ΑΝΑΦΟΡΑ ΜΕΤΑΦΡΑΣΤΕΣ

Όνομα: Κωνσταντίνος Ρουμελιώτης

A.M: 3165

Email: cs03165@uoi.gr

Έτος: 7ο

Πριν ξεκινήσω την εξήγηση του κώδικα, θα ήθελα να αναφέρω ότι το project αναπτύχθηκε στο visual studio code και υλοποιήθηκε με πάρα πολλές γραμμές κώδικα. Θα προσπαθήσω να γίνω όσο πιο σαφής και περιεκτικός μπορώ. Στο μεγαλύτερο μέρος όπως θα δείτε ακολούθησα πιστά, τα όσα εσείς μας δίνετε στις διαφάνειες, εκτός από ελάχιστες περιπτώσεις, με γνώμονα να είναι σωστή η υλοποίηση και να τρέχει ο κώδικας. Ας ξεκινήσουμε…

Εισαγωγή & Λεκτικός Αναλυτής

Αρχικά , κάνουμε εισαγωγή τις απαραίτητες βιβλιοθήκες, οι οποίες κατά κύριο λόγο θα μας βοηθήσουν, στο να εισάγουμε το όνομα του προγράμματος μας από το τερματικό και παράλληλα, να ελέγξουμε αν το αρχείο υπάρχει, και αν δούμε ότι υπάρχει, ύστερα να ελέγξουμε αν έχει την σωστή για τα files μας κατάληξη .ci . Αυτός ο έλεγχος γίνεται με την δικιά μας συνάρτηση givefilename, η οποία μας οδηγεί στο να δώσουμε σαν είσοδο το όνομα του αρχείου από το cmd, και αν αυτό υπάρχει και έχει κατάληξη .ci τότε επιστρέφει το όνομα του αρχείου, διαφορετικά μας ενημερώνει ότι είτε το αρχείο μας δεν έχει την κατάληξη που έπρεπε να έχει για να περάσει από τον μεταφραστή μας, είτε ότι το όνομα που δώσαμε δεν αντιστοιχεί σε κάποιο από τα αρχεία που βρίσκονται στον τρέχοντα κατάλογο όπου βρισκόμαστε. Ύστερα η συνάρτηση καλείται και αν όλα πάνε καλά, στην μεταβλητή name αποθηκεύεται το όνομα του αρχείου που δώσαμε από το cmd.Αμέσως μετά ανοίγουμε το αρχείο μας, ώστε να το περάσουμε από λεκτική ανάλυση.

Αρχικά μαζί με μία σειρά μεταβλητών που δηλώνουμε, ορίζουμε πριν το while την κατάσταση state=”start” , δηλαδή αρχική. Ύστερα στο while λέω ότι όσο δεν έχει δημιουργηθεί κάποιο κομμάτι το οποίο αναγνωρίζει ο μεταφραστής διάβασε ακόμη έναν χαρακτήρα, από το αρχείο .ci . Αν βρισκόμαστε στην κατάσταση start και έρχεται στο αυτόματο μας κενό, τότε δεν αλλάζουμε κατάσταση, αλλά παραμένουμε στην ίδια. Αν είμαστε στην state=”start” και έρθει ψηφίο, μεταβαίνουμε στην κατάσταση digit και προσθέτουμε το ψηφίο στην λίστα ώστε αργότερα να γυρίσουμε αυτό που θα σχηματιστεί, είτε αν μείνει μόνο του, είτε αν προστεθούν και άλλα. Ακόμη αν η κατάσταση είναι start και το ψηφίο που διαβάζουμε είναι λατινικό γράμμα είτε κεφαλαίο είτε πεζό (a-z,A-Z), μεταβαίνουμε στην κατάσταση idk, προσθέτουμε τον χαρακτήρα στην λίστα και επιπλέον

