ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

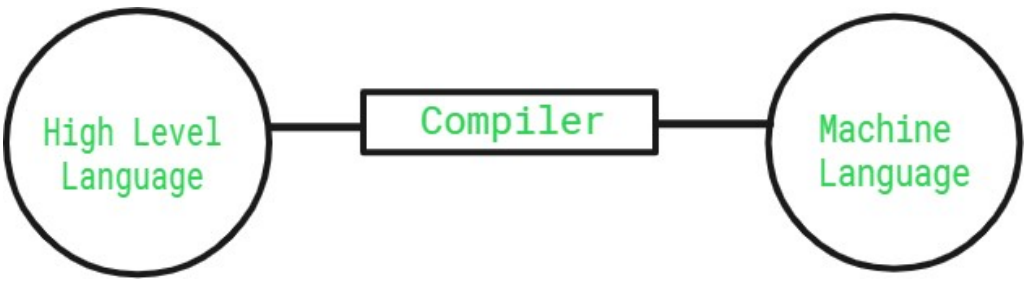
ΜΑΘΗΜΑ: ΜΕΤΑΦΡΑΣΤΕΣ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2023-2024

ΟΜΑΔΑ

ΑΛΕΞΑΝΔΡΟΣ-ΔΗΜΗΤΡΙΟΣ ΤΖΙΜΟΓΙΑΝΝΗΣ (Α.Μ.: 4179)

ΓΕΩΡΓΙΟΣ ΤΡΙΑΝΤΟΣ (Α.Μ.: 4184)



Σκοπός της άσκησης είναι η δημιουργία ενός μεταγλωττιστή για τη γλώσσα cpy, μία μικρή γλώσσα προγραμματισμού η οποία αντλεί ιδέες και δομές από τη γλώσσα Python.

Οι φάσεις της μεταγλώττισης είναι οι ακόλουθες:

- Λεκτική ανάλυση
- Συντακτική ανάλυση
- Ενδιάμεσος κώδικας
- Πίνακας Συμβόλων/Σημασιολογική ανάλυση
- Τελικός κώδικας

ΑΡΧΙΚΕΣ ΠΑΡΑΤΗΡΗΣΕΙΣ ΓΙΑ ΤΟΝ ΚΩΔΙΚΑ:

- Έχουμε δημιουργήσει μια κλάση TokenType η οποία κληρονομεί από την κλάση Enum της Python, στην οποία δίνουμε τιμή για κάθε τύπο του εκάστοτε token που συναντά ο λεκτικός αναλυτής και αναγνωρίζει (π.χ. ανβρούμε το σύμβολο "/" (SLASH), τότε έχει την τιμή 18).

- Λαμβάνοντας βοήθεια από την κλάση TokenType, αρχικοποιήσαμε ένα dictionary "tokens", το οποίο έχει σαν keys όλες τις δεσμευμένες λέξεις της cpy καθώς και όλα τα tokens που μπορούμε να συναντήσουμε (εξαιρούνται τα identifiers καθώς είναι λέξεις που επιλέγονται από το συντάκτη του αρχείου cpy, τα οποία αναγνωρίζονται ως identifiers).

- Φτιάξαμε επίσης μια συνάρτηση errorMessage, η οποία καλείται όταν ο μεταγλωττιστής εντοπίσει λάθος στον κώδικα του αρχείου που θέλουμε να μεταγλωττίσουμε. Η συνάρτηση αυτή επιστρέφει στο τερματικό ε΄να κείμενο με το λάθος και το σημείο που βρίσκεται (γραμμή-χαρακτήρας) καιτέλος τερματίζει τη μεταγλώττιση με την εντολή sys.exit(0).

ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

Η λεκτική ανάλυση αποτελεί την αρχική φάση της μεταγλώττισης. Παίρνει ωςείσοδο το πηγαίο πρόγραμμα σε γλώσσα cpy διαβάζοντας κάθε χαρακτήρα ξεχωριστά και παράγονται οι λεκτικές μονάδες (Lexemes) οι οποίες περνάνε στη συντακτική ανάλυση για έλεγχο.

Ο λεκτικός αναλυτής είναι μία συνάρτηση (lex()) η οποία διαβάζει κάθε χαρακτήρα ξεχωριστά από το αρχικό πρόγραμμα και επιστρέφει την επόμενηλεκτική μονάδα που θα αναγνωρίσει. Σε περίπτωση που βρει κάποιο σφάλματότε καλείται η errorMessage() η οποία επιστρέφει το μήνυμα του συγκεκριμένου λάθους καθώς και το σημείο που βρίσκεται.

