquad() . Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def callStat():** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_call.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis , tokens_rightParenthesis και για tokens_id.Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις actualparlist(),genquad(). Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def printStat():**Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_print.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis και για tokens_rightParenthesis .Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις expression(),genquad(). Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def inputStat():** Στη συνάρτηση αυτή εξετάζεται αρχικά έαν η λεκτική μονάδα είναι ισοδύναμη με tokens_input.Στη περίπτωση που είναι τότε καλείται η lex() και γίνεται ο αντίστοιχος έλεγχος για tokens_leftParenthesis , tokens_rightParenthesis και για tokens_id .Εάν όλοι οι έλεγχοι είναι αληθείς τότε καλείται ο λεκτικός αναλυτής, καλούνται οι συναρτήσεις expression(),genquad(). Εάν οι συνθήκες είναι ψευδείς τότε τυπώνονται τα αντίστοιχα μυνήματα λάθους.

- **def actualparlist():**Στη συνάρτηση αυτή αρχικά καλείται η actualparitem() και έπειτα για όσο η λεκτική μονάδα είναι ισοδύναμη με tokens_coma τότε καλείται ο λεκτικός αναλυτής και η actualparitem().

- **def actualparitem():** Στη συνάρτηση αυτή εξετάζουμε τα εξής ενδεχόμενα, είτε εάν η λεκτική μονάδα ισούται με tokens_in είτε με tokens_inout.Στη πρώτη περίπτωση καλείται η lex(),expression() και η genquad().Στη δεύτερη περίπτωση καλείται η lex() και ακολουθεί ένας ακόμα έλεγχος για το εάν η λεκτική μονάδα αντιστοιχεί σε tokens_id.Εάν είναι αληθής τότε καλείται η lex() και η genquad().Σε οποιοδήποτε έλεγχο που βρεθεί αναληθής τότε το πρόγραμμα τυπώνει το αντίστοιχο μύνημα λάθους.

- **def condition():**Στη συνάρτηση αυτή αρχικά ορίζουμε τους Btrue[] και Bfalse[] και για όσον η λεκτική μονάδα ισοδυναμεί με tokens_or τότε καλείται η lex(),backpatch() και γεμίζουν οι δύο πίνακες με στοιχεία από τον Q2list[].

- **def boolterm():**Στη συνάρτηση αυτή αρχικά ορίζουμε τους Qtrue[] και Qfalse[] , καλείται η boolfactor() και έπειτα για όσον η λεκτική μονάδα

αντιστοιχεί σε tokens_and καλείται η lex(),backpatch() ,boolfactor() και γεμίζουν οι Qfalse[] και Qtrue[].

- **def boolfactor():**Στη συνάρτηση αυτή αρχικά ορίζουμε τους Rtrue[] και Rfalse[].Εάν η λεκτική μονάδα ισοδυναμεί με tokens_not τοτε καλείται ο λεκτικός αναλυτής και ακολουθεί ο έλεγχος για την ύπαρξη των tokens_leftBracket και tokens_rightBracket.Εάν οι έλεγχοι είναι αληθείς τότε καλούνται οι lex(),condition() και προστίθονται στοιχεία στις Rtrue και Rfalse.Εάν η πρώτη συνθήκη είναι ψευδείς τότε ακολουθεί νέος έλεγχος για τα tokens_leftBracket και tokens_rightBracket.Εάν δεν ισχύει καμία συνθήκη τότε ενημερώνεται ο πίνακας συμβόλων, καλούνται οι expression(),REL_OP(),makelist(),genquad().Σε οποιοδήποτε εμφολευμένο έλεγχο που βρεθεί αναληθής τότε το πρόγραμμα τυπώνει το αντίστοιχο μύνημα λάθους.

- **def expression():**Στη συνάρτηση αυτή καλείται η optionalSign() (θα αναλυθεί παρακάτω) και η term().Για όσο ισχύει οτί η λεκτική μονάδα ισοδυναμεί με tokens_plus ή tokens_minus τοτε καλούνται οι ADD_OP(),term(),newtemp(),genquad().

