

MYY802:CutePy Compiler

Κωνσταντίνα Σκούρα ΑΜ:4168

Κωνσταντίνος Ρέτσας ΑΜ:4163

Περιεχόμενα

- 1.Λεκτικός αναλυτής
- 2.Συντακτικος αναλυτής
- 3.Ενδιάμεσος κώδικας
- 4.Πίνακας Συμβόλων
- 5.Τελικός κώδικας

Εισαγωγή

Στόχος της συγκεκριμένης προγραμματιστικής άσκησης ήταν να κατασκευάσουμε έναν μεταγλωττιστή, με την χρήση της γλώσσας προγραμματισμού Python, όπου θα παίρνει σαν είσοδο ένα αρχείο της γλώσσας προγραμματισμού cutePy (δηλ. αρχείο με κατάληξη .cpx) και θα το μετατρέπει σε μια μορφή που είναι κατανοητή και εκτελέσιμη από τον υπολογιστή (δηλ. αρχείο με κατάληξη .asm). Η cutePy είναι μια μικρή, εκπαιδευτική γλώσσα προγραμματισμού η οποία θυμίζει την γλώσσα Python αλλά είναι αρκετά πιο μικρή, τόσο στις υποστηριζόμενες δομές, όσο φυσικά και στις προγραμματιστικές της δυνατότητες και τα προγράμματα της είναι γραμμένα με τέτοιο τρόπο ώστε να μπορούν να τρέξουν και από έναν διερμηνέα Python. Η συγκεκριμένη προγραμματιστική άσκηση είχε διάρκεια ολόκληρο το χειμερινό εξάμηνο και η παράδοση της χωρίστηκε στις εξής 3 φάσεις:

1. Λεκτικός Αναλυτής - Συντακτικός Αναλυτής
2. Ενδιάμεσος Κώδικας - Πίνακας Συμβόλων
3. Τελικός Κώδικας

1.Λεκτικός αναλυτής

Η λεκτική ανάλυση αποτελεί την πρώτη φάση της μεταγλώττισης. Κατά τη φάση αυτή, διαβάζεται το αρχικό πρόγραμμα και παράγονται οι λεκτικές μονάδες. Χρησιμοποιούμε τον όρο λεκτική μονάδα για να αναπαραστήσουμε οτιδήποτε έχει νόημα να θεωρηθεί ως αυτόνομο σύνολο συνεχόμενων χαρακτήρων που μπορεί να συναντηθεί σε ένα πρόγραμμα και βρίσκει σημασιολογία στην υπό υλοποίηση γλώσσα. Οι λεκτικές μονάδες στην γλώσσα cytePy είναι οι ακόλουθες:

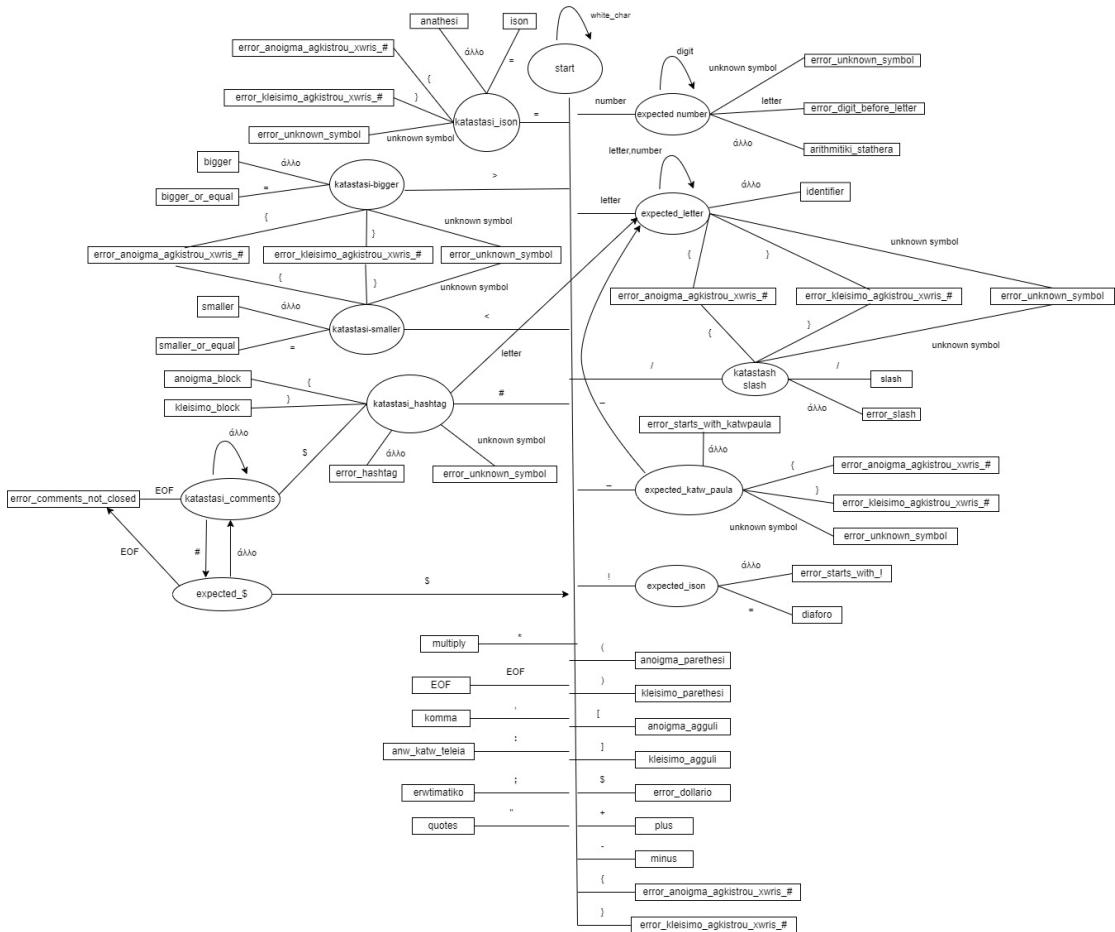
- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,...,Z και a,...,z),
- τα αριθμητικά ψηφία (0,...,9),
- η κάτω παύλα (_),
- τα σύμβολα των αριθμητικών πράξεων (+, -, *, //),
- οι τελεστές συσχέτισης (>,<, !=, <=, >=, ==)
- το σύμβολο ανάθεσης (=)
- τους διαχωριστές (;, "", :)
- τα σύμβολα ομαδοποίησης ([], (), #{, #})
- και διαχωρισμού σχολίων (#\\$)

Όταν ο λεκτικός αναλυτής αναγνωρίσει μία λεκτική μονάδα, τότε την επιστρέφει στον συντακτικό αναλυτή, επιστρέφοντας του και τον έλεγχο:

Μια λεκτική μονάδα μπορεί να ανήκει σε μια ευρύτερη κατηγορία/οικογένεια που είναι αναγκαία για τον συντακτικό αναλυτή. Πιο συγκεκριμένα υπάρχουν οι

- a. identifiers: ονόματα μεταβλητών, συναρτήσεων, σταθερών και ό,τι άλλο μπορεί να έχει ονομασία σε ένα πρόγραμμα.
- b. numbers: αριθμητικές σταθερές
- c. keywords: περιέχει τις λέξεις κλειδιά της γλώσσας, π.χ.: while, if, else , def,__main__,print,return, #declare, and, or, not, input, int
- d. addOperators: προσθετικοί αριθμητικοί τελεστές: +, -
- e. mulOperators: πολλαπλασιαστικοί αριθμητικοί τελεστές: *,//
- f. relOperators: λογικοί τελεστές π.χ.: ==, >=, !=, <=, <, >
- g. assignment: τελεστής εκχώρησης (=)
- h. delimiters: διαχωριστές : ; ,
- i. groupSymbol: σύμβολα ομαδοποίησης π.χ.: (), #{, #}, [,]

Ο λεκτικός αναλυτής υλοποιείται από μία συνάρτηση η οποία καλείται από τον συντακτικό αναλυτή και κάθε φορά που καλείται επιστρέφει την επόμενη διαθέσιμη στην είσοδο λεκτική μονάδα. Αυτό σημαίνει ότι σε κάθε του κλήση, ο λεκτικός αναλυτής σημειώνει το σημείο στο οποίο σταματάει την ανάγνωση του αρχικού προγράμματος και την επόμενη φορά που καλείται συνεχίζει την ανάγνωση από το σημείο αυτό. Η λειτουργία του βασίζεται στο παρακάτω αυτόματο.



Αυτόματο γλώσσας cutePy

Με την κλήση του λεκτικού αναλυτή το αυτόματο αρχικοποιείται στην κατάσταση start. Εάν στην είσοδο εμφανιστεί κάποιο ψηφίο, τότε το αυτόματο μεταβαίνει στην κατάσταση expected_number, η οποία είναι μη τελική. Όσο στην είσοδο εμφανίζονται ψηφία, τότε για κάθε ψηφίο που διαβάζεται, γίνεται μία μετάβαση, η οποία όμως δεν αλλάζει την κατάσταση του αυτομάτου, αφού είναι μετάβαση από την κατάσταση expected_number στην κατάσταση expected_number. Από την expected_number θα φύγει μόλις έρθει κάτι διαφορετικό, κάτι άλλο που δεν είναι ψηφίο. Όταν συμβεί αυτό το αυτόματο θα μεταβεί στην τελική κατάσταση arithmitiki_stathera και θα έχει αναγνωρίσει μία αριθμητική (ακέραια)

σταθερά. Αντίστοιχα, αν στην είσοδο εμφανιστεί κάποιο letter τότε το αυτόματο θα μεταβεί στην κατάσταση expected_letter, η οποία είναι μη τελική, και θα γίνεται μετάβαση στην ιδιά κατάσταση όσο ακόμα έρχονται letters,numbers ή κάτω παύλα καθώς ονόματα συναρτήσεων/μεταβλητών μπορούν να περιέχουν και την κάτω παύλα. Όταν στην είσοδο εμφανιστεί κάτι διαφορετικό θα μεταβεί στην τελική κατάσταση identifier/keyword και θα έχει αναγνωρίσει μια ακόμα λεκτική μονάδα.

Επιπρόσθετα αν στην είσοδο ο λεκτικός αναλυτής εντοπίσει κάποιο από τα group symbols [],(),#{,#} θα πάει κατευθείαν στην τελική κατάσταση του αντίστοιχου συμβόλου δηλαδή άνοιγμα παρένθεσης, άνοιγμα αγκύλης άνοιγμα μπλοκ... καθώς και αν εντοπίσει delimiters δηλαδή ,;; θα πάει αντίστοιχα στις τελικές καταστάσεις του αντίστοιχου συμβόλου και θα πάρει την συγκεκριμένη λεκτική μονάδα.

Αν ο λεκτικός αναλυτής βρει το σύμβολο = θα μεταβεί στην μη τελική κατάσταση katastasi_ison και σε περίπτωση που βρει ένα ακόμα = θα πάει στην τελική κατάσταση και θα αναγνωρίσει τη λεκτική μονάδα του σχεσιακού τελεστή της ισότητας ειδάλλως αν βρει στις δήποτε άλλο θα μεταβεί στην τελική κατάσταση και θα αναγνωρίσει τον τελεστή ανάθεσης . Οι σχεσιακοί τελεστές > και < όταν ο λεκτικός αναλυτής τους εντοπίσει στην είσοδο το αυτόματο θα μεταβεί στις μη τελικές καταστάσεις katastasi_bigger και katastasi_smaller αντίστοιχα στις οποίες αν εντοπίσει στην είσοδο το = θα μεταβεί στις τελικές καταστάσεις και θα πάρει την λεκτική μονάδα του μεγαλύτερου ή ίσου και του μικρότερου ή ίσου. Άλλιώς αν βρει στις δήποτε άλλο θα πάει στην τελική κατάσταση και θα πάρει την λεκτική μονάδα του > και του < αντίστοιχα. Επειδή στην γλώσσα cutePy ο αριθμητικός τελεστής της διαίρεσης αποτελείται από δυο σύμβολα // σε αντίθεση με τον τελεστή του πολλαπλασιασμού *, με τον οποίο μεταβαίνουμε κατευθείαν σε τελική κατάσταση και βρίσκουμε την λεκτική μονάδα του συγκεκριμένου τελεστή, πηγαίνουμε σε μη τελική κατάσταση την katastasi_slash όπου στην περίπτωση που ο λεκτικός αναλυτής εντοπίσει και δεύτερο / στην είσοδο μεταβαίνουμε σε τελική κατάσταση και παίρνουμε το πολλαπλασιαστικό τελεστή // αλλιώς θα εμφανιστεί λάθος. Οι αθροιστικοί τελεστές + και - όταν εντοπιστούν στην είσοδο το αυτόματο θα μεταβεί στις τελικές καταστάσεις του plus_tk και του minus_tk αντίστοιχα.

Όσον αφορά το σύμβολο # θα μετάβουμε στην προσωρινή κατάσταση katastasi_hashtag και αν το επόμενο σύμβολο είναι είτε το { είτε το } θα μετάβουμε σε τελική κατάσταση και θα πάρουμε το άνοιγμα μπλοκ και το κλείσιμο μπλοκ αντίστοιχα. Στην περίπτωση όμως όπου βρει ο αναλυτής \$ θα μεταβεί στην μη τελική κατάσταση katastasi_comments όπου όσα σύμβολα από εδώ και πέρα εντοπιστούν θα οδηγούν πάλι στην ίδια κατάσταση. Όταν ξαναβρεί # θα μεταβεί στην μη τελική κατάσταση expected_\$ όπου εδώ πρακτικά περιμένουμε να βρούμε το δολάριο ώστε να κλείσουν τα σχολια και να μετάβουμε στην αρχική κατάσταση και αν δεν είναι το επόμενο σύμβολο που θα έρθει θα ξαναοδηγηθούμε στην κατάσταση σχόλια. Λόγω της δεσμευμένης λέξης #declare σε περίπτωση που στην katastasi_hashtag το αυτόματο βρει γράμμα θα πάει στην κατάσταση expected_letter.

Επιπρόσθετα, αν βρει την κάτω παύλα στην αρχή το αυτόματο θα πάει στη μη τελική κατάσταση katastasi_katwaula και θα περιμένει να βρει μια ακόμα κάτω παύλα ώστε να πάει στην κατάσταση expected_letter λόγω της δυσμενής λέξης __main__. Αν βρει πάλι το ! θα πάει στην κατάσταση expected_ison και θα περιμένει να έρθει το = ώστε να πάει στην τελική κατάσταση και να πάρει την λεκτική μονάδα του σχεσιακού τελεστή diaforo_tk.

Τέλος αν ο λεκτικός τελεστής εντοπίσει το τέλος του αρχαίου θα μεταβεί σε τελική κατάσταση και θα πάρει τη λεκτική μονάδα του eof.

Στον συντακτικό αναλυτή θα επιστραφεί , ο αριθμός που χαρακτηρίζει τη λεκτική μονάδα, το recognized_string της λεκτικής μονάδας και φυσικά ο αριθμός γραμμής στον οποίο βρέθηκε η λεκτική μονάδα, το line_number. Το line_number χρειάζεται στον συντακτικό αναλυτή για τη διαχείριση σφαλμάτων ώστε να εκτυπώνει σε ποια γραμμή υπάρχει το αντίστοιχο σφάλμα.

Υλοποίηση λεκτικού αναλυτή

```
#input_file = open(sys.argv[1], 'r')
#In dictionary we have assigned integer values for every state,every symbol,every token,every keyword and every error.
my_dict = {
    # katastaseis
    "start": 0,
    "expected_number": 1,
    "expected_letter": 2,
    "katastasi_json": 3,
    "katastasi_bigger": 4,
    "katastasi_smaller": 5,
    "katastasi_slash": 6,
    "katastasi_#": 7,
    "katastasi_comments": 8,
    "expected_$": 9,
    "expected_katwpaula": 10,
    "expected_json": 11,
    # chars
    "white_char": 0,
    "letter": 1,
    "number": 2,
    "(": 3,
    ")": 4,
    "[": 5,
    "]": 6,
    "#": 7,
    "{": 8,
    "}": 9,
    "+": 10,
    "-": 11,
    "*": 12,
    "/": 13,
    "'": 14,
    "akuros_xaraktiras": 15,
    ".": 16,
    ",": 17,
    ";": 18,
    "=": 19,
    "<": 20,
    ">": 21,
    "...": 22,
    "$": 23,
    "_": 24,
    "\n": 25,
    "!": 26,
    # tokens
    "arithmitiki_stathera_tk": 50,
    "identifier_tk": 51,
    "plus_tk": 52,
    "minus_tk": 53,
    "erwtimatiiko_tk": 54,
    "komma_tk": 55,
    "anw_katw_teleia_tk": 56,
    "slash_tk": 57,
    "multiply_tk": 58,
    "bigger_tk": 59,
    "bigger_or_equal_tk": 60,
```

```

    "ison_tk": 61,
    "smaller_tk": 62,
    "smaller or equal_tk": 63,
    "diaforo_tk": 64,
    "anathesi_tk": 65,
    "quotes_tk": 66,
    "EOF_tk": 67,
    "kleisimo_agguli_tk": 68,
    "anoigma_agguli_tk": 69,
    "kleisimo_parenthesi_tk": 70,
    "anoigma_parenthesi_tk": 71,
    "anoigma_block_tk": 72,
    "kleisimo_block_tk": 73,
    # desmeymenes lexeis
    "def": 100,
    "__main__": 101,
    "#declare": 102,
    "if": 103,
    "else": 104,
    "and": 105,
    "or": 106,
    "not": 107,
    "return": 108,
    "while": 109,
    "__name__": 110,
    "input": 111,
    "int": 112,
    "print": 113,
    # errors
    "error_unknown_symbol": -1,
    "error_digit_before_letter": -2,
    "error_slash": -3,
    "error_hashtag": -4,
    "error_comments_not_closed": -5,
    "error_starts_with_katwpaula": -6,
    "error_dolario": -7,
    "error_panw_apo_30_xarakthres": -8,
    "error_arithmos_ektos_diasthmatos": -9,
    "error_anoigma_agkistrou_xwris_#": -10,
    "error_kleisimo_agkistrou_xwris_#": -11,
    "error_starts_with_!": -12,
}
}

#For our array we have the states that come from finite automata
my_array = [
    # start
    [my_dict["start"], my_dict["expected_letter"], my_dict["expected_number"], my_dict["anoigma_parenthesi_tk"],
     my_dict["kleisimo_parenthesi_tk"], my_dict["anoigma_agguli_tk"], my_dict["kleisimo_agguli_tk"], my_dict["katastasi_#"],
     my_dict["error_anoigma_agkistrou_xwris_#"], my_dict["error_kleisimo_agkistrou_xwris_#"], my_dict["plus_tk"], my_dict["minus_tk"],
     my_dict["multiply_tk"], my_dict["katastasi_slash"], my_dict["EOF_tk"], my_dict["error_unknown_symbol"],
     my_dict["komma_tk"], my_dict["anw_katw_telia_tk"], my_dict["rwrtimatiwo_tk"], my_dict["katastasi_ison"],
     my_dict["katastasi_smaller"], my_dict["katastasi_bigger"], my_dict["quotes_tk"], my_dict["error_dolario"],
     my_dict["expected_katwpaula"], my_dict["start"], my_dict["expected_ison"]],


    # expected_number
    [my_dict["arithmitiki_stathera_tk"], my_dict["error_digit_before_letter"], my_dict["expected_number"], my_dict["arithmitiki_stathera_tk"],
     my_dict["arithmitiki_stathera_tk"], my_dict["arithmitiki_stathera_tk"], my_dict["arithmitiki_stathera_tk"], my_dict["arithmitiki_stathera_tk"]],


    # expected_letter
    [my_dict["identifier_tk"], my_dict["expected_letter"], my_dict["expected_letter"], my_dict["identifier_tk"],
     my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"],
     my_dict["error_anoigma_agkistrou_xwris_#"], my_dict["error_kleisimo_agkistrou_xwris_#"], my_dict["identifier_tk"], my_dict["identifier_tk"],
     my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["error_unknown_symbol"],
     my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"],
     my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"], my_dict["identifier_tk"],
     my_dict["expected_letter"], my_dict["identifier_tk"], my_dict["identifier_tk"]],


    # katastasi_ison
    [my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"],
     my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"],
     my_dict["error_anoigma_agkistrou_xwris_#"], my_dict["error_kleisimo_agkistrou_xwris_#"], my_dict["anathesi_tk"], my_dict["anathesi_tk"],
     my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["error_unknown_symbol"],
     my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["ison_tk"],
     my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"],
     my_dict["anathesi_tk"], my_dict["anathesi_tk"], my_dict["anathesi_tk"]],
```

