MYY802- Μεταφραστές
Εαρινό Εξάμηνο
Αθανάσιος Μασούρας-2752

Προγραμματιστική Άσκηση - cutePy

1 Γενικές πληροφορίες

Στην εργασία μας ζητήθηκε η ανάπτυξη ενός compiler για την γλώσσα προγραμματισμού cutePy. Παρακάτω παρουσιάζονται πληροφορίες για την cutePy, οι υλοποιήσεις, περιγραφή του κώδικα και αποτελέσματα.

1 Γλώσσα cutePy

Η γλώσσα προγραμματισμού cutePy είναι μια εκπαιδευτική γλώσσα που θυμίζει τη γλώσσα Python, αλλά είναι αρκετά πιο απλή. Δεν υποστηρίζει όλες τις δομές που υποστηρίζει η Python. Τα χαρακτηριστικά της είναι :

- ακέραιοι αριθμοί
- τελεστές και εκφράσεις
- αριθμητικές πράξεις
- σχόλια
- δομές: if, while
- συναρτήσεις, μετάδοση παραμέτρων με τιμή
- αναδρομικές κλήσεις
- είσοδος και έξοδος δεδομένων
- κανόνες εμβέλειας
- φώλιασμα

2 Εκτέλεση

Για την εκτέλεση του compiler απλα πληκτρολογούμε στον τερματικό το παρακάτω.

```
$ python3 cutePy_2752.py program.cpy
```

Όπου program.cpy είναι οποιοδήποτε πρόγραμμα σε cutePy.

Μετά την εκτέλεση του προγράμματος θα παραχθούν αρχεία με τα αποτελέσματα με κάθε φάση της μετάφρασης :

- lex_tokens.txt : λεκτικός αναλυτής
- int_code.int : ενδιάμεσος κώδικας
- symbol_table.txt : πίνακας συμβόλων
- final_code.asm : τελικός κώδικας

3 Υλοποιήσεις

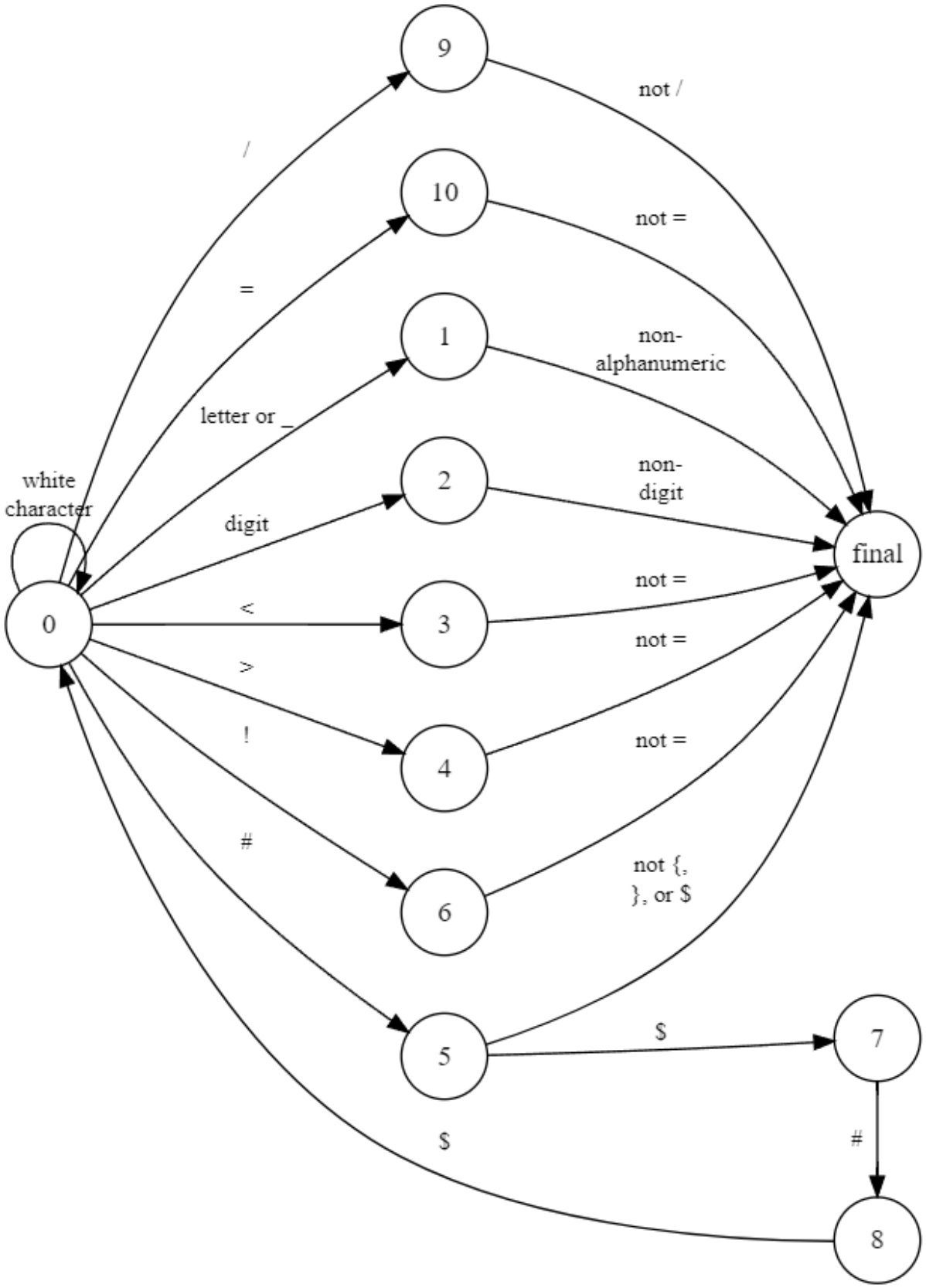
Στο παραδοτέο υπάρχει

- το `cutePy_2752.py` που περιέχει την υλοποίηση σχετικά με την λεκτική ανάλυση, συντακτική ανάλυση, ενδιάμεσος κώδικα, πίνακα συμβόλων και τον τελικό κώδικα.

3.1 Λεκτικός Αναλυτής

Η λεξική ανάλυση και tokenization εκτελείται στην συνάρτηση lex(). Η συνάρτηση διαβάζει τον κώδικα, χαρακτήρα προς χαρακτήρα και αναγνωρίζει τα tokens με βάση ορισμένους κανόνες που ορίζονται από τη μηχανή κατάστασης. Η συνάρτηση ελέγχει πρώτα αν η τρέχουσα κατάσταση είναι προσωρινή και στη συνέχεια διαβάζει τον επόμενο χαρακτήρα από το αρχείο. Στη συνέχεια προσθέτει τον χαρακτήρα στη λίστα 'word' και καθορίζει την επόμενη κατάσταση με βάση τους κανόνες. Εάν η τρέχουσα κατάσταση είναι τελική κατάσταση, το αναγνωρισμένο token περνά στη συνάρτηση 'create_token' και δημιουργείται το αντικείμενο και η λίστα 'word' αδειάζει. Εάν ο χαρακτήρας λευκός χαρακτήρας, ο τελευταίος χαρακτήρας αφαιρείται από τη λίστα 'word' και εάν ο χαρακτήρας είναι νέα γραμμή, ο αριθμός της γραμμής αυξάνεται.Η συνάρτηση χειρίζεται επίσης ειδικούς χαρακτήρες και σχόλια εντός του κώδικα. Συνολικά, εκτελεί το έργο του εντοπισμού και της εξαγωγής token από το αρχείο εισόδου, το οποίο αποτελεί το πρώτο βήμα στη διαδικασία μεταγλώττισης μιας γλώσσας προγραμματισμού.