- **def term():**Στη συνάρτηση αυτή καλείται η factor() (θα αναλυθεί παρακάτω) και για όσον η λεκτική μονάδα ισοδυναμεί είτε με tokens_mult είτε με tokens_div τότε καλούνται οι MUL_OP(),factor(),newtemp() και η genquad()

- **def factor():**Στη συνάρτηση αυτή γίνεται έλεγχος για το εάν η λεκτική μονάδα ισοδυναμεί είτε με tokens_digit , είτε με tokens_leftParenthesis

είτε με tokens_id.Στη πρώτη περίπτωση καλείται η lex().Στη δεύτερη περίπτωση καλείται ο λεκτικός αναλυτής και η expression() και γίνεται ένας ακόμα έλεγχος για tokens_rightParenthesis.Στη Τρίτη περίπτωση καλείται πάλι η lex() και η idtail().Εάν δεν ισχύει καμία συνθήκη τότε το πρόγραμμα κάνει έναν τελευτάιο έλεγχο για ισοδυναμία της λεκτικής μονάδας με tokens_rightParenthesis.Σε περίπτωση που είναι αληθής τότε καλείται η lex().

- **def idtail():**Στη συνάρτηση αυτή γίνεται έλεγχος για το εάν αντιστοιχεί η λεκτική μονάδα με tokens_leftParenthesis.Εάν ισχύει, τότε καλείται η lex(), actualparlist() , η newtemp() και η genquad().Ακολουθεί ο κλασσικός έλεγχος για την ύπαρξη tokens_rightParenthesis . Εάν δεν ισχύει καμία συνθήκη η συνάρτηση επιστρέφει name.

- **def optionalSign():** Στη συνάρτηση αυτή γίνεται έλεγχος για το εάν η λεκτική μονάδα ισοδυναμεί με tokens_plus ή με tokens_minus.Εάν είναι αληθής αυτή η συνθήκη καλείται η ADD_OP().

- **def REL_OP():**Στη συνάρτηση αυτή ελέγχεται εάν η λεκτική μονάδα αντιστοιχεί σε ένα απο τα tokens_equal,tokens_lessThanOrEq,tokens_greaterThanOrEq,tokens_greaterThan,tokens_lessThan,tokens_diff.Σε οποιαδήποτε συνθήκη που είναι αληθής ενημερώνεται η relop και καλείται η lex().Εάν δεν ισχύει καμία συνθήκη τυπώνεται μύνημα λάθους.Στο τέλος της συνάρτησης επιστρέφεται η relop.

- **def ADD_OP():** Στη συνάρτηση αυτή ελέγχεται εάν η λεκτική μονάδα αντιστοιχεί σε ένα απο τα tokens_plus ή tokens_minus.Σε οποιαδήποτε

συνθήκη που είναι αληθής ενημερώνεται η plusOrminus και καλείται η lex().Εάν δεν ισχύει καμία συνθήκη τυπώνεται μύνημα λάθους.Στο τέλος της συνάρτησης επιστρέφεται η plusOrminus.

- **def MUL_OP():**Στη συνάρτηση αυτή ελέγχεται εάν η λεκτική μονάδα αντιστοιχεί σε ένα απο τα tokens_mult ή tokens_div.Σε οποιαδήποτε συνθήκη που είναι αληθής ενημερώνεται η multOrdiv και καλείται η lex().Εάν δεν ισχύει καμία συνθήκη τυπώνεται μύνημα λάθους.Στο τέλος της συνάρτησης επιστρέφεται η multOrdiv.

Δεύτερη Φάση

Ενδιάμεσος κώδικας

Σε αυτή τη φάση το πρόγραμμα είναι στην ενδιάμεση γλώσσα και αποτελείται από μία σειρά από τετράδες, οι οποίες είναι αριθμημένες έτσι ώστε σε κάθε τετράδα να μπορούμε να αναφερθούμε χρησιμοποιώντας τον αριθμό της ως ετικέτα. Για να απλοποιήσουμε όσο το δυνατόν τους συμβολισμούς , θεωρούμε ότι έχουμε έναν τελεστή και τρία τελούμενα .Δημιουργούνται με τη βοηθητική συνάρτηση def genquad(op ,x,y,z) και κάθε τετράδα λαμβάνει τη σωστή αρίθμηση με τη def nextQuad().Ολοκληρώνοντας , η κάθε τετράδα τοποθετείται στον global πίνακα listQuads. Οι υπόλποιπες βασικές συναρτήσεις του ενδιάμεσου κώδικα είναι οι newtemp() , η emptylist() ,η makelist() , η merge() και η backpatch().