Για την υλοποίηση του λεκτικού μας αναλυτή αποθηκεύσαμε σε ένα λεξικό όλες τις μη τελικές καταστάσεις, τα σύμβολα, τις δεσμευμένες λέξεις, τις λεκτικές μας μονάδες και τα πιθανά λάθη με ένα μοναδικό αριθμό που τα χαρακτηρίζει. Το αυτόματο μας υλοποιείται με τον πίνακα καταστάσεων ο οποίος περιέχει όλες τις καταστάσεις και για κάθε κατάσταση περιέχει όλες τις δυνατές μεταβάσεις βάση των συμβόλων που υπάρχουν στην *cutePy*.

```
# lektikos analytis
state = 0
line = 1

def lex():
    global state
    state = my_dict["start"]
    num1 = 0
    string = ""
    global line
    array = [] #we store the token
    array1 = [] #we store the value of the token, the token and the line

    while state >= 0 and state <= 11:
        char = input_file.read(1) #we read one byte from our file
        # print(char)
        if char == '\t' or char == ' ':
            num1 = my_dict["white_char"]
            string += char

        elif char.isalpha():
            num1 = my_dict["letter"]
            string += char

        elif char.isdigit():
            num1 = my_dict["number"]
            string += char
```

```
elif char == '(':
    num1 = my_dict["("]
    string += char

elif char == ')':
    num1 = my_dict[")"]
    string += char

elif char == '[':
    num1 = my_dict["["]
    string += char

elif char == ']':
    num1 = my_dict["]"]
    string += char

elif char == '#':
    num1 = my_dict["#"]
    string += char

elif char == '{':
    num1 = my_dict["{"]
    string += char

elif char == '}':
    num1 = my_dict["}"]
    string += char
```

```
    elif char == '+':
        num1 = my_dict["+"]
        string += char

    elif char == '-':
        num1 = my_dict["-"]
        string += char

    elif char == '*':
        num1 = my_dict["*"]
        string += char

    elif char == '/':
        num1 = my_dict["/"]
        string += char

    elif char == '':
        num1 = my_dict[""]
        string += char

    elif char == ',':
        num1 = my_dict[","]
        string += char

    elif char == ':':
        num1 = my_dict[":"]
        string += char
```

```
elif char == ';':
    num1 = my_dict[";"]
    string += char

elif char == '=':
    num1 = my_dict["="]
    string += char

elif char == '<':
    num1 = my_dict["<"]
    string += char

elif char == '>':
    num1 = my_dict[">"]
    string += char

elif char == '':
    num1 = my_dict['']
    string += char

elif char == '$':
    num1 = my_dict["$"]
    string += char

elif char == '_':
    num1 = my_dict["_"]
    string += char
```

```

        elif char == '\n':
            num1 = my_dict["\n"]
            line += 1
            string += char

        elif char == '!':
            num1 = my_dict[!"!"]
            string += char

        else:
            num1 = my_dict["_akuros_xaraktiras"]
            string += char

        state = my_array[state][num1]
        #When we have comments or we are at the start of our finite automata we empty the string
        if state == my_dict["katastasi_comments"] or state == my_dict["start"]:
            string = ""

        #when we have found these tokens we add them at the array and we check the last char and then we go back a byte at the file so the aytomato can read
        if state == my_dict["identifier_tk"] or state == my_dict["arithmitiki_stathera_tk"] or state == my_dict["anathesi_tk"] or state == my_dict["bigger_tk"]:
            char1 = string[-1]
            if char1 == '\n':
                line -= 1
            array.append(string[:-1])
            if string[-1] in my_dict.keys() and state != my_dict["anathesi_tk"] and state != my_dict["bigger_tk"] and state != my_dict["smaller_tk"]:
                state = my_dict[string[-1]]
                if string[-1] == "number" or string[-1] == "letter":
                    state = my_dict["identifier_tk"]
            input_file.seek(input_file.tell() - 1, 0) #we go back one byte from the pos we already are

    else:
        array.append(string)

#we check the length of id_tk and the range at numerical constant
if state == my_dict["identifier_tk"]:
    if len(array[0]) > 30:
        state = my_dict["error_panw_apo_30_xarakthres"]

if state == my_dict["arithmitiki1_stathera_tk"]:
    if (int(array[0])) < -(2 ** 32 - 1) or (int(array[0])) > (2 ** 32 - 1):
        state = my_dict["error_arithmos_ektos_diasthmatos"]

if state == my_dict["error_unknown_symbol"]:
    print("Error unknown symbol")

elif state == my_dict["error_digit_before_letter"]:
    print("Error there is a digit before letter")

elif state == my_dict["error_slash"]:
    print("Error there is a / alone")

elif state == my_dict["error_hashtag"]:
    print("Error there is a # alone")

elif state == my_dict["error_comments_not_closed"]:
    print("Error comments not closed")

elif state == my_dict["error_starts_with_katwpaula"]:
    print("Error starts with _ ")

elif state == my_dict["error_dolario"]:
    print("Error there is $ alone")

elif state == my_dict["error_anoigma_agkistrou_xwris_#"]:
    print("Error there is { without #")

elif state == my_dict["error_kleisimo_agkistrou_xwris_#"]:
    print("Error there is } without #")

elif state == my_dict["error_panw_apo_30_xarakthres"]:
    print("Error word over 30 chars")

elif state == my_dict["error_arithmos_ektos_diasthmatos"]:
    print("Error number out of range")

elif state == my_dict["error_starts_with_!"]:
    print("Error starts with !")
    array1.append(state)
    array1.append(array[0])
    array1.append(line)
    print(array1)
    return array1

```

Η συνάρτηση του λεκτικού αναλυτή διαβάζει έναν χαρακτήρα από το input αρχείο και αλλάζει καταστάσεις μέχρι να φτάσει σε τελική κατάσταση. Οι μη τελικές καταστάσεις είναι από το 0 έως το 11 στο λεξικό μας ενώ οι τελικές είναι από το 50 και πάνω. Ιδιαίτερη έμφαση δόθηκε όταν βρίσκουμε identifier, anathesi_tk, bigger_tk, smaller_tk και arithmitiki_stathera_tk όπου πάντα κοιτάζουμε το επόμενο σύμβολο και με αυτό τον τρόπο

καταναλώνουμε έναν χαρακτήρα από την επόμενη λεκτική μονάδα και για αυτό πηγαίνουμε αν είμαστε σε αυτές τις τελικές καταστάσεις ένα byte πίσω στο αρχείο μας ώστε όταν ξανακινήσει το αυτόματο να ξεκινήσει από τον χαρακτήρα που καταναλώσαμε.

Επίσης γίνεται έλεγχος αν είναι identifier να ελέγχουμε αν είναι keyword όπου ψάχνουμε αν υπάρχει η συγκεκριμένη λέξη μέσα στο λεξικό μας όπου έχουμε αποθηκεύσει τις δεσμευμένες λέξεις, οι οποίες είναι από το 100 και πάνω.

Έλεγχος εγκυρότητας λεκτικού αναλυτή

Για να ελεγχουμε στην πρωτη φαση της εργασιας μας τον λεκτικο αναλυτη

Εκτελουσαμε την εντολη `python cutePy_4168_4163.py test1.cpy` οπου το αρχειο `cutePy_4168_4163.py` ηταν ο λεκτικος μας αναλυτης και το `test1.cpy` το αρχειο της γλωσσας `cutePy` που θελαμε να βρουμε τις λεκτικες μοναδες.Το πρωτο αρχειο που φτιαξαμε για να βρουμε τις λεκτικες μοναδες ηταν το παρακατω:

```
def main_calculate():
    #\$ our own calculator    #\$

    #{
        #declare x, y,num1,num2,result,choice
        def add(x,y):
            #{           return (x + y);
            #
        def subtract(x,y):
            #{           return (x - y);
            #
        def multiply(x,y):
            #{           return (x - y);
            #

        def divide(x,y):
            #{           return (x //y);
            #

        num1 = int(input());
        num2 = int(input());
        while(choice > 0 and choice < 5):
            #{

                choice = int(input());
                if (choice == 1):
                    result = add(num1,num2);
                if (choice == 2):
                    result = subtract(num1,num2);
                if (choice == 3):
                    result = multiply(num1,num2);
                if (choice == 4):
                    result = divide(num1,num2);
            #}
```

```
    print(result);
}

if __name__ == "__main__":
    #\$ call of main function #\$

    main_calculate();
```

Αρχικά ελέγχουμε αν θα βρει σωστά το keyword def με τον σωστό ακέραιο του που είναι ο αριθμός 100 και δε θα βάλει τον ακέραιο 50 που χαρακτηρίζει γενικά όλα τα identifiers. Επίσης παρατηρούμε αν τα σχόλια που τα έχουμε βάλει με τα #\\$ #\\$ θα τα προσπεράσει και δε θα τα εκτυπώσει σαν λεκτικές μονάδες. Ελέγχουμε ακόμα τις δεσμευμένες λέξεις #declare, return, print, input, int, __main__, if, while και and. Το #{} αν θα το πάρει και τα 2 σύμβολα ως μια λεκτική μονάδα δηλαδή το άνοιγμα μπλοκ και το ίδιο ισχύει και για το κλείσιμο μπλοκ. Τους τελεστές +, -, * καθώς και το // αν θα τους αναγνωρίσει σαν λεκτικές μονάδες. Επίσης έχουμε τους σχεσιακούς τελεστές >, < καθώς και τον σχεσιακό τελεστή == με τους δικούς τους ακέραιους που τους χαρακτηρίζουν. Την ανάθεση = καθώς και τα ,; : () που είναι το καθένα ξεχωριστά λεκτική μονάδα. Τέλος ελέγχουμε αν θα καταναλωθεί κάποιος χαρακτήρας μετρά την ανάθεση, κάποιο όνομα μεταβλητής, κάποια αριθμητική σταθερά το > και το < ώστε να σιγουρευτούμε ότι ο λεκτικός μας αναλυτής προλαμβάνει αυτό το λάθος. Όπως παρατηρούμε παρακάτω αν τον τρέξουμε βγαίνουν όλα με τους σωστούς ακέραιους καθώς και τον σωστό αριθμό γραμμής.

```
C:\python\ex3>python cutePy_4168_4163.py test3.cpy
[100, 'def', 3]
[51, 'main_calculate', 3]
[71, '(', 3]
[79, ')', 3]
[56, '::', 3]
[72, '#!', 5]
[102, '#declare', 6]
[51, 'x', 6]
[55, ',', 6]
[51, 'y', 6]
[55, ',', 6]
[51, 'num1', 6]
[55, ',', 6]
[51, 'num2', 6]
[55, ',', 6]
[51, 'result', 6]
[55, ',', 6]
[51, 'choice', 6]
[100, 'def', 7]
[51, 'add', 7]
[71, '(', 7]
[51, 'x', 7]
[55, ',', 7]
[51, 'y', 7]
[79, ')', 7]
[56, '::', 7]
[72, '#!', 8]
[108, 'return', 9]
[71, '(', 9]
[51, 'x', 9]
[52, '+', 9]
[51, 'y', 9]
[79, ')', 9]
[54, ',', 9]
[73, '#!', 10]
[100, 'def', 11]
[51, 'subtract', 11]
[71, '(', 11]
[51, 'x', 11]
```

```
[71, '(', 22]
[51, 'x', 22]
[57, '/', 22]
[51, 'y', 22]
[79, ')', 22]
[54, '::', 22]
[73, '#{', 23]
[51, 'num1', 26]
[65, '=', 26]
[112, 'int', 26]
[71, '(', 26]
[111, 'input', 26]
[71, '(', 26]
[78, ')', 26]
[78, ')', 26]
[54, '::', 26]
[51, 'num2', 27]
[65, '=', 27]
[112, 'int', 27]
[71, '(', 27]
[111, 'input', 27]
[71, '(', 27]
[78, ')', 27]
[78, ')', 27]
[54, '::', 27]
[72, '#!', 28]
[51, 'choice', 28]
[59, '>', 28]
[58, '0', 28]
[105, 'and', 28]
[51, 'choice', 28]
[62, '0', 28]
[58, '5', 28]
[78, ')', 28]
[56, '::', 28]
[56, '1', 28]
[72, '#!', 29]
[51, 'choice', 30]
[65, '=', 30]
[112, 'int', 30]
```

```
[71, '(', 30]
[111, 'input', 30]
[71, '(', 30]
[78, ')', 30]
[78, ')', 30]
[54, '::', 30]
[103, 'if', 31]
[71, '(', 31]
[51, 'choice', 31]
[61, '==', 31]
[50, '1', 31]
[78, ')', 31]
[56, '::', 31]
[51, 'result', 32]
[65, '=', 32]
[51, 'add', 32]
[71, '(', 32]
[51, 'num1', 32]
[59, '0', 32]
[51, 'num2', 32]
[79, ')', 32]
[54, '::', 32]
[103, 'if', 33]
[71, '(', 33]
[51, 'choice', 33]
[61, '==', 33]
[50, '2', 33]
[78, ')', 33]
[56, '::', 33]
[51, 'result', 34]
[65, '=', 34]
[51, 'subtract', 34]
[71, '(', 34]
[51, 'num1', 34]
[55, ',', 34]
[51, 'num2', 34]
[79, ')', 34]
[54, '::', 34]
[103, 'if', 35]
[71, '(', 35]
```

```

[51, 'choice', 35]
[61, '==', 35]
[56, '3', 35]
[60, '1', 35]
[66, ':', 35]
[51, 'result', 36]
[65, '=', 36]
[51, 'multiply', 36]
[71, '(', 36]
[51, 'num1', 36]
[55, ',', 36]
[51, 'num2', 36]
[60, ')', 36]
[50, '...', 36]
[103, 'if', 37]
[71, '(', 37]
[51, 'choice', 37]
[61, '==', 37]
[56, '4', 37]
[78, ')', 37]
[60, '...', 37]
[51, 'result', 38]
[65, '=', 38]
[51, 'divide', 38]
[71, '(', 38]
[51, 'num1', 38]
[55, ',', 38]
[51, 'num2', 38]
[78, ')', 38]
[73, '#', 39]
[113, 'print', 40]
[71, '(', 40]
[51, 'result', 40]
[78, ')', 40]
[54, ',', 40]
[73, '#', 41]
[103, 'if', 42]
[110, 'else', 42]
[61, '==', 42]

```

```

[66, "'", 42]
[101, '__main__', 42]
[66, "'", 42]
[56, ':', 42]
[51, 'main_calculate', 44]
[71, '(', 44]
[70, ')', 44]
[54, ';', 44]
[67, "'", 47]

```

Ένα ακόμα τεστ αρχείο που δημιουργήσαμε:

```

#§ program to find the L.C.M of two input number #§

def main_compute_lcm():
    #{
        #declare x,y
        def compute_lcm(x,y):
            #{
                #§choose the greater number#§
                #declare greater,lcm,u1,u2

                if  (x>y):
                    greater = x;
                else:
                    greater = y;
                u1 = greater - (greater // x)* x ;
                u2 = greater - (greater // y)* y ;
                while([u1 != 0] or [u2 !=0]):
                    greater = greater + 1;
                lcm = greater;
                return (lcm);
            #}
            num1 = int(input());
            num2 =int(input());

            print(compute_lcm(num1,num2));
    #}

    if __name__ == "__main__":
        main_compute_lcm();

```

Ελέγχουμε τις υπόλοιπες δεσμευμένες λέξεις else ,or καθώς και τις αγκύλες [] ,τον σχεσιακό τελεστή του διάφορου αν θα το πάρει σαν μια λεκτική μονάδα καθώς και τα ονόματα των συναρτήσεων με κάτω παύλες αν θα τα πάρει σαν μια λεκτική μονάδα.