Το αλφάβητο της cpy αποτελείται από τα παρακάτω:

- Όλα τα γράμματα του λατινικού αλφάβητου (κεφαλαία, μικρά)
- Αριθμητικά ψηφία (0-9)
- Σύμβολα αριθμητικών πράξεων (+, -, *, //, %)
- Τελεστές συσχέτισης (<, >, !=, <=, >=, ==)
- Σύμβολο ανάθεσης (=)
- Διαχωριστές (“,”, :)
- Σύμβολα ομαδοποίησης ((,), #{, #})
- Δεσμευμένες λέξεις (main, def, #def, #int, global, if, elif, else, while, print, return, input, int, and, or, not)

ΠΑΡΑΤΗΡΗΣΕΙΣ:

- Οι ακέραιες σταθερές πρέπει να έχουν τιμές από $-(2^{32} - 1)$ έως $2^{32} - 1$.
- Οι δεσμευμένες λέξεις της γλώσσας δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές.
- Τα identifiers (συμβολοσειρές που επιλέγει ως όνομα μεταβλητής ή συνάντησης ο συντάκτης του πηγαίου προγράμματος) έχουν μήκος το πολύ 30 χαρακτήρων, αν ξεπεραστεί αυτό το όριο, τότε θεωρείται εσφαλμένο.
- Οι λευκοί χαρακτήρες (tab, space) αγνοούνται αρκεί να μην είναι μέσα σε δεσμευμένες λέξεις ή identifiers.

ΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ ΤΗΣ ΛΕΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ

Αρχικοποιούμε μια μεταβλητή lexeme σαν μια κενή συμβολοσειρά και ένακενό πίνακα bufferList, ο οποίος θα παίρνει ένα ένα τους χαρακτήρες που διαβάζει η lex() από το πηγαίο πρόγραμμα. Ανάλογα το χαρακτήρα που βρίσκει η συνάρτηση πηγαίνουμε σε μία συγκεκριμένη κατάσταση (state) μέχρι να φτάσουμε στην κατάσταση OK όπου αναγνωρίζεται η επόμενη λεκτική μονάδα και περνάει στη συντακτική ανάλυση.
Κάθε φορά που η κατάσταση δεν είναι στο OK, διαβάζουμε τον επόμενο χαρακτήρα και αναλόγως τι είναι αυτός προχωράμε στην επόμενη κατάσταση.

Όσον αφορά τη boolean μεταβλητή filePointerMove, η οποία αρχικοποιείται ως false στην αρχή της lex(), αυτή έχει να κάνει με τους χαρακτήρες μετά από ένα lexeme. Για παράδειγμα, αν έχουμε μία συμβολοσειρά τύπου identifier και ο επόμενος χαρακτήρας είναι το σύμβολο ανάθεσης (=), τότε επιστρέφουμε τη συμβολοσειρά χωρίς το = και ταυτόχρονα πηγαίνουμε μία θέση πίσω (ένα χαρακτήρα πίσω) στο κείμενο (καθώς επίσης διαγράφουμε το τελευταίο element του bufferList με την τελευταία συμβολοσειρά) έτσι ώστε να ελέγξουμε το σύμβολο ανάθεσης την επόμενη φορά που θα κληθεί η lex().Αυτό γίνεται μέσω της εντολής testFile.seek(testFile.tell() - 1).

Επίσης, μέσω της εντολής lexeme = ‘’.join(bufferList), περνάμε στο lexeme τη συμβολοσειρά του πίνακα χωρίς κενά (π.χ. αν είχαμε bufferList = [‘h’, ‘e’, ‘l’, ‘l’, ‘o’], τότε το lexeme θα γινόταν “hello”) και, τέλος, η συνάρτηση επιστρέφει ένα αντικείμενο τύπου Token με tokenType, τον τύπο του εκάστοτε lexeme (π.χ. identifier), tokenValue, την τιμή του(π.χ. “hello”), καθώς και το σημείο που βρίσκεται στο αρχικό πρόγραμμα (σειρά και χαρακτήρας).