- **def newtemp()**: Δημιουργεί και επιστρέφει τις προσωρινές μεταβλητές όποτε αυτή καλεστεί.
- **def emptylist()**: Δημιουργεί και επιστρέφει μια κενή λίστα στην οποία στη συνέχεια θα τοποθετηθούν ετικέτες τετράδων.
- **def makelist(x)**: Δημιουργεί και επιστρέφει μία νέα λίστα η οποία οπο'ια έχει σαν μοναδικό στοιχείο της την ετικέτα τετράδας x.
- **def merge(list1,list2**): Δημιουργεί την totalList και συνενώνει τις list1 και list2.
- **def backpatch(list,z)**: Διαβάζει μία μία τις τετράδες που σημειώνονται στη list (for i list) και για κάθε τετράδα του πίανακα ListQuads που αντιστοιχεί στην ετικέτα αυτή , συμπληρώνουμε το τελευταίο πεδίο της με το z(for i in range(len(listQuads))). Όταν συμπληρωθούν όλες οι τετράδες που βρίσκονται στη λίστα αυτή, η λίστα δε χρειάζεται άλλο και μπορεί να αποδεσμεύσει τη μνήμη που κατέχει.

Block

Η δήλωση block στον ενδιάμεσο κώδικα γίνεται με τη χρήση του "begin_block,name _,_" και του "end_block,_,_". Με αυτό τον τρόπο οριοθετούμε την αρχή και το τέλος του ενδιάμεσου κώδικα που παρήχθη . Επίσης , είναι σημαντικό να αναφέρουμε ότι οι εντολές αυτές δεν επιτρέπεται να είναι εμφωλευμένες . Δηλαδή, πρέπει κάθε begin_block να τερματίζεται με end_block πριν ανοίξει μια άλλη ενότητα με end_block διατηρώντας έτσι τη σειρά εμφώλευσης . Συγκεκριμένα, στην def block(name,flag) με τη χρήση της genquad δημιουργείται η τετράδα "begin_block ,name ,_,_" Εάν βρίσκεται μέσα στο κύριο πρόγραμμα πριν τερματιστεί γίνεται χρήση της genquad("halt,_,_,_") και έπειτα το block ολοκληρώνεται με την παραγωγη της τετράδας genquad("end_block, name, _, _").

Αριθμητικές παραστάσεις

Σύμφωνα με τη γραμματική της γλώσσας Cimple, υποστηρίζονται αριθμητικές παραστάσεις με τις τέσσερις αριθμητικές πράξεις (πρόσθεση, αφαίρεση, διαίρεση, πολλαπλασιασμός) καθώς και η προτεραιότητα των πράξεων μέσω των παρενθέσεων. Συγκεκριμένα , η ανάλυση και η επεξεργασία των αριθμητικών πράξεων στο πρόγραμμα μας ξεκινάει στη συνάρτηση def expression().

Πιο αναλυτικά , η γραμματική που ακολουθεί η πρόσθεση ή η αφαίρεση αντίστοιχα είναι E-> T1 (+/- T2 {p1})* {p2}. Μέσα στη while-loop όσο η λεκτική μονάδα είναι ίση με tokens_plus ή tokens_minus παράγουμε μία νέα τετράδα η οποία όταν εκτελείται θα προσθέτει τη μεταβλητή που βρίσκεται στο T1place και τη μεταβλητή που βρίσκεται στο T2place . Δεν μπορούμε δηλαδή να χρησιμοποιήσουμε μία ήδη υπάρχουσα μεταβλητή και γι’ αυτό το λόγο μετά το T2place = term() , δημιουργούμε μία προσωρινή μεταβλητή (w = newtemp()). Κατόπιν , κάνουμε χρήση της genquad(+/- , T1place , T2place ,w) και τοποθετούμε στο {p1} τη νέα μεταβήτη T1place . Κλείνοντας, έξω από τη while-loop θέτουμε Eplace = T1place διότι ο κανόνας μπορεί να αναγνωρίσει παραπάνω από μία προσθέσεις ή αφαιρέσεις .

Στη συνάρτηση def term() ακολουθείται ακριβώς η ίδια διαδικασία που περιγράψαμε στη def expression() με τις εξης διαφορές. Η γραμματική που

ακολουθεί ο πολλαπλασιασμός και η διαίρεση είναι E -> F1 (*ή/ F2)* καθώς και η λεκτική μονάδα αναμένει tokens_mult ή tokens_div.