Διάγραμμα κατάστασης



3.1.1 Λεκτικός Αναλυτής - Κλάση

Για τον λεκτικό αναλυτή χρησιμοποιούμε την **κλάση Token()**, με τρία χαρακτηριστικά:

- **family** : η κατηγορία που ανήκει το token
- **value** : η τιμή του token
- **line** : η γραμμή στην οποία διαβάστηκε

3.1.2 Λεκτικός Αναλυτής - Συναρτήσεις

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του λεκτικού αναλυτή:

- lex() : εδώ βρίσκεται η βασική λογική του συντακτικού αναλυτή που περιγράφηκε παραπάνω
- create_token(word,line) : δημιουργεί το αντικείμενο token
- check_id(word, line) : ελέγχει αν ένα αναγνωριστικό είναι σύμφωνο με τους κανόνες

3.1.3 Λεκτικός Αναλυτής - Δομές

Η μοναδική δομή που χρησιμοποιήθηκε είναι το λεξικό :

```
tokens_dict = {
      '+' : 'TOKEN_plus',
      '-' : 'TOKEN_minus',
      '*' : 'TOKEN_times',
      '//' : 'TOKEN_divide',
      '<' : 'TOKEN_less',
       ...
}
```

Το λεξικό tokens_dict μας βοηθάει να αναγνωρίσουμε σε ποια κατηγορία ανήκει το token που θα δημιουργηθεί.

3.2 Συντακτικός Αναλυτής

Μετά την φάση της λεκτικής ανάλυσης, ακολουθεί η συντακτική ανάλυση, κατά την οποία ελέγχεται εάν η ακολουθία των λεκτικών μονάδων που δημιουργείται από τον λεκτικό αναλυτή αποτελεί μια συμβατή ακολουθία βάσει της γραμματικής της γλώσσας. Οποιαδήποτε ακολουθία που δεν αναγνωρίζεται από τη γραμματική θεωρείται μη συμβατή και οδηγεί στον εντοπισμό συντακτικού σφάλματος. Ο σχεδιασμός του κώδικα βασίζεται στην γραμματική της cutePy. Οπότε για κάθε γραμματικό κανόνα - δομή έχουμε υλοποιήσει και μια συνάρτηση. Ο συντακτικός αναλυτής ξεκινάει με την συνάρτηση parser(), όπου καλούμε για πρώτη φορά τον lex() ο οποίος θα παράξει την πρώτη λεκτική μονάδα. Στην συνέχεια εκτελείται η συνάρτηση start_rule(), δηλαδή ο πρώτος συντακτικός κανόνας. Με βάση την κατηγορία που ανήκει η λεκτική μονάδα καλούνται και οι αντίστοιχες συναρτήσεις των κανόνων γραμματικής και στην συνέχεια καλείται πάλι ο lex() για να καταναλώσουμε την επόμενη λεκτική μονάδα. Σε γενικές γραμμές έχουμε μια ακολουθία καλέσματος της συνάρτησης lex() και των συναρτήσεων. Επίσης να σημειωθεί ότι μέσα την δομή του συντακτικού αναλυτή υπάρχουν και συναρτήσεις που βοηθούν στην υλοποίηση του ενδιάμεσου κώδικα και του πίνακα συμβόλων, αλλά θα εξηγηθούν αργότερα.

3.2.1 Συντακτικός Αναλυτής - Συναρτήσεις

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του συντακτικού αναλυτή:

- parser()
- start_rule()
- def_main_part()
- def_main_function()
- def_function()
- declarations()
- declaration_line()
- statements()
- statement()
- simple_statement()
- structured_statement()
- assignment_stat(operand3)
- print_stat()
- return_stat()
- if_stat()
- while_stat()
- id_list(type)
- expression()
- term()
- factor()
- idtail()

- actual_par_list()
- optional_sign()
- condition()
- bool_term()
- bool_factor()
- call_main_part()
- main_function_call()
- error(line, missing_token)

Η δομή όλων των παραπάνω είναι η ίδια. Πολλα εμφωλευμένα if και κάποια while που ελέγχουν την κατηγορία του token. Αν η κατηγορία είναι σύμφωνη τότε καταναλώνουμε την επόμενη λεκτική μονάδα και προχωράμε στον επόμενο if.

Για παράδειγμα:

```
def declarations():
     while(token.family == "TOKEN_hashtag"):
          lex()
          if token.family == "TOKEN_declare":
               lex()
               declaration_line()
          else:
               error(token.line, "declare")
```

Γραμμή μέσα σε αρχείο .cpy

```
#declare x
```

3.3 Ενδιάμεσος Κώδικας

Μέσα στις συναρτήσεις του συντακτικού κώδικα έχουμε και τις συναρτήσεις για την δημιουργία του ενδιάμεσου κώδικα. Αποτελείται από μια σειρά τετράδων που είναι αριθμημένες με μια ετικέτα και αποτελούνται από έναν τελεστή και τρία τελούμενα. Οι ετικέτες χρησιμοποιούνται για να καθορίσουμε τη σειρά των τετράδων, με κάθε τετράδα να αναφέρεται στην επόμενή της.Οι τετράδες αποτελούνται από τέσσερα στοιχεία, με τον τελεστή να καθορίζει τη δράση που θα εκτελεστεί και τα τελούμενα να αποτελούν τα αντικείμενα πάνω στα οποία θα εφαρμοστεί η δράση. Παρόλο που ο τελεστής μπορεί να χρειάζεται λιγότερα από τρία τελούμενα, για απλότητα θεωρούμε ότι υπάρχουν πάντα τρία τελούμενα. Αν ο τελεστής χρειάζεται λιγότερα από τρία τελούμενα, τα αδιάφορα τελούμενα θεωρούνται κενά. Σε αντίθετη περίπτωση που χρειαζόμαστε περισσότερα τελούμενα, χρησιμοποιούνται τεχνικές όπως της προσωρινής μεταβλητής, του merge και του

backpatch.Ο ενδιάμεσος κώδικας που παράγεται στη συνέχεια θα χρησιμοποιηθεί ως είσοδος για τη φάση παραγωγής του τελικού κώδικα. Παρακάτω γίνεται αναφορά των περιπτώσεων που γίνεται μετατροπή σε quad και σε ποια σημεία του συντακτικού αναλυτή γίνεται αυτή η μετατροπή.

Η πρώτη δημιουργία quad που βλέπουμε στον συντακτικό κώδικα έχει να κάνει με την ομαδοποίηση εντολών ενδιάμεσων κώδικα.

- begin_block, name, _, _
- end_block, name, _, _

Δημιουργούνται στις συναρτήσεις του συντακτικού κώδικα που έχουν να κάνουν με την main και τις function, συγκεκριμένα :

- def_main_function()
- def_function()
- call_main_part()

Καλούνται στα σημεία πριν και μετά την ανάπτυξη των μπλοκ των παραπάνω δομών

Μετά θα αναφερθούμε στις περιπτώσεις που έχουν να κάνουν με την εκχώρηση τιμών, την εκτύπωση δεδομένων και την επιστροφή δεδομένων.

