**2η φάση: Παραγωγή ενδιάμεσου κώδικα, Πίνακας Συμβόλων και Σημασιολογική Ανάλυση**

1. *Παραγωγή ισοδύναμου κώδικα σε C*

Για την παραγωγή του ισοδύναμου αρχείου σε C, ουσιαστικά χρησιμοποιήθηκαν οι τετράδες που παρήχθησαν από τον ενδιάμεσο κώδικα και έχουν αποθηκευτεί στο λεξικό *quad\_program\_list*. Για την ευκολότερη αναγνώριση των ονομάτων των μεταβλητών που πρέπει να δηλωθούν στο C αρχείο, έχουν οριστεί οι global λίστες *allFunctions* και *bindedCharacters*.

Η πρώτη διατηρεί τα ονόματα όλων των συναρτήσεων που μπορεί να περιέχει το πρόγραμμα που μεταφράζεται. Τα ονόματα αυτά προστίθενται στη λίστα *allFunctions* κατά τη διάρκεια της κλήσης της συνάρτησης *subprogram,* όπου γίνεται η μετάφραση της κάθε συνάρτησης.

Η λίστα *bindedCharacters* διατηρεί δεσμευμένους χαρακτήρες και εκφράσεις που εμφανίζονται στις τετράδες και αφορούν το πέρασμα παραμέτρων στις συναρτήσεις. Πρόκειται για το χαρακτήρα *‘\_’,* που συμβολίζει το κενό στοιχείο σε μια τετράδα, καθώς και τις εκφράσεις ‘*CV’* (πέρασμα παραμέτρου με τιμή), *‘CP’* (πέρασμα παραμέτρου με αντιγραφή τιμής), *‘REF’* (πέρασμα παραμέτρου με αναφορά) και *‘RET’* (πέρασμα παραμέτρου επιστροφής της συνάρτησης).

Η παραγωγή του ισοδύναμου κώδικα σε C πραγματοποιείται μέσω της συνάρτησης ***produceCFile***, η οποία δέχεται ως όρισμα το domain name του αρχείου που θα δημιουργηθεί (το ίδιο με το όνομα του αρχικού προγράμματος καθώς και του ισοδύναμου ενδιάμεσου κώδικα).

Οι μεταβλητές που θα δηλωθούν στο C αρχείο διατηρούνται στη λίστα *variables*. Σε αυτή τη λίστα προστίθενται όλα τα μοναδικά ονόματα μεταβλητών που έχουν χρησιμοποιηθεί στις τετράδες. Εάν υπάρχουν δηλωμένες μεταβλητές στο αρχικό πρόγραμμα που δεν έχουν χρησιμοποιηθεί για την παραγωγή κάποιας τετράδας, παραβλέπεται η δήλωσή τους στο αρχείο C. Η συνάρτηση *checkIfNegative* χρησιμοποιείται για την αναγνώριση των προσημασμένων μεταβλητών και την προσθήκη των κατάλληλων μεταβλητών στη λίστα μεταβλητών. Πριν πραγματοποιηθεί ο έλεγχος για την προσθήκη της εκάστοτε μεταβλητής στη λίστα *variables* αφαιρείται το πρόσημο (εάν υπάρχει), χωρίς, ωστόσο, να επηρεάζεται η ορθή λειτουργία του προγράμματος. Επιπλέον, ελέγχεται εάν η μεταβλητή που δόθηκε είναι μεταβλητή ή αριθμητική σταθερά. Στη δεύτερη περίπτωση, δεν προστίθενται στη λίστα μεταβλητών.

Πριν την καταγραφή των μεταβλητών στο αρχείο C, γίνεται έλεγχος για την ύπαρξη χαρακτήρων από τις λίστες *bindedCharacters* και *allFunctions* στη λίστα *variables* και απομακρύνονται κατάλληλα.