Οι λεκτικές μονάδες που εκτυπώνονται στην οθόνη μας

```
C:\python\ex3>python cutePy_4168_4163.py test2.cpy
[100, 'def', 5]
[51, 'main_compute_lcm', 5]
[71, '(', 5]
[70, ')', 5]
[56, ':', 5]
[72, '#!', 6]
[102, '#declare', 7]
[51, 'x', 7]
[55, ',', 7]
[51, 'y', 7]
[100, 'def', 8]
[51, 'compute_lcm', 8]
[71, '(', 8]
[51, 'x', 8]
[55, ',', 8]
[51, 'y', 8]
[70, ')', 8]
[56, ':', 8]
[72, '#!', 9]
[102, '#declare', 11]
[51, 'greater', 11]
[55, ',', 11]
[51, 'lcm', 11]
[55, ',', 11]
[51, 'u1', 11]
[55, ',', 11]
[51, 'u2', 11]
[103, 'if', 13]
[71, '(', 13]
[51, 'x', 13]
[55, ',', 13]
[51, 'y', 13]
[70, ')', 13]
[56, ':', 13]
[51, 'greater', 14]
[65, '=', 14]
[51, 'x', 14]
[54, ',', 14]
[104, 'else', 15]
```

```
[56, ':', 15]
[51, 'greater', 16]
[65, '=', 16]
[51, 'y', 16]
[54, ',', 16]
[51, 'u1', 17]
[65, '=', 17]
[51, 'greater', 17]
[53, '!', 17]
[71, '(', 17]
[51, 'greater', 17]
[57, '//', 17]
[51, 'x', 17]
[70, ')', 17]
[58, '*', 17]
[51, 'x', 17]
[54, ',', 17]
[51, 'u2', 18]
[65, '=', 18]
[51, 'greater', 18]
[53, '!', 18]
[71, '(', 18]
[51, 'greater', 18]
[57, '//', 18]
[51, 'y', 18]
[70, ')', 18]
[58, '*', 18]
[51, 'y', 18]
[54, ',', 18]
[109, 'while', 19]
[71, '(', 19]
[69, '[', 19]
[51, 'u1', 19]
[64, '!=', 19]
[58, '0', 19]
[68, ')', 19]
[106, 'or', 19]
[69, '[', 19]
[51, 'u2', 19]
```

```
[60, '!=', 19]
[66, '0', 19]
[68, ')', 19]
[70, ')', 19]
[56, ':', 19]
[51, 'greater', 20]
[65, '=', 20]
[51, 'greater', 20]
[52, '*', 20]
[50, '1', 20]
[54, ':', 20]
[51, 'lcm', 21]
[65, '=', 21]
[51, 'lcm', 21]
[54, ':', 21]
[108, 'return', 22]
[71, '(', 22]
[51, 'lcm', 22]
[70, ')', 22]
[50, '0', 22]
[73, '#!', 23]
[51, 'num1', 24]
[65, '=', 24]
[112, 'int', 24]
[71, '(', 24]
[51, 'input', 24]
[71, '(', 24]
[70, ')', 24]
[70, ')', 24]
[54, ':', 24]
[51, 'num2', 25]
[65, '=', 25]
[112, 'int', 25]
[71, '(', 25]
[111, 'input', 25]
[71, '(', 25]
[70, ')', 25]
[70, ')', 25]
[54, ':', 25]
[113, 'print', 27]
```

```

[71, '(' , 27]
[51, 'compute_lcm' , 27]
[71, ')' , 27]
[51, 'num1' , 27]
[55, ',' , 27]
[51, 'num2' , 27]
[70, ')' , 27]
[70, ')' , 27]
[54, ';' , 27]
[73, '#' , 28]
[103, 'if' , 30]
[110, '__name__' , 30]
[61, '==' , 30]
[66, '' , 30]
[56, ':' , 30]
[51, 'main_compute_lcm' , 31]
[71, '(' , 31]
[70, ')' , 31]
[54, ';' , 31]
[67, '' , 32]

```

Διαχείριση σφαλμάτων στον λεκτικό αναλυτή

Στη φάση της λεκτικής ανάλυσης αναγνωρίζονται τα πρώτα από τα σφάλματα που μπορεί να ανακαλύψει ένας μεταγλωττιστής. Πρόκειται για σφάλματα τα οποία σχετίζονται με λεκτικές μονάδες. Σφάλματα που σχετίζονται με σύνταξη (π.χ. παράλειψη παρένθεσης) ή σημασιολογικά (π.χ. μία μεταβλητή δεν έχει δηλωθεί) θα εντοπιστούν σε φάσεις που θα ακολουθήσουν.

Αρχικά τα errors βρίσκονται στο λεξικό μας με αρνητικούς ακέραιους από το -1,-2,... Στο αυτόματο μας μπορούμε από κάποιες καταστάσεις να οδηγηθούμε σε σφάλματα. Αν εμφανιστεί στη γλώσσα μας ένας χαρακτήρας που δεν υπάρχει σαν σύμβολο στη γλώσσας μας θα μας εμφανιστεί η φράση << error unknown symbol >>. Ένα άλλο λάθος είναι όταν είμαστε στην κατάσταση expected_number και εντοπίσει ο λεκτικός μας αναλυτής ένα letter τότε θα εμφανίσει το error «Error there is a digit before letter». Εεπιπλέον, αν δει μόνο του ένα δολάριο θα βγάλει λάθος καθώς δεν μπορεί να υπάρξει μόνο του στη cutePy ένα δολάριο αλλά και αν δεν έχουν κλείσει τα σχόλια δηλαδή αν δεν έχει δει το #\\$ και βρει το EOF. Επιπρόσθετα δεν εμφανίζονται στην cutePy μόνα τους τα { και } οπότε πάλι θα βγάλει μήνυμα λάθους.

Η cutePy ορίζει τα identifiers να μην έχουν μήκος παραπάνω από 30 χαρακτήρες οπότε θέσαμε το λάθος <<error word over 30 chars>> να εμφανίζεται στην οθόνη. Αντίστοιχα και οι αριθμητικές σταθερές να έχουν τιμές από -(2³² - 1) έως 2³² - 1 επομένως ομοίως βάλαμε έναν έλεγχο και για την εμφάνιση του λάθους για ακέραιους με μη επιτρεπτές τιμές.

Υπάρχουν και αρκετά άλλα λάθη όπως φαίνεται στον κώδικα μας που μπορεί να εντοπίσει ο λεκτικός αναλυτής όπως να αρχίζει μια λέξη με κάτω παύλα, ένα θαυμαστικό μόνο του χωρίς το ιόν, ένα / μόνο του όπως και η δίεση μονή της. Σε καθένα από αυτά θα εμφανίζεται στην οθόνη το αντίστοιχο μήνυμα.

Ενδεικτικά παρακάτω είναι κάποια λάθη να εμφανίζονται στην οθόνη τα οποία θα εντοπίζει ο λεκτικός αναλυτής:

```

-----^
def compute_lcm(x,y):
{
#$choose the greater number
    #declare greater,lcm,u1,u2

```

```
C:\>python ex3>python cutePy_4168_4163.py test2.cpy
[100, 'def', 5]
[51, 'main_compute_lcm', 5]
[71, '(' , 5]
[70, ')', 5]
[56, ':', 5]
[72, '#{', 6]
[102, '#declare', 7]
[51, 'x', 7]
[55, ',', 7]
[51, 'y', 7]
[100, 'def', 8]
[51, 'compute_lcm', 8]
[71, '(' , 8]
[51, 'x', 8]
[55, ',', 8]
[51, 'y', 8]
[70, ')', 8]
[56, ':', 8]
[72, '#{', 9]
Error comments not closed
[-5, '', 32]
Error,something is wrong with your statement in line 32
```

Ενδεικτικά εδώ που δεν κλείσαμε τα σχόλια μας βγάζει το αντίστοιχο μήνυμα λάθους και έχει τον ακέραιο που είναι για το μη κλείσιμο σχολίων από το λεξικό μας το -5 και δεν συνεχίζει παρακάτω ο λεκτικός μας αναλυτής.

```
if (x>y) :  
    5greater = x;  
else:  
    greater = y;  
  
C:\python\ex3>python cutePy_4168_4168.py test2.cpy  
[[0, 'def', 5]  
[[51, 'main_compute_lcm', 5]  
[[71, '(' , 5]  
[[79, ')', 5]  
[[56, '::', 5]  
[[72, '#!', 6]  
[[102, '#declare', 7]  
[[81, 'x', 7]  
[[55, ',', 7]  
[[51, 'y', 7]  
[[100, 'def', 8]  
[[51, 'compute_lcm', 8]  
[[71, '(', 8]  
[[51, 'x', 8]  
[[55, '::', 8]  
[[51, 'y', 8]  
[[79, ')', 8]  
[[56, '::', 8]  
[[72, '#!', 9]  
[[102, '#declare', 11]  
[[61, 'greater', 11]  
[[55, ',', 11]  
[[51, 'lcm', 11]  
[[55, '::', 11]  
[[51, 'u1', 11]  
[[55, '::', 11]  
[[51, 'u2', 11]  
[[103, 'if', 13]  
[[71, '(', 13]  
[[51, 'x', 13]  
[[59, '>', 13]  
[[51, 'y', 13]  
[[79, ')', 13]  
[[56, '::', 13]  
[[51, 'u1', 13]]  
Error: there is a digit before letter  
[-2, '5g' 14]  
Error: something is wrong with your statement in line 14
```

Στη συγκεκριμένη περίπτωση βάλαμε έναν αριθμό πριν από μια μεταβλητή και μας έβγαλε πάλι το αντίστοιχο μήνυμα λάθους.

Αντίστοιχα εδώ πέρα είχαμε έναν identifier όπου το μήκος του ξεπερνούμε τους 30 χαρακτήρες.

Συντακτικός αναλυτής

Τη φάση της λεκτικής ανάλυσης ακολουθεί η φάση της συντακτικής ανάλυσης. Κατά τη συντακτική ανάλυση ελέγχουμε αν οι λεκτικές μονάδες που δημιουργούνται από τον λεκτικό αναλυτή είναι μια νόμιμη ακολουθία με βάση τη γραμματική της γλώσσας. Όποια ακολουθία δεν αναγνωρίζεται από τη γραμματική, αποτελεί μη νόμιμο κώδικα και οδηγεί στον εντοπισμό συντακτικού σφάλματος. Έτσι, ο συντακτικός αναλυτής παίρνει σαν είσοδο μία ακολουθία από λεκτικές μονάδες.

Οι δομές της γλώσσας είναι οι απλές εντολές και οι δομημένες εντολές. Έχουμε τις εντολές εκχώρησης, επιστροφής τιμής συνάρτησης, εισόδου και εξόδου δεδομένων οι οποίες είναι οι απλές εντολές και τις δομημένες όπου είναι οι εντολές απόφασης και επανάληψης. Όπως και στην python έχουμε το πέρασμα παραμέτρου με τιμή και οι συναρτήσεις χωρίζονται σε κυρίες και τοπικές. Οι κύριες συναρτήσεις καλούνται μόνο από το κυρίως πρόγραμμα. Δεν επιστρέφουν τιμή, δεν δέχονται παραμέτρους, δεν καλούν άλλες κύριες συναρτήσεις, παρά μόνο εκτελούν τον δικό τους κώδικα και καλούν τις δικές τους απλές συναρτήσεις.

Υλοποίηση συντακτικού αναλυτή

```
# syntaktikos_analytis
def syn():
    global fullist
    global line
    global tokens
    tokens = lex()
    line = tokens[2]
    add_scope('main')
    def startRule():

        def_main_part()
        call_main_part()

    def def_main_part():
        global tokens
        global line

        def_main_function()
        while tokens[0] == my_dict["def"]:
            def_main_function()

    def def_main_function():
        global line
        global tokens
        global_list_of_scopes
        if tokens[0] == my_dict["def"]:
            tokens = lex()
            line = tokens[2]

    if tokens[0] == my_dict["identifier_tk"]:
        ID = tokens[1]
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
        entity = Entity()
        entity.datatype = 'FUNCTION'
        entity.name = ID
        entity.function.nestingLevel = list_of_scopes[-1].nestingLevel + 1
        add_entity(entity)
    if tokens[0] == my_dict["anw_katw_teleia_tk"]:
        tokens = lex()
        line = tokens[2]
        add_scope(ID)
    if tokens[0] == my_dict["anoigma_block_tk"]:
        tokens = lex()
        line = tokens[2]
        declarations()
        while tokens[0] == my_dict["def"]:
            def_function()
            calculate_startingQuad()
            genQuad('begin_block', ID, '_', '_')

statements()
if tokens[0] == my_dict["kleisimo_block_tk"]:
    calculate_framelength()
    genQuad('end_block', ID, '_', '_')
    show_symbols(my_file2)

tokens = lex()
line = tokens[2]
generate_final()
delete_scope()
else:
    print("Error, the program does not have #{ at main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have #{} after main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have : after main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have ( at main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have ( after main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have a ) after ( at main function in line", line)
    exit(-1)
else:
    print("Error, the program does not have a name for main function in line", line)
    exit(-1)
```

```

        else:
            print("Error, the program doesnt start with def at main function in line", line)
            exit(-1)

    def def_function():
        global line
        global tokens
        global list_of_scopes
        global isArgument

        if tokens[0] == my_dict["def"]:
            tokens = lex()
            line = tokens[2]
            if tokens[0] == my_dict["identifier_tk"]:
                ID = tokens[1]
                tokens = lex()
                line = tokens[2]
                entity = Entity()
                entity.datatype = 'FUNCTION'
                entity.name = ID
                entity.function.nestingLevel = list_of_scopes[-1].nestingLevel + 1
                add_entity(entity)
                if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
                    tokens = lex()
                    line = tokens[2]
                    isArgument = True
                    id_list()
                    if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
                        tokens = lex()
                        line = tokens[2]
                        add_scope(ID)
                        if tokens[0] == my_dict["anw_katw_teleia_tk"]:
                            tokens = lex()
                            line = tokens[2]
                            if tokens[0] == my_dict["anoigma_block_tk"]:
                                tokens = lex()
                                line = tokens[2]
                                add_parameters()
                                declarations()
                                while tokens[0] == my_dict["def"]:
                                    def_function()
                                calculate_startingQuad()
                                genQuad('begin_block', ID, '_', '_')
                                statements()
                                calculate_framelength()
                                if tokens[0] == my_dict["kleisimo_block_tk"]:
                                    genQuad('end_block', ID, '_', '_')
                                    show_symbols(my_file2)

                                tokens = lex()
                                line = tokens[2]
                                generate_final()
                                delete_scope()

                            else:
                                print("Error, the program does not have #} after #{ at function in line", line)
                                exit(-1)
                        else:
                            print("Error, the program does not have #{ after function in line", line)
                            exit(-1)
                    else:
                        print("Error, the program does not have : after function in line", line)
                        exit(-1)
                else:
                    print("Error, the program does not have a ) after ( at function in line", line)
                    exit(-1)
            else:
                print("Error, the program does not have ( after function in line", line)
                exit(-1)
        else:
            print("Error, the program does not have a name for function in line", line)
            exit(-1)
    else:
        print("Error, the program doesnt start with def in line", line)
        exit(-1)

    def declarations():
        global tokens
        global line

        while tokens[0] == my_dict["#declare"]:
            declaration_line()

```

```

def declaration_line():
    global tokens
    global line
    global isArgument

    if tokens[0] == my_dict["#declare"]:
        tokens = lex()
        line = tokens[2]
        isArgument = False
        id_list()

def statement():
    global tokens
    global line

    if tokens[0] == my_dict["identifier_tk"] or tokens[0] == my_dict["print"] or tokens[0] == my_dict["return"]:
        simple_statement()
    elif tokens[0] == my_dict["if"] or tokens[0] == my_dict["while"]:
        structured_statement()
    else:
        print("Error,something is wrong with your statement in line", line)
        exit(-1)

def statements():
    global tokens

    while tokens[0] != my_dict["end_of_file"]:
        statement()

def simple_statement():
    global tokens
    global line

    if tokens[0] == my_dict["identifier_tk"]:
        assignment_stat()
    elif tokens[0] == my_dict["print"]:
        print_stat()
    elif tokens[0] == my_dict["return"]:
        return_stat()

def structured_statement():
    global tokens
    global line

    if tokens[0] == my_dict["if"]:
        if_stat()
    elif tokens[0] == my_dict["while"]:
        while_stat()

def assignment_stat():
    global tokens
    global line

    if tokens[0] == my_dict["identifier_tk"]:
        idplace = tokens[1]
        tokens = lex()
        line = tokens[2]
        if tokens[0] == my_dict["anaphesi_tk"]:
            tokens = lex()
            line = tokens[2]
            if tokens[0] == my_dict["int"]:
                tokens = lex()
                line = tokens[2]
                genQuad('inp', idplace, '_', '_')
            if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
                tokens = lex()
                line = tokens[2]
                if tokens[0] == my_dict["input"]:
                    tokens = lex()
                    line = tokens[2]
                    if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
                        tokens = lex()
                        line = tokens[2]
                        if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
                            tokens = lex()
                            line = tokens[2]

```

```

        if tokens[0] == my_dict["erwtimatiiko_tk"]:
            tokens = lex()
            line = tokens[2]
        else:
            print("Error, there is not ; after assignment in line", line)
            exit(-1)
        else:
            print("Error, there is not ) after int( in line", line)
            exit(-1)
        else:
            print("Error, there is not ) after input( in line", line)
            exit(-1)
        else:
            print("Error, there is not ( after input in line", line)
            exit(-1)
        else:
            print("Error, there is not input after ( in line", line)
            exit(-1)
        else:
            print("Error, there is not ( after int in line", line)
            exit(-1)
    else:
        Eplace = expression()
        genQuad('=', Eplace, '_', idplace)
        if tokens[0] == my_dict["erwtimatiiko_tk"]:


```

```

            tokens = lex()
            line = tokens[2]
        else:
            print("Error ,there is not ; after assignment in line", line)
            exit(-1)
        else:
            print("Error, there is not = for assignment statement in line", line)
            exit(-1)
    else:
        print("Error,there is not an identifier for assignment in line", line)
        exit(-1)

def print_stat():
    global tokens
    global line

    if tokens[0] == my_dict["print"]:
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
        Eplace = expression()
        genQuad('out', Eplace, '_', '_')
    if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["erwtimatiiko_tk"]:


```

```

            tokens = lex()
            line = tokens[2]
        else:
            print("Error,there is not ; after print in line", line)
            exit(-1)
        else:
            print("Error,there is not ) after print( in line", line)
            exit(-1)
        else:
            print("Error,there is not ( after print in line", line)
            exit(-1)
    else:
        print("Error,there is not print in line", line)
        exit(-1)

def return_stat():
    global tokens
    global line

    if (tokens[0] == my_dict["return"]):
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
        Eplace = expression()
        genQuad('retv', Eplace, '_', '_')
    if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:


```

```

tokens = lex()
line = tokens[2]
if tokens[0] == my_dict["erwtimatiiko_tk"]:
    tokens = lex()
    line = tokens[2]
else:
    print("Error,there is not ; after return in line",line)
    exit(-1)
else:
    print("Error,there is not ) after return( in line",line)
    exit(-1)
else:
    print("Error,there is not ( after return in line",line)
    exit(-1)
else:
    print("Error,there is not return in line",line)
    exit(-1)

def if_stat():
    global tokens
    global line
    global ifList
    Btrue = []
    Bfalse = []
    ifList = []

    if tokens[0] == my_dict["if"]:
        tokens = lex()

        line = tokens[2]
        if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
            tokens = lex()
            line = tokens[2]
            B = condition()
            Btrue = B[0]
            Bfalse = B[1]
            backPatch(Btrue, nextQuad())
        if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
            tokens = lex()
            line = tokens[2]
            if tokens[0] == my_dict["anw_katw_teleia_tk"]:
                tokens = lex()
                line = tokens[2]
                if tokens[0] == my_dict["anoigma_block_tk"]:
                    tokens = lex()
                    line = tokens[2]

                    statements()
                    ifList = makeList(nextQuad())
                    genQuad("jump", "_", "_", "_")
                    backPatch(Bfalse, nextQuad())
                if tokens[0] == my_dict["kleisimo_block_tk"]:
                    tokens = lex()
                    line = tokens[2]
                else:
                    print("Error there is not #{ after if #{ in line", line)
                    exit(-1)

            else:
                statement()
                ifList = makeList(nextQuad())
                genQuad('jump', '_', '_', '_')
                backPatch(Bfalse, nextQuad())
        if tokens[0] == my_dict["else"]:
            tokens = lex()
            line = tokens[2]
            if tokens[0] == my_dict["anw_katw_teleia_tk"]:
                tokens = lex()
                line = tokens[2]
                if tokens[0] == my_dict["anoigma_block_tk"]:
                    tokens = lex()
                    line = tokens[2]
                    statements()
                    backPatch(ifList, nextQuad())
                if tokens[0] == my_dict["kleisimo_block_tk"]:
                    tokens = lex()
                    line = tokens[2]
                else:
                    print("Error there is not #{ after else: #{ in line", line)
                    exit(-1)
            else:
                statement()
                backPatch(ifList, nextQuad())
        else:
            print("Error,there is not : after else in line",line)
            exit(-1)
    
```

```

        else:
            backPatch(ifList,nextQuad())
        else:
            print("Error,there is not : after if in line",line)
            exit(-1)
        else:
            print("Error,there is not ) after if( in line",line)
            exit(-1)
        else:
            print("Error,there is not ( after if in line",line)
            exit(-1)
    else:
        print("Error,there is not if in line",line)
        exit(-1)

def while_stat():
    global tokens
    global line
    Btrue = []
    Bfalse = []

    if tokens[0] == my_dict["while"]:
        tokens = lex()
        line = tokens[2]
        if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
            tokens = lex()
            line = tokens[2]
            Bquad = nextQuad()
            tokens = lex()

            B = condition()
            Btrue = B[0]
            Bfalse = B[1]
            backPatch(Btrue, nextQuad())
            if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
                tokens = lex()
                line = tokens[2]
                if tokens[0] == my_dict["anw_katw_telaia_tk"]:
                    tokens = lex()
                    line = tokens[2]
                    if tokens[0] == my_dict["anoigma_block_tk"]:
                        tokens = lex()
                        line = tokens[2]
                        statements()
                        genQuad("jump", "_", "_", Bquad)
                        backPatch(Bfalse, nextQuad())
                        if tokens[0] == my_dict["kleisimo_block_tk"]:
                            tokens = lex()
                            line = tokens[2]
                        else:
                            print("Error there is not #} after while #{ in line", line)
                            exit(-1)
                    else:
                        statement()
                        genQuad("jump", "_", "_", Bquad)
                        backPatch(Bfalse, nextQuad())
                else:
                    print("Error,there is not : after while in line", line)
                    exit(-1)
            else:
                print("Error,there is not ) after while( in line", line)
                exit(-1)
        else:
            print("Error,there is not ( after while in line", line)
            exit(-1)
    else:
        print("Error,there is not while in line", line)
        exit(-1)

def id_list():
    global tokens
    global line
    global _isArgument

    if tokens[0] == my_dict["identifier_tk"]:
        ID = tokens[1]
        tokens = lex()
        line = tokens[2]

        if _isArgument == True: #if it is argument and it was called from the def_function
            argument = Argument()
            argument.name = ID
            add_argument(argument) # we add a new argument at the argumentList
        else: #if it is a variable and it was called from the declaration_line
            entity = Entity()
            entity._assign(entity.ID, "VARIABLE", calculate_offset()) #we assign values at the fields of the object entity

```

```

add_entity(entity) #we add the entity at the Entitylist

while tokens[0] == my_dict["komma_tk"]:
    tokens = lex()
    line = tokens[2]
    ID = tokens[1]
    if tokens[0] == my_dict["identifier_tk"]:
        tokens = lex()
        line = tokens[2]
        if isArgument == True: #if it is an argument and it was called from the def_function
            argument = Argument()
            argument.name = ID
            add_argument(argument) #we add the argument at the argumentList
        else:
            entity = Entity() #if it is a variable and it was called from the declaration_line
            entity_assign(entity, ID, "VARIABLE", calculate_offset())
            add_entity(entity) #we add the entity at the entityList
    else:
        print("Error,there is not identifier after , in line", line)
        exit(-1)

def expression():
    global tokens
    global line

    optional_sign()
    Tplace = term()
    while tokens[0] == my_dict["plus_tk"] or tokens[0] == my_dict["minus_tk"]:


```

```

        add_operator = ADD_OP()
        T2place = term()
        w = new_temp()
        genQuad(add_operator, T1place, T2place, w)
        T1place = w
        Eplace = T1place
        return Eplace

def term():
    global tokens
    global line

    Fplace = factor()
    while tokens[0] == my_dict["multiply_tk"] or tokens[0] == my_dict["slash_tk"]:
        mul_operator = MUL_OP()
        F2place = factor()
        w = new_temp()
        genQuad(mul_operator, F1place, F2place, w)
        F1place = w
        Tplace = F1place
        return Tplace

def factor():
    global tokens
    global line

    if tokens[0] == my_dict["arithmitiki_stathera_tk"]:


```

```

        Fplace = tokens[1]
        tokens = lex()
        line = tokens[2]

    elif tokens[0] == my_dict["anoigma_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
        Eplace = expression()
        if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
            Fplace = Eplace
            tokens = lex()
            line = tokens[2]
        else:
            print("Error,there is not ) after ( at factor in line", line)
            exit(-1)
    elif tokens[0] == my_dict["identifier_tk"]:
        function_name = tokens[1]
        tokens = lex()
        line = tokens[2]
        Fplace = idtail(function_name)
    else:
        print("Error,there is not numerical constant or expression or identifier at factor in line", line)
        exit(-1)
    return Fplace

def idtail(function_name):
    global tokens
    global line
```

```

if tokens[0] == my_dict["anoigma_parenthesi_tk"]:
    tokens = lex()
    line = tokens[2]
    actual_par_list()
    w = new_temp()
    genQuad('par', w, 'RET', '_')
    genQuad('call', function_name, '_', '_')
if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
    tokens = lex()
    line = tokens[2]
    return w
else:
    print("Error ,there is not ) after ( at idtail ",line)
    exit(-1)
else:
    return function_name

def actual_par_list():
    global tokens
    global line

    if tokens[0] == my_dict["arithmitiki_stathera_tk"] or tokens[0] == my_dict["anoigma_parenthesi_tk"] or tokens[0]== my_dict["identifier_tk"]:
        Eplace = expression()
        genQuad('par', Eplace, 'CV', '_')
        while tokens[0] == my_dict["komma_tk"]:
            tokens = lex()
            line = tokens[2]

Eplace = expression()
genQuad('par', Eplace, 'CV', '_')

def optional_sign():
    global tokens
    global line

    if tokens[0] == my_dict["plus_tk"] or tokens[0] == my_dict["minus_tk"]:
        ADD_OP()

def ADD_OP():
    global tokens
    global line
    add_operator = tokens[1]
    if tokens[0] == my_dict["plus_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["minus_tk"]:
        tokens = lex()
        line = tokens[2]
    return add_operator

def MUL_OP():
    global tokens
    global line
    mull_operator = tokens[1]

    if tokens[0] == my_dict["multiply_tk"]:
        tokens = lex()

        line = tokens[2]
    elif tokens[0] == my_dict["slash_tk"]:
        tokens = lex()
        line = tokens[2]
    return mull_operator

def condition():
    global tokens
    global line
    Btrue = []
    Bfalse = []
    Q1 = bool_term()
    Q1true = Q1[0]
    Q1false = Q1[1]
    Btrue = Q1true
    Bfalse = Q1false
    while tokens[0] == my_dict["or"]:
        backPatch(Q1false, nextQuad())
        tokens = lex()
        line = tokens[2]
        Q2 = bool_term()
        Q2true = Q2[0]
        Q2false = Q2[1]
        Btrue = merge(Q1true,Q2true)
        Bfalse = Q2false
    return Btrue,Bfalse

```

```

def bool_term():
    global tokens
    global line
    Btrue = []
    Bfalse = []

    B1 = bool_factor()
    Qtrue = B1[0]
    Qfalse = B1[1]
    Btrue = Qtrue
    Bfalse = Qfalse

    while tokens[0] == my_dict["and"]:
        backPatch(Qtrue, nextQuad())
        tokens = lex()
        line = tokens[2]
        B2 = bool_factor()
        Q2true = B2[0]
        Q2false = B2[1]
        Qfalse = merge(Qfalse, Q2false)
        Qtrue = Q2true
        Btrue = Qtrue
        Bfalse = Qfalse
    return Btrue, Bfalse


def bool_factor():
    global tokens
    global line
    Btrue = []
    Bfalse = []

    if tokens[0] == my_dict["not"]:
        tokens = lex()
        line = tokens[2]
        if tokens[0] == my_dict["anoigma_aggioi_tk"]:
            tokens = lex()
            line = tokens[2]
            R = condition()
            if tokens[0] == my_dict["kleisimo_aggioi_tk"]:
                tokens = lex()
                line = tokens[2]
                Rtrue = R[0]
                Rfalse = R[1]
                Btrue = Rtrue
                Bfalse = Rtrue

            else:
                print("Error, there is not ] after not[ at boolfactor in line", line)
                exit(-1)
        else:
            print("Error, there is not [ after not at boolfactor in line", line)
            exit(-1)

    elif tokens[0] == my_dict["anoigma_aggioi_tk"]:
        tokens = lex()
        line = tokens[2]
        R = condition()
        if tokens[0] == my_dict["kleisimo_aggioi_tk"]:
            tokens = lex()
            line = tokens[2]
            Rtrue = R[0]
            Rfalse = R[1]
            Btrue = Rtrue
            Bfalse = Rfalse

        else:
            print("Error, there is not ] after [ at boolfactor in line", line)
            exit(-1)
    else:
        E1place = expression()
        rel_operator = REL_OP()
        E2place = expression()
        Rtrue = makeList(nextQuad())
        genQuad(rel_operator, E1place, E2place, "-")
        Rfalse = makeList(nextQuad())
        genQuad("jump ", " - ", " - ", "-")
        Btrue = Rtrue
        Bfalse = Rfalse
    return Btrue, Bfalse

```

```

def REL_OP():
    global tokens
    global line
    rel_operator = tokens[1]

    if tokens[0] == my_dict["ison_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["diafore_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["smaller_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["smaller or equal_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["bigger_tk"]:
        tokens = lex()
        line = tokens[2]
    elif tokens[0] == my_dict["bigger or equal_tk"]:
        tokens = lex()
        line = tokens[2]
    else:
        print("Error, there is not == or != or > or >= or < or <= in line", line)
        exit(-1)
    return rel_operator

```

```

def call_main_part():
    global tokens
    global line

    if tokens[0] == my_dict["if"]:
        tokens = lex()
        line = tokens[2]
        if tokens[0] == my_dict["__name__"]:
            tokens = lex()
            line = tokens[2]
            if tokens[0] == my_dict["ison_tk"]:
                tokens = lex()
                line = tokens[2]
            if tokens[0] == my_dict["quotes_tk"]:
                tokens = lex()
                line = tokens[2]
                if tokens[0] == my_dict["__main__"]:
                    tokens = lex()
                    line = tokens[2]
                    if tokens[0] == my_dict["quotes_tk"]:
                        tokens = lex()
                        line = tokens[2]
                        genQuad('begin_block', 'main', '_', '_')
                        if tokens[0] == my_dict["anw_kata_teleia_tk"]:
                            tokens = lex()
                            line = tokens[2]
                            main_function_call()
                            while tokens[0] == my_dict["identifier_tk"]:
                                main_function_call()

                                genQuad('halt', '_', '_', '_')
                                genQuad('end_block', 'main', '_', '_')
                                show_symbols(my_file2)
                                generate_final()
                                delete_scope()
                            else:
                                print("Error, there is not : after call_main_part in line", line)
                                exit(-1)
                        else:
                            print("Error, there is not quotes at call_main_part in line", line)
                            exit(-1)
                    else:
                        print("Error, there is not __main__ at call_main_part in line", line)
                        exit(-1)
                else:
                    print("Error, there is not quotes at call_main_part in line", line)
                    exit(-1)
            else:
                print("Error, there is not == at call_main_part in line", line)
                exit(-1)
        else:
            print("Error, there is not __name__ at call_main_part in line", line)
            exit(-1)
    else:
        print("Error, there is not if at call_main_part in line", line)
        exit(-1)

```

```

def main_function_call():
    global tokens
    global line

    if tokens[0] == my_dict["identifier_tk"]:
        ID = tokens[1]
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["enoigma_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
        tokens = lex()
        line = tokens[2]
    if tokens[0] == my_dict["erwtimatiiko_tk"]:
        genQuad('call', ID, '_', '_')
        tokens = lex()
        line = tokens[2]
    else:
        print("Error, there is not ; at main_function_call in line", line)
        exit(-1)
    else:
        print("Error, there is not ) at main_function_call in line", line)
        exit(-1)
    else:
        print("Error, there is not ( at main_function_call in line", line)
        exit(-1)
    else:
        print("Error, there is not identifier at main_function_call in line", line)

if tokens[0] == my_dict["kleisimo_parenthesi_tk"]:
    tokens = lex()
    line = tokens[2]
if tokens[0] == my_dict["erwtimatiiko_tk"]:
    genQuad('call', ID, '_', '_')
    tokens = lex()
    line = tokens[2]
else:
    print("Error, there is not ; at main_function_call in line", line)
    exit(-1)
else:
    print("Error, there is not ) at main_function_call in line", line)
    exit(-1)
else:
    print("Error, there is not ( at main_function_call in line", line)
    exit(-1)
else:
    print("Error, there is not identifier at main_function_call in line", line)
    exit(-1)

startRule()
syn()
write_Quads(my_file)
my_file.close()
my_file2.close()
final_file.close()

```

Όπως παρατηρούμε στις παραπάνω εικόνες για κάθε κανόνα της γραμματικής υλοποιούμε μία συνάρτηση. Δίνουμε στη συνάρτηση το ίδιο όνομα με τον κανόνα ή τουλάχιστον έναν όνομα που να αντιστοιχίζεται μονοσήμαντα στον κανόνα. Σε όλες τις περιπτώσεις γίνεται έλεγχος αν η λεκτική μονάδα που επιστρέφει ο λεκτικός αναλυτής αντιστοιχεί στη λεκτική μονάδα που αναμένεται από τη γραμματική, χρησιμοποιώντας το recognized_string που περιέχει το token. Σε περίπτωση που δεν είναι το επιθυμητό πηγαίνει στο else του κάθε ελέγχου και εμφανίζει το κατάλληλο λάθος καθώς και την γραμμή που υπάρχει το λάθος. Για παράδειγμα στο while _stat πρώτα ελέγχουμε αν το string του token είναι η λέξη while αλλιώς βγάζει το μήνυμα λάθους ότι δεν υπάρχει η λέξη while στην γραμμή που έχουμε μέσα στο token μας. Στη συνέχεια ξανακαλεί τη συνάρτηση του λεκτικού μας αναλυτή και ελέγχει αν είναι το άνοιγμα παρένθεσης και καλεί τη συνάρτηση του condition. Έπειτα καλεί ξανά τη συνάρτηση του λεκτικού και ξαναελέγχει αν αυτή τη φορά είναι το κλείσιμο παρένθεσης.

Έλεγχος ορθότητας συντακτικού αναλυτή

Για να ελέγχουμε ότι λειτουργεί σωστά ο συντακτικός μας αναλυτής δημιουργήσαμε συγκεκριμένα τεστ αρχεία.

```
##$ program to find the L.C.M of two input number ##

def main_compute_lcm():
    #{

        #declare x,y
        def compute_lcm(x,y):
            #{

                #choose the greater number##
                #declare greater,lcm,u1,u2

                if (x>y):
                    greater = x;
                else:
                    greater = y;
                u1 = greater - (greater // x)* x ;
                u2 = greater - (greater // y)* y ;
                while([u1 != 0] or [u2 !=0]):
                    greater = greater + 1;
                lcm = greater;
                return (lcm);
            //}

            num1 = int(input());
            num2 =int(input());

            print(compute_lcm(num1,num2));
        //}

        if __name__ == "__main__":
            main_compute_lcm();
    #}
}

if __name__ == "__main__":
    main_compute_lcm();
```

Αρχικά ελέγχουμε ότι οι κύριες συναρτήσεις είναι σύμφωνα με την γραμματική μας δομημένες όπως και ο τρόπος που τις καλούμε στο κυρίως πρόγραμμα. Ελέγχουμε τη δομή ελέγχου if – else με μια εντολή μέσα τους χωρίς άνοιγμα block και κλείσιμο μπλοκ καθώς και τις αναθέσεις τιμών στις μεταβλητές. Επιπλέον ελέγχουμε τις δηλώσεις μεταβλητών αν ορίζονται όπως στην cutePy με το #declare και με κόμματα μεταξύ των μεταβλητών. Το ίδιο συμβαίνει και με την δομημένη εντολή while όπου ελέγχουμε αν λειτουργεί σωστά με τις αγκύλες ο λογικός έλεγχος που έχει μέσα του. Αντίστοιχα, μπορούμε να δούμε αν και εντολές εισόδου input και εντολές εξόδου print ακολουθούν το σωστό συντακτικό τη γλώσσα cutePy. Τέλος στο συγκεκριμένο παράδειγμα ελέγχεται τόσο η εντολή return αλλά και οι expressions και η δομή τους.