μετράμε τον χαρακτήρα στην μεταβλητή token_length, ώστε να γνωρίζουμε μελλοντικά, αν η συμβολοσειρά υπερβαίνει ή όχι το επιτρεπόμενο πλήθος χαρακτήρων . Αν στην αρχική κατάσταση δούμε + ή – τότε για εμάς η κατάσταση, γίνεται addOperator, προσθέτουμε το αριθμητικό σύμβολο στην λίστα που θα επιστρέψουμε(μόνο του). Επειδή η κατάσταση, αν δούμε ένα εκ των δύο συμβόλων γίνεται τελική, στο element_type θα βάλουμε την κατάσταση addOperator, και ύστερα για να σπάσουμε το loop , state=”token”(Το πως επιστρέφουμε κάθε κομμάτι που αναγνωρίζει ο λεκτικός αναλυτής, θα το περιγράψουμε, μετά από όλα τα if, αφού είναι κοινός για όλα ο τρόπος) . Στην περίπτωση που η κατάσταση είναι αρχική και έρθει το * ή το / , πάλι η κατάσταση θα γίνει τελική, θα δώσουμε τύπο mullOperator και θα ακολουθήσουμε την αντίστοιχη διαδικασία με από πάνω. Αν από state=”start” δούμε κάποιο χαρακτήρα που ανήκει στα groupSymbol, τότε πάμε πάλι σε κατάσταση token, ενημερώνουμε τον τύπο στοιχείου και επιστρέφουμε. Αν αρχίσουμε και πάρουμε τον χαρακτήρα ; ή , τότε έχουμε πάλι τελική κατάσταση, ενημερώνουμε τον τύπο του στοιχείου, το προσθέτουμε στην λίστα για να το επιστρέψουμε και βγαίνουμε από την επανάληψη. Τώρα στις ειδικές περιπτώσεις , που ξεκινήσουμε από αρχή και πάρουμε < ή = ή >, δεν βγαίνουμε από την επανάληψη, καθώς μπορεί να έχουμε επιτρεπόμενους συνδυασμούς τους στο μέλλον. Έτσι απλά ενημερώνουμε αντίστοιχα τους τύπους τους, που δεν είναι εντελώς σίγουρα τελικοί και απλά προσθέτουμε όποιο από τα τρία σύμβολα στην λίστα, περιμένοντας να πράξουμε, ανάλογα με τον επόμενο χαρακτήρα που θα έρθει. Ακόμη αν στην αρχική κατάσταση έρθει ο χαρακτήρας : , μεταβαίνουμε στην κατάσταση asgn , τον προσθέτουμε στην λίστα και περιμένουμε να έρθει και το =. Αν δούμε #, αυτό σημαίνει ότι πάμε σε κατάσταση rem και δεχόμαστε όλα τα κομμάτια μέχρι να κλείσουν τα σχόλια. Τέλος μεταβαίνουμε εκτός επανάληψης, αν από την αρχική δούμε finalOperator (.) ή ‘ ‘. Στην

περίπτωση της τελείας ενημερώνουμε και τον τύπο του στοιχείου. Ακόμη στην περίπτωση που από κατάσταση start, δεν δούμε καμία από τις περιπτώσεις που περιέγραψα από πάνω, τότε οδηγούμαστε στο συμπέρασμα ότι υπάρχει λάθος, καθώς ο χαρακτήρας που είδαμε δεν θα ανήκει στην γλώσσα Cimple. Αμέσως μετά, θα περιγράψουμε τον κώδικα από το elif state==”digit” και κάτω. Μετά από αυτό το στιγμιότυπο του κώδικα, μας μένει ένα ακόμη μικρό κομμάτι.

Αν η κατάσταση είναι digit λοιπόν, όσο έρχονται χαρακτήρες που είναι αριθμοί, τους δεχόμαστε παραμένοντας στην ίδια κατάσταση, εκτός από την περίπτωση που ξεπεράσουν τα αριθμητικά όρια που θέσαμε εξ’ αρχής. Τότε αναγκαζόμαστε να βγάλουμε value error. Στην περίπτωση που από αυτή την κατάσταση διαβάσουμε χαρακτήρα, ο οποίος δεν είναι ψηφίο, τότε οδηγούμαστε σε τελική κατάσταση, ενημερώνουμε τον τύπο του στοιχείου που θα επιστρέψουμε, ωστόσο είναι αναγκαίο να μεταφερθούμε 1 χαρακτήρα πίσω στο αρχείο, ώστε να μην τον «φάμε» , καθώς θα ανήκει στο επόμενο στοιχείο που θα επιστρέψουμε. Αν ο χαρακτήρας που διαβάσαμε είναι αλλαγή γραμμής, τότε οφείλουμε να γυρίσουμε πίσω μία γραμμή στον μετρητή γραμμών, που τον χρησιμοποιούμε για να δείχνουμε σε ποια γραμμή υπάρχει error . Επίσης, όταν βρισκόμαστε στο state=idk, όσο διαβάζουμε χαρακτήρα μεταξύ των επιτρεπόμενων που αναφερθήκαμε και προηγουμένως, τον προσθέτουμε στην λίστα και αυξάνουμε τον μετρητή, ώστε να ελέγξουμε αν ξεπεράσαμε το όριο των 30 χαρακτήρων. Στο σενάριο που έχουμε διαβάσει συνεχόμενα περισσότερους από 30 επιτρεπόμενους χαρακτήρες, σηκώνουμε Value error, διαφορετικά αν δεν γίνει τίποτα από όσα είπαμε, τότε μπαίνουμε στην τελική