- := , source, _, target
- in, x, _, _
- out, x, _, _
- ret, source, _, _

Η δημιουργία των quads τους γίνονται αντίστοιχα στις συναρτήσεις:

- assignment_stat(operand3)
- print_stat()
- return_stat()

Και στις 3 περιπτώσεις στην σύνταξη τους υπάρχει και η συνάρτηση expression(), που με την σειρά του καλεί τις συναρτήσεις term() και factor(). Όλες αυτές οι συναρτήσεις καλύπτουν τις περιπτώσεις που για παράδειγμα γίνεται εκχώρηση σε μια μεταβλητή μια ολόκληρη συμβολοσειρά ή μια μεγάλη αριθμητική πράξη αριθμών και μεταβλητών. Οπότε με την βοήθεια των προσωρινών μεταβλητών παράγουμε τον ενδιάμεσο κώδικα για όλες αυτές τις περιπτώσεις. Εδώ θα πρέπει να αναφέρουμε ότι αυτές οι πράξεις που αναφέρθηκαν γίνονται με τα quads που σχετίζονται με τις αριθμητικές πράξεις

- op, operand1, operand2, target

Οι επόμενες περιπτώσεις αφορούν τις δομές if και while. Η ιδιαιτερότητα τους βρίσκεται στις λογικές συνθήκες. Πρέπει να καλύπτονται όλα τα αποτελέσματα των λογικών παραστάσεων, ώστε να γίνονται τα σωστά άλματα μεταξύ των τετράδων. Οι δύο θεμελιώδης τετράδες των δύο δομών δηλαδή είναι:

- op, operand1, operand2, label (όπου op = <, > , == ,κτλ)
- jump , _, _, labe

Η δημιουργία των quads γίνεται αντίστοιχα στις συναρτήσεις

- if_stat()
- while_stat()

που με την σειρά τους καλούν τις συναρτήσεις condition(), bool_term(), bool_factor(). Σε αυτές τις συναρτήσεις δημιουργείται ο ενδιάμεσος κώδικας για τις σύνθετες λογικές παραστάσεις Απαραίτητη είναι η χρήση της βοηθητικής συνάρτησης backpatch ώστε να έχουμε σωστά άλματα στις τετράδες ανάλογα το αποτέλεσμα των συνθηκών.

Μία ακόμα περίπτωση για τον ενδιάμεσου κώδικα είναι η το κάλεσμα συναρτήσεων. Μέσα στην συνάρτηση factor() παράγονται οι τετράδες :

- par, name, mode, _
- call, name, _, _

όπου στην πρώτη δημιουργούμε μια temp μεταβλητή που θα αποθηκεύσει την τιμή στην περίπτωση που η συνάρτηση επιστρέφει κάτι.

Τέλος στην συνάρτηση actual_par_list() παράγονται οι τετράδες για το πέρασμα παραμέτρων μια συνάρτησης.

- par, operand1, cv, _

3.3.1 Ενδιάμεσος Κώδικας - Κλάση

Για τον ενδιάμεσο κώδικα χρησιμοποιούμε την **κλάση Quad()**, με πέντε χαρακτηριστικά:

- **id**: η ετικέτα της τετράδας
- **operator** : η διαδικασία που θα εκτελεστεί
- **operand1**: τελούμενο 1 - πηγή που θα συμμετέχει στην διαδικασία
- **operand2**: τελούμενο 2 - πηγή που θα συμμετέχει στην διαδικασία
- **operand3**: τελούμενο που αποθηκευτεί το αποτέλεσμα

3.3.2 Ενδιάμεσος Κώδικας - Συναρτήσεις

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του ενδιάμεσου κώδικα:

- gen_quad(operator, operand1, operand2, operand3): δημιουργεί το αντικείμενο quad
- next_quad(): επιστρέφει την ετικέτα του επόμενου quad
- empty_list(): επιστρέφει άδεια λίστα
- make_list(label): δημιουργεί μια νέα λίστα με ένα μόνο στοιχείο, το οποίο είναι η ετικέτα
- merge(list1, list2): συνδέει δύο λίστες σε μια και την επιστρέφει
- backpatch(list, label): αντικαθιστά τον τρίτο τελεστέο όλων των quad στη δεδομένη λίστα με τη δοθούσα ετικέτα.
- new_temp(): δημιουργεί μια νέα προσωρινή μεταβλητή με μοναδικό όνομα.

3.4 Πίνακας Συμβόλων

Το επόμενο σημαντικό μέρος της μετάφρασης της γλώσσας είναι ο πίνακας συμβόλων. Ο πίνακας συμβόλων αντιπροσωπεύει μια δυναμική δομή δεδομένων, όπου αποθηκεύονται πληροφορίες που σχετίζονται με τα συμβολικά ονόματα που χρησιμοποιούνται σε ένα πρόγραμμα κατά τη μεταγλώττισή του. Αυτή η δομή ακολουθεί δυναμικά τη διαδικασία της μεταγλώττισης, προσαρμόζοντας το περιεχόμενό της καθ' όλη τη διάρκεια της διαδικασίας με την προσθήκη ή αφαίρεση πληροφοριών ανάλογα με τις απαιτήσεις. Έτσι, σε κάθε στιγμή της μεταγλώττισης, ο πίνακας περιλαμβάνει ακριβώς τις πληροφορίες που απαιτούνται για τη συγκεκριμένη στιγμή. Παρακάτω είναι μια σύντομη αναφορά για το που καλούνται οι συναρτήσεις του πίνακα συμβόλων και ποιες περιπτώσεις καλύπτει.

Το πρώτο πράγμα που γίνεται σε αυτή την φάση είναι να δημιουργούνται τα scopes. Εδώ έχει γίνει μια διαφοροποίηση σε σχέση με την θεωρία επειδή δεν μπορούσα να το υλοποιήσω αλλιώς. Ενώ κανονικά το πρώτο scope έπρεπε να είναι για την main του προγράμματος, σαν αρχικό scope έχω την εκάστοτε main_function καθώς δεν υπάρχει σύνδεση μεταξύ main function. Αυτή η διαφοροποίηση έγινε επειδή σύμφωνα με την ροή που έχει ο συντακτικός αναλυτής η main του προγράμματος διαβάζεται τελευταία και δεν μπορούσα να βρω τρόπο ώστε να δημιουργώ το scope του σωστά.

Οπότε σύμφωνα με αυτή την παραμετροποίηση τα επίπεδα δημιουργούνται κάθε φορά που αναπτύσεται μια function ή main function, δηλαδή μέσα στις συναρτήσεις:

- def_function()
- def_main_function()

Μέσα σε αυτές τις συναρτήσεις του συντακτικού αναλυτή με την add_new_scope() δημιουργείται νέο επίπεδο ενώ με την remove_scope() αφαιρείται το scope αφού έχει

τελειώσει η μετάφραση της συνάρτησης. Ανάμεσα σε αυτές τις δύο συναρτήσεις δημιουργούνται τα διαφορετικής κατηγορίας αντικείμενα entities και την δυναμική ενημέρωση των χαρακτηριστικών τους.

Αφού έχει δημιουργηθεί το scope, σειρά έχει η μετάφραση των περιεχομένων των συναρτήσεων και η προσθήκη entities στα επίπεδα. Δημιουργία entity έχουμε στις περιπτώσεις που:

- δημιουργείται νέα προσωρινή μεταβλητή
- συναντάμε δήλωση μεταβλητής
- συναντάμε δήλωση νέας συνάρτησης
- συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

Η δημιουργία entity για τις προσωρινές μεταβλητές γίνεται μέσα στην new_temp() για δήλωση νέα function στις ίδιες συναρτήσεις που αναφέρθηκαν σχετικά με την δημιουργία των scopes, ενώ για τις υπόλοιπες περιπτώσεις μέσα στην id_list(type).

Πέρα όμως από την δημιουργία των scopes και των entity, έγιναν και υλοποιήσεις σχετικά με την ενημέρωση του μήκους πλαισίου αλλά και της απόστασής από την αρχή του εγγραφήματος δραστηριοποίησης, τα οποία είναι απαραίτητα για να γνωρίζουμε την εμβέλεια κάθε συνάρτησης και μεταβλητής.