Ενα ακόμα παράδειγμα είναι:

```
def main_factorial():
    #{

        ## declarations ##
        #declare x
        #declare i,fact

        ## body of main_factorial ##
        x = int(input());
        fact = 1;
        i = 1;
        while (i<=x):
            #
                fact = fact * i;
                i = i + 1;
            #
        print(fact);
    #}
```

Όπου εδώ ελέγχουμε την while με άνοιγμα και κλείσιμο μπλοκ καθώς έχει μέσα τις παραπάνω από μια εντολές καθώς και οι αναθέσεις που κάνουμε με expressions που έχουν αθροιστικούς και πολλαπλασιαστικούς τελεστές.

```
def main_primes():
#{
    #declare x
    def isPrime(x):
    #{
        #declare i
        def divides (x ,y):
        #{
            if (y == (y//x) * x):
                return (1);
            else:
                return (0);
        }
        i = 2;
        while (i<x):
        #{
            if (divides(i,x)==1):
                return (0);
            i = i + 1;
        }
    }
    return (1) ;
}
```

Στον παραπάνω έλεγχο βλέπουμε την δομή της τοπικής συνάρτησης η οποία βρίσκεται μέσα σε μια κύρια συνάρτηση καθώς και ένα εμφωλιασμένο if μέσα στο while μόνο του χωρίς else για να σιγουρευτούμε ότι λειτουργεί και σωστά μόνη της η δομή του if . Ακόμα παρατηρούμε το condition μέσα στο if όπου περιλαμβάνει τη κλήση μιας τοπικής συνάρτησης με τις παραμέτρους που της είχαμε θέσει.

Για να τρέξουμε τα τεστ αρχεία πάλι γράφαμε στον τερματικό την εντολή

Python cutePy_4168_4163.py test1.cpy

Και επαληθεύαμε ότι έτρεχε σωστά ο συντακτικός μας όταν μας εμφανιζόταν στην οθόνη το παρακάτω μήνυμα

Syntax analysis finished without errors!

Διαχείριση σφαλμάτων στον συντακτικό αναλυτή

Ο συντακτικός αναλυτής είναι υπεύθυνος για να εμφανίζει στην οθόνη μας τα καταλληλά μηνύματα λάθους σε περίπτωση που παραλείψουμε σε κάποιο πρόγραμμα μας κάποιον χαρακτήρα ή δεν έχουμε την σωστή δομή στις προκαθορισμένες εντολές και συναρτήσεις της cutePy. Για να εξετάσουμε ότι θα μας εμφανίσει το κατάλληλο λάθος δοκιμάσαμε στα τεστ αρχεία μας να κάνουμε κάποια λάθη ώστε να δούμε τι μηνύματα λάθους θα εμφανίστούν

Ενδεικτικά:

```

def main_func():
    #{ 
        #declare number
    def is_square(number):
        #{ 
            #declare i,sq
            i = 0;
            sq = 0;
            while (sq < number):
                #{ 
                    i = i + 1;
                    sq = i * i;
                }
                if (sq == number):
                    return (1);
                else:
                    return(0);
            }
        def is_cube(number):
            #{ 
                #declare i,cub
                i = 0 ;
                cub = 0;
                while (cub < number):
                    #{ 
                        i = i +1;
                        cub = i * i * i;
                    }
                    if (cub == number):
                        return (1);
                    else:
                        return(0);
                }
            }

```

Δεν κλείσαμε την παρένθεση στη συνάρτηση μας και εμφανίστηκε το παρακάτω λάθος

```

C:\python\ex3>python cutePy_4168_4163.py test4.py
[100, 'def', 3]
[51, 'main_func', 3]
[71, '(', 3]
[70, ')', 3]
[56, ':', 3]
[72, '#{', 4]
[102, '#declare', 5]
[51, 'number', 5]
[100, 'def', 7]
[51, 'is_square', 7]
[71, '(', 7]
[51, 'number', 7]
[56, ':', 7]
Error, the program does not have a ) after ( at function in line 7

```

Στο παρακάτω δεν βάλαμε το άνοιγμα block μετά την συνάρτηση μας

```

def main_calculate () :
    #§ our own calculator  #§
    #{ 
        #declare x, y,num1,num2,result,choice
    def add(x,y):
        -   return (x + y);
    }
    def subtract(x,y):
        -   return (x - y);
    }
    def multiply(x,y):
        -   return (x - y);
    }
    def divide(x,y):
        -   return (x //y);
    }

```

Και εμφανίστηκε πάλι το αντίστοιχο λάθος στο τερματικό μας

```
C:\python\ex3>python cutePy_4168_4163.py test3.cpy
[100, 'def', 3]
[51, 'main_calculate', 3]
[71, '(', 3]
[70, ')', 3]
[56, ':', 3]
[72, '#{', 5]
[102, '#declare', 6]
[51, 'x', 6]
[55, ',', 6]
[51, 'y', 6]
[55, ',', 6]
[51, 'num1', 6]
[55, ',', 6]
[51, 'num2', 6]
[55, ',', 6]
[51, 'result', 6]
[55, ',', 6]
[51, 'choice', 6]
[100, 'def', 7]
[51, 'add', 7]
[71, '(', 7]
[51, 'x', 7]
[55, ',', 7]
[51, 'y', 7]
[70, ')', 7]
[56, ':', 7]
[108, 'return', 9]
Error, the program does not have #! after function in line 9
```

Στο συγκεκριμένο δεν βάλαμε μετά το input τις παρενθέσεις. Παρατηρούμε επίσης ότι εμφανίζει και το λάθος στην κατάλληλη γραμμή που πρέπει.

```
5  def main_compute_lcm():
6  #{
7      #declare x,y
8      def compute_lcm(x,y):
9      #{
10         #choose the greater number#
11         #declare greater,lcm,u1,u2
12
13         if (x>y):
14             greater = x;
15         else:
16             greater = y;
17             u1 = greater - (greater // x)* x ;
18             u2 = greater - (greater // y)* y ;
19             while([u1 != 0] or [u2 !=0]):
20                 greater = greater + 1;
21                 lcm = greater;
22                 return (lcm);
23             #
24             num1 = int(input());
25             num2 =int(input());
26
27             print(compute_lcm(num1,num2));
28         #
29
30         if __name__ == "__main__":
31             main_compute_lcm();
```

```
[70, ')', 24]
Error,there is not ( after input in line 24
```

Με αυτό τον τρόπο συμπεραίνουμε ότι ο συντακτικός μας αναλυτής μπορεί να εμφανίσει τα καταλληλά μηνύματα λάθους στις γραμμές που εμφανίζονται. Αυτό το καταφέραμε βάζοντας σε κάθε συνάρτηση του συντακτικού μας αναλυτή, ελέγχους αναφέροντας πάντοτε το όνομα της δομής που έχει λάθος.

Ενδιάμεσος κώδικας

Η μετατροπή του κώδικα από την αρχική γλώσσα στην τελική δεν γίνεται απευθείας. Μεσολαβεί η μετατροπή του σε μία ενδιάμεση γλώσσα, την οποία συνηθίζουμε να λέμε και ενδιάμεση αναπαράσταση, η οποία εξακολουθεί να θεωρείται γλώσσα υψηλού επιπέδου, αλλά οι δομές της δεν είναι τόσο σύνθετες, όσο αυτές της αρχικής γλώσσας. Ένα πρόγραμμα συμβολισμένο στην ενδιάμεση γλώσσα αποτελείται από μία σειρά από τετράδες, οι οποίες είναι αριθμημένες έτσι ώστε σε κάθε τετράδα να μπορούμε να αναφερθούμε

χρησιμοποιώντας τον αριθμό της ως ετικέτα. Κάθε τετράδα αποτελείται από έναν τελεστή ,τρία τελούμενα και μαζί με την ετικέτα αποτελούν πρακτικά μια πεντάδα. Στον κώδικα μας τις τετράδες τις αποθηκεύουμε σε ένα αρχείο που δημιουργούμε κατά την εκτέλεση του μεταφραστή μας το endiamesos.int.

Υλοποίηση ενδιάμεσου κώδικα

```

fullist = []
fullist2 = []
quadlabel = 1
counter = -1
isArgument = True

def nextQuad():
    global fullist
    global quadlabel
    return quadlabel

def genQuad(op,x,y,z):
    global fullist
    global fullist2
    global quadlabel
    my_list = []
    my_list = [nextQuad()]
    my_list.insert(1,str(op))
    my_list.insert(2,str(x))
    my_list.insert(3,str(y))
    my_list.insert(4,str(z))
    fullist += [my_list]
    fullist2 += [my_list]
    quadlabel = quadlabel + 1
    return my_list

def new_temp():
    global fullist
    global counter
    counter += 1
    entity = Entity()
    entity_assign(entity,"%"+ str(counter),"TEMPORARYVARIABLE",calculate_offset())
    add_entity(entity)

    return "%" + str(counter)

def empty_list():
    return []

def makeList(label):
    return [label]

def merge(list1,list2):
    return list1 +list2

def backPatch(list,z):
    global fullist
    list.sort()
    y = 0
    count = 0
    for i in list:
        for j in fullist[y:]:
            if j[0] == i:
                j[4] = str(z)
                count += 1
                y = count
                break
            else:
                count += 1
        count = 0

def print_listOfAllQuads():
    global fullist
    for i in fullist:
        print(str(i)+"\n")

```

Με τις παραπάνω βιοηθητικές συναρτήσεις ,τις οποίες καλούμε μέσα στον συντακτικό μας αναλυτή προκειμένου να πράξουμε τις τετράδες, υλοποιείται ο ενδιάμεσος κώδικας.

Θα πρέπει να αναζητήσουμε τα σημεία εκείνα της γραμματικής, και κατ' επέκταση του κώδικα του συντακτικού αναλυτή, στα οποία πρέπει να εισαχθούν σημασιολογικές ρουτίνες που θα παραγάγουν ενδιάμεσο κώδικα. Λόγω της ανεξαρτησίας των κανόνων της γραμματικής μας επιτρέπεται να κάνουμε σχέδιο ενδιάμεσου κώδικα για κάθε κανόνα ξεχωριστά.

Ας δούμε για παράδειγμα έναν κανόνα που εφαρμόσαμε κώδικα ενδιάμεσου κώδικα..

```
def expression():
    global tokens
    global line
    optional_sign()
    T1place = term()
    while tokens[0] == my_dict["plus_tk"] or tokens[0] == my_dict["minus_tk"]:
        add_operator = ADD_OP()
        T2place = term()
        w = new_temp()
        genQuad(add_operator, T1place, T2place, w)
        T1place = w
    Eplace = T1place
    return Eplace

def term():
    global tokens
    global line

    F1place = factor()
    while tokens[0] == my_dict["multiply_tk"] or tokens[0] == my_dict["slash_tk"]:
        mul_operator = MUL_OP()
        F2place = factor()
        w = new_temp()
        genQuad(mul_operator, F1place, F2place, w)
        F1place = w
    Tplace = F1place
    return Tplace
```

Μια αριθμητική παράσταση (expression) γνωρίζουμε ότι έχει σχέδιο ενδιάμεσου κώδικα

$E \rightarrow T_1 (+ T_2 \{P1\})^* \{P2\}$ όπου T_1, T_2 είναι αθροιστικοί όροι. Στον κώδικα μας παράγονται από τον κανόνα term, ο οποίος αντίστοιχα καλεί άλλον κανόνα τον factor, και αποθηκεύονται στο $T_1.place$ και στα $T_2.place$. Τα $P1$ και $P2$ μας υποδεικνύουν που πρέπει να καλέσουμε τις βιοθητικές συναρτήσεις προκειμένου να παραχθεί η τετράδα του ενδιάμεσου που χρειάζεται. Μια παράσταση μπορεί να έχει από έναν μέχρι όσους θέλει αθροιστικούς ορούς. Στην πρώτη θέση μετά την ετικέτα φυσικά θα μπει ο αθροιστικός τελεστής μετρά οι 2 αθροιστικοί όροι και θα αποθηκευτούν σε μια προσωρινή μεταβλητή που θα δημιουργήσουμε και μετρά αυτή θα την εκχωρήσουμε στην μεταβλητή $T_1.place$. Στο τέλος βέβαια θα επιστραφεί η μεταβλητή $E.place$ στην οποία αποθηκεύουμε την $T_1.place$.

Έτσι αν έχουμε την παράσταση $a + b$ θα δημιουργηθεί η τετράδα $(+ \alpha \beta T_1)$ όπου το T_1 αποθηκεύεται στο $T_1.place$ και αποθηκεύεται στη συνέχεια στο $E.place$, το οποίο επιστρέφουμε.

Έλεγχος τετράδων ενδιάμεσου κώδικα

Αρχικά φτιάξαμε κάποια αρχεία τεστ για να δοκιμάσουμε ότι δημιουργούνται σωστά όλες οι τετράδες του ενδιάμεσου κώδικα.

```

def main_compute():
#{ 
    #declare x,z
    def compute():
    #{ 
        #declare k,m
        def iscube(num):
        #{ 
            #declare i,cub
            i = 0;
            cub = 0;
            while (cub < num):
                #{ 
                    i = i + 3;
                    cub = i * i * i;
                }
                if (cub == num):
                    return(1); #$if the num is cube returns 1 ,otherwise returns0 $#
                else:
                    return(0);
            }
        #$ main body of compute $#
        k = int(input());
        while( k <= 0 ): #$we want numbers positive $#
            k = int(input());
            print(k);

        m = iscube(k);
        print(m);
        return(m);
    }
}

```

```

def issquare(d):
#{ 
    #declare p,sq
    p = 0;
    sq = 0;
    while (sq < d):
        #{ 
            p = p + 1;
            sq = p * p;
        }
        if (sq == d):
            return (1); #$if the num is square returns 1 ,otherwise returns 0 $#
        else:
            return (0);
    }
    #$ main body of main_compute $#
    z = compute();
    x = 4;
    print(issquare(x));
    ex = 5*x // 2 + 1;
    print(ex);
}
#}

if __name__ == "__main__":
    main_compute();

```

Οι τετράδες που δημιουργούνται στο αρχείο endiamesos.int από το συγκεκριμένο αρχείο τεστ είναι:

```

1: begin_block iscube --
2: = 0 _ i
3: = 0 _ cub
4: < cub num 6
5: jump _ _ 12
6: + i 3 %0
7: = %0 _ i
8: * i i %1
9: * %1 i %2
10: = %2 _ cub
11: jump _ _ 4
12: == cub num 14
13: jump _ _ 16
14: retv 1 _ _ 16
15: jump _ _ 17
16: retv 0 _ _ 17
17: end_block iscube
18: begin_block compute _ _
19: inp k _ _
20: <= k 0 22
21: jump _ _ 24
22: inp k _ _ 24
23: jump _ _ 20
24: out k _ _
25: par k CV_
26: par %3 RET
27: call iscube _ _
28: = %3 _ m
29: out m _ _
30: retv m _ _
31: end_block compute
32: begin_block issquare _ _
33: = 0 _ p
34: = 0 _ sq
35: < sq d 37
36: jump _ _ 42
37: + p 1 %4
38: = %4 _ p

```

```

39: * p p %5
40: = %5 _ sq
41: jump _ _ 35
42: == sq d 44
43: jump _ _ 46
44: retv 1 _ _ 46
45: jump _ _ 47
46: retv 0 _ _
47: end_block issquare
48: begin_block main_compute _ _
49: par %6 RET
50: call compute _ _
51: = %6 _ z
52: = 4 _ x
53: par x CV_
54: par %7 RET
55: call issquare _ _
56: out %7 _ _
57: * 5 x %8 _ _
58: // %8 2 %9
59: + %9 1 %10
60: = %10 _ ex
61: out ex
62: end_block main_compute
63: begin_block main _ _
64: call main_compute _ _
65: halt
66: end_block main

```

Όπως παρατηρούμε δημιουργήθηκε η πρώτη τετράδα για το begin block του υποπρογράμματος iscube καθώς έχουμε βάλει στον κανόνα του def_function να δημιουργείται η τετράδα όταν ξεκινάει η συνάρτηση. Στη συνέχεια δημιουργούνται οι τετράδες για τις αναθέσεις των μεταβλητών της συνάρτησης στον κανόνα assignment_stat. Έπειτα έχουμε το πρώτο condition στο while που δημιουργεί δυο τετράδες. Στον κανόνα condition που καλεί δημιουργούνται δυο λίστες οι Btrue και Bfalse και αντίστοιχα καλεί τον κανόνα boolterm, ο οποίος καλεί τον κανόνα bool_factor ο οποίος με τη σειρά δημιουργεί 2 τετράδες μια για το έλεγχο δηλαδή (< cub num 6) και την αποθηκεύει στη Btrue και μια άλλη τετράδα που είναι ένα (jump _ _ _) και την αποθηκεύει στο Bfalse και αντίστοιχα τα επιστρέφει στο bool_term όπου με τη σειρά του τα επιστρέφει το condition. Με αυτές τις τετράδες αν ισχύει η συνθήκη θα εκτελεστούν οι εντολές μέσα στο while ειδάλλως όταν θα εκτελεστεί το πρόγραμμα θα πάει στις παρακάτω εντολές που αρχίζουν μετρά το while δηλαδή στην τετράδα με την ετικέτα 12 όπου κάνει το αντίστοιχο έλεγχο για το if μετά το while.

Η επόμενη τετράδα είναι για την εντολή $i = i + 3$ του προγράμματος μας. Σε αυτή την τετράδα έχουμε μια expression, η οποία εκχωρείται σε μια μεταβλητή. Πρώτα θα δημιουργηθεί η τετράδα για την αριθμητική παράσταση όπου θα αποθηκευτεί σε μια τοπική μεταβλητή $\%0$ δηλαδή η $<<6: + i 3 \%0>>$ και στη συνέχεια η προσωρινή μεταβλητή θα

εικωρηθεί στη μεταβλητή `I <<7: = %0 _ i>>`. Το ίδιο θα συμβεί και για την επομένη εντολή του προγράμματος μας `cub = i * i * i;` Όπου θα αποθηκευτεί το αποτέλεσμα της πράξης `i * i` σε μια προσωρινή μεταβλητή `%1` και μετέπειτα αυτή η προσωρινή μεταβλητή θα πολλαπλασιαστεί με το τρίτο `i` και θα αποθηκευτεί σε μια άλλη προσωρινή μεταβλητή `%2`. Στη συγκεκριμένη περίπτωση το expression κάλεσε το term το οποίο κάλεσε με τη σειρά του τον κανόνα factor. Το `%2` θα εικωρηθεί μετέπειτα στη μεταβλητή `cub` όπως λέει ο κώδικας μας.