κατάσταση, ενημερώνουμε τον τύπο ως keyword και πριν επιστρέψουμε με την file.seek(file.tell()-1), γυρίζουμε έναν χαρακτήρα πίσω και στην περίπτωση που ο χαρακτήρας ήταν αλλαγή γραμμής αφαιρούμε 1 από τον μετρητή γραμμών par. Στην κατάσταση asgn, περιμένουμε να δούμε = για να συμπληρωθεί το := , ώστε να ενημερώσουμε τον τύπο σε assignment, να μπούμε σε τελική κατάσταση token και να επιστρέψουμε. Σε οποιοδήποτε άλλο σενάριο εμφανίζουμε Value error. Στην περίπτωση που βρισκόμαστε ήδη στην κατάσταση smaller, αν δούμε = ή < , το προσθέτουμε στην επιστρεπτέα λίστα, περνάμε σε τελική κατάσταση token, ενημερώνοντας παράλληλα τον τύπο σε relOperator. Σε διαφορετικό ενδεχόμενο γυρίζουμε απλά ένα από τα δύο λογικά σύμβολα, γυρνώντας πίσω κατά ένα σύμβολο, από αυτά που διαβάσαμε από το αρχείο.Αν το σύμβολο που δεν καταναλώσαμε είναι η αλλαγή γραμμής, τότε αφαιρούμε 1 από τον μετρητή γραμμών. Αν state=larger τότε κάνουμε ότι ακριβώς κάναμε και για το state smaller, με την μόνη διαφορά ότι μετά το > περιμένουμε μόνο = αντί για ένα από τα δύο πιθανά σύμβολα που περιμέναμε πριν. Αμέσως πιο κάτω παραθέτω το τελευταίο κομμάτι του λεκτικού αναλυτή.

Στο ενδεχόμενο που βρισκόμαστε στην κατάσταση των σχολίων rem ,όσο διαβάζουμε σύμβολα διαφορετικά της αλλαγής γραμμής, παραμένουμε στην ίδια κατάσταση και τα προσθέτουμε στην επιστρεπτέα λίστα. Διαφορετικά, ενημερώνουμε τον τύπο του στοιχείου (άτυπα), προσθέτουμε από μόνοι μας # και απλά καθαρίζουμε την λίστα, χωρίς να γυρίσουμε κάτι και επιστρέφουμε στην αρχική κατάσταση start. Αν ο χαρακτήρας που διαβάσουμε είναι αλλαγή γραμμής τότε προσθέτουμε 1 στον μετρητή γραμμών.Αν η κατάσταση είναι τελική token, ανεξαρτήτως από πως προήλθε στο lineNo αναθέτουμε την τιμή του μετρητή γραμμών, που θα είναι και ο αριθμός της γραμμής, που θα ανήκει το στοιχείο που θα

επιστρέψουμε. Έξω από την επανάληψη, με το “”.join(x) in reservedWord στην ουσία κάνουμε ενωμένη λέξη το περιεχόμενο της λίστας, χωρίς τα , και ελέγχουμε αν η λέξη ανήκει στην λίστα των δεσμευμένων λέξεων, που τη δηλώσαμε στις πρώτες γραμμές του κώδικα μας. Τέλος πριν επιστρέψουμε το στοιχείο, την γραμμή που ανήκει και τον τύπο του, κάνω έναν έλεγχο ο οποίος αφορά τον ενδιάμεσο κώδικα, με το flag, ώστε προγράμματα που περιέχουν συναρτήσεις, να μην δημιουργείται αρχείο .c . Σε σχόλια επίσης , υπάρχει ο κώδικας ο οποίος τυπώνει με τη βοήθεια του format, καθαρά, όλα όσα επιστρέφει ο λεκτικός αναλυτής, δηλαδή το στοιχείο, την γραμμή που βρέθηκε και τον τύπο του.