3.4.1 Πίνακας Συμβόλων - Κλάση

Για τον πίνακα συμβόλων χρησιμοποιούμε διάφορες κλάσεις

Την **κλάση Scope()**, με τρία χαρακτηριστικά:

- nested_level: βάθος φωλιάσματος
- entities: λίστα με τα entities του scope
- offset: η απόστασή από την αρχή του εγγραφήματος δραστηριοποίησης

Και τις συναρτήσεις:

- add_entity(self, entity): προσθήκη entity στο scope
- get_offset(self): επιστρέφει το offset και υπολογίζει το επόμενο offset
- get_nested_level(self): επιστρέφει βάθος φωλιάσματος

Την **κλάση Variable()**, με τρία χαρακτηριστικά:

- name: το αναγνωριστικό
- datatype: ο τύπος
- offset: η απόστασή από την αρχή του εγγραφήματος δραστηριοποίησης

Την **κλάση TemporaryVariable()**, με τρία χαρακτηριστικά:

- name: το αναγνωριστικό
- datatype: ο τύπος
- offset: η απόστασή από την αρχή του εγγραφήματος δραστηριοποίησης

Την **κλάση Parameter()**, με τέσσερα χαρακτηριστικά:

- name: το αναγνωριστικό
- datatype: ο τύπος
- mode: η πληροφορία για την κατηγορία περάσματος τιμής
- offset: η απόστασή από την αρχή του εγγραφήματος δραστηριοποίησης

Την **κλάση FormalParameter()**, με τρία χαρακτηριστικά:

- name: το αναγνωριστικό
- datatype: ο τύπος
- mode: η πληροφορία για την κατηγορία περάσματος τιμής

Την **κλάση Function()**, με πέντε χαρακτηριστικά:

- name: το αναγνωριστικό
- datatype: ο τύπος
- startingQuad: το id της πρώτης τετράδας της συνάρτησης
- frameLength: το μήκος του εγγραφήματος δραστηριοποίησης της συνάρτησης σε bytes
- formalParameters: λίστα με τις παραμέτρους της συνάρτησης

Και τις συναρτήσεις:

- set_startingQuad(self, Quad): ενημερώνουμε την πρώτη τετράδα της συνάρτησης
- set_frameLength(self, frameLength): ενημερώνουμε το μήκος του εγγραφήματος δραστηριοποίησης της συνάρτησης σε bytes
- add_formalParameter(self, fParameter): προσθήκη παραμέτρου στην συνάρτηση

3.4.2 Πίνακας Συμβόλων - Συναρτήσεις

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του πίνακα συμβόλων:

- add_new_scope(): δημιουργεί νέο scope
- remove_scope(): διαγράφει το τελευταίο scope
- add_var_entity(name): προσθέτει μια entity μεταβλητή σε ένα scope.
- add_parameter_entity(name): προσθέτει μια entity παράμετρο σε ένα scope.
- add_func_entity(name): προσθέτει μια entity συνάρτηση σε ένα scope.
- search_entity(name): αναζητά ένα entity με βάση το όνομά.
- update_func_startingQuad(name): ενημερώνει την αρχική τετράδας μιας συνάρτησης.

- update_func_frameLength(name, frameLength): ενημερώνει το μήκος πλαισίου μιας entity συνάρτησης.
- add_func_formalParameter(name, func_name): προσθέτει μια τυπική παράμετρο σε μια entity συνάρτηση.

3.5 Τελικός Κώδικας

Η παραγωγή του τελικού κώδικα είναι το στάδιο της μεταγλώττισης, όπου παράγεται ο κώδικας σε γλώσσα μηχανής. Για την cutePy θα παραγάγουμε τελικό κώδικα σε συμβολική γλώσσα μηχανής (assembly code) του επεξεργαστή RISC-V. Κατά τη διάρκεια της φάσης αυτής, παράγουμε τις αντίστοιχες εντολές του τελικού κώδικα για κάθε quad του ενδιάμεσου κώδικα. Κατά την εκτέλεση αυτής της φάσης, οι μεταβλητές απεικονίζονται στη μνήμη.Αυτό σημαίνει ότι οι τιμές των μεταβλητών αποθηκεύονται σε συγκεκριμένες θέσεις μνήμης που μπορούν να προσπελαστούν από τον τελικό κώδικα.Επιπλέον, σε αυτήν τη φάση, πραγματοποιείται και το πέρασμα παραμέτρων καθώς και η κλήση συναρτήσεων.

Η παραγωγή του τελικού κώδικα ξεκινάει με την εκτέλεση της gen_asm_code(id) μέσα στις συναρτήσεις def_main_function() και def_function(). Όπως γίνεται αντιληπτό η μετατροπή των quads σε τελικό κώδικα γίνεται ανά μπλοκ και παράλληλα με την δημιουργία του πίνακα συμβόλων για να γνωρίζουμε τις απαραίτητες πληροφορίες για κάθε scope και entity. Έτσι λοιπόν για κάθε quad καλείται η quad_to_asm(quad) που παράγει τον τελικό κώδικα που αντιστοιχεί στο quad. Ανάλογα την περίπτωση χρησιμοποιούνται και οι συναρτήσεις gnlvcode(), loadvr(v,r) και storerv(r,v) που έχουν να κάνουν με την μεταφορά και ανάκτηση πληροφορίας.

3.5.1 Τελικός Κώδικας - Συναρτήσεις

Οι συναρτήσεις που χρησιμοποιήθηκαν για την υλοποίηση του τελικού κώδικα:

- gen_asm_code(id): βοηθητική συνάρτηση για μετατροπή όλων των quads σε τελικό κώδικα
- quad_to_asm(quad): μετατρέπει την τετράδα ενδιάμεσου κώδικα σε τελικό κώδικα
- get_block_function_name(): βοηθητική συνάρτηση για να δούμε το όνομα της συνάρτησης στο οποίο βρισκόμαστε κατά την παραγωγή τελικού κώδικα
- loadvr(v,r): μεταφορά δεδομένων στον καταχωρητή
- storerv(r,v): μεταφορά δεδομένων από τον καταχωρητή στη μνήμη
- gnvlcode(v): μεταφέρει στον t0 την διεύθυνση μιας μη τοπικής μεταβλητής

3.6 Αρχεία

Παρακάτω είναι οι συναρτήσεις που αφορούν την διαχείριση αρχείων :

- open_cpy_file(): ανοίγει το αρχείο με τον κώδικα σε cutePy
- open_asm_file(): δημιουργεί το αρχείο στο οποίο θα γραφεί ο τελικός κώδικας
- open_table_file(): δημιουργεί το αρχείο στο οποίο θα γραφεί ο πίνακας συμβόλων
- save_lex_tokens_to_file(): δημιουργεί αρχείο και αποθηκεύει τις λεκτικές μονάδες
- save_int_code_to_file(): δημιουργεί αρχείο αποθηκεύει τον ενδιάμεσο κώδικα
- save_symbol_table_to_file(): δημιουργεί αρχείο αποθηκεύει τον πίνακα συμβόλων

4. Τεστ Αρχεία .cpy

Μαζί με τον κώδικα του compiler, υπάρχουν και 4 αρχεία με κώδικα cutePy.

- test-noerrors.cpy: είναι το παράδειγμα των διαφανειών χωρίς error
- test-error1.cpy είναι το παράδειγμα των διαφανειών με error στο λεκτικό αναλυτή. Το λάθος βρίσκεται στην γραμμή 4, όπου γίνεται αρχικοποίηση μια μεταβλητή με συμβολοσειρά μεγαλύτερη από 30 μήκος.
- test-error2.cpy είναι το παράδειγμα των διαφανειών με error στο συντακτικό αναλυτή. Το λάθος βρίσκεται στην γραμμή 15, όπου η σύνταξη της print είναι λάθος.
- my_example.cpy είναι ένα μικρό παράδειγμα ώστε να εξηγήσω όλη την διαδικασία της μετάφρασης

4.1 Περιγραφή μετάφρασης αρχείου my_example.cpy

Παρακάτω είναι μια περιγραφή του κώδικα του μεταφραστή με βαση το αρχείο my_example.cpy.

Στην αρχή του προγράμματος διαβάζουμε το .cpy αρχείο και ανοίγουμε τα αρχεία στα οποία θα γράψουμε τα αποτελέσματα για πίνακα συμβόλων και τελικό κώδικα. Μετά ξεκινάει η μετάφραση με την εκτέλεση της parser(), η οποία εκτελεί για πρώτη φορά την lex(). Το τι γίνεται στην lex() θα εξηγηθεί μια φορά αναλυτικά. Σε γενικές γραμμές επιστρέφει την επόμενη λεκτική μονάδα.