Ολοκληρώνοντας , στη συνάρτηση def factor() υλοποιείται η γραμματική F -> (E) {p1} όταν η λεκτική μονάδα αναμένει tokens_leftParenthesis . Το Eplace περιέχει τη μεταβλητή που έχει το αποτέλεσμα των πράξεων που έχουν δημιουργηθεί από τον κανόνα αυτόν. Αυτή η ίδια μεταβλητή είναι που θα αποτελέσει και το Fplace αφού καμία αριθμητική πράξη δεν αναγνωρίζεται και δεν δημιουργείται από τον κανόνα F. Τέλος, παραθέτουμε το ίδιο σχέδιο ενδιάμεσου κώδικα για τον κανόνα που αναγνωρίζει τα τερματικά σύμβολα F-> ID {p1}. Στην ίδια συνάρτηση def factor() εάν η λεκτική μονάδα είναι ίση με tokens_id ,τότε καλείται ο lex() και έπειτα γράφουμε Fplace = IDtail(id).

Λογικές Συνθήκες

Οι κανόνες των λογικών παραστάσεων επιστρέφουν σαν αποτέλεσμα τις τετράδες που έχουν μείνει ασυμπλήρωτες , έτσι ώστε να συμπληρωθούν στον τελικό κώδικα από άλλους κανόνες όταν θα γνωρίζουμε που πρέπει να γίνει το λογικό άλμα . Κάθε κανόνας προετοιμάζει για τον κανόνα που τον κάλεσε δύο λίστες:

- Λίστα true: αποτελείται από όλες εκείνες τις τετράδες που έχουν μείνει ασυμπλήρωτες, διότι ο κανόνας δεν μπορεί να της συμπληρώσει. Οι τετράδες θα συμπληρωθούν με την ετικέτα εκείνης της τετράδας στην οποία θα πρέπει να μεταβεί ο έλεγχος αν η λογική συνθήκη ισχύει.
- Λίστα false: αποτελείται από όλες εκείνες τις τετράδες που έχουν μείνει ασυμπλήρωτες, διότι ο κανόνας δεν μπορεί να της συμπληρώσει. Οι τετράδες θα συμπληρωθούν με την ετικέτα εκείνης της τετράδας στην οποία θα πρέπει να μεταβεί ο έλεγχος αν η λογική συνθήκη δεν ισχύει.

Τις λίστες αυτές θα τις ορίσουμε μέσα στις συναρτήσεις def condition(), def boolterm() και def boolfactor(). Στην def condition() περιγράφεται η συνένωση λογικών συνθηκών με τον τελεστή "or". Η γραμματική του κανόνα αυτού είναι B * Q1 (or Q2)* . Μέσα στην συνάρτηση ορίζονται οι λίστες Btrue και Bfalse. Όσο η λεκτική μονάδα είναι ίση με tokens_or καλείται ο λεκτικός αναλυτής κάνουμε backpatch() την λίστα Bfalse και κάνουμε merge() τις νέες τετράδες που θα

έρθουν από το Q2true. Έτσι, η λίστα Btrue μεγαλώνει χωρίς να αλλάζει ο σκοπός της. Τέλος, επιστρέφουμε την Btrue και Bfalse.

Στην def boolterm() περιγράφεται η συνένωση λογικών συνθηκών με τον τελεστή "and". Η γραμματική του κανόνα αυτού είναι Q → R1 {p1} (and {p2} R2 {p3})* . Ακολουθείται η ίδια διαδικασία που αναφέραμε στην def condition() με τη διαφορά ότι η λεκτική μονάδα περιμένει tokens_and και η κάνουμε backpatch() την λίστα Qtrue.

Στην def boolfactor() όταν συναντάμε την σύγκριση δύο αριθμητικών εκφράσεων, πρέπει να παράγουμε την τετράδα που θα πραγματοποιήσει το λογικό άλμα στην περίπτωση που η συνθήκη ισχύει, αλλά και στην περίπτωση που δεν ισχύει. Αρχικά για την δεύτερη περίπτωση εάν η λεκτική μονάδα είναι ίση με tokens_not τότε καλείται ο lex() και στην συνέχεια η λεκτική μονάδα ελέγχει εάν είναι ίση με tokens_leftBracket τότε πάλι καλείται ο λεκτικός αναλυτής και οι τετράδες από την λίστα Btrue γίνονται οι τετράδες τις λίστας Rfalse και αντίστοιχα οι τετράδες από την λίστα Bfalse γίνονται οι τετράδες τις λίστας Rtrue. Στην περίπτωση τώρα που η λεκτική μονάδα δεν είναι tokens_not δημιουργούνται οι λίστες Rtrue και Rfalse καθώς επίσης και οι τετράδες των αλμάτων. Η makelist() καλείται πριν την genquad() διότι πρέπει να πάρει ως παράμετρο την nextQuad(). Με τον τρόπο αυτό θα δημιουργηθεί μία νέα λίστα, μοναδικό στοιχείο της οποίας θα είναι μία μη συμπληρωμένη τετράδα που θα δημιουργηθεί αμέσως μετά. Τέλος, επιστρέφονται οι λίστες Rtrue και Rfasle.