Ιδιαίτερη αναφορά πρέπει να γίνει στον έλεγχο της δίεσης (#). Αν η lex() βρει δίεση τότε περνάμε στο state = 6 και εκεί ελέγχει αν ο επόμενος χαρακτήραςείναι “{“ ή “}”. Αν ναι, τότε περνάμε στην κατάσταση OK και επιστρέφει το lexeme. Επειδή όμως υπάρχουν οι δεσμευμένες λέξεις #def και #int, οφείλουμε να ελέγξουμε τον επόμενο χαρακτήρα αν δεν είναι “{“ ή “}”. Αν δεν είναι το γράμμα “d” ούτε το γράμμα “i”, τότε επιστρέφει μήνυμα λάθους. Αν όμως είναι “d” ή “i”, τότε περνάμε σε μια άλλη κατάσταση που μας ενδιαφέρει μόνο η λέξη #def ή #int αντίστοιχα και εκεί πρέπει να ελέγξουμε αν ο επόμενος χαρακτήρας είναι το “e” ή το “n” αντίστοιχα.
Συνεχίζουμε με τον ίδιο τρόπο για κάθε γράμμα και αν όντως η ακολουθία των χαρακτήρων είναι #def ή #int, τότε μεταβαίνουμε στην κατάσταση OK και το επιστρέφει. Αν όμως βρεθεί κάποιος άλλος χαρακτήρας ενδιάμεσα, η συνάρτηση θα εμφανίσει μήνυμα σφάλματος και η μεταγλώττιση θα τερματιστεί.

ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

Μετά τη λεκτική ανάλυση ακολουθεί η συντακτική ανάλυση. Κατά τη φάση της συντακτικής ανάλυσης ελέγχεται αν η ακολουθία των lexemes που επιστρέφονται από τη lex() είναι νόμιμη, δηλαδή ακολουθεί τη γραμματική της cpy. Όταν μία ακολουθία δεν είναι σωστή τότε η συντακτική ανάλυση καλεί την errorMessage() η οποία επιστρέφει συντακτικό λάθος και τερματίζεται η μεταγλώττιση.

ΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ ΤΗΣ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ

Τώρα, θα εξηγήσουμε την υλοποίηση κάθε συνάντησης που αφορά τη φάσητης συντακτικής ανάλυσης.

- **startRule():** Πρόκειται για την εναρκτήρια συνάρτηση της φάσης, η οποία δημιουργεί το αρχείο του ενδιάμεσου κώδικα και καλεί πρώτη φορά τη λεκτική ανάλυση (lex()) ώστε να αναγνωριστεί η πρώτη λεκτική μονάδα του πηγαίου προγράμματος. Μετά καλούμε με τη σειρά τις global_check() και όσο η επόμενη λεκτική μονάδα είναι #def, καλούμε τη def_function(). Αφού βγούμε από τη λούπα, καλείται η def_main_function(). Τέλος, σε αυτό το σημείο θα πρέπει η μεταγλώττιση να έχει ολοκληρωθεί σωστά και θα ψάξει η συντακτική ανάλυση να βρει EOF (end of file), κάτι που αν δε γίνει θα εμφανιστεί μήνυμα λάθους.
- **global_check():** Η συνάρτηση αυτή είναι υπεύθυνη για την αναγνώριση των πρώτων μεταβλητών στην αρχή του προγράμματος, οι οποίες, αν υπάρχουν, αρχικοποιούνται με το "#int …var_name…# και είναι global μεταβλητές.
- **function_globals_check():** Η συνάρτηση αυτή ελέγχει αν οι global μεταβλητές που ορίζονται μέσα σε απλές (και όχι main) συναρτήσεις, έχουν οριστεί πάνω από τις συναρτήσεις με το #int.
- **main_globals_check():** Ίδια με την παραπάνω, για συνάρτηση main.
- **def_main_function():** Σε αυτή τη συνάρτηση έχουμε βρει το lexeme από τη startRule() και ελέγχουμε αν είναι "#def". Αν ναι, πρόκειται για main συνάρτηση και στη συνέχεια καλούμε κατά σειρά τις declarations(), main_global_check(), statements().
- **def_function():** Κάθε φορά που βρίσκουμε τη λεκτική μονάδα def, καλείται η συνάρτηση αυτή. Μετά περιμένουμε να δούμε κατά σειρά "identifier(var_list): #{". Στη συνέχεια καλείται η declarations() και η function_globals_check(). Μετά, όσο βρίσκουμε το def, ξανακαλούμε αναδρομικά τη συνάρτηση και αφού τελειώσει η δομή επανάληψης, καλείται η statements().
- **declarations():** Κάθε φορά που βρίσκουμε το token "#int" καλούμε τη declaration_line().
- **declaration_line():** Ορίζουμε ένα σύνολο από declarations, τα οποία

υπάγονται στα identifiers με τη βοήθεια της id_list().