Όλη η λογική του λεκτικού αναλυτή βρίσκεται μέσα στην while.

(172): while state in temporary_states: (174): char = file.read(1) (1)def main_check_positive(): (175): word.append(char) (177): if state == 0: (183): if char.isalpha() or char == "_": (184): state = 1	ξεκίνημα while Διαβάζουμε χαρακτήρα Κρατάμε του χαρακτήρες για να δημιουργηθεί στο τέλος η λέξη Είμαστε στο state 0 άρα μπαίνουμε στην if Μέσα στην if του (177) υπάρχουν διάφορα if που ανάλογα τον χαρακτήρα διαβάζουμε γίνεται και η ενημέρωση του state. Εδώ έχουμε διαβάζει τον χαρακτηρα d άρα πάμε σε state 1
(172): while state in temporary_states: (174): char = file.read(1) (1)def main_check_positive(): (175): word.append(char) (201): elif state == 1: (202) if not char.isalnum() and char != "_" and char != "" and char != "”":	Δεύτερη επανάληψη while Από την στιγμή που έχουμε state=1 Ο χαρακτήρας τώρα είναι το e άρα δεν μπαίνει σε αυτό το if και συνεχίζει η επόμενη while
(172): while state in temporary_states: (174): char = file.read(1) (1)def main_check_positive(): (175): word.append(char) (201): elif state == 1: (202) if not char.isalnum() and char != "_" and char != "" and char != "”":	Επόμενη επανάληψη while
(172): while state in temporary_states: (174): char = file.read(1) (1)def main_check_positive(): (175): word.append(char) (201): elif state == 1: (202): if not char.isalnum() and char != "_" and char != "" and char != "”": (203): state = final (242): if char.isspace(): (243): del word[-1]	Επόμενη επανάληψη while Τώρα διαβάσαμε το “ “ Τώρα που ο χαρακτήρας είναι το “ “, ενημερώνεται το state ως τελικό από την στιγμή που έχουμε διαβάσει την πρώτη λέξη του κώδικα Διορθώνουμε την λέξη ώστε να μην περιλαμβάνει το κενό που διαβάσαμε
(172): while state in temporary_states: (250): if state == final : (251): word = ''.join(word)	Επόμενη επανάληψη while Είμαστε σε state final Ενώνουμε τους χαρακτήρες σε μια λέξη

(252): create_token(word,line) (257): if word in tokens_dict.keys(): (258): token = Token(tokens_dict[word], word, line)	Και καλούμε την συνάρτηση για να δημιουργήσουμε το αντικείμενο token Η λέξη def είναι δεσμευμένη λέξη άρα βρίσκεται στην δομή με τα tokens, οπότε δημιουργείται το αντικείμενο με: family = TOKEN_def value = def line = 1

Η παραπάνω λογική είναι η ίδια κάθε φορά που καλούμε την συνάρτηση lex() για να πάρουμε την επόμενη λεκτική μονάδα

Αφού τελείωσε η lex(), συνεχίζουμε στην parser() και όπου θα ξεκινήσει η συντακτική ανάλυση.

(299): start_rule() (303): def_main_part() (308): while(token.family == "TOKEN_def"): (309): def_main_function() (1)def main_check_positive(): (313): if token.family == "TOKEN_def": (314): lex() (1)def main_check_positive(): (315): if token.family == "TOKEN_id": (316): name = token.value (318): add_new_scope() (877): scopes.append(Scope(0))	 Η λεκτική μονάδα είναι το def που διαβάσαμε άρα συνεχίζουμε Μετά την ακολουθία καλεσμάτων της συνάρτησης φτάνουμε στο σημείο του ελέγχου και της συνέχειας με την επόμενη λεκτική μονάδα. Αυτή η λογική υπάρχει κατά την διάρκεια όλης της συντακτικής ανάλυσης. Το επόμενο token είναι το id main_check_positive. Συνεχίζουμε με τον έλεγχο της σύνταξης. Κρατάμε το όνομα main_check_positive διότι θα μας βοηθήσει αργότερα Από την στιγμή που είμαστε σύνταξη μιας συνάρτησης πρέπει να δημιουργηθεί και το πρώτο επίπεδο. Οπότε καλείται η συνάρτηση. Δημιουργείται το πρώτο scope με τιμές: nested_level = 0 entities = άδεια λίστα για τώρα offset = 12

(319): lex() (1)def main_check_positive(): (320): if token.family == "TOKEN_leftParenthesis": (321): lex() (1)def main_check_positive(): (322): if token.family == "TOKEN_rightParenthesis": (323): lex() (1)def main_check_positive(): (324): if token.family == "TOKEN_colon": (325): lex() (2) #{	Συνεχίζει η συντακτική ανάλυση Σε αυτή την lex αναγνωρίζεται και η αλλαγή γραμμής χωρίς να επηρεάζεται το αποτέλεσμα την επόμενης λέξης
(326): if token.family == "TOKEN_left_hashbracket": (327):lex() (3) #declare x (328): declarations()	Συνεχίζουμε στον συντακτικό κανόνα για για τις αρχικοποιήσεις
(400): while(token.family == "TOKEN_hashtag"): (401): lex() (3) #declare x (402): if token.family == "TOKEN_declare": (403): lex() (3) #declare x (404): declaration_line() (410): id_list("declare")	Συνεχίζουμε στον συντακτικό κανόνα Συνεχίζουμε στον συντακτικό κανόνα δίνοντας και την πληροφορία ότι κλήθηκε από τον συνακτικό κανόνα των αρχικοποιήσεων
(604): if token.family == "TOKEN_id": (605): if type == "declare": (606): add_var_entity(token.value)	Έλεγχος για να δούμε ποιος κάλεσε την id_list(type) Στο σημείο αυτό προσθέτουμε το πρώτο entity για το επίπεδο 0. Οπότε καλούμε την συνάρτηση
(887): offset = scopes[-1].get_offset()	Υπολογίζουμε το offset που θα τοποθετηθεί στο var entity που θα προσθέσουμε στο scope
(888): scopes[-1].add_entity(Variable(name,"int", offset))	Και δημιουργείται το entity var με τιμές: name = x datatype = int offset = 12

(100): self.entities.append(entity)	Και το τοποθετούμε στην λίστα entities του scope
(610): lex() (5)def check_positive(num): (329) def_function()	Από την στιγμή που δεν έχουμε άλλο declaration επιστρέφουμε στην def_main_function και συνεχίζουμε στον επόμενο κανόνα σύνταξης.
(356): while(token.family == "TOKEN_def"): (357): lex() (5)def check_positive(num): (358): if token.family == "TOKEN_id": (360): add_func_entity(name)	Από την στιγμή που διαβάζουμε σύνταξη function, αρχικά προσθέτουμε το entity func στο scope που είμαστε δηλαδή στο 0.
(897) scopes[-1].add_entity(Function(name,"func", 0, 0, 0))	Και δημιουργείται το entity func με τιμές: name = check_positive datatype = func startingQuad = 0 frameLength = 0 formalParameters = άδεια λίστα
(100): self.entities.append(entity)	Και το τοποθετούμε στην λίστα entities του scope Ο πίνακας συμβόλων μέχρι στιγμής: scope level: 0 entities: x/12, check_positive/16,
(361): add_new_scope()	Από την στιγμή που είμαστε στη σύνταξη μιας συνάρτησης πρέπει να δημιουργηθεί και το επόμενο επίπεδο. Οπότε καλείται η συνάρτηση.
(879): scopes.append(Scope(scopes[-1].nested_level + 1)) (362): lex() (5)def check_positive(num): (363): if token.family == "TOKEN_leftParenthesis": (364): lex() (5)def check_positive(num):	Δημιουργείται το δεύτερο scope με τιμές: nested_level =1 entities = άδεια λίστα για τώρα offset = 12