Συντακτικός Αναλυτής

Σε αυτό το σημείο θα περιγράψουμε τις συναρτήσεις, του Συντακτικού Αναλυτή. Επειδή τυγχάνει, αυτό το κομμάτι της εργασίας, μετράει τις περισσότερες γραμμές κώδικα, για να ολοκληρωθεί σε φυσιολογικό μέγεθος η αναφορά, δεν θα περιγράψουμε τόσο εκτενώς τον κώδικα όπως στον λεκτικό αναλυτή. Θα σταθούμε ωστόσο, στην ουσία των συναρτήσεων που φτιάξαμε. Αρχικά να πούμε ότι σχεδόν σε όλες τις συναρτήσεις του συντακτικού αναλυτή έχουμε τις global μεταβλητές lex, line,e_type που αντιπροσωπεύουν το

στοιχείο, την γραμμή που βρίσκεται καθώς και τον τύπο του. Ξεκινώντας λοιπόν στη συνάρτηση **program(),** περιμένουμε να δούμε το αλφαριθμητικό program και αφού καλέσουμε τον λεκτικό αναλυτή, αμέσως μετά ο τύπος του στοιχείου που πρέπει να δούμε είναι keyword(όνομα προγράμματος). Καλούμε ξανά τον λεκτικό αναλυτή και την συνάρτηση block, και αν όλα πάνε καλά, αφού ολοκληρωθεί η block, πρέπει να δούμε . . Αν κάπου εμφανιστεί κάτι άλλο τότε εμφανίζουμε τα αντίστοιχα μηνύματα λάθους. Μέσα στην **block()** αν εξαιρέσουμε τις συναρτήσεις που αφορούν τον πίνακα συμβόλων και τις τετράδες, δεν κάνουμε τίποτα άλλο σε επίπεδο συντακτικού αναλυτή, από το να καλέσουμε τις απαραίτητες συναρτήσεις που ζητάτε και επιστρέφουμε. Στη συνάρτηση **declarations(),** περιμένουμε να δούμε την λέξη declare, και ακολούθως δηλώσεις. Καλούμε την varlist και περιμένουμε να δούμε ερωτηματικό, ενώ σε διαφορετική περίπτωση εμφανίζουμε μήνυμα λάθους. Στη συνάρτηση **varlist()**, αναμένουμε να δούμε στοιχείο τύπου keyword, καλούμε τον λεκτικό αναλυτή και όσο βλέπουμε την ακολουθία , και στοιχείο τύπου keyword, τότε δεν εμφανίζουμε μήνυμα λάθους και στο τέλος επιστρέφουμε. Αμέσως μετά ακολουθεί η συνάρτηση **subprograms()** που στην ουσία διαχειρίζεται τα υποπρογράμματά μας. Αν δει function ή procedure καλεί την **subprogram()** . Η subprogram περιμένει να λάβει από τον λεκτικό αναλυτή την λέξη function ή procedure και αν αυτό συμβεί, αμέσως μετά θέλει να δει το σύμβολο (, καλεί την formal_par_list και αν όλα πάνε καλά, τότε πρέπει να κλείσει η παρένθεση και καλείται η block. **Η formal_par_list()** καλεί με τη σειρά της