- **statements():** Καλείται και μπάινει σε λούπα όσο βρίσκει σύμβολο από την ακόλουθη λίστα: identifier, print, return, if, while.
- **statement():** Αν το token είναι if ή while καλείται η structured_statement(), αλλιώς καλείται η simple_statement().
- **simple_statement():** Αν το token είναι identifier, τότε καλούμε την input_stat(), αν έχουμε το print καλούμε την print_stat(), αλλιώς έχουμε το return και καλούμε τη return_stat().
- **structured_statement():** Ίδια δομή με τη simple, αλλά τώρα έχουμε τις if_stat() και while_stat().
- **input_stat():** Έχουμε δύο περιπτώσεις, η πρώτη θα ελέγξει αν μετάτο σύμβολο ανάθεσης έχουμε τη συμβοσειρά "= int(input())", ενώ αν η πρώτη λεκτική μονάδα μετά το σύμβολο ανάθεσης ("=") δεν είναι "int" τότε θα καλέσουμε την expression().
- **print_stat():** Καλούμε την expression() αφού βεβαιωθούμε ότι περικλείεται από παρενθέσεις.
- **return_stat():** Ακριβώς ίδια δομή με την print_stat(). Έχουμε μία ειδική περίπτωση αν ο χρήστης θέλει να επιστρέψει αρνητικό αριθμό (π.χ. return -7), οπότε ελέγχουμε αν υπάρχει παύλα και κατόπιν αριθμός μετά το return.
- **if_stat():** Καλούμε στην αρχή την condition, και μετά όσο βρίσκουμε and ή or ξανακαλούμε την condition(). Μετά, ελέγχουμε αν το επόμενο token είναι το colon. Αν ναι, καλούμε τη statements(). Ομοίως λειτουργούμε για το elif και, τέλος, για το else αφού ολοκληρωθεί το block του if.
- **while_stat():** Παρόμοια δομή με το block της if (κατά σειρά condition(), colon, έλεγχος για #{, statements(), έλεγχος για #}) .
- **id_list():** Λίστα με τους identifiers χωρισμένους με κόμμα.
- **expression():** Ορίζουμε αριθμητικές παραστάσεις. Καλούμε την optional_sign() και την term() και μπαίνουμε σε λούπα όσο βρίσκουμε μέσω της λεκτικής ανάλυσης τους τελεστές πράξεων + και -. Μέσα στη λούπα καλούμε την ADD_OP() καθώς και πάλι την term() για το 2ο μέρος της αριθμητικής παράστασης.
- **term():** Καλούμε τη factor() και όσο βρίσκουμε μέσω της λεκτικής ανάλυσης τους τελεστές *, // και %, και μπαίνουμε σε λούπα όπου καλείται η MUL_OP() και η factor() για το δεύτερο μέρος της παράστασης.

- **factor():** Εντοπίζουμε ένα παράγοντα της αριθμητικής έκφρασης ο οποίος μπορεί να είναι είτε σταθερός αριθμός (θετικός ή αρνητικός), είτε expression() που περικλείεται από παρενθέσεις, είτε ένας identifier, ανάλογα τη λεκτική μονάδα που επιστρέφεται από την τελευταία κλήση της lex().
- **id_tail():** Καλούμε την actual_par_list() ανάμεσα από παρενθέσεις.
- **actual_par_list():** Καλούμε την expression() και όσο μετά έχουμε κόμμα, ξανακαλούμε την expression().
- **optional_sign():** Καλείται όταν βρίσκουμε τους τελεστές + ή -.
- **condition():** Ορίζουμε μια λογική παράσταση από λογικούς και σχεσιακούς τελεστές, η οποία θα αποτιμηθεί ως true ή false. Καλούμε επαναληπτικά τη bool_term().
- **bool_term():** Φτιάχνει μια λογική παράσταση από παράγοντες που χωρίζονται από τον τελεστή and. Καλούμε επαναληπτικά τη bool_factor().
- **bool_factor():** Παράγουμε μια έκφραση (expression) η οποία είναι του τύπου not condition(), είτε υπάρχει σχεσιακός τελεστής (REL_OP()), τότε καλούμε 2 φορές την expression(), μία για το κάθε μέρος της λογικής έκφρασης.
- **ADD_OP():** Αναγνωρίζονται και επιστρέφονται οι τελεστές + και -.
- **REL_OP():** Το ίδιο για * , // και %.

ΣΗΜΕΙΩΣΗ:

- Κάθε φορά που βρίσκουμε identifier σε κάποια συνάρτηση, τοποθετούμε το όνομα της μεταβλητής σε μία λίστα localVariables ή localGlobalVariables (ανάλογα με το αν είναι global ή τοπική μεταβλητή) και όσο διατρέχουμε τον κώδικα της συνάρτησης, αν βρούμε άλλο identifier, ψάχνουμε στις παραπάνω λίστες αν έχει ήδη αρχικοποιηθεί, ώστε στην περίπτωση που ο χρήστης χρησιμοποιήσει μια μεταβλητή, η οποία δεν έχει αρχικοποιηθεί, να πετάει μήνυμα σφάλματος ο μεταφραστής μας. Τέλος, όταν βγαίνουμε από τη συνάρτηση, καθαρίζουμε τη συνάρτηση με τις τοπικές μεταβλητές, ώστε να ξαναγεμίσει την επόμενη φορά που θα βρει ο μεταφραστής μας νέα συνάρτηση.

ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

ΒΟΗΘΗΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΟΝ ΠΙΝΑΚΑ ΣΥΜΒΟΛΩΝ:

- **add_new_entity(entity):** Πρόσθεση νέας εγγραφής (entity) στον πίνακα συμβόλων.
- **add_new_scope():** Πρόσθεση νέου επιπέδου.
- **delete_scope():** Διαγραφή επιπέδου.
- **update_entity(entity, startingQuad, framelength):** Ενημέρωση υπάρχουσας εγγραφής.
- **add_formal_par(name, datatype, mode):** Προόσθεση τυπικής παραμέτρου.
- **search_entity(name):** Αναζήτηση υπάρχουσας εγγραφής με όνομα name στον πίνακα συμβόλων. Αν δε βρεθεί, επιστρέφεται μήνυμα σφάλματος.

ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

ΒΟΗΘΗΤΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΟΝ ΕΝΔΙΑΜΕΣΟ ΚΩΔΙΚΑ:

- **next_quad():** επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
- **gen_quad(op, x, y, z):** δημιουργεί την επόμενη τετράδα (op, x, y, z)
- **newtemp():** δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή της μορφής T_1, T_2 κτλ.
- **emptylist():** δημιουργεί μία κενή λίστα ετικετών τετράδων
- **makelist(x):** δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x
- **mergelist(list1, list2):** δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών list1, list2
- **backpatch(list1, z):** η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

ΠΑΡΑΤΗΡΗΣΕΙΣ ΓΙΑ ΤΟΝ ΕΝΔΙΑΜΕΣΟ ΚΩΔΙΚΑ:

- Δημιουργήσαμε μία κλάση Quad που αρχικοποιεί τετράδες με τα πεδία (op, x, y, z). Υλοποιήσαμε επίσης και τη συνάρτηση __str__ στην κλάση αυτή ώστε να μπορούμε να τοποθετούμε τις τετράδες στο αρχείο καταγραφής ως string.
- Προσθέσαμε τις απαραίτητες εντολές στις συναρτήσεις της συντακτικής ανάλυσης σύμφωνα με τις διαφάνειες του ενδιάμεσου κώδικα, όπως για παράδειγμα στην def_function() τις ακόλουθες εντολές:
  gen_quad("begin_block",name,"_","_")
  gen_quad("halt","_","_","_")
  ----statements----
  gen_quad("end_block",name,"_","_")
- Φτιάξαμε, τέλος, μια βοηθητική συνάρτηση generate_int_code_file(), η οποία διατρέχει τη λίστα με τις υπάρχουσες τετράδες και τις γράφει σε αρχείο με αύξουσα ταμπέλα (label).
- Παρατηρούμε ότι βγάζει σε κάποιες τετράδες λανθασμένο 4ο στοιχείο (z). Πιθανώς να υπάρχει κάποιο ζήτημα στην εντολή gen_quad("=", e_place, "_", tok) που βρίσκεται στο τέλος της συνάρτησης input_stat() (γραμμή 615 του κώδικα), λόγω του tok που υποθέτουμε ότι παίρνει λάθος όρισμα η global μεταβλητή tok που ορίσαμε.
- Συνοπτικά, λειτουργεί κατά μεγάλο ποσοστό σωστά (π.χ. begin_block, αριθμητικές εκφράσεις κτλ) με εξαίρεση το στοιχείο z που αναφέρθηκε ακριβώς από πάνω.

ΠΑΡΑΤΗΡΗΣΕΙΣ ΓΙΑ ΤΟΝ ΠΙΝΑΚΑ ΣΥΜΒΟΛΩΝ:

- Υλοποιήσαμε τις βοηθητικές συναρτήσεις, αλλά είχαμε errors κατά την προσπάθεια να προσθέτουμε νέα entities (μας έβγαζε out of range λάθος καθώς διατρέχαμε τη λίστα με τα entities). Οπότε βάλαμε σε σχόλια τις εντολές που χρησιμοποιήσαμε για την υλοποίηση του πίνακα συμβόλων (π.χ. γραμμή 564 στη συνάρτηση simple_statement(): add_new_entinty(token.tokenValue)).