(365): id_list(name)	Και περνάμε στον συντακικό κανόνα μαζί με το όνομα της συνάρτησης
(608): add_func_formalParameter(token.value,type)	Αναγνωρίζουμε ότι καλείται η id_list από συνάρτηση άρα πρέπει να προσθέσουμε την παράμετρο σαν argument στο func entity που δημιουργήσαμε προηγουμένως
(928) formal_parameter = FormalParameter(name,"int", "CV")	Οπότε δημιουργείται πρώτα το αντικείμενο FormalParameter με τις τιμές: name = num datatype = int mode = CV
(929) func, level = search_entity(func_name)	Μετά βρίσκουμε το entity της function στην οποία θα βάλουμε την παράμετρο.
(930): func.add_formalParameter(formal_parameter) (155): self.formalParameters.append(fParameter)	Στην συνέχεια καλείται η συνάρτηση του αντικειμένου για να προσθέσουμε στην λίστα του την παράμετρο.
(609): add_parameter_entity(token.value)	Πέρα όμως την προσθήκη της παραμέτρου σαν argument, πρέπει να προστεθεί και σαν entity στο scope 1 που βρισκόμαστε τωρα
(893): scopes[-1].add_entity(Parameter(name,"int","CV",offset))	Και δημιουργείται το entity Parameter με τιμές: name = num datatype = int Mode = CV Offset = 16 formalParameters = άδεια λίστα
(100)self.entities.append(entity)	Προσθήκη στο scope 1
(610): lex() (5)def check_positive(num):	Ο πίνακας συμβόλων μέχρι στιγμής: scope level: 0 entities: x/12, check_positive/16, scope level: 1 entities: num/12,
(611): while(token.family == "TOKEN_comma"):	

(366): if token.family == "TOKEN_rightParenthesis": (367): lex() (5)def check_positive(num): (368): if token.family == "TOKEN_colon": (369): lex() (6)#{ (370): if token.family == "TOKEN_left_hashbracket": (371): lex() (7) if (num > 0):	
(372): declarations()	Συνεχίζουμε τις συντακτικές συναρτήσεις με βάση τους κανόνες. Από την στιγμή όμως που δεν έχουμε declarations συνεχίζουμε στον επόμενο κανόνα
(373): def_function()	Δεν έχουμε όμως ούτε εμβόλιμη σύνταξη συνάρτησης και προχωράμε στο ίδιο block
(374): update_func_startingQuad(name)	Σε αυτό το σημείο θα ενημερώσουμε το αντικείμενο func με το ποιο θα είναι id της πρώτης του quad.
(917) quad = next_quad()	Παίρνουμε το επόμενο id της quad. Θα είναι η πρώτη quad που θα δημιουργήσουμε στο πρόγραμμα άρα έχει id = 1
(918): func, level = search_entity(name)	Βρίσκουμε το entity της συνάρτησης
(919): func.set_startingQuad(quad)	Γίνεται η ενημέρωση στο αντικείμενο
(375): gen_quad("begin_block", name, "_", "_")	Επιστρέφουμε λοιπόν στο μπλοκ του συντακτικού της συνάρτησης και δημιουργούμε το πρώτο quad
(376): start_quad_id = quad_list[-1].id	Ενημερώνουμε και μια βοηθητική μεταβλητή
(377): statements()	Συνεχίζουμε στον επόμενο συντακτικό κανόνα. Εδώ να θυμίσουμε ότι η τελευταία λεκτική μονάδα είναι το if
(421): elif token.family in ("TOKEN_if", "TOKEN_while"): (422): structured_statement()	Συνεχίζουμε στον επόμενο συντακτικό κανόνα
(434): if token.family == 'TOKEN_if': (435): if_stat() (520): lex() (7) if (num > 0):	Συνεχίζουμε στον συνακτικό κανόνα

(521): if token.family == "TOKEN_leftParenthesis": (522): lex() (7) if (num > 0):	
(523): (b_true, b_false) = condition()	Είμαστε στο σημείο που θα ελέγξουμε την λογική συνθήκη, οπότε τρέχουμε την συνάρτηση condition(). Η επιστροφή της condition έχει να κάνει με την παραγωγή των quads της if.
(714): (b_true, b_false) = (q1_true, q1_false) = bool_term()	Με την σειρά της εκτελούμε bool_term()
(726): (q_true, q_false) = (r1_true, r1_false) = bool_factor()	Και με την σειρά της την bool_factor()
(756): exp1 = expression()	Στη συνέχεια καλείται η expression() για να βρούμε το πρώτο κομμάτι της λογικής έκφρασης. Σε αυτό το σημείο υπάρχουν ακολουθία από καλέσματα συναρτήσεων συντακτικής ανάλυσης για τις περιπτώσεις σύνθετών εκφράσεων.
(626): term1 = term() (648): factor1 = factor() (676): elif token.family == "TOKEN_id": (677): factor_value = token.value (678): lex() (7) if (num **>** 0):	Εδώ η τιμή του factor_value = γίνεται num
(685): return factor_value	Καλείται και ο κανόνας idtail() αλλά από την στιγμή που έχουμε διαβάσει το > η συνάρτηση factor τελειώνει με την επιστροφή της τιμής num στο factor1
(661): return factor1	Η συνάρτηση term() τελειώνει με την επιστροφή΄της τιμής num στο term1
(643): return term1	Η συνάρτηση expression() τελειώνει με την επιστροφή΄της τιμής num στο exp1
(757): if token.value in ['==','<','>','!=','<=','>=']: (758): op = token.value (759): lex() (7) if (num **> 0**):	Επιστρέφουμε στο μπλοκ της συνάρτησης bool_factor() Κρατάμε το > για την παραγωγή quad
(760): exp2 = expression()	Και τώρα γίνεται το ίδιο πράγμα για το δεύτερο κομμάτι της λογικής έκφρασης.
(626): term1 = term() (648): factor1 = factor() (666): if token.family == "TOKEN_number":	

(667): factor_value = token.value (668): lex() (7) if (num > 0): (685): return factor_value (661): return factor1 (643): return term1	factor_value = 0 Και αφού βρήκαμε το δεύτερο κομμάτι της λογικής έκφρασης επιστρέφουμε ξανά στο block της bool_factor
(761): r_true = make_list(next_quad())	δημιουργούμε μια νέα λίστα με την επόμενη ετικέτα. Αφορά το κομμάτι των quads για τις περιπτώσεις που η λογική έκφραση είναι True
(762): gen_quad(op, exp1, exp2, "_")	Σε πρώτη φάση δημιουργεί το quad >, num, 0 _ Αφήνοντας κενό το πεδίο σχετικά με το άλμα που θα κάνουμε στην περίπτωση True
(763): r_false = make_list(next_quad())	Αφορά το κομμάτι των quads για τις περιπτώσεις που η λογική έκφραση είναι False
(764): gen_quad('jump', "_", "_", "_")	Δημιουργία quad jump χωρίς την ετικέτα προορισμού
(765): retval = (r_true, r_false) (766): return retval	Ενημέρωση retval με τις δύο λίστες Και επιστροφή στην συνάρτηση bool_term()
(733): return (q_true, q_false)	Επιστροφή στην συνάρτηση condition()
(722): return (b_true, b_false)	Επιστροφή στην συνάρτηση if_stat()
(524): if token.family == "TOKEN_rightParenthesis": (525): lex() (7) if (num > 0): (526): if token.family == "TOKEN_colon": (527): lex() (8) return(1);	
(540): backpatch(b_true, next_quad())	αντικαθιστά τον τρίτο τελεστέο όλων των quad στη δεδομένη λίστα με τη δοθούσα ετικέτα.