formal_par_item() και όσο βλέπει ‘,’ , καλεί τον λεκτικό αναλυτή και την **formal_par_item().** Η προαναφερθείσα συνάρτηση διαχειρίζεται τα in και inout μέσα στις παρενθέσεις. Αν δει ένα από τα δύο, αμέσως μετά περιμένει ένα στοιχείο τύπου keyword, ενώ σε κάθε άλλη περίπτωση εμφανίζονται τα αντίστοιχα μηνύματα σφαλμάτων. Αργότερα ακολουθεί η συνάρτηση **statements()**, η οποία τσεκάρει για το σύμβολο {, μετά την κλήση του λεκτικού αναλυτή καλεί την statement() και περιμένει κλείσιμο του προηγούμενου συμβόλου, προκειμένου να επιστρέψει. Αν αντί για { στην αρχή δεί οτιδήποτε άλλο, τότε καλεί την statement και αν όλα κυλίσουν ομαλά περιμένει ; . Η συνάρτηση **statement()** στην ουσία ελέγχει αν ο τύπος του στοιχείου που μας έδωσε ο λεκτικός αναλυτής είναι keyword και καλεί την assignment_stat, ενώ σε διαφορετική περίπτωση ελέγχει το ίδιο το στοιχείο και αν ανήκει σε κάποια από τις συγκεκριμένες δεσμευμένες λέξεις μέσα στα if else, τότε καλεί την αντίστοιχη συνάρτηση και επιστρέφει. Στην περίπτωση της **assignment_stat()** γίνεται έλεγχος αν ο τύπος της λέξης είναι αλφαριθμητικό και αφού καλεστεί ο λεκτικός ελέγχει αν η επόμενη λέξη είναι το σύμβολο := . Αν κάτι τέτοιο ισχύει επιστρέφουμε, διαφορετικά εμφανίζουμε τα απαραίτητα μηνύματα λάθους. Ακολούθως μετά η συνάρτηση **if_stat()** αφορά τη συνθήκη του if και ελέγχει την δομή του, δηλαδή αρχικά περιμένει τη λέξη if, αργότερα την αριστερή παρένθεση, καλεί την συνάρτηση condition και αν κλείνει η παρένθεση καλείται η statements, η else_part και επιστρέφει. Η **else_part(),** είναι πολύ πιο απλή, καθώς περιμένει να δει την δεσμευμένη λέξη else, καλεί τον λεκτικό

,την statements() και σε κάθε περίπτωση επιστρέφει. Μία συνάρτηση η οποία βοηθάει στο έλεγχο του while είναι η **while_stat(),** η οποία αρχικά αν δει την λέξη while, αμέσως μετά περιμένει την αριστερή παρένθεση και την συνθήκη. Αν όλα πάνε καλά και στο τέλος κλείσει και η παρένθεση, τότε επιστρέφουμε, διαφορετικά είμαστε υποχρεωμένοι να ενημερώσουμε με μήνυμα συντακτικού λάθους. Ακολούθως στον κώδικά μας βρίσκουμε τη συνάρτηση **switchcase_stat().** Αυτή η συνάρτηση αν δει τη δεσμευμένη λέξη που τη χαρακτηρίζει, αμέσως μετά σε περίπτωση που δει το case, μπαίνει σε loop για όσο παίρνει συνολικά case(condition). Βέβαια δεν παραβλέπουμε το κίνδυνο για λάθος όσον αφορά τις παρενθέσεις και γι'αυτό κάνουμε και ελέγχους.Αν δει χωρίς πρόβλημα και δεξιά παρένθεση καλείται η statements(). Στην περίπτωση που δει default, καλείται ο λεκτικός αναλυτής και αμέσως μετά η συνάρτηση statements(). Σε διαφορετική περίπτωση θα εμφανιστεί μήνυμα σφάλματος. Μία δομή επίσης είναι η **forcase_stat().** Στην αρχή περιμένει να λάβει τη λέξη forcase και αν αυτό γίνει, ύστερα όσο λαμβάνει case(καλεί condition) και καλεί statements, σημαίνει ότι όλα πάνε καλά . Αν αυτό σταματήσει και δει τη λέξη default, τότε μετά τον λεκτικό αναλυτή θα καλέσει την συνάρτηση statements(). Σε κάθε άλλη περίπτωση περιγράφεται αναλυτικά στα μηνύματα, το λάθος στην σύνταξη. Η **incase_stat()** για να μην πλατιάζουμε, έχει ακριβώς την ίδια συμπεριφορά σε επίπεδο συντακτικού αναλυτή με την προηγούμενη, με τη μόνη διαφορά ότι δεν υπάρχει default σε αυτή την περίπτωση. Ακόμα μία συνάρτηση στον συντακτικό αναλυτή