Δομή Εκχώρησης

Η μετατροπή της εκχώρησης σε ενδιάμεσο κώδικα γίνεται στην συνάρτηση def assignStat(). Συγκεκριμένα, εάν η λεκτική μονάδα αντιστοιχεί στο tokens_asgn καλείται ο lex() και με την χρήση της genquad(':=',Eplace,'_',IDplace) καταχωρείται η τιμή του Eplace στο IDplace. Τέλος, το Eplace μπορεί να είναι είτε μεταβλητή είτε αριθμητική ή συμβολική σταθερά.

Δομή απόφασης if

Η μετατροπή της δομής if σε ενδιάμεσο κώδικα γίνεται στην συνάρτηση def ifStat(). Στη γραμμή 1049 γίνεται το backpatch() στο nextQuad() διότι οι τετράδες που έχουν αποθηκευτεί στη λίστα conditionlist[0] έχουν αποτίμηση true των συνθηκών της if και θα πρέπει να οδηγηθούν στο statements . Το backpatch() του conditionList[1] εκτελείται μετά το τέλος της if .Αν λεκτική μονάδα στη def elsepart() είναι διάφορη του tokens_else τότε οι τελευταίες τετράδες που θα παραχθούν θα είναι από το statements και άρα το backpatch() καλείται μετά την τελευταία τετράδα που θα παράγει το if .

Δομή while

Η μετατροπή της δομής while σε ενδιάμεσο κώδικα γίνεται μέσα στη whileStat().Στη γραμμή 1100 με backpatch() συμπληρώνονται όλες οι ασυμπλήρωτες τετράδες της conditionlist[0] οι οποίες έχουν αποτίμηση true . Εάν η αμέσως μετά η λεκτική μονάδα αντιστοιχεί στο tokens_rightParentheseis τότε δημιουργούμε στη γραμμή 1107 τη genquad(jump,_,_,conQuad) η οποία θα κάνει jump στη πρώτη συνθήκη της while .Τέλος, οι ασυμπλήρωτες τετράδες της Conditionlist[1] οι οποίες έχουν αποτίμηση σε false τοποθετούνται ακριβώς στο τέλος διότι εκέι πρέπει να τοποθετηθεί η backpatch() όταν ο έλεγχος δεν ισχύει.

Δομή επιλογής switchcase

Η μετατροπή της switchcase σε ενδιάμεσο κώδικα γίνεται μέσα στη def switchcaseStat() . Όσο η λεκτική μονάδα αντιστοιχεί στο tokens_switchcase ελέγχονται ένα ένα τα condition και όταν ένα αποτιμηθεί ως αληθές ,τότε εκτελούνται τα αντίστοιχα statements και ο έλεγχος στη συνέχεια μεταβαίνει έξω από τη δομή. Όταν η συνθήκη αποτιμηθεί ως ψευδής τότε ο κώδικας που ακολουθεί το statements1 είναι το statements2 κάτι το οποίο επιζητούμε. Σε κάθε περίπτωση λοιπόν, μετά την statements1 θα τοποθετήσουμε t=makeList(nextQuad()) η οποία θα δημιουργήσει μία λίστα με το ασυμπλήρωτο άλμα, κάνουμε χρήση της genQuad('jump','_','_','_') , η οποία θα δημιουργήσει την μη συμπληρωμένη τετράδα και μία merge() η οποία θα συνενώνει την t με τη λίστα που τελικά θα κάνει στο τέλος backpatch() την exitList. Αυτός ο κώδικας

είναι στις γραμμές 1146 έως 1150. Ολοκληρώνοντας, στην αρχή της def switchcaseStat() δημιουργούμε μία κενή exitlist, μία μη συμπληρωμένη τετράδα μετά από κάθε statements και στο τέλος συμπληρώνουμε την exitlist με backpatch() ώστε να δείχνει στην πρώτη τετράδα μετά το τέλος της switchcase.