(541): statement() (420): simple_statement() (431): return_stat() (502): lex() (8) return(1); (503): if token.family == "TOKEN_leftParenthesis": (504): lex() (8) return(1); (505): operand1 = expression() (626): term1 = term() (648): factor1 = factor() (666): if token.family == "TOKEN_number": (667): factor_value = token.value (668): lex() (8) return(1); (685): return factor_value (661): return factor1 (643): return term1 (506): gen_quad("ret", "_","_", operand1) (507): if token.family == "TOKEN_rightParenthesis": (508): lex() (8) return(1); (509): if token.family == "TOKEN_semiColon": (510): lex() (9) else: (542): skip_list = make_list(next_quad()) (543): gen_quad("jump", "_", "_", str(next_quad()+1)) (544): backpatch(b_false, next_quad())	συνεχίζουμε στον κανόνα σύνταξης συνεχίζουμε στον κανόνα σύνταξης συνεχίζουμε στον κανόνα σύνταξης Από αυτό το σημείο ξεκινάει η ίδια ακολουθία συναρτήσεων που περιγράψαμε και παραπάνω factor_value = 1 Και επιστρέφουμε στην return_stat() όπου δημιουργούμε την quad ret , _ ,_ , 1 Επιστροφή στην if_stat() και δημιουργούμε βοηθητική λίστα Εδώ δημιουργείται το quad jump, _, _, 6 αντικαθιστά τον τρίτο τελεστέο όλων των quad στη δεδομένη λίστα με τη δοθούσα ετικέτα.

(550): if token.family =="TOKEN_else": (551): lex() (9) else**:** (552): if token.family =="TOKEN_colon": (553): lex() (10) return(0); (562): statement() (420): simple_statement() (431): return_stat() (502): lex() (10) return**(**0); (503): if token.family == "TOKEN_leftParenthesis": (504): lex() (10) return(0); (505): operand1 = expression() (626): term1 = term() (648): factor1 = factor() (666): if token.family == "TOKEN_number": (667): factor_value = token.value (668): lex() (10) return(0**)**; (685): return factor_value (661): return factor1 (643): return term1 (506): gen_quad("ret", "_","_", operand1) (507): if token.family == "TOKEN_rightParenthesis": (508): lex() (10) return(0)**;** (509): if token.family == "TOKEN_semiColon": (510): lex() (12) **#}** (563) backpatch(skip_list, next_quad()) (414): while(token.family in ('TOKEN_id', 'TOKEN_print', 'TOKEN_return', 'TOKEN_if', 'TOKEN_while')):	Factor_value = 0 Και επιστρέφουμε στην return_stat() όπου δημιουργούμε την quad ret , _ ,_ , 0 Επιστρέφουμε στην statements()

(378): gen_quad("end_block", name, "_", "_")	Επιστρέφουμε στο block της συνάρτησης def_function που σημαίνει ότι τελειώσαμε με το περιεχόμενο της οπότε δημιουργούμε το αντίστοιχο quad. Ο ενδιάμεσος κώδικας που δημιουργήθηκε QUAD: 0 :: begin_block, check_positive, _, _ QUAD: 1 :: >, num, 0, 3 QUAD: 2 :: jump, _, _, 5 QUAD: 3 :: ret, _, _, 1 QUAD: 4 :: jump, _, _, 6 QUAD: 5 :: ret, _, _, 0 QUAD: 6 :: end_block, check_positive, _, _
(379): update_func_frameLenght(name,scopes[-1].offset)	Σε αυτό το σημείο ενημερώνουμε και το framelegth του αντικειμένου της συνάρτησης που μόλις μεταφράσαμε Ο πίνακας συμβόλων μέχρι στιγμής: scope level: 0 entities: x/12, check_positive/16, scope level: 1 entities: num/12,
(381): gen_asm_code(start_quad_id)	Για τα quads που παράχθηκαν σε αυτό το block, παράγουμε και τον τελικό κώδικα
(943): for quad in quad_list[id:]: (944): quad_to_asm(quad) (949): if quad.id == 0: (950): file_asm.write(f".data\n") (951): file_asm.write(f"str_nl: .asciz '\\n' \n") (952): file_asm.write(f".text\n")	Όταν πάμε να μεταφράσουμε το πρώτο quad , φροντίζουμε να γράψουμε και τον τελικό κώδικα για την αρχή του προγράμματος.Από εδώ και πέρα για όλα τα quads ισχύουν οι περιπτώσεις του τελικού κώδικα όπως περιγράφηκαν στην θεωρία
(953):file_asm.write(f"Label_{quad.id}:\n") (1016):elif quad.operator == 'begin_block': (1017): file_asm.write(f"sw ra, 0(sp)\n")	QUAD: 0 :: begin_block, check_positive, _, _

(953):file_asm.write(f"Label_{quad.id}:\n") (988): elif quad.operator == '>': (989): loadvr(quad.operand1,'1') (1073): entity, entity_level = search_entity(v) (1074): current_level = scopes[-1].nested_level (1079):elif isinstance(entity, Parameter) and entity_level == current_level: (1080): file_asm.write(f"lw t{r} -{entity.offset}(sp)\n") (990): loadvr(quad.operand2,'2') (1070): if str(v).isdigit(): (1071): file_asm.write(f"li t{r}\n") (991): file_asm.write(f"bgt t1, t2, Label_{quad.operand3}\n")	QUAD: 1 :: >, num, 0, 3
(953):file_asm.write(f"Label_{quad.id}:\n") (954): if quad.operator == 'jump': (955): file_asm.write(f"b Label_{quad.operand3}\n")	QUAD: 2 :: jump, _, _, 5
(953): file_asm.write(f"Label_{quad.id}:\n" (968): elif quad.operator == 'ret': (969): loadvr(quad.operand3, '1') (1071): file_asm.write(f"li t{r}\n") (970): file_asm.write("lw t0, -8(sp)\n" (971): file_asm.write("sw t1, 0(t0)\n")	QUAD: 3 :: ret, _, _, 1
(953):file_asm.write(f"Label_{quad.id}:\n") (955): file_asm.write(f"b Label_{quad.operand3}\n")	QUAD: 4 :: jump, _, _, 6
(953):file_asm.write(f"Label_{quad.id}:\n") (968): elif quad.operator == 'ret': (969): loadvr(quad.operand3, '1') (1071): file_asm.write(f"li t{r}\n") (970): file_asm.write("lw t0, -8(sp)\n") (971): file_asm.write("sw t1, 0(t0)\n")	QUAD: 5 :: ret, _, _, 0
(953):file_asm.write(f"Label_{quad.id}:\n") (1018): elif quad.operator == 'end_block':	QUAD: 6 :: end_block, check_positive, _, _

(1019): file_asm.write(f"lw ra, 0(sp)\n") (1020): file_asm.write(f"jr ra\n")	
(382): remove_scope()	Από την στιγμή που τελειώσαμε την μετατροπή των quads σε τελικό κώδικα, διαγράφουμε το scope 1, δηλαδή το επίπεδο της συνάρτησης Ο πίνακας συμβόλων μέχρι στιγμής: scope level: 0 entities: x/12, check_positive/16,
(383): if token.family == "TOKEN_right_hashbracket": (384): lex() (14)x = int(input());	
(330): gen_quad("begin_block", name, "_", "_")	Από την στιγμή που δεν υπάρχει άλλη function, γυρνάμε στο block της def_main_function() και ξεκινάει η μετάφραση της
(332): statements() (415): statement() (420): simple_statement() (426): if token.family == 'TOKEN_id': (427): assignment_stat(token.value) (441): lex() (14)x = int(input());	
(442): if token.family == "TOKEN_assignment": (443): lex() (14)x = int(input());	
(444): if token.value == "int": (445): lex() (14)x = int(input());	
(446): if token.family == "TOKEN_leftParenthesis": (447): lex() (14)x = int(input());	

(448): if token.family == "TOKEN_input":
(449): lex()
(14)x = int(input());
....
(456): if token.family ==
"TOKEN_semiColon":
(457): lex()
(15) print(check_positive(x));

(458): gen_quad("in", operand3, "_", "_")
(415): statement()
(420): simple_statement()
(429): print_stat()
(484): lex()
(15) print(check_positive(x));