μας είναι η **return_stat(),** η οποία περιμένει να δει τη λέξη return, ακολούθως (,καλεί expression,) και στο τέλος αφού καλέσει τον λεκτικό αναλυτή επιστρέφει, εκτός και αν έχει πάει κάτι στραβά πιο πάνω. Λίγες γραμμές κώδικα πιο κάτω βρίσκεται η **print_stat() .** Αυτή η συνάρτηση μόλις κληθεί, περιμένει να δει τη λέξη print. Μόλις αυτό γίνει, για να μην υπάρξει πρόβλημα, είναι αναγκαίο να δει με τη σειρά (, να κληθεί η expression και τελικά να κλείσει η παρένθεση. Όπως σε κάθε άλλη συνάρτηση φυσικά, έχουμε έτοιμα μηνύματα λάθους στην περίπτωση συντακτικής αστοχίας. Επόμενη συνάρτηση του συντακτικού αναλυτή, είναι η **input_stat().** Σε αυτή τη συνάρτηση, αρχικά περιμένουμε ο λεκτικός αναλυτής να γυρίσει τη λέξη input, αμέσως μετά να υπάρχει παρένθεση και ενδιάμεσα πριν κλείσει η δεξιά παρένθεση, να δει ένα στοιχείο τύπου keyword. Αν όλα αυτά γίνουν με επιτυχία, τότε δεν επιστρέφει λάθος. Η συνάρτηση **actual_par_list()** καλεί τη συνάρτηση actual_par_item() και όσο η επόμενη λέξη από αυτή που η έχει λάβει η actual_par_item είναι , , τότε συνεχίζει να καλεί τον λεκτικό αναλυτή και την **actual_par_item()**. Η προαναφερθείσα συνάρτηση μόλις καλείται θέλει να λάβει από τον λεκτικό αναλυτή που κάλεσε η προηγούμενη τη λέξη in ή inout, διαφορετικά πρέπει να μας εμφανίζει μήνυμα λάθους. Αν τελικά λάβει όντως in , τότε καλεί τον λεκτικό και την expression στην περίπτωση του in .Στην περίπτωση του inout, καλεί τον λεκτικό, αν όμως δεν έρθει λέξη τύπου keyword, τότε εμφανίζεται error , διαφορετικά ξανακαλεί τον λεκτικό αναλυτή.Η συνάρτηση **bool_factor()** πρέπει να δει τη λέξη not και αν αυτό γίνει, τότε καλείται ο λεκτικός και

περιμένει για [, καλεί την condition() και περιμένει να λάβει],για να καλέσει τον λεκτικό ξανά και να τελειώσει. Μία άλλη περίπτωση, είναι αντί στην αρχή να λάβει not, να λάβει [, πάρει λέξη, η οποία να είναι] και να κλείσει καλώντας τον λεκτικό αναλυτή. Η συνάρτηση **expression**, καλεί την optional_sign() και την term(). Όσο η λέξη που παίρνει είναι τα σύμβολα – ή +, τόσο καλεί την add_oper() και την term(). Η **term()** στην περίπτωση που δει * , / καλεί την mul_oper και την **factor()**. Η τελευταία αν δει τύπο ψηφίου, καλεί τον λεκτικό, ενώ αν δεί (, καλεί τον λεκτικό, την expression και φυσικά περιμένει δεξιά παρένθεση. Αν πάλι στην αρχή αντί για τύπος με ψηφία δοθεί, τύπος με αλφαριθμητικό, τότε καλείται η idtail, αφού πρώτα κληθεί ο lex_analyzer. Στην **idtail() ,** αν αρχικά έρθει (, τότε αμέσως μετά ξανακαλεί για να λάβει τόκεν και αφού το λάβει, καλεί την actual_par_list.Τέλος περιμένει κλείσιμο της δεξιάς παρένθεσης. Η **optional_sign()** περιμένει – ή + για να καλέσει την add_oper(). Ακόμη η συνάρτηση **rel_op()** ελέγχει τα λογικά-σχεσιακά σύμβολα και ανάλογα ποιο απ’όλα είναι, το καταχωρεί στην μεταβλητή rel_token και το επιστρέφει. Αν κάποιο δεν ανήκει στην γλώσσα, τότε επιστρέφεται μήνυμα λάθους. Από την άλλη πλευρά η **add_oper()** , ελέγχει αν το σύμβολο είναι + ή -, το καταχωρεί σε μία μεταβλητή την οποία επιστρέφει και φυσικά καλεί τον λεκτικό. Τέλος η **mull_**oper τσεκάρει να δει αν η λέξη είναι * ή /, αποθηκεύει το σύμβολο, το επιστρέφει και καλεί τον λεκτικό αναλυτή. Τέλος καλείται η program και επιστρέφει με return ο συντακτικός.