Η επομένη τετράδα πρέπει να κάνει `jump` πάνω στο condition του while καθώς έχουμε μια δομή επανάληψης και θέλουμε να ελέγχουμε αν η συνθήκη ισχύει ή όχι ώστε όταν το εκτελούμε να ξαναεκτελεστούν οι εντολές του while ή όχι. Επομένως η τετράδα 11: `jump _ _ 4` μας ξαναπιγαίνει στην τετράδα με την ετικέτα `4` όπου γίνεται ο έλεγχος της συνθήκης του while. Μετά στο πρόγραμμα μας έχουμε την εντολή `eléγχου` η οποία αν ισχύει η συνθήκη θα καλέσει την εντολή `return` και θα επιστρέψει 1 αλλιώς 0. Παλι θα δημιουργηθούν για το condition 2 τετράδες μια θα έχει τον έλεγχο δηλαδή `12: == cub num 14` και θα προχωράει στις τετράδες των εντολών μέσα στο if δηλαδή θα πάει στην τετράδα με την ετικέτα `14` όπου είναι η τετράδα του `return 14: retv 1 _ _` ενώ η δεύτερη τετράδα είναι πάλι ένα `jump _ _ 16` για να μην μπει μέσα στο if αν δεν ισχύει η συνθήκη αλλά στο else και την τετράδα με την ετικέτα `16` όπου είναι η εντολή του `return 0`. Η επόμενη τετράδα φυσικά θα είναι το end block της συνάρτησης `iscube` μιας και δεν υπάρχουν άλλες εντολές μέσα της.

Μετά είναι επόμενο να έχουμε την τετράδα με το begin block της compute μιας και αρχίζουν οι εντολές της. Επειδή έχουμε την εντολή `εισόδου k = int(input())` θα δημιουργηθεί η τετράδα 19: `inp k _ _`. Αντίστοιχα πάλι έχουμε το while ξανά που θα γίνει η ίδια διαδικασία που περιγράφαμε παραπάνω τετράδα της εντολής `print` θα είναι 24: `out k _ _` όσον αφορά την εντολή `print(k)`. Επίσης ιδιαίτερη σημασία πρέπει να δώσουμε όταν καλούμε την συνάρτηση `iscube(k)` όπου θα δημιουργηθούν 3 τετράδες

25: `par k CV _` όπου το `k` είναι η παράμετρος με τιμή (CV)

26: `par %3 RET _` η επιστροφή της συνάρτησης αποθηκεύεται στην προσωρινή μεταβλητή `%3`

27: `call iscude _ _` και η τετράδα που είναι η κλήση της συνάρτησης

Οι άλλες τετράδες ακολουθούν όσα αναφέραμε παραπάνω ανάλογα την εντολή του προγράμματος μας.

Επιλογή των προγραμμάτων τεστ μας

Για να σιγουρευτούμε ότι ο ενδιάμεσος κώδικας μας λειτουργεί σωστά για όλες τις περιπτώσεις δημιουργήσαμε 2 αρχεία `test.cpy`. Το πρώτο τεστ αρχείο σχολιάζεται παραπάνω ενώ το δεύτερο αρχείο είναι το εξής:

```

def main_maximum():
    #declare num1,num2,num3
    #declare maximum
    num1 = 5;
    num2 = 2;
    num3 = int(input());
    maximum = num1;
    if(num2 >= num1):
        if (num3>=num2):
            maximum = num3;
        else:
            maximum = num2;
    else:
        if(num3 >= num1):
            maximum = num3;
    print(maximum); #$ prints the maximum num out of three nums $$

}

def main_menu():
{
    #declare choice
    def menu(ch,l):
    {
        #declare num4,num5,res
        num4 = 12;
        num5 = 6;
        res = 0;
        if ((num4- num5) > (num5 - num4)):
        {
            res = num4 // num5;
            print(res);
            l = 1;
            print(l);
        }

        if(ch == 1):
        {
            res = num4 * num5; #$ if the user gives 1 multiplies the nums $$ 
            return(res);
        }
        if(ch >=2):
            return(res + 1); #$ if the user gives 2 or more returns res + 1 $$

    }
    #$ main body of main_menu $$ 
    q = 0;
    choice = int(input());
    print(menu(choice,q));
    print(3*choice + 5);
}
if __name__ == "__main__":
    main_maximum();
    main_menu();

```

Και όταν το τρέχουμε μας βγαίνουν οι εξής τετράδες:

```
1: begin_block main_maximum --
2: = 5 _ num1
3: = 2 _ num2
4: inp num3 _-
5: = num1 _ maximum
6: >= num2 num1 8
7: jump _- 14
8: >= num3 num2 10
9: jump _- 12
10: = num3 _ maximum
11: jump _- 13
12: = num2 _ maximum
13: jump
14: >= num3 _ num1 16
15: jump _- 18
16: = num3 _ maximum
17: jump _- 18
18: out maximum _-
19: end_block main_maximum --
20: begin_block menu --
21: = 12 _ num4
22: = 6 _ num5
23: = 0 _ res
24: - num4 num5 %0
25: - num5 num4 %1
26: > %0 %1 28
27: jump _- 34
28: // num4 num5 %2
29: = %2 _ res
30: out res _-
31: = 1 1 --
32: out I
33: jump _- 34
34: == ch 1 36
35: jump _- 40
36: * num4 num5 %3
37: = %3 _ res
38: retv res
```

```
39: jump _- 40
40: >= ch 2 42
41: jump _- 45
42: + res 1 %4
43: retv %4
44: jump _- 45
45: end_block menu
46: begin_block main_menu --
47: = 0 _ q
48: inp choice _-
49: par choice CV _-
50: par q CV _-
51: par %5 RET _-
52: call menu _-
53: out %5
54: * 3 choice %6
55: + %6 5 %7
56: out %7
57: end_block main_menu --
58: begin_block main --
59: call main_maximum --
60: call main_menu _-
61: halt
62: end_block main --
```

Αρχικά θέλαμε να ελέγξουμε την εντολή print:

Πήραμε την περίπτωση όπου έχει μεταβλητή (identifier),όπου έχει αριθμητική σταθερά ,την περίπτωση όπου έχει μέσα της αριθμητική παράσταση και την περίπτωση όπου έχει μέσα της την κλήση μιας συνάρτησης.

Αντίστοιχα για το return το καλούμε περνώντας του σαν όρισμα αριθμητική σταθερά, μεταβλητή (identifier) και αριθμητική παράσταση.

Επιπλέον στα conditions ελέγχουμε συγκρίσεις μεταξύ αριθμητικών παραστάσεων ,μεταβλητών(identifier)μόνο και μεταβλητών και αριθμητικών σταθερών.

Οι λογικές εκφράσεις με and or not δεν μπορούν να ελεγχθούν καθώς η γλώσσα python δεν δέχεται τα [].

Ακόμα έχουμε αριθμητικές παραστάσεις με αθροιστικούς και πολλαπλασιαστικούς τελεστές .

Επιπρόσθετα στην ανάθεση ελέγχουμε τις περιπτώσεις ανάθεσης αριθμητικής σταθεράς, αριθμητικής παράστασης , επιστροφής συνάρτησης ,μεταβλητής και συνάρτησης εισόδου.

Στις δηλώσεις μεταβλητών κάνουμε με μια μεταβλητή και αντίστοιχα με πολλές όπου διαχωρίζονται με κόμματα. Στις συναρτήσεις έχουμε αντίστοιχα μια ή παραπάνω παραμέτρους.

Τέλος έχουμε κλήσεις μιας ή και παραπάνω κύριων συναρτήσεων όπως και μέσα στις κύριες συναρτήσεις τοπικές συναρτήσεις.

Πίνακας συμβόλων

Ο πίνακας συμβόλων είναι δυναμική δομή στην οποία αποθηκεύεται πληροφορία σχετιζόμενη με τα συμβολικά ονόματα που χρησιμοποιούνται στο υπό μεταγλώττιση πρόγραμμα. Η δομή αυτή παρακολουθεί τη μεταγλώττιση και μεταβάλλεται δυναμικά, με την προσθήκη ή αφαίρεση πληροφορίας σε και από αυτήν, ώστε σε κάθε σημείο της διαδικασίας της μεταγλώττισης να περιέχει ακριβώς την πληροφορία που εκείνη τη στιγμή πρέπει να έχει. Λέγοντας πρέπει να έχει, εννοούμε τις εγγραφές εκείνες και μόνο αυτές, που σύμφωνα με τους κανόνες εμβέλειας της γλώσσας, το υπό μεταγλώττιση πρόγραμμα έχει δικαίωμα να έχει πρόσβαση τη συγκεκριμένη στιγμή της μεταγλώττισης. Πέρα από την αρωγή στην παραγωγή του τελικού κώδικα, με τον πίνακα συμβόλων είναι συνυφασμένη η σημασιολογική ανάλυση. Ο πίνακας συμβόλων παρέχει την πληροφορία που απαιτείται για τη σημασιολογική ανάλυση, ενώ μέρος της σημασιολογικής ανάλυσης υλοποιείται μέσα στον πίνακα συμβόλων. Σε σχέση με τις φάσεις ανάπτυξης ενός μεταγλωττιστή, τις οποίες περιγράψαμε στο εισαγωγικό κεφάλαιο, ο πίνακας συμβόλων αντλεί πληροφορία από τις φάσεις της συντακτικής ανάλυσης και της παραγωγής ενδιάμεσου κώδικα και διαθέτει την πληροφορία που έχει συλλέξει για την παραγωγή τελικού κώδικα.

Υλοποίηση του πίνακα συμβόλων

```
#pinakas_symbolwn -----
list_of_scopes = []
class Scope():
    def __init__(self):
        self.name = '' #we initialize our class Scope with name,list of entities and nestinglevel
        self.entityList = []
        self.nestingLevel = 0

class Entity():
    def __init__(self):
        self.name = '' #we initialize our class entity with the name and datatype(variable,function_parameter,temporaryvariable)
        self.datatype = ''
        self.variable = self.Variable() #we create 4 objects from the classes who inherit the name and datatype from entity
        self.function = self.Function()
        self.parameter = self.Parameter()
        self.temporaryVariable = self.TemporaryVariable()

    class Variable:
        def __init__(self):
            self.offset = 0 # the distance from the start of the stack

class Function:
    def __init__(self):
        self.startingQuad = 0 # first quad since the function begins
        self.frameLength = 0 #length activation record
        self.argumentList = [] # the arguments of the function
        self.nestingLevel = 0

class Parameter:
    def __init__(self):
        self.offset = 0 # distance from the start of the stack

class TemporaryVariable:
    def __init__(self):
        self.offset = 0 # distance from the start of the stack

class Argument():
    def __init__(self):
        self.name = '' #we initialize the name of the argument
        self.datatype = 'INT'#the datatype of the arguments is int

def add_entity(entity): # we add a new entity at the entityList of the last scope in the list_of_scopes
    global list_of_scopes
    list_of_scopes[-1].entityList = list_of_scopes[-1].entityList + [entity]

count = 0 #global variable that we increment when we add a new scope in the list_of_scopes
def nestingLevel_assign(scope): #we assign the global var count at the nestinglevel of a scope
    global count
    scope.nestingLevel = count

def add_argument(argument): # we add a new argument at the argumentlist of the last entity
    global list_of_scopes
    list_of_scopes[-1].entityList[-1].function.argumentList = list_of_scopes[-1].entityList[-1].function.argumentList + [argument]

def add_scope(name): # we add a new scope at the list_of_scopes
    global list_of_scopes
    global count

    nextScope = Scope()
    nextScope.name = name
    if len(list_of_scopes) >= 1:
        count += 1
        nestingLevel_assign(nextScope)

    list_of_scopes.append(nextScope)
```

```

def delete_scope():
    global list_of_scopes
    global count
    count = 0
    for i in range(len(list_of_scopes[-1].entityList)): #we delete from the list each entity of the last scope in the list_of_scopes
        list_of_scopes[-1].entityList.pop()
    del list_of_scopes[-1] #we delete the last scope

def calculate_offset():
    global list_of_scopes
    offset = 12 #the first entity is 12 bytes

    if len(list_of_scopes[-1].entityList) >= 1: #for each entity except the entity (Function) we increment the offset by 4 bytes
        for i in range(len(list_of_scopes[-1].entityList)):
            if list_of_scopes[-1].entityList[i].datatype == 'VARIABLE' or list_of_scopes[-1].entityList[i].datatype == 'TEMPORARYVARIABLE' \
                or list_of_scopes[-1].entityList[i].datatype == 'PARAMETER':
                offset += 4
    return offset

def calculate_framelength(): #we calculate the framelength of the last entity of the scope that is before the last scope
    global list_of_scopes
    list_of_scopes[-2].entityList[-1].function.frameLength = calculate_offset()

def calculate_startingQuad(): #we find the first quad of the last entity of the scope that is before the last scope
    global list_of_scopes
    list_of_scopes[-2].entityList[-1].function.startingQuad = nextQuad()

def find_entity(x): #we search for the entity by the name
    global list_of_scopes
    if len(list_of_scopes) > 0:
        for j in range(len(list_of_scopes)-1,-1,-1):
            for i in list_of_scopes[j].entityList:
                if i.name == x:
                    return(list_of_scopes[j],i)
    print("There is not entity with name : " + str(x))
    exit()

def entity_assign(en,name1,type1,offset1): #we assign the name,type and offset at the object entity that we give
    en.name = name1
    en.datatype = type1
    if en.datatype == "PARAMETER":
        en.parameter.offset = offset1
    elif en.datatype == "VARIABLE":
        en.variable.offset = offset1
    elif en.datatype == "TEMPORARYVARIABLE":
        en.temporaryVariable.offset = offset1

def add_parameters(): # we add a new entity for each argument in the scope before the last scope
    global list_of_scopes

    for i in list_of_scopes[-2].entityList[-1].function.argumentList:
        entity = Entity()
        entity_assign(entity,i.name,"PARAMETER",calculate_offset())
        add_entity(entity)

def show_symbols(file): #we print our symbol table at the file
    global list_of_scopes
    file.write("-----")
    file.write("\n")
    for i in list_of_scopes:
        file.write("\n")
        file.write("Scope: " + str(i.name) + " " + "nestingLevel: " + str(i.nestingLevel))
        for j in i.entityList:
            file.write("\n")
            file.write("Entity: " + str(j.name) + " " + str(j.datatype))
            file.write("\n")
            if j.datatype == "VARIABLE":
                file.write("Offset is :" + str(j.variable.offset))
                file.write("\n")
            elif j.datatype == "PARAMETER":
                file.write("Offset is :" + str(j.parameter.offset))
                file.write("\n")
            elif j.datatype == "TEMPORARYVARIABLE":
                file.write("Offset is :" + str(j.temporaryVariable.offset))
                file.write("\n")

```

```

        elif j.datatype == "FUNCTION":
            file.write("      FrameLength of " + str(j.name) + " is : " + str(j.function.frameLength))
            file.write("\n")
            file.write("      StartingQuad of " + str(j.name) + " is : " + str(j.function.startingQuad))
            file.write("\n")
            for k in j.function.argumentList:
                file.write("          Argument: " + str(k.name) + " " + str(k.datatype))
                file.write("\n")

```

Ο πίνακας συμβόλων μας εμφανίζεται στο αρχείο symbol_table.txt. Με τις παραπάνω κλάσεις και συναρτήσεις τις οποίες καλούμε μέσα στο συντακτικό αναλυτή δημιουργείται ο πίνακας μας.

Έλεγχος εγκυρότητας πίνακα συμβόλων

Χρησιμοποιήσαμε τα ίδια αρχεία ελέγχου που παραδώσαμε και στον ενδιάμεσο πίνακα .Πιο συγκεκριμένα

```

def main_compute():
    #{

        #declare x,z
        def compute():
            #{

                #declare k,m
                def iscube(num):
                    #{

                        #declare i,cub
                        i = 0;
                        cub = 0;
                        while (cub < num):
                            #{

                                i = i + 3;
                                cub = i * i * i;
                            }

                            if (cub == num):
                                return(1); #$if the num is cube returns 1 ,otherwise returns0 $#
                            else:
                                return(0);
                    }

                    #$ main body of compute $#
                    k = int(input());
                    while( k <= 0): #$we want numbers positive $#
                        k = int(input());
                        print(k);

                    m = iscube(k);
                    print(m);
                    return(m);
            }

            def issquare(d):
                #{

                    #declare p,sq
                    p = 0;
                    sq = 0;
                    while (sq < d):
                        #{

                            p = p + 1;
                            sq = p * p;
                        }

                        if (sq == d):
                            return (1); #$if the num is square returns 1 ,otherwise returns 0 $#
                        else:
                            return (0);
                }

                #$ main body of main_compute $#
                z = compute();
                x = 4;
                print(issquare(x));
                ex = 5*x // 2 + 1;
                print(ex);
            }
        }

        if __name__ == "__main__":
            main_compute();
    }