Πριν την καταγραφή των εκτελέσιμων εντολών, προστίθενται οι κατάλληλες κεφαλίδες στο αρχείο C («includes»). Στη συνέχεια, για κάθε τετράδα στο λεξικό *quad\_program\_list* γίνονται οι παρακάτω έλεγχοι:

* Εάν πρόκειται για τετράδα *begin\_block* με το όνομα του προγράμματος, γίνεται έναρξη της συνάρτησης int main() και δήλωση των μεταβλητών της λίστας *variables*.
* Εάν πρόκειται για τετράδα ανάθεσης, δημιουργείται εκτελέσιμη εντολή της μορφής a = b;, όπου a η μεταβλητή στην οποία πραγματοποιείται η ανάθεση και b η ανατιθέμενη τιμή.
* Εάν πρόκειται για τετράδα με αριθμητική πράξη, δημιουργείται εκτελέσιμη εντολή της μορφής a = b op c;, όπου a η μεταβλητή στην οποία πραγματοποιείται η ανάθεση και b ο πρώτος τελεστέος, op ο αριθμητικός τελεστής (ένας εκ των +,-,\*,/) και c ο δεύτερος τελεστέος.
* Εάν πρόκειται για τετράδα με συγκριτικό τελεστή, δημιουργείται εκτελέσιμη εντολή της μορφής if (x op y) goto L\_z;, όπου x, ο πρώτος τελεστέος, y ο δεύτερος τελεστέος, op ο συγκριτικός τελεστής (ένας εκ των <,>,=,<>,<=,>=) στην κατάλληλη αποδεκτή μορφή για το αρχείο C (π.χ. ο τελεστής <> μετατρέπεται σε !=) και z η ετικέτα κάποιας τετράδας.
* Εάν πρόκειται για τετράδα jump σε κάποια ετικέτα, δημιουργείται εκτελέσιμη εντολή της μορφής goto L\_z;, όπου z η ετικέτα της τετράδας.
* Εάν πρόκειται για τετράδα retv και όνομα μεταβλητής/σταθεράς, δημιουργείται εκτελέσιμη εντολή της μορφής return x;, όπου x το όνομα της μεταβλητής/σταθεράς.
* Εάν πρόκειται για τετράδα inp και όνομα μεταβλητής, δημιουργείται εκτελέσιμη εντολή της μορφής scanf(“%d”, x);, όπου x το όνομα της μεταβλητής που ζητάμε την τιμή της ως είσοδο στο πρόγραμμα.
* Εάν πρόκειται για τετράδα out και όνομα μεταβλητής/σταθεράς, δημιουργείται εκτελέσιμη εντολή της μορφής printf("%d, x\\n");, όπου x το όνομα της μεταβλητής/σταθεράς που θα εμφανιστεί στην οθόνη.
* Εάν πρόκειται για τετράδαhalt προστίθεται η εντολή return 0; και το πρόγραμμα ολοκληρώνεται.

**Σημείωση:** Όλες οι εκτελέσιμες εντολές διαθέτουν μια ετικέτα L\_z, όπου z η τιμή της ετικέτας. Επιπλέον, στο τέλος κάθε εντολής έχει προστεθεί με σχόλια η εκάστοτε ισοδύναμη εντολή σε ενδιάμεσο κώδικα.

1. *Πίνακας Συμβόλων*

Για την υλοποίηση των οντοτήτων που περιέχει ο πίνακας συμβόλων δημιουργήθηκαν οι παρακάτω κλάσεις:

* **Entity**: Μητρική κλάση που περιέχει τα πεδία ***Name*** και ***Entity\_type*.** Το πεδίο Entity\_type μπορεί να πάρει μία από τις παρακάτω τιμές: *Variable* (αφορά μεταβλητές και προσωρινές μεταβλητές), *Function* (για συναρτήσεις) και *Parameter* (για παραμέτρους συναρτήσεων). Η γλώσσα Starlet δεν χρησιμοποιεί σταθερές. Περιλαμβάνει τις μεθόδους *\_\_init\_\_* (constructor) και *printEntity* που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.
* **Variable:** «Παιδί» της κλάσης *Entity.*Ορίζει ένα επιπλέον πεδίο ***Offset,*** που υποδηλώνει την απόσταση στη μνήμη από την αρχή του block που ανήκει. Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες του επιπλέον πεδίου.
* **Function:** «Παιδί» της κλάσης *Entity.*Ορίζει ένα επιπλέον πεδίο ***Framelength,*** που υποδηλώνει το μέγεθος του block της συνάρτησης στη μνήμη. Επιπλέον, περιλαμβάνει το πεδίο ***returnType***, που δείχνει τον τύπο της τιμής επιστροφής (integer), το πεδίο ***startQuad***, που είναι ένας δείκτης στην πρώτη τετράδα που θα εκτελέσει η συνάρτηση και το πεδίο ***argList***, που είναι μια λίστα η οποία διατηρεί τον τύπο των ορισμάτων της συνάρτησης. Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες των επιπλέον πεδίων.
* **Parameter:** «Παιδί» της κλάσης *Entity.*Περιλαμβάνει ένα επιπλέον πεδίο ***Offset,*** που υποδηλώνει την απόσταση στη μνήμη από την αρχή του block που ανήκει, καθώς και το πεδίο ***parMode***, που υποδεικνύει τον τύπο περάσματος του ορίσματος. Το πεδίο parMode μπορεί να πάρει μία εκ των τιμών «*in*», «*inout*», «*inandout*». Περιλαμβάνει τις ίδιες μεθόδους με την γονική κλάση Entity αλλά overridden, καλύπτοντας και τις ανάγκες του επιπλέον πεδίου.
* **Scope:** Κλάση που υποδηλώνει το βάθος φωλιάσματος. Περιλαμβάνει τα πεδία ***nestingLevel,*** που δείχνει το βάθος φωλιάσματος και παίρνει τιμές >=0 καθώς και το πεδίο ***framelength,*** το οποίο ενημερώνεται κάθε φορά που προστίθεται στο scope νέα μεταβλητή ή παράμετρος. Περιλαμβάνει τις μεθόδους *\_\_init\_\_* (constructor) και *printScope*, που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.
* **Argument:** Κλάση που αφορά τα ορίσματα που δέχεται μια συνάρτηση. Περιλαμβάνει ένα πεδίο ***argMode***, που συμβολίζει τον τύπο περάσματος του ορίσματος, καθώς και τις μεθόδους *\_\_init\_\_* (constructor) και *printArgument*, που επιστρέφει ένα string με τις τιμές των πεδίων του αντικειμένου.

Εκτός από τις παραπάνω κλάσεις, διατηρείται η global λίστα **symbolList,** που περιέχει όλα τα ενεργά scopes του προγράμματος. Η συνάρτηση **printSymbolList**() τυπώνει τα περιεχόμενα της λίστας *symbolList* και καλείται πριν τη διαγραφή ενός scope και αμέσως μετά από αυτή.

Επιπρόσθετα, προστέθηκαν οι παρακάτω συναρτήσεις που αποσκοπούν στη δημιουργία αντικειμένων τύπου Variable, Function, Parameter, Scope και Argument, στη διαγραφή ενός Scope καθώς και στην αναζήτηση Entities, μεταβλητών ή παραμέτρων. Συγκεκριμένα:

* **createArgument** (mode, arglist): Δημιουργεί ένα αντικείμενο τύπου Argument με argMode = mode και το προσθέτει στη λίστα arglist (λίστα ορισμάτων κάποιας συνάρτησης που δίνεται ως όρισμα).
* **createVariable**(name, entity\_type, offset, scopelist): Δημιουργεί ένα αντικείμενο τύπου Variable και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createFunction**(name, entity\_type, scopelist): Δημιουργεί ένα αντικείμενο τύπου Function και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createParameter**(name, entity\_type, offset, parMode, scopelist): Δημιουργεί ένα αντικείμενο τύπου Parameter και το προσθέτει στη λίστα scopelist (λίστα οντοτήτων κάποιου scope που δίνεται ως όρισμα).
* **createScope**(nestingLevel): ): Δημιουργεί ένα αντικείμενο τύπου Scope και το προσθέτει στη λίστα symbolList. Καθορίζει το βάθος φωλιάσματος του αντικειμένου με βάση το πλήθος των ενεργών scopes στη λίστα symbolList.
* **deleteScope**(): Διαγράφει το τελευταίο scope της λίστας symbolList.
* **searchEntity**(name, nl): Αναζητά και επιστρέφει την οντότητα με όνομα name, ξεκινώντας από βάθος φωλιάσματος nl. Εάν δεν υπάρχει τέτοιο αντικείμενο, τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.
* **searchVariableOrParameter**(sc): Αναζητά και επιστρέφει την τελευταία μεταβλητή ή παράμετρο που έχει προστεθεί στη λίστα οντοτήτων του scope sc. Εάν δεν υπάρχει τέτοιο αντικείμενο, επιστρέφει None.