Ενδιάμεσος Κώδικας

Σε αυτό το κομμάτι θα αναφερθούμε στο τι κάνουν οι συναρτήσεις του ενδιάμεσου κώδικα, που καλούνται και στον συντακτικό. Επειδή πριν περιγράψαμε καθαρά τι κάνει ο συντακτικός, δεν τις αναφέραμε μέσα στις συναρτήσεις του. Οπότε θα πούμε εδώ τι κάνει η κάθε μία όταν καλείται. Η **nextquad()** όπου καλείται επιστρέφει τον κωδικό της τετράδας, που θα παραχθεί. Η **genquad** κατασκευάζει την επόμενη τετράδα (5αδα μαζί με τον κωδικό στην αρχή). Αυξάνει τον num_quad κατά 1, προσθέτει την τετράδα, στον πίνακα όλων των τετράδων και επιστρέφει την τετράδα που μόλις κατασκεύασε. Η **newtemp()** όταν καλείται, δημιουργεί μία νέα προσωρινή μεταβλητή Tk, συνενώνοντας το T με έναν αριθμό και εκτός από το γεγονός ότι το επιστρέφει το προσθέτει και στην λίστα όλων των προσωρινών μεταβλητών. Η **emptylist()** κατασκευάζει μία νέα κενή λίστα, για τετράδες. Επιπλέον η συνάρτηση **makelist** φτιάχνει μία λίστα τετράδων που περιέχουν μόνο το όρισμα, το οποίο θα δοθεί και την επιστρέφει. Η **merge** συγχωνεύει τις δύο λίστες που δίνονται σαν όρισμα και τις επιστρέφει σε μία λίστα. Στην **backpatch,** η λίστα που δίνεται σαν όρισμα, αποτελείται από κωδικούς οι οποίοι δείχνουν συγκεκριμένες τετράδες στην quads_list, στις οποίες ο τελευταίος τελεστής είναι κενός. Έτσι με την συνάρτηση αυτή, ψάχνουμε αυτές τις τετράδες, για να τις συμπληρώσουμε με το δεύτερο όρισμα.Όπου βρίσκουμε _, τότε συμπληρώνουμε και ολοκληρώνουμε την τετράδα. Συγκεκριμένα στις συναρτήσεις

expression,term,factor,assignment_stat,return_stat,print_stat,input_stat,bool_term,bool_factor,if_stat,condition,while_stat,call_stat,idtail,actual_par_item,block κτλπ, που περιγράψαμε πιο πριν, χωρίς να αναφέρουμε κάτι για τις συναρτήσεις του ενδιάμεσου που βρίσκονται μέσα, δημιουργούνται στην ουσία οι τετράδες του ενδιάμεσου κώδικα όπως είχε ζητηθεί στις διαφάνειες.Η **write_intc,** χρησιμεύει στο να κατασκευάσουμε το αρχείο .int που μας ζητήθηκε για τον ενδιάμεσο. Η **conv_c,** το αρχείο .c. Βέβαια δεν φτιάχνεται τίποτα χωρίς να καλέσουμε την **create_c() ,** η οποία ανοίγει τα αρχεία με τις επιθυμητές καταλήξεις, στο αρχείο .c από επιλογή μας θα γραφτούν στην αρχή 2 βιβλιοθήκες, και μετά από τροποποίηση στην declarations και οι δηλώσεις. Για να λειτουργήσει όμως ο ενδιάμεσος πρέπει να καλέσουμε και τον συντακτικό αναλυτή, καθώς οι συναρτήσεις του ενδιάμεσου χρησιμοποιήθηκαν στις συναρτήσεις του συντακτικού, όπου κάναμε αναφορά από πάνω. Τέλος αφού γράψουμε αυτά που θέλουμε στα αρχεία μας, τα κλείνουμε. Η **check_create** έχει δημιουργηθεί από εμένα ώστε με ένα flag να κάνει έλεγχο αν μέσα στο αρχείο υπάρχει η λέξη procedure ή function και να μη δημιουργεί αρχείο .c .Το λάθος μου εδώ είναι ότι τα αρχεία .c και .int δημιουργούνται στην create_c() , επομένως όποτε το flag ήταν 1, δεν δημιουργούταν ούτε αρχείο .int . Η printlist(): είχε δημιουργηθεί για να τυπώνονται οι λίστες.