Δομή επιλογής forcase

Η μετατροπή της forcase υλοποιείται μέσα στην def forcaseStat(), ελέγχει τα conditions που βρίσκονται μέσα τα case. Όταν μία από αυτές αποτιμηθεί σε true εκτελούνται οι αντίστοιχες statements. Αν καμία από τις case δεν ισχύει τότε ο λεκτικός αναλυτής είναι ίσος με το tokens_default και υλοποιείται το αντίστοιχο statements. Στην συνέχεια ο έλεγχος μεταβαίνει έξω από την forcase και ακολουθείται η ίδια διαδικασία όπως στη switchcase.

Δομή πολλαπλής επιλογής incase

Η συνάρτηση αυτή υλοποιείται με στην def incaseStat(). Η incase ελέγχει όλες τις condition που βρίσκονται μετά τα case. Αφότου εξεταστούν όλες οι case, εάν καμία από τις statements1 δεν έχει εκτελεστεί, τότε ο έλεγχος μεταβαίνει έξω από την incase. Αλλιώς μεταβαίνει στην αρχή της incase. Η πληροφορία εάν έστω και μία από τις statements1 έχει εκτελεστεί πρέπει να συλλεχθεί κατά τη διάρκεια της αποτίμησης των εκφράσεων ή της εκτέλεσης των statements1 και να αξιολογηθεί όταν φτάσουμε στο τέλος της δομής. Αυτό επιτυγχάνεται με μία μεταβλητή flag. Η μεταβλητή αυτή είναι προσωρινή δημιουργείται δηλαδή με την χρήση newtemp() στην γραμμή 1233. Αρχικοποιούμε με την χρήση της genquad το flag στην τιμή μηδέν δηλαδή false. Πιο αναλυτικά, το flag λειτουργεί ως boolean τιμή για το εάν έχει εκτελεστεί έστω και ένα από τα statements. Εάν η απάντηση είναι true με χρήση της genquad το flag μετατρέπεται σε ένα στη γραμμή 1253. Τέλος, σε κάθε επανάληψη της incase η τιμή του flag πρέπει να επαναρχικοποιείται στο μηδέν και αυτό επιτυγχάνεται τοποθετώντας μετά την αρχικοποιήση την firstCondQuad = nextQuad().

Συναρτήσεις και διαδικασίες

Σε αυτή την φάση του ενδιάμεσου κώδικα υλοποιούνται η def actualparitem(), η def actualparlist() και η def callStat(). Η κλήση των υποπρογραμμάτων γίνεται με την κλήση της def callStat(). Πρώτα από όλα πρέπει να μεταφραστούν τα ορίσματα της συνάρτησης σε ενδιάμεσο κώδικα. Πιο αναλυτικά, στην def actualparitem() εάν η λεκτική μονάδα είναι ίση με tokens_in τότε κάνουμε χρήση της genquad('par',a,'CV','_') και το αποθηκεύουμε στο expression a, αυτό γίνεται στις γραμμές 1432-1433. Στην περίπτωση τώρα που η λεκτική μονάδα είναι ίση με tokens_inout, τότε κάνουμε χρήσης της genquad('par',b,'REF','_') στην γραμμή 1444. Με αυτό τον τρόπο η def callStat() όταν καλεί μία διαδικασία ή συνάρτηση μπορεί να δει της παραμέτρους της. Στην περίπτωση που είναι συνάρτηση στην def returnStat() με χρήση της genquad('retv',Eplace,'_','_') στην γραμμή 1283 αποθηκεύουμε το return στο Eplace για να μπορεί η συνάρτηση όταν καλείται να έχει τιμή επιστροφής.

Είσοδος και έξοδος δεδομένων

Οι δύο εντολές εισόδου-εξόδου υλοποιούνται στον ενδιάμεσο κώδικα στις def printStat() και def inputStat(). Συγκεκριμένα, στην def printStat() η δημιουργία της print γίνεται στην γραμμή 1354 με χρήση genquad('out',Eplace,'_','_') και η δημιουργία της input γίνεται στην γραμμή 1386 με την χρήση της genquad('inp',IDplace,'_','_').

Παραγωγή αρχείου σε .int

Στην συνάρτηση def creatIntFile() παράγεται ένα αρχείο που έχει το ίδιο όνομα με αυτό του αρχείου σε .ci αλλά είναι τύπου .int. Σε αυτό τυπώνονται όλες οι τετράδες που φτιάχτηκαν και βρίσκονται στην λίστα listQuads.