Για τη συμπλήρωση του πίνακα συμβόλων έχουν τροποποιηθεί τα εξής σημεία στον κώδικα του μεταγλωττιστή:

1. Στη συνάρτηση **newTemp**() δημιουργείται μεταβλητή με το όνομα της προσωρινής μεταβλητής που επιστρέφεται και ορίζεται κατάλληλα το framelength του scope που τροποποιήθηκε.
2. Στη συνάρτηση **program**() δημιουργείται το scope του main προγράμματος.
3. Στη συνάρτηση **block**(name, returnList = []), εάν πρόκειται για block συνάρτησης και όχι του βασικού προγράμματος, ορίζεται το πεδίο startQuad να δείχνει στην εντολή begin\_block της συνάρτησης. Κατόπιν, πριν το τέλος του block, ορίζεται η τιμή του framelength της κάθε συνάρτησης και στη συνέχεια διαγράφεται το τελευταίο scope που προστέθηκε. Εάν πρόκειται για το block βασικού προγράμματος, το framelength αποθηκεύεται σε μία global μεταβλητή με το όνομα **mainFramelength**.
4. Στη συνάρτηση **varlist**() για κάθε μεταβλητή που δηλώνεται, δημιουργείται ένα αντικείμενο Variable και ενημερώνεται κατάλληλα το framelength του scope που τροποποιήθηκε.
5. Στη συνάρτηση **subprogram** () για κάθε συνάρτηση που ορίζεται, δημιουργείται ένα αντικείμενο Function με το όνομα της συνάρτησης στο scope που ανήκει και στη συνέχεια δημιουργείται ένα καινούριο αντικείμενο Scope με μεγαλύτερο βάθος φωλιάσματος που αντιστοιχίζεται στη συνάρτηση αυτή.
6. Στη συνάρτηση **formalparitem**() για κάθε παράμετρο που διαβάζεται, δημιουργείται δημιουργείται ένα αντικείμενο Argument στη λίστα ορισμάτων της συνάρτησης που μεταφράζεται μέσω της συνάρτησης createArgument και στο τελευταίο scope προστίθεται ένα νέο αντικείμενο Parameter με τον κατάλληλο τύπο περάσματος και ενημερώνοντας κατάλληλα το framelength του scope που τροποποιήθηκε.
7. *Σημασιολογική Ανάλυση*

Για τη σημασιολογική ανάλυση έχει προστεθεί κώδικας που ικανοποιεί τους παρακάτω περιορισμούς:

1. *Κάθε συνάρτηση έχει μέσα της τουλάχιστον ένα return.*
2. *Δεν υπάρχει return έξω από συνάρτηση.*
3. *Υπάρχει exit μόνο μέσα σε βρόχους loop-endloop.*
4. *Κάθε μεταβλητή ή συνάρτηση που έχει δηλωθεί να μην έχει δηλωθεί πάνω από μία φορά στο βάθος φωλιάσματος στο οποίο βρίσκεται.*
5. *Κάθε μεταβλητή ή συνάρτηση που χρησιμοποιείται έχει δηλωθεί και μάλιστα με τον τρόπο που χρησιμοποιείται (σαν μεταβλητή ή σαν συνάρτηση).*
6. *Οι παράμετροι με τις οποίες καλούνται οι συναρτήσεις είναι ακριβώς αυτές με τις οποίες έχουν δηλωθεί και με τη σωστή σειρά.*

Συγκεκριμένα:

1. Στη συνάρτηση **block**(name, returnList = []) προστέθηκε ως επιπλέον –προαιρετικό- όρισμα μια λίστα *returnList*, στην οποία αποθηκεύονται όλες οι εμφανίσεις των return statements μέσα σε μια συνάρτηση. Αυτή η λίστα αρχικοποιείται κάθε φορά που εκτελείται η συνάρτηση *subprogram()* και μεταβάλλεται με την κλήση της συνάρτησης return\_stat. Στη συνάρτηση **subprogram()** γίνεται έλεγχος για το μέγεθος της λίστας returnList και εάν δεν έχει βρεθεί κανένα return statement τυπώνεται κατάλληλο μήνυμα λάθους και η μετάφραση του προγράμματος διακόπτεται.
2. Η global boolean μεταβλητή ***enableReturnSearch*** χρησιμοποιείται για την ανίχνευση return statements εκτός συναρτήσεων. Στη συνάρτηση **block**(name, returnList = []) γίνεται έλεγχος για το εάν πρόκειται για κλήση της block μέσα στο βασικό πρόγραμμα και αλλάζει η τιμή της μεταβλητής. Στην πρώτη εμφάνιση return statement (κλήση της *return\_stat*) μέσα στο βασικό πρόγραμμα τυπώνεται κατάλληλο μήνυμα λάθους και ο compiler τερματίζει.
3. Η global λίστα **loopCounterList** διατηρεί το πλήθος των εμφανίσεων των loop statements. Κάθε φορά που ξεκινά ένα καινούριο loop statement προστίθεται ένα νέο στοιχείο στο τέλος της λίστας (κατά σύμβαση η πρώτη τετράδα του statement) και μόλις αυτό ολοκληρωθεί (μετά το endloop) αφαιρείται από τη λίστα το τελευταίο στοιχείο της. Όσο η λίστα αυτή δεν είναι κενή, υπάρχουν ενεργά loop statements, επομένως είναι επιτρεπτή η εμφάνιση exit statements. Σε κάθε κλήση της συνάρτησης *exit\_stat(),* εάν η λίστα είναι κενή, τότε το exit statement έχει τοποθετηθεί εκτός loop-endloop, επομένως τυπώνει κατάλληλο μήνυμα λάθους και τερματίζει.
4. Στη συνάρτηση **declarations**() γίνεται έλεγχος για τη δήλωση περισσότερων από μία φορές της ίδιας μεταβλητής μέσα στο πιο μεγάλο βάθος φωλιάσματος. Εάν υπάρχουν τουλάχιστον δύο μεταβλητές με το ίδιο όνομα, τυπώνεται κατάλληλο μήνυμα λάθους και ο compiler τερματίζει. Στη συνάρτηση **subprogram(),** κάθε φορά που προστίθεται ένας νέος ορισμός συνάρτησης γίνεται έλεγχος για την ύπαρξη οντότητας (μεταβλητής, συνάρτησης ή παραμέτρου) με το ίδιο όνομα και εάν βρεθεί τέτοια οντότητα, τυπώνεται κατάλληλο μήνυμα λάθους και η μετάφραση διακόπτεται.
5. Στη συνάρτηση **factor**() και στην περίπτωση του κανόνα id <idtail>, γίνεται έλεγχος για τον τύπο της οντότητας που έχει δοθεί. Εάν το id αντιστοιχεί σε όνομα συνάρτησης αλλά δεν δίνεται λίστα παραμέτρων ή αντίστροφα, εάν το id αντιστοιχεί σε όνομα μεταβλητής/παραμέτρου και δίνεται και λίστα παραμέτρων, τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.
6. Σε κάθε κλήση συνάρτησης, η συνάρτηση **factor**() βρίσκει τον ορισμό της συνάρτησης αυτής στο scope που έχει δηλωθεί και καλεί την συνάρτηση *idtail* περνώντας ως όρισμα τη λίστα των ορισμάτων που δέχεται αυτή. Η συνάρτηση **actualparlist**() επιστρέφει μια λίστα που περιέχει τους τύπους των παραμέτρων που δίνονται ως όρισμα σε κάποια συνάρτηση κατά την κλήση της. Η σύγκριση αυτών των δύο λιστών γίνεται στη συνάρτηση *actualpars*, όπου γίνεται σύγκριση των μεγεθών των δύο λιστών καθώς και της διάταξης των στοιχείων τους. Εάν το πλήθος διαφέρει ή έχουν δοθεί παράμετροι με διαφορετική σειρά τότε τυπώνεται κατάλληλο μήνυμα λάθους και το πρόγραμμα τερματίζει.