Πίνακας Συμβόλων

Στην περίπτωση του κώδικα συμβόλων είναι πολύ πιο εύκολη η κατανόηση, αν το κάνουμε με αντικειμενοστρέφεια, καθώς σε αυτή την περίπτωση έχουμε να κάνουμε με σχήματα, που κάθε σχήμα έχει ορισμένα πεδία. Έχουμε την κλάση **Argument**, η οποία έχει τα στοιχεία name,parMode και type. Παρότι δεν μας ζητείται ο τύπος, εγώ τον έβαλα int σε όλα, αφού μόνο int έχουμε, ώστε όταν τυπώνεται ο πίνακας, να φαίνονται καλύτερα(όχι μισά να έχουν και άλλα μισά όχι τύπο). Έχουμε την κλάση **Entity,** η οποία έχει στοιχεία το όνομα, τον τύπο, τη μεταβλητή, την παράμετρο, την προσωρινή μεταβλητή και το υποπρόγραμμα. Τα 4 τελευταία είναι κλάσεις εντός της Entity, με δικά τους υποπεδία. Η μεταβλητή έχει το offset, ο τύπος της πάντοτε θα είναι int, ενώ η παράμετρος θα έχει το mode και το offset. Η προσωρινή μεταβλητή όπως προείπαμε, πάντα θα είναι ακέραια. Τέλος το SubProgram, περιέχει τη λίστα με τα arguments, την startQuad, το frameLength και τον τύπο. Η κλάση **scope**, περιέχει την λίστα από τα Entities, το βάθος φωλιάσματος, το όνομα και το δείκτη στο scope που περιέχει. Η **new_arg** κατασκευάζει ένα νέο argument, όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης και το κάνει append στο argumentlist. Με τη **new_entity**, προσθέτουμε ένα entity στην αντίστοιχη λίστα, όταν συναντάμε δήλωση μεταβλητής, προσωρινή μεταβλητή, δήλωση καινούριου υποπρογράμματος ή δήλωση τυπικής παραμέτρου συνάρτησης. Αμέσως μετά έχουμε τη **new_scope**, την οποία την καλούμε και στην

block. Αυτή στην ουσία δημιουργεί ένα νέο κύκλο scope με τα χαρακτηριστικά που περιγράψαμε προηγουμένως. Αν το topScope είναι διαφορετικό του none(έτσι το είχαμε αρχικοποιήσει) τότε αυξάνουμε κατά ένα το nestingLevel, διαφορετικά το θέτουμε 0. Όσον αφορά την **delete_scope(),** την καλώ στο τέλος της block, προκειμένου να κάνει διαγραφή scope. Η **compute_offset()**, μετράει τον αριθμό των Entities, εκτός από τα υποπρογράμματα. Οι compute_StarQuad() και compute_framelength() υπολογίζουν αντίστοιχα, σύμφωνα με τις διαφάνειες, αυτά που υποδηλώνουν και τα ονόματά τους, τα οποία θα τα χρησιμοποιήσουμε μόλις τυπώσουμε τον πίνακα συμβόλων. Το τύπωμα συμβόλων γίνεται με τη **Symbol_table(),** μέσα στην οποία βέβαια περνάμε σαν global μεταβλητή τη cFs , η οποία μας δίνει τη δυνατότητα, με τη χρήση της write, ότι τυπωθεί στην οθόνη, να τυπωθεί και σε αρχείο txt. Προκειμένου να γίνει αυτό, δημιουργούμε με πρωτοβουλία μας την **create_symbol_table(),** μέσα στην οποία ανοίγει το αρχείο, για να γίνουν η εγγραφές και το κλείνει. Φυσικά καλεί και την check_create, γιατί δίχως αυτή, δεν θα γινόταν να τρέξει το πρόγραμμα μας, ακόμη και στην περίπτωση που καλούσαμε την create_symbol_table(), όπως και κάνουμε στο τέλος.