Παραγωγή αρχείου σε .c

Στην συνάρτηση def creatCfile() παράγεται ένα αρχείο που έχει το ίδιο όνομα με αυτό του αρχείου σε .ci αλλά είναι τύπου .c, με την προϋπόθεση βέβαια ότι δεν υπάρχει function , procedure ή κλήση συνάρτησης με την εντολή call. Σε αυτό το αρχείο μεταφράζονται σε γλώσσα C και τυπώνονται όλες οι τετράδες που φτιάχτηκαν και βρίσκονται στην λίστα listQuads.

Πίνακας συμβόλων

Σε αυτή την φάση δημιουργούμε τον πίνακα συμβόλων. Η μορφή του πίνακα συμβόλων αποτελείται από Scope, Argument, Entity. Συνεπώς, φτιάχνονται αντίστοιχα οι class Argument(), class Entity(), class Scope().

Για κάθε μεταβλητή, συνάρτηση/διαδικασία, παράμετρο και προσωρινή μεταβλητή δημιουργούνται οι οντότητες Entities. Πιο αναλυτικά, μέσα στο Record Entity δημιουργείται η class Variable, η οποία αποτελείται από το type και το intOffset. Η class SubProgram (διαδικασία ή συνάρτηση) που περιλαμβάνει το type, startingQuad, frameLength και argumentList. Στη συνέχεια φτιάχνεται η class Parameter που περιλαμβάνει το parMode και το intOffset. Τέλος , περιέχεται και η class TempVar που περιέχει το intOffset.

Το Record Argument αποτελείται από το name, type και το parMode και το Record Scope περιέχει το name, την pointerEntityList, το nestingLevel και το pointerEnclosingScope.

Οι ενέργειες στον πίνακα συμβόλων πραγματοποιούνται με την χρήση συναρτήσεων. Αναλυτικά, έχουμε την def newScope() η οποία δημιουργεί ένα scope, το όνομα του scope καθώς και το newScope είναι ίσο με το upperComparingScope. Εάν το upperComparingScope είναι το μόνο scope τότε το nestingLevel του είναι ίσο με το μηδέν, αλλιώς newscope.nestingLevel = upperComparingScope.nestingLevel+1 και ανανεώνεται το upperComparingScope σε newScope. Έπειτα δημιουργείται η def deleteScope(). Κάθε φορά διαγράφεται το πιο πάνω scope δηλαδή deletedScope = upperComparingScope ,το upperComparingScope = upperComparingScope.pointerEnclosingScope και τέλος διαγράφουμε το deletedScope. Επιπρόσθετα, δημιουργούμε την def

newArgument(object). Στην συγκεκριμένη συνάρτηση πάμε στο τελευταίο Entity της λίστας μας άρα στο pointerEntityList(-1) και τοποθετούμε τα Argument του subprogram. Κατόπιν δημιουργούμε την def computeOffset(), συγκεκριμένα αρχικοποιούμε ένα counter στο μηδέν. Η συνάρτηση ψάχνει πόσα Entities εκτός από subprogram υπάρχουν στην λίστα από Entities και θα πάει να υπολογίσει το intOffset. Τέλος επιστρέφουμε το intOffset. Κατόπιν, στη def computeFramelength() υπολογιζουμε το framelength. Η def addParametres() θα καλεστεί όταν δε βρίσκεται στο κυρίως πρόγραμμα , θα πάει στο pointerEnclosingScope.pointerEntityList[-1].subprogram.argumentList θα πάρει τις παραμέτρους και θα τα τοποθετήσει στο παραπάνω scope. Κλείνοντας , η def SymbolBoard() τυπώνει όλο τον πίνακα συμβόλων.

Συνοψίζοντας , όλες οι ενέργειες που αναφέραμε τοποθετούνται και καλούνται μέσα στο συντακτικό αναλυτή. Δηλαδή , στη def block() καλώ την newScope στην αρχή του block για να φτιάξω το scope στη γραμμη 685. Η SymbolBoardPrint() και η deleteScope() καλούνται στο τέλος του block στις γραμμες 710-712 για να κάνουν τις αντίστοιχες λειτουργίες της η καθεμία. Επίσης , προσθήκη newEntity() συναντάμε στις γραμμές 748-752 στη def varlist() , στις γραμμές 306-310 στη def newtemp() και στις γραμμές 798-802 , 836-840 στη def subprogram() . Τέλος , προσθήκη newArgument συναντάμε στις γραμμές 895-899 , 913-914 στην def formalparitem()

Τρίτη Φάση

Τελικός κώδικας

Όπως αναφέραμε και στην αρχή, η τρίτη φάση αποτελείται από τον τελικό κώδικα. Ο τελικός κώδικας προκύπτει από τον ενδιάμεσο κώδικα με τη βοήθεια του πίνακα συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μια σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων.