```

Ο πινακας συμβολων μας εμφανιζεται ως εξης:

```
|-----  
Scope: main nestingLevel: 0  
  Entity: main_compute FUNCTION  
  Framelength of main_compute is : 0  
  StartingQuad of main_compute is : 0  
  
Scope: main_compute nestingLevel: 1  
  Entity: x VARIABLE  
  Offset is :12  
  
    Entity: z VARIABLE  
    Offset is :16  
  
    Entity: compute FUNCTION  
    Framelength of compute is : 0  
    StartingQuad of compute is : 0  
  
Scope: compute nestingLevel: 2  
  Entity: k VARIABLE  
  Offset is :12  
  
  Entity: m VARIABLE  
  Offset is :16  
  
  Entity: iscube FUNCTION  
  Framelength of iscube is : 36  
  StartingQuad of iscube is : 2  
    Argument: num INT  
  
Scope: iscube nestingLevel: 3  
  Entity: num PARAMETER  
  Offset is :12  
  
  Entity: i VARIABLE  
  Offset is :16  
  
  Entity: cub VARIABLE  
  Offset is :20  
  
Entity: %0 TEMPORARYVARIABLE  
Offset is :24  
  
Entity: %1 TEMPORARYVARIABLE  
Offset is :28  
  
Entity: %2 TEMPORARYVARIABLE  
Offset is :32  
-----  
  
Scope: main nestingLevel: 0  
  Entity: main_compute FUNCTION  
  Framelength of main_compute is : 0  
  StartingQuad of main_compute is : 0  
  
Scope: main_compute nestingLevel: 1  
  Entity: x VARIABLE  
  Offset is :12  
  
  Entity: z VARIABLE  
  Offset is :16  
  
  Entity: compute FUNCTION  
  Framelength of compute is : 24  
  StartingQuad of compute is : 19  
  
Scope: compute nestingLevel: 2  
  Entity: k VARIABLE  
  Offset is :12
```

```
Entity: m VARIABLE
Offset is :16

Entity: iscube FUNCTION
Framelength of iscube is : 36
StartingQuad of iscube is : 2
    Argument: num INT

Entity: %3 TEMPORARYVARIABLE
Offset is :20
-----

Scope: main nestingLevel: 0
    Entity: main_compute FUNCTION
    Framelength of main_compute is : 0
    StartingQuad of main_compute is : 0

Scope: main_compute nestingLevel: 1
    Entity: x VARIABLE
    Offset is :12

    Entity: z VARIABLE
    Offset is :16

    Entity: compute FUNCTION
    Framelength of compute is : 24
    StartingQuad of compute is : 19

    Entity: issquare FUNCTION
    Framelength of issquare is : 32
    StartingQuad of issquare is : 33
        Argument: d INT

Scope: issquare nestingLevel: 1
    Entity: d PARAMETER
    Offset is :12

Entity: p VARIABLE
Offset is :16

Entity: sq VARIABLE
Offset is :20

Entity: %4 TEMPORARYVARIABLE
Offset is :24

Entity: %5 TEMPORARYVARIABLE
Offset is :28
-----

Scope: main nestingLevel: 0
    Entity: main_compute FUNCTION
    Framelength of main_compute is : 40
    StartingQuad of main_compute is : 49

Scope: main_compute nestingLevel: 1
    Entity: x VARIABLE
    Offset is :12

    Entity: z VARIABLE
    Offset is :16

    Entity: compute FUNCTION
    Framelength of compute is : 24
    StartingQuad of compute is : 19

    Entity: issquare FUNCTION
    Framelength of issquare is : 32
    StartingQuad of issquare is : 33
        Argument: d INT
```

```
Entity: %6 TEMPORARYVARIABLE  
Offset is :20  
  
Entity: %7 TEMPORARYVARIABLE  
Offset is :24  
  
Entity: %8 TEMPORARYVARIABLE  
Offset is :28  
  
Entity: %9 TEMPORARYVARIABLE  
Offset is :32  
  
Entity: %10 TEMPORARYVARIABLE  
Offset is :36
```

```
-----  
  
Scope: main nestingLevel: 0  
Entity: main_compute FUNCTION  
Framelength of main_compute is : 40  
StartingQuad of main_compute is : 49
```

Πρώτα εμφανίζεται ο πίνακας με τα scope της main, της main_compute, της compute και της iscube. Η iscube είναι πάνω πάνω στον πίνακα συμβόλων μας όπως φαίνεται και από το nestingLevel. Οπως βλέπουμε και από το τεστ πρόγραμμα μας η iscube έχει μια παράμετρο ,2 δικές της μεταβλητές και 3 προσωρινές μεταβλητές που προκύπτουν από τις τετράδες της και αυτό που παρατηρούμε είναι ότι το offset σε κάθε μια από αυτές αυξάνεται κατά 4 ενώ της πρώτης είναι 12. Αντιστοιχα και η συνάρτηση που κάλεσε την iscube η compute έχει σαν entities τις δικές της μεταβλητές και τη συνάρτηση iscube, της οποίας τώρα το framelength έχει τιμή 36 και η startingquad είναι η 2 κάτι που είναι σωστό όπως φαίνεται και στον ενδιάμεσο . Επιπλέον έχει και το argument που βλέπουμε στον κώδικα το num.

Στη συνέχεια έχουμε την main_compute η οποία έχει μικρότερο nestingLevel και έχει σαν entities τις δικές της μεταβλητές και τη συνάρτηση compute όπως και η main με entity την main_compute.

Έπειτα σβήνεται η iscube και έχουμε πλέον μόνο την main, main_compute και την compute. Η main_compute θα έχει σαν entity την compute με framelength 24 και startingquad 19.

Όταν σβήστεί και η compute τότε θα έχουμε τη main, main_compute και issquare. Η issquare θα έχει την παράμετρο d, τις μεταβλητές r και s_q και 2 προσωρινές μεταβλητές σαν entities που αυξάνονται κατά 4 η κάθε μια. Και η main_compute θα έχει τις μεταβλητές την compute και την issquare σαν entity με framelenght 32 και startingquad την 33 δηλαδή εκεί που ξεκινάει συνάρτηση.

Ακόμα όταν σβήστεί η issquare θα μείνουν πλέον η main με την main_compute . Και τέλος όταν σβήστεί και η main_compute θα παραμείνει μόνο η main με entity την main_compute και θα έχει framelength 40 και startingquad 49.

Η επιλογή του συγκεκριμένου προγράμματος έγινε διότι έχουμε πολλές συναρτήσεις και ελέγχουμε κάθε φορά πως δημιουργείται ο πίνακας συμβόλων όταν τελειώνει η κάθε μια καθώς και τις τιμές των διαφόρων προσωρινών μεταβλητών και παραμέτρων.

Τελικός κώδικας

Η τελευταία φάση της παραγωγής κώδικα είναι η παραγωγή του τελικού κώδικα. Ο τελικός κώδικας προκύπτει από τον ενδιάμεσο κώδικα με τη βοήθεια του πίνακα συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων.

Με την υλοποίηση του τελικού κώδικα μάλιστα μπορούμε να βρούμε και σημασιολογικά λάθη δηλαδή αν κάποιος παραλείψει να δηλώσει μια μεταβλητή και δεν υπάρχει στον πίνακα συμβολών. Για παράδειγμα

```
def main_compute():
#{
    #declare x,z
    def compute(x):
    #{
        #declare k,m,f
        def iscube(num):
        #{
            #declare i
            i = 0;
            cub = 0;
            while (cub < num):
                #{
                    i = i + 3;
                    f = 7;
                    x = x + 1;

                    cub = i * i * i;
                }
            if (cub == num):
                return(1); #If the num is cube returns 1 ,otherwise returns0 #§
            else:
                return(0);
        #1
    #1
}
```

Τότε θα βγει το εξής μήνυμα λάθους:

```
C:\python\ex6>python cutePy_4168_4163.py file1.cpy
There is not entity with name : cub
```

Υλοποίηση τελικού κώδικα

```
#####
telikos_kwdkas #####
def gnlvcode(y):
    global list_of_scopes
    (my_scope, my_entity) = find_entity(y)
    n = list_of_scopes[-1].nestingLevel - my_scope.nestingLevel - 1
    final_file.write('lw t0,-4($p)\n')
    while n > 0:
        final_file.write('lw t0,-4($t0)\n')
        n = n - 1
    if my_entity.datatype == 'VARIABLE':
        my_offset = my_entity.variable.offset
        final_file.write('addi t0,t0,%d\n' % my_offset)
    elif my_entity.datatype == 'PARAMETER':
        my_offset = my_entity.parameter.offset
        final_file.write('addi t0,t0,%d\n' % my_offset)

def loadvr(v,r):
    if v.isdigit():
        final_file.write('li t$,%s\n' % (r,v))
    else:
        (my_scope,my_entity) = find_entity(v)
        if my_scope.nestingLevel == 0:
            if my_entity.datatype == "VARIABLE":
                final_file.write('lw t$,-%d($gp)\n' % (r, my_entity.variable.offset))
```

```

        elif my_entity.datatype == "TEMPORARYVARIABLE":
            final_file.write('lw t%,-%d(gp)\n' % (r, my_entity.temporaryVariable.offset))

    elif my_scope.nestingLevel == list_of_scopes[-1].nestingLevel:
        if my_entity.datatype == "VARIABLE":
            final_file.write('lw t%,-%d(sp)\n' % (r, my_entity.variable.offset))
        elif my_entity.datatype == "TEMPORARYVARIABLE":
            final_file.write('lw t%,-%d(sp)\n' % (r, my_entity.temporaryVariable.offset))
        elif my_entity.datatype == "PARAMETER":
            final_file.write('lw t%,-%d(sp)\n' % (r, my_entity.parameter.offset))

    elif my_scope.nestingLevel < list_of_scopes[-1].nestingLevel:
        if my_entity.datatype == "VARIABLE":
            gnlvcode(v)
            final_file.write('lw t%,(t0)\n' % r)
        elif my_entity.datatype == "PARAMETER":
            gnlvcode(v)
            final_file.write('lw t%,(t0)\n' % r)

def storerv(r,v):
    (my_scope, my_entity) = find_entity(v)
    if my_scope.nestingLevel == 0:
        if my_entity.datatype == "VARIABLE":
            final_file.write('sw t%d,-%d(gp)\n' % (r,my_entity.variable.offset))
        elif my_entity.datatype == "TEMPORARYVARIABLE":
            final_file.write('sw t%d,-%d(gp)\n' % (r, my_entity.temporaryVariable.offset))

```

```

    elif my_scope.nestingLevel == list_of_scopes[-1].nestingLevel:
        if my_entity.datatype == "VARIABLE":
            final_file.write('sw t%d,-%d(sp)\n' % (r, my_entity.variable.offset))
        elif my_entity.datatype == "TEMPORARYVARIABLE":
            final_file.write('sw t%d,-%d(sp)\n' % (r, my_entity.temporaryVariable.offset))
        elif my_entity.datatype == "PARAMETER":
            final_file.write('sw t%d,-%d(sp)\n' % (r, my_entity.parameter.offset))

    elif my_scope.nestingLevel < list_of_scopes[-1].nestingLevel:
        if my_entity.datatype == "VARIABLE":
            gnlvcode(v)
            final_file.write('sw t%d,(t0)\n' % r)
        elif my_entity.datatype == "PARAMETER":
            gnlvcode(v)
            final_file.write('sw t%d,(t0)\n' % r)

    final_file.write('\n')
    my_count = 0

def generate_final():
    global fullist
    global fullist2
    global my_count
    relop_operators = ['==', '!=', '<', '<=', '>', '>=']
    assembly_operators = ['beq', 'bne', 'blt', 'ble', 'bgt', 'bge']
    numerical_op = ['+', '-', '*', '//']

```

```

assembly_op = ['add', 'sub', 'mul', 'div']
final_flag = 0
for i in fullist2:
    final_file.write('L' + str(i[0]) + ': \n')
    if i[1] == "jump":
        final_file.write('j L'+str(i[4])+'\n')
    elif i[1] in relop_operators:
        x = assembly_operators[relop_operators.index(i[1])]
        loadvr(i[2], 1)
        loadvr(i[3], 2)
        final_file.write(x + ' ,t1, t2, L' + i[4] + '\n')
    elif i[1] in numerical_op:
        x = assembly_op[numerical_op.index(i[1])]
        loadvr(i[2], 1)
        loadvr(i[3], 2)
        final_file.write(x + ' ,t1, t1,t2 \n')
        storerv(1, i[4])
    elif i[1] == '=':
        loadvr(i[2], 1)
        storerv(1, i[4])
    elif i[1] == 'retv':
        loadvr(i[2], '1')
        final_file.write('lw t0, -8(sp)\n')
        final_file.write('sw t1, 0(t0)\n')
        final_file.write('lw ra, 0(sp)\n')
        final_file.write('jr ra\n')

    elif i[1] == 'inp':
        final_file.write('li a7,5' + '\n')
        final_file.write('ecall' + '\n')
        final_file.write('mv t1,a0' + '\n')
        storerv(1, i[2])
    elif i[1] == 'out':
        loadvr(i[2], 1)
        final_file.write('mv a0,t1' + '\n')
        final_file.write('li a7,1' + '\n')
        final_file.write('ecall' + '\n')
    elif i[1] == 'halt':
        final_file.write('li a0,0\n')
        final_file.write('li a7,93\n')
        final_file.write('ecall\n')
    elif i[1] == 'begin_block' and list_of_scopes[-1].nestingLevel != 0:
        final_file.write('sw ra,(sp)\n')
    elif i[1] == 'end_block' and list_of_scopes[-1].nestingLevel != 0:
        final_file.write('lw ra,(sp)\n')
        final_file.write('jr ra\n')
    elif i[1] == 'begin_block' and list_of_scopes[-1].nestingLevel == 0:
        final_file.seek(0,0)
        final_file.write('j L%d\n' % i[0])
        final_file.seek(0,2)
        final_file.write('addi sp,sp,%d\n' % calculate_offset())
        final_file.write('mv gp,sp\n')

    elif i[1] == 'par':
        j = fullist.index(i)
        for k in fullist[j:]:
            if k[1] == 'call' and final_flag == 0:
                (my_scope,my_entity) = find_entity(k[2])
                final_file.write('addi fp,sp,%d\n' % my_entity.function.frameLength)
                final_flag = 1
                break
            if i[3] == 'CV':
                loadvr(i[2],0)
                final_file.write('sw t0, -%d(fp)\n' % (12 + 4 * my_count))
                my_count = my_count + 1
            elif i[3] == 'RET':
                (my_scope, my_entity) = find_entity(i[2])
                final_file.write('addi t0,sp,-%d\n' % my_entity.temporaryVariable.offset)
                final_file.write('sw t0, -8(fp)\n')
            elif i[1] == 'call':
                final_flag = 0
                my_count = 0
                (my_scope,my_entity) = find_entity(i[2])
                if list_of_scopes[-1].nestingLevel < my_entity.function.nestingLevel:
                    final_file.write('sw sp,-4(fp)\n')
                elif list_of_scopes[-1].nestingLevel == my_entity.function.nestingLevel:
                    final_file.write('lw t0,-4(sp)\n')
                    final_file.write('sw t0,-4(fp)\n')
                final_file.write('addi sp,sp,%d\n' % my_entity.function.frameLength)
                final_file.write('jal L%d\n' % my_entity.function.startingQuad)
                final_file.write('addi sp,sp,-%d\n' % my_entity.function.frameLength)

fullist2 = []

```

Στον τελικό κώδικα χρησιμοποιήσαμε τις βιοηθητικές συναρτήσεις `loadvr`,`storevr`,`gnlvcod` καθώς και την `generate_final`. Στον συντακτικό μας καλούμε την `generate_final` προκείμενου να παραχθεί η συμβολική γλώσσα μηχανής (assembly code) του επεξεργαστή RISC-V.

Ελεγχος εγκυρότητας τελικού κώδικα

Για να ελέγχουμε ότι λειτουργεί ο τελικός μας κώδικας χρησιμοποιήσαμε 2 τεστ αρχεία στη τελευταία φάση της εργασίας. Το πρώτο αρχείο είναι το παρακάτω:

```
def main_compute():
    #{
        #declare x,z
        def compute(x):
            #{
                #declare k,m,f
                def iscube(num):
                    #{
                        #declare i,cub
                        i = 0;
                        cub = 0;
                        while (cub < num):
                            #{
                                i = i + 3;
                                f = 7;
                                x = x + 1;
                                cub = i * i * i;
                            }
                            if (cub == num):
                                return(1); #$if the num is cube returns 1 ,otherwise returns0 $#
                            else:
                                return(0);
                #}

                #$ main body of compute #$ 
                k = int(input());
                f = 2;
                m = iscube(k);
                print(f);
                print(m);
                print(x);
                return(m);
            #}

            def issquare(d):
                #{
                    #declare p,sq
                    p = 0;
                    sq = 0;
                    while (sq < d):
                        #{
                            p = p + 1;
                            sq = p * p;
                        }
                        if (sq == d):
                            return (1); #$if the num is square returns 1 ,otherwise returns 0 $#
                        else:
                            return (0);
                #}
                #$ main body of main_compute #$ 
                x = 4;
                z = compute(x);
                print(issquare(x));
            #}

            if __name__ == "__main__":
                main_compute();
    #}
```