(485): if token.family ==
"TOKEN_leftParenthesis":
(486): lex()
(15) print(check_positive(x));

(487): operand1 = expression()
(626): term1 = term()
(648): factor1 = factor()
(676): elif token.family == "TOKEN_id":
(677): factor_value = token.value
(678): lex()
(15) print(check_positive(x));

(679): tail = idtail()
(689): if token.family ==
"TOKEN_leftParenthesis":
(690): lex()
(15) print(check_positive(x));

(691): actual_par_list()
(698): operand1 = expression()
(626): term1 = term()
(648): factor1 = factor()
(676): elif token.family == "TOKEN_id"
(677): factor_value = token.value
(678): lex()
(15) print(check_positive(x));

(685): return factor_value
(661): return factor1
(643): return term1

(699): gen_quad("par", operand1, "cv", "_") (692): if token.family == "TOKEN_rightParenthesis": (693): lex() (15) print(check_positive(x)); (694): return True (680): if tail: (681): temp_var = new_temp() (862):scopes[-1].add_entity(TemporaryVariable(tmpVar, "int",offset)) (682): gen_quad("par", temp_var, "ret", "_") (683):gen_quad("call", factor_value, "_", "_") (685): return factor_value (661): return factor1 (643): return term1 (488): gen_quad("out",operand1, "_","_") (489): if token.family == "TOKEN_rightParenthesis": (490): lex() (15) print(check_positive(x)); (491): if token.family == "TOKEN_semiColon": (492): lex() (17) #} (414): while(token.family in ('TOKEN_id', 'TOKEN_print', 'TOKEN_return', 'TOKEN_if', 'TOKEN_while')):	Δημιουργεί το quad για την παράμετρο x για κάλεσμα της συνάρτησης check_positive Επιστρέφουμε από την id_tail στην factor() Δημιουργούμε μια προσωρινή μεταβλητή για να αποθηκευτεί η τιμή που θα επιστρέψει η συνάρτηση check_positive Σε κάθε δημιουργία προσωρινής μεταβλητής, πρέπει να την προσθέσουμε σαν entity στο scope που βρισκόμαστε, δηλαδή στο 0 Το quad της μεταβλητής για την επιστροφή τιμής Το quad για το κάλεσμα της check_positive Επιστρέφουμε από την expression() στην print_stat() και δημιουργούμε το quad για το print

(333):gen_quad("end_block", name, "_", "_")	Επιστρέφουμε στην statements() Επιστρέφουμε στο block της def_main_function() παράγουμε το quad, από την στιγμή που τελειώσαμε την μετάφραση του περιεχομένου της Ο ενδιάμεσος κώδικας που δημιουργήθηκε QUAD: 7 :: begin_block, main_check_positive, _, _ QUAD: 8 :: in, x, _, _ QUAD: 9 :: par, x, cv, _ QUAD: 10 :: par, %_0, ret, _ QUAD: 11 :: call, check_positive, _, _ QUAD: 12 :: out, %_0, _, _ QUAD: 13 :: end_block, main_check_positive, _, _
(335): gen_asm_code(start_quad_id2)	Παραγωγή τελικού κώδικα για τα παραπάνω quads
(953):file_asm.write(f"Label_{quad.id}:\n") (1016): elif quad.operator == 'begin_block': (1017): file_asm.write(f"sw ra, 0(sp)\n")	QUAD: 7 :: begin_block, main_check_positive, _, _
(953):file_asm.write(f"Label_{quad.id}:\n") (959): elif quad.operator == 'in': (960): file_asm.write(f"li a7, 5\n") (961): file_asm.write(f"ecall\n") (962): loadvr(quad.operand1, '1') (1073): entity, entity_level = search_entity(v)(1074): current_level = scopes[-1].nested_level (1075): if isinstance(entity, Variable) and entity_level == current_level: (1076) file_asm.write(f"lw t{r} -{entity.offset}(sp)\n") (963): file_asm.write(f"mv t1, a0\n")	QUAD: 8 :: in, x, _, _

(953):file_asm.write(f"Label_{quad.id}:\n") (1044): elif quad.operator == 'par': (1045): if quad.operand2 == 'ret': (1049): elif quad.operand2 == 'cv': (1050): block_function = get_block_function_name() (1051): if block_function.startswith("main_"): (1052): calling_framelength = scopes[-1].offset (1056): if number_of_parameters == 0: (1057): file_asm.write(f" addi fp, sp, {calling_framelength}\n") (1058): number_of_parameters += 1 (1059): loadvr(quad.operand1, '0') (1074): current_level = scopes[-1].nested_level (1075): if isinstance(entity, Variable) and entity_level == current_level: (1076): file_asm.write(f"lw t{r} -{entity.offset}(sp)\n") (1060): file_asm.write(f"sw t0, -{12+4*number_of_parameters}(fp)\n")	QUAD: 9 :: par, x, cv, _
(953):file_asm.write(f"Label_{quad.id}:\n") (1044): elif quad.operator == 'par': (1045): if quad.operand2 == 'ret': (1046): entity, entity_level = search_entity(quad.operand1) (1047): file_asm.write(f"addi t0, sp, -{entity.offset}\n") (1048):file_asm.write(f"sw t0, -8(fp)\n")	QUAD: 10 :: par, %_0, ret, _
(953):file_asm.write(f"Label_{quad.id}:\n") (1025): elif quad.operator == 'call': (1026): number_of_parameters = 0 (1027): called_function, called_level = search_entity(quad.operand1) (1028): block_function = get_block_function_name() (1029):if block_function.startswith("main_"): (1030):calling_level = 0 (1031): calling_framelength = scopes[-1].offset (1036): if calling_level == called_level: (1037): file_asm.write(f"lw t0, -4(sp)\n") (1038): file_asm.write(f"sw t0, -4(sp)\n")	QUAD: 11 :: call, check_positive, _, _

(1041): file_asm.write(f"addi sp, sp, {calling_framelength}\n") (1042): file_asm.write(f"jal {called_function.startingQuad}\n") (1043): file_asm.write(f"addi sp, sp, -{calling_framelength}\n")	
(953):file_asm.write(f"Label_{quad.id}:\n") (964): elif quad.operator == 'out': (965): loadvr(quad.operand1, '1') (1073):entity, entity_level = search_entity(v) (1074): current_level = scopes[-1].nested_level (1077): elif isinstance(entity, TemporaryVariable) and entity_level == current_level: (1078): file_asm.write(f"lw t{r} -{entity.offset}(sp)\n") (966): file_asm.write(f"li a0,t1\n") (967): file_asm.write(f"li a7,1\n")	QUAD: 12 :: out, %_0, _, _
(953):file_asm.write(f"Label_{quad.id}:\n") (1018): elif quad.operator == 'end_block': (1019): file_asm.write(f"lw ra, 0(sp)\n") (1020): file_asm.write(f"jr ra\n")	QUAD: 13 :: end_block, main_check_positive, _, _
(336): remove_scope() (337): if token.family == "TOKEN_right_hashbracket": (338): lex() (22) if __name__ == ”__main__”:	Επιστροφή στην def_main_function() και αφού τελειώσαμε με την μετάφραση της, διαγράφουμε το επίπεδο της
(308): while(token.family == "TOKEN_def"): (304): call_main_part()	Επιστροφή στην def_main_function() Επιστροφή def start_rule() για να καλέσουμε το συντακτικό κανόνα της main. Σε αυτό το κομμάτι που έμεινε γίνονται κανονικά λεκτικές μονάδες, συντακτική ανάλυση, ενδιάμεσος κώδικας. Δεν γίνεται πίνακας συμβόλων λόγο της ιδιαιτερότητας

	που εξήγησα. Στον τελικό κώδικα γίνεται μόνο το κομμάτι με το τέλος του προγράμματος. Όταν τελειώσει και η μετάφραση της main, τα αποτελέσματα αποθηκεύονται στα διάφορα αρχεία