Σημείωση: Ο τελικός κώδικας δεν είναι πλήρης ,λείπουν αρκετά κομμάτια ακόμα αλλά δεν μπήκε σε σχόλια εφόσον δεν εμποδίζει τη λειτουργία του προγράμματος. Μόνο η κλήση του print έχει μπει σε σχόλια καθώς δεν έκανε run. Παρακάτω εξηγούμε τις συναρτήσεις που έχουμε υλοποιήσει.

- **def connector():** Στην βοηθητική συνάρτηση αυτή πρακτικά θέλουμε να βρούμε την συσχέτιση μεταξύ ενός Entity και σε ποιό Scope βρίσκεται. Αρχικά ορίζουμε μια τοπική μεταβλητή sc να αντιστοιχίζεται με το ανώτερο Scope και στη συνέχεια με χρήση της while ψάχνουμε τα υπόλοιπα scopes. Εάν βρεθεί συσχέτιση , η συνάρτηση την επιστρέφει αλλιώς τυπώνεται μήνυμα λάθους.

- **def gnlvcode():**Πρόκειται για κύρια συνάρτηση όπου ο σκοπός μας είναι να εξάγει τον τελικό κώδικα περί προσπέλασης πληροφορίας που είναι αρχικά αποθυκευμένη στο εγγράφημα δρασηριοποίησης μιας συνάρτησης ή διαδικασίας. Για να το πετύχουμε αυτό, αρχικά γράφουμε στο αρχείο μας για πρώτη φορά να φορτώσει στον t0 απο την μνήμη και ώς offset μείον 4 από τον καταχωρητή δείκτη sp. Καλείται η συνάρτηση connector(),έχουμε ορίσει μια μεταβλητή ως μετρητή για την while που θα χρησιμοποιήσουμε για να ελέγξουμε το πόσες φορές το πρόγραμμα θα γράψει το *“lw t0,-4(t0)”* στο αρχείο assembly που εξάγεται.Επίσης στο τέλος της συνάρτησης γίνεται και άλλη εγγραφή στο αρχείο assembly “addi t0,t0, -intOffset”.
- **def loadvr(v,reg):**Η συνάρτηση αυτή παίρνει δυο ορίσματα, μια μεταβλητή και έναν καταχωρητή. Καλείται η connector(), και έπειτα εξετάζει εάν η μεταβλητή είναι αριθμός ώστε να γίνει η αντίστοιχη εγγραφή στο αρχείο assembly. Εάν δεν είναι , ακολουθεί μια σειρά από ελέγχους όπου εξετάζεται το εάν η μεταβλητή είναι καθολική, τοπική ή προσωρινή και τους δυνατούς συνδυασμούς τους. Σε κάθε περίπτωση γράφει στο αρχείο την κατάλληλη εντολή που αναφορά την ανάγνωση μιας μεταβλητής που είναι αποθηκευμένη στη μνήμη και την μεταφέρει σε έναν καταχωρητή.

- **def storerv(reg,v):** Η συνάρτηση αυτή παίρνει δυο ορίσματα, μια μεταβλητή και έναν καταχωρητή. Καλείται η connector(),και έπειτα ακολουθεί η αντίστροφη διαδικασία της προαναφερθέν loadvr().Δηλαδή κάνει τους ίδιους ελέγχους, απλώς αντί να φορτώσει από την μνήμη, γράφει σε αυτήν την τιμή μιας μεταβλητής.

- **def lastCheck():**Η τελευταία συνάρτηση του τελικού κώδικα διατρέχει όλες τις διαθέσιμες τετράδες (listQuads) και γράφει στο αρχείο τον αριθμό τους.Έπειτα,εξετάζει το δέυτερο στοιχείο τους εάν ισοδυναμέι με ένα απο τα Jump, = , <,>,<=,>=,<>,:=,-,*,div,retv και σε κάθε περίπτωση θα καλέσει την loadvr() και θα γράψει στο assembly αρχείο το κατάλληλο μύνημα.Για παράδειγμα, στη περίπτωση που το δεύτερο στοιχείο μιας τετράδας είναι ίσο με «<>» τότε στο αρχείο assembly θα εγγραφεί το “bne,t1,t2,label”.