Και ο τελικός μας κώδικας που δημιουργείται σε ένα αρχείο `final_file.asm` είναι ο παρακάτω:

```
1 j L60
2
3 L1:
4 sw ra,(sp)
5 L2:
6 li t1,0
7 sw t1,-16(sp)
8 L3:
9 li t1,0
10 sw t1,-20(sp)
11 L4:
12 lw t1,-20(sp)
13 lw t2,-12(sp)
14 blt ,t1, t2, L6
15 L5:
16 j L15
17 L6:
18 lw t1,-16(sp)
19 li t2,3
20 add ,t1, t1,t2
21 sw t1,-24(sp)
22 L7:
23 lw t1,-24(sp)
24 sw t1,-16(sp)
25 L8:
26 li t1,7
27 lw t0,-4(sp)
28 addi t0,t0,-24
29 sw t1,(t0)
30 L9:
31 lw t0,-4(sp)
32 addi t0,t0,-12
33 lw t1,(t0)
34 li t2,1
35 add ,t1, t1,t2
36 sw t1,-28(sp)
37 L10:
38 lw t1,-28(sp)
```

```
39 lw t0,-4(sp)
40 addi t0,t0,-12
41 sw t1,(t0)
42 L11:
43 lw t1,-16(sp)
44 lw t2,-16(sp)
45 mul ,t1, t1,t2
46 sw t1,-32(sp)
47 L12:
48 lw t1,-32(sp)
49 lw t2,-16(sp)
50 mul ,t1, t1,t2
51 sw t1,-36(sp)
52 L13:
53 lw t1,-36(sp)
54 sw t1,-20(sp)
55 L14:
56 j L4
57 L15:
58 lw t1,-20(sp)
59 lw t2,-12(sp)
60 beq t1, t2, L17
61 L16:
62 j L19
63 L17:
64 li t1,1
65 lw t0, -8(sp)
66 sw t1, 0(t0)
67 lw ra, 0(sp)
68 jr ra
69 L18:
70 j L20
71 L19:
72 li t1,0
73 lw t0, -8(sp)
74 sw t1, 0(t0)
75 lw ra, 0(sp)
76 jr ra
```

```
77 L20:
78 lw ra,(sp)
79 jr ra
80 L21:
81 sw ra,(sp)
82 L22:
83 li a7,5
84 ecall
85 mv t1,a0
86 sw t1,-16(sp)
87 L23:
88 li t1,2
89 sw t1,-24(sp)
90 L24:
91 addi fp,sp,40
92 lw t0,-16(sp)
93 sw t0, -12(fp)
94 L25:
95 addi t0,sp,-28
96 sw t0, -8(fp)
97 L26:
98 sw sp,-4(fp)
99 addi sp,sp,40
100 jal L1
101 addi sp,sp, 40
102 L27:
103 lw t1,-28(sp)
104 sw t1,-20(sp)
105 L28:
106 lw t1,-24(sp)
107 mv a0,t1
108 li a7,1
109 ecall
110 L29:
111 lw t1,-20(sp)
112 mv a0,t1
113 li a7,1
114 ecall
```

```
115 L30:  
116 lw t1,-12(sp)  
117 mv a0,t1  
118 li a7,1  
119 ecall  
120 L31:  
121 lw t1,-20(sp)  
122 lw t0, -8(sp)  
123 sw t1, 0(t0)  
124 lw ra, 0(sp)  
125 jr ra  
126 L32:  
127 lw ra, (sp)  
128 jr ra  
129 L33:  
130 sw ra, (sp)  
131 L34:  
132 li t1,0  
133 sw t1,-16(sp)  
134 L35:  
135 li t1,0  
136 sw t1,-20(sp)  
137 L36:  
138 lw t1,-20(sp)  
139 lw t2,-12(sp)  
140 blt ,t1, t2, L38  
141 L37:  
142 j L43  
143 L38:  
144 lw t1,-16(sp)  
145 li t2,1  
146 add ,t1, t1,t2  
147 sw t1,-24(sp)  
148 L39:  
149 lw t1,-24(sp)  
150 sw t1,-16(sp)  
151 L40:  
152 lw t1,-16(sp)
```

```
153 lw t2,-16(sp)  
154 mul ,t1, t1,t2  
155 sw t1,-28(sp)  
156 L41:  
157 lw t1,-28(sp)  
158 sw t1,-20(sp)  
159 L42:  
160 j L36  
161 L43:  
162 lw t1,-20(sp)  
163 lw t2,-12(sp)  
164 beq ,t1, t2, L45  
165 L44:  
166 j L47  
167 L45:  
168 li t1,1  
169 lw t0, -8(sp)  
170 sw t1, 0(t0)  
171 lw ra, 0(sp)  
172 jr ra  
173 L46:  
174 j L48  
175 L47:  
176 li t1,0  
177 lw t0, -8(sp)  
178 sw t1, 0(t0)  
179 lw ra, 0(sp)  
180 jr ra  
181 L48:  
182 lw ra, (sp)  
183 jr ra  
184 L49:  
185 sw ra, (sp)  
186 L50:  
187 li t1,4  
188 sw t1,-12(sp)  
189 L51:  
190 addi fp,sp,32
```

```
191 lw t0,-12(sp)  
192 sw t0, -12(fp)  
193 L52:  
194 addi t0,sp,-20  
195 sw t0, -8(fp)  
196 L53:  
197 sw sp,-4(fp)  
198 addi sp,sp,32  
199 jal L21  
200 addi sp,sp,-32  
201 L54:  
202 lw t1,-20(sp)  
203 sw t1,-16(sp)  
204 L55:  
205 addi fp,sp,32  
206 lw t0,-12(sp)  
207 sw t0, -12(fp)  
208 L56:  
209 addi t0,sp,-24  
210 sw t0, -8(fp)  
211 L57:  
212 sw sp,-4(fp)  
213 addi sp,sp,32  
214 jal L33  
215 addi sp,sp,-32  
216 L58:  
217 lw t1,-24(sp)  
218 mv a0,t1  
219 li a7,1  
220 ecall  
221 L59:  
222 lw ra, (sp)  
223 jr ra  
224 L60:  
225 addi sp,sp,12  
226 mv gp,sp  
227 L61:  
228 sw sp,-4(fp)
```

```

229 addi $sp,$sp,28
230 jal L49
231 addi $sp,$sp,-28
232 L62:
233 li a0,0
234 li a7,93
235 ecall
236 L63:
237

```

Και ο αντίστοιχος ενδιάμεσος κώδικας για αυτό το αρχείο είναι ο :

```

1 1: begin_block iscube --
2 2: = 0 _ i
3 3: = 0 _ cub
4 4: < cub num 6
5 5: jump _ 15
6 6: + i 3 %0
7 7: = %0 _ i
8 8: = 7 _ f
9 9: + x 1 %1
10 10: = %1 _ x
11 11: * i i %2
12 12: * %2 i %3
13 13: = %3 _ cub
14 14: jump _ 4
15 15: == cub num 17
16 16: jump _ 19
17 17: retv 1 -
18 18: jump _ 20
19 19: retv 0 -
20 20: end_block iscube --
21 21: begin_block compute --
22 22: inp k -
23 23: = 2 _ f -
24 24: par k CV -
25 25: par %4 RET -
26 26: call iscube --
27 27: = %4 _ m
28 28: out f --
29 29: out m --
30 30: out x --
31 31: retv m -
32 32: end_block compute --
33 33: begin_block issquare --
34 34: = 0 _ p
35 35: = 0 _ sq
36 36: < sq d 38
37 37: jump _ 43
38 38: + p 1 %5

```

```

39 39: = %5 _ p
40 40: * p p %6
41 41: = %6 _ sq
42 42: jump _ 36
43 43: == sq d 45
44 44: jump _ 47
45 45: retv 1 -
46 46: jump _ 48
47 47: retv 0 --
48 48: end_block issquare --
49 49: begin_block main_compute --
50 50: = 4 _ x
51 51: par x CV
52 52: par %7 RET -
53 53: call compute --
54 54: = %7 _ z
55 55: par x CV
56 56: par %8 RET -
57 57: call issquare --
58 58: out %8 --
59 59: end_block main_compute --
60 60: begin_block main --
61 61: call main_compute --
62 62: halt
63 63: end_block main --
64

```

Η πρώτη τετράδα με το begin block αντιστοιχεί στην εντολή sw ra,(sp) καθώς είμαστε στη συνάρτηση του iscube .

Η δεύτερη τετράδα ακόμα αντιστοιχεί στην ετικέτα L2 και τις εντολές li t1,0 sw t1,-16(sp) καθώς αναθέτω μια τιμή σε μια μεταβλητή και αυτό το κάνω φορτώνοντας την τιμή σε έναν καταχωρητή και αποθηκεύοντας την στη μνήμη .,Αντίστοιχα η τετράδα με την ετικέτα 4 αντιστοιχεί στο L4 του αρχείου με το τελικό κώδικα καθώς συγκρίνω τις 2 μεταβλητές και αν

ισχύει πηγαίνω στο L6 οπού εκεί αυξάνω την μεταβλητή i ,την οποία έχω αποθηκεύσει σε ένα καταχωρητή ,με την αριθμητική σταθερά 3 οπού αντίστοιχα έχω φορτώσει σε έναν καταχωρητή. Επιπρόσθετα, η ετικέτα L5 κάνει jump στην ετικέτα L15 όπου εκεί ελέγχεται ο έλεγχος του if αν είναι ίσες οι μεταβλητές .Η εκχώρηση στο i γίνεται στην L7 με τις εντολές lw t1,-24(sp) sw t1,-16(sp) και η εκχώρηση στην μεταβλητή f όπου είναι δηλωμένη στη συνάρτηση που καλεί την iscube γίνεται με τις εντολές

li t1,7

lw t0,-4(sp)

addi t0,t0,-24

sw t1,(t0) καθώς καλείται η gnlvcode και πρέπει να βρει τη διεύθυνση της μεταβλητής που βρίσκεται στη compute.

Επιπλέον, η τετράδα retv 1 __ αντιστοιχεί στο L17 όπου φορτώνεται το 1 στην τρίτη θέση του εγγραφήματος δραστηριοποίησης .

Αντίστοιχα οι τετράδες των συναρτήσεων εισόδου για παράδειγμα η iop k __ γίνεται με τις εντολές li a7,5 ecall και πρέπει να γίνει ένα move από τον καταχωρητή a0 που αποθηκεύεται στο t1 ώστε μετά να αποθηκεύσουμε την τιμή στη μνήμη

Όπως και στις τετράδες των συναρτήσεων εξόδου για παράδειγμα out m __ γίνεται με τις εντολές li a7,1 ecall αφού πρώτα φορτώσουμε την τιμή της μεταβλητής m και την μεταφέρουμε στον καταχωρητή a0.

Ακόμα η τετράδα του halt γίνεται στις εντολές li a0,0 li a7,93

Οι παράμετροι για παράδειγμα η par k CV __ αντιστοιχεί στις εντολές

addi fp,sp,40

lw t0,-16(sp)

sw t0, -12(fp) καθώς πρώτα τοποθετούμε τον fp να δείχνει στη στοίβα της συνάρτησης που θα δημιουργηθεί και μετά αποθηκεύουμε την παράμετρο στη στοίβα.

Τέλος η τετράδα του call για παράδειγμα call iscube __ επειδή η iscube έχει μεγαλύτερο βαθμός φωλιάσματός από την compute που την καλεί θα έχουμε αυτές τις εντολές

sw sp,-4(fp)

addi sp,sp,40

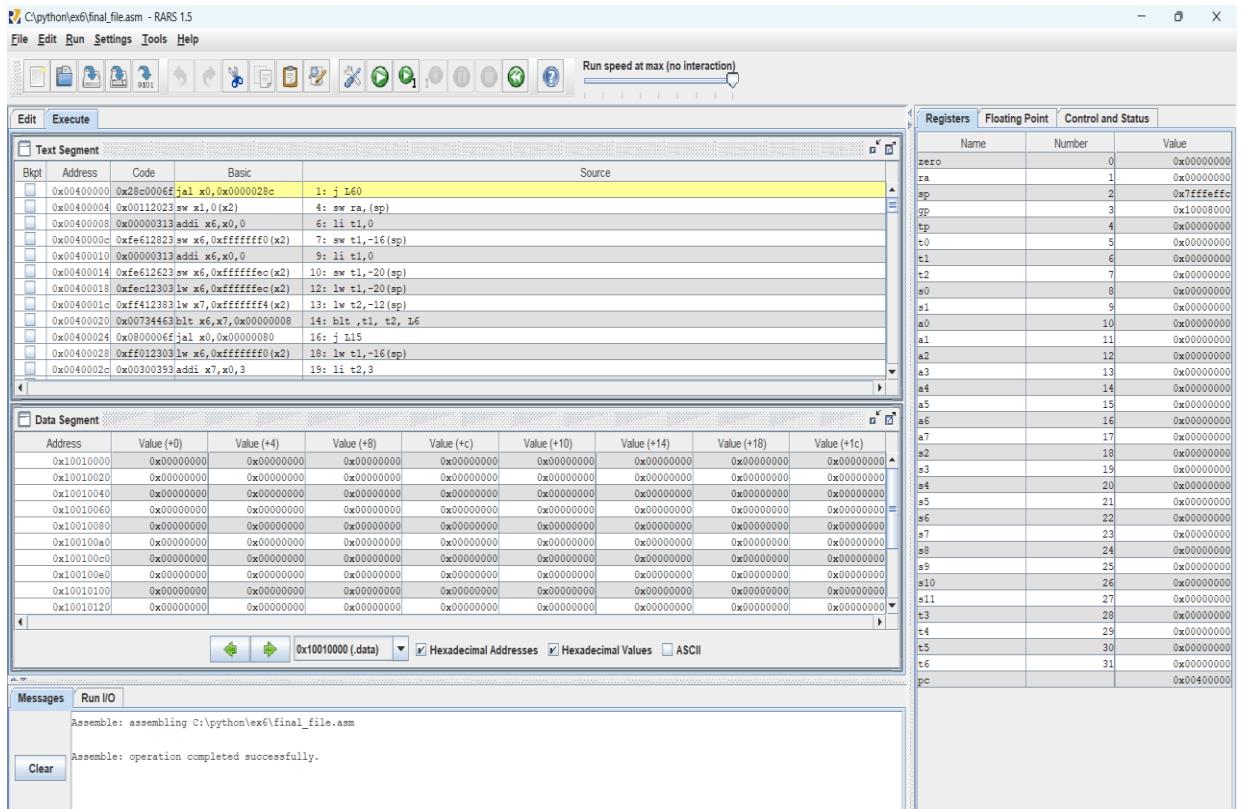
jal L1

addi sp,sp,-40

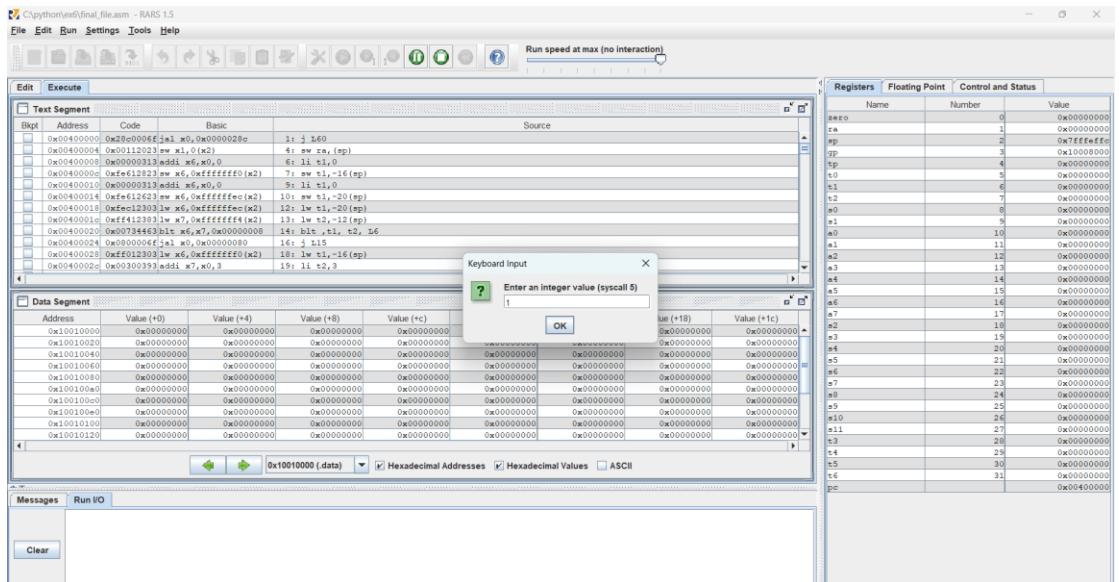
Ομοίως για κάθε τετράδα έχει παραχθεί ο αντίστοιχος τελικός κώδικας.

Όταν τον τρέχουμε τον κώδικα στο RISC :

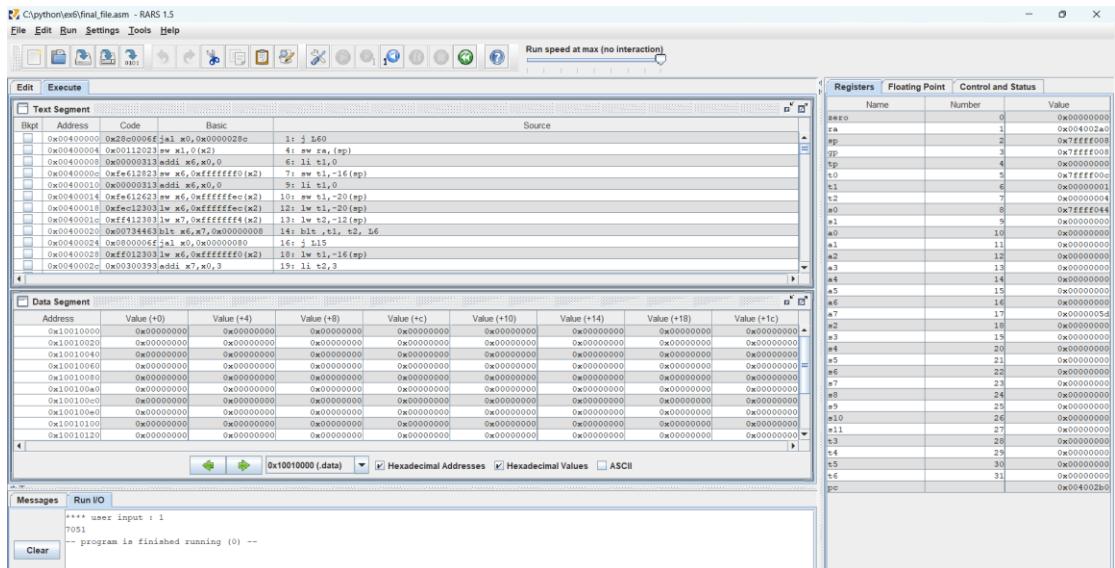
Πρώτα θα κάνουμε assemble



Στη συνέχεια όταν θα το τρέξουμε θα πρέπει να βάλουμε μια τιμή εισόδου καθώς στο πρόγραμμα μας έχουμε input. Στο παρακάτω βάζουμε την τιμή 1.



Και θα μας βγάλει το παρακάτω αποτέλεσμα:



Όπως παρατηρούμε μας τυπώνει για input = 1 τις τιμές 7,0,5 ,1 οπού αν παρατηρήσουμε στο πρόγραμμα .cργ θέλουμε να τυπώσουμε το f στο οποίο όταν καλείται η iscube εκχωρείται η τιμή 7, μετα εκτυπώνουμε το m που είναι η τιμή επιστροφής της συνάρτησης iscube(1) δηλαδή το 0. Επειτα τυπώνεται το x το οποίο έχει τιμή 4 και όταν καλείται η iscube γίνεται 5 και τέλος τυπώνεται το αποτέλεσμα της συνάρτησης issquare(4) που είναι 1.

Άρα ο μεταγλωττιστής μας λειτουργεί σωστά.

Επιλογή των προγραμμάτων για τον τελικό κώδικα

Τα 2 προγράμματα που ελέγχαμε αν λειτουργεί ο τελικός κώδικας περιλαμβάνουν όλες τις εντολές της γλώσσας cutePy. Αρχικά έχουν τις εντολές εξόδου print, τις εντολές εισόδου input, την εντολή return, το κάλεσμα συναρτήσεων που έχουν μεγαλύτερο βάθος φωλιάσματος αλλά και κάλεσμα συναρτήσεων που έχουν ίδιο βάθος φωλιάσματος όπως η Fibonacci παρακάτω στο δεύτερο τεστ αρχείο

```
def main_fibonacci():
#{
    #declare x
    def fibonacci(x):
    #{
        if (x <=1):
            return(x);
        else:
            return (fibonacci(x-1)+fibonacci(x-2));
    #}
    x = int(input());
    print(fibonacci(x));
#}
```

Ελέγχουμε τις αναθέσεις καθώς και τις παραμέτρους που έχουν οι συναρτήσεις μας. Ακόμα στα αρχεία μας καλείται και η gnlvncode τόσο για δήλωση μεταβλητής στον πατερά της συνάρτησης που της εκχωρεί τιμή αλλά και για παράμετρο μεταβλητής στον πατερά της συνάρτησης που της αυξάνει την τιμή. Τέλος στα αρχεία μας έχουμε αριθμητικές πράξεις αλλά και συγκρίσεις για να ελέγχουμε και εκεί τον τελικό κώδικα ότι λειτουργησε σωστά.