

Λεκτικός Αναλυτής

Σχήμα Αυτόματου

Για την υλοποίηση του παραπάνω αυτομάτου έχει δημιουργηθεί ο πίνακας `transition_table` στην συνάρτηση `lex()` η οποία υλοποιεί τον λεκτικό αναλυτή. Για τον λεκτικό αναλυτή έχει δημιουργηθεί η κλάση `Token` καθώς και οι βοηθητικές συναρτήσεις `check_keyword(current_token, state)` , `find_family(state)` , `peek(file, num_chars=1)` και η `find_state(char, transition_table)`.

Για κάθε λεκτική μονάδα ορίστηκε ένας μοναδικός αριθμός. Με βάση αυτόν καθορίζεται αν το αυτόματο βρίσκεται σε τελική κατάσταση. Αν βρεθεί σε μία από τις τελικές καταστάσεις θα επιστραφεί η αντίστοιχη λεκτική μονάδα στον συντακτικό αναλυτή.

Transition_table:

Στον πίνακα `transition_table` αποθηκεύονται οι κωδικοί των λεκτικών μονάδων (`token`) και οι διάφορες καταστάσεις. Οι στήλες του πίνακα αντιστοιχούν στις εισόδους που μπορεί να δεχτεί το αυτόματο, όπως φαίνεται παρακάτω:

```
#Inputs(transition_table columns):  "Space" ,  
"Alpha" , "Digit" , "+" , "-" , "*" , "/" , "%" ,  
"<" , ">" , "=" , "!" , "," , ":" , "(" , ")" ,  
"#" , "EOF" , "{" , "}"
```

Οι γραμμές αναφέρονται στις καταστάσεις του αυτόματου.

Προσπέλαση του πίνακα:

Ξεκινώντας από την αρχική κατάσταση 0 διαβάζεται ένας χαρακτήρας από το αρχείο εισόδου κάθε φορά, καλείται η συνάρτηση `find_state` η οποία είναι υπεύθυνη να μεταβάλει την τρέχουσα κατάσταση ανάλογα με τον χαρακτήρα που μόλις διαβάστηκε και την κατάσταση που βρίσκεται το αυτόματο. Οι θέσεις του πίνακα που περιέχουν τους κωδικούς των λεκτικών μονάδων αντιστοιχούν και στις τελικές καταστάσεις. Για τις περιπτώσεις των σφαλμάτων στον πίνακα `transition_table` υπάρχουν οι κωδικοί “ERROR”.

Συνάρτηση `lex()`:

Όπως αναφέρθηκε και προηγουμένως διαβάζεται ένας χαρακτήρας από το αρχείο εισόδου και αν δεν πρόκειται για λευκό χαρακτήρα (`white space`) τότε προστίθεται στο `current_token` (τρέχουσα λεκτική μονάδα) και αποφασίζεται η επόμενη κατάσταση. Αν η τιμή της κατάστασης είναι μικρότερη από 25 τότε δεν βρισκόμαστε σε κάποια από τις τελικές καταστάσεις και συνεχίζουμε με την ανάγνωση του αμέσως επόμενου χαρακτήρα. Η προσπέλαση του `transition_table` θα γίνει με βάση την τιμή του επόμενου χαρακτήρα που διαβάσαμε.

Peek:

Για την ανάγνωση του επόμενου χαρακτήρα χρησιμοποιήθηκε η συνάρτηση “peek” η οποία μετακινεί τον δείκτη του αρχείου (file pointer) στην επόμενη θέση και μόλις διαβάσει τον χαρακτήρα επιστρέφει στην προηγούμενη θέση που βρίσκονταν.

Τα condition-blocks που ακολουθούν υλοποιούν τον σχεδιασμό του αυτόματου που ορίστηκε παραπάνω. Σε αυτό το σημείο αξιοποιείται η έξοδος της συνάρτησης “peek” έτσι ώστε να υπολογιστεί η επόμενη κατάσταση. Στις περιπτώσεις που υπάρχουν εμφωλευμένα if-condition-blocks η μετάβαση στην επόμενη κατάσταση απαιτεί την γνώση όχι μόνο του επόμενου αλλά και του τρέχοντος χαρακτήρα. Αυτό συμβαίνει διότι οι συγκεκριμένες λεκτικές μονάδες αποτελούνται από περισσότερους από έναν χαρακτήρες.

Στο τέλος των condition-block ελέγχεται αν το αυτόματο βρίσκεται σε κάποια από τις τελικές καταστάσεις ώστε να αποφευχθεί η κατανάλωση ενός χαρακτήρα που διαφορετικά θα άνηκε στην επόμενη λεκτική μονάδα.

```
if state < 25:  
    char = inputFile.read(1)  
    current_token += char
```

Comment handling:

Αν η τρέχουσα λεκτική μονάδα είναι η “##” τότε το αυτόματο βρίσκεται σε κατάσταση ανάγνωση ενός σχολίου. Πιο συγκεκριμένα μέχρι την ανάγνωση της επόμενης λεκτικής μονάδας που θα είναι ίση με “##” οι ενδιάμεσοι χαρακτήρες αγνοούνται. Στην περίπτωση που συναντηθεί το τέλος του αρχείου (EOF) τότε τυπώνεται μήνυμα λάθους και γίνεται τερματισμός της εκτέλεσης του προγράμματος.

```
print(f"\033[3m\033[91mError: '##' delimiter found  
on line {newLine}.\nExpected '##' delimiter before  
end of file.\033[0m")  
exit(0)
```

Μετά την ολοκλήρωση του while-block ελέγχεται αν η λεκτική μονάδα που αναγνωρίστηκε ανήκει στα keywords μέσω της συνάρτησης “check_keyword”. Στην συνέχεια εξετάζεται αν η λεκτική μονάδα περιέχει περισσότερους από τριάντα χαρακτήρες και στην περίπτωση που ισχύει αυτό διατηρεί μόνο τους τριάντα πρώτους. Την ολοκλήρωση των προηγούμενων ελέγχων ακολουθεί ο καθορισμός της οικογένειας στην η οποία ανήκει η λεκτική μονάδα. Στην περίπτωση της “integer_family” ελέγχεται αν ο ακέραιος αριθμός βρίσκεται μεταξύ των επιθυμητών ορίων [-32767,32767]. Τελικά η συνάρτηση του λεκτικού αναλυτή “lex()” επιστρέφει ένα αντικείμενο της κλάσης Token που θα χρησιμοποιηθεί σε επόμενη φάση από τις συναρτήσεις του συντακτικού αναλυτή.

def lex()	Επιστρέφει τις λεκτικές μονάδες διαβάζοντας συνεχόμενους χαρακτήρες από το αρχείο εισόδου. Για τον σκοπό αυτό αξιοποιεί τον πίνακα καταστάσεων transition_table.
def check_keyword(current_token,state)	Ελέγχεται αν η λεκτική μονάδα ανήκει στα keywords και επιστρέφεται ο κωδικός της.
def find_family(state)	Καθορίζει την οικογένεια της τρέχουσας λεκτικής μονάδας.
def peek(file, num_char=1)	Διαβάζει τον επόμενο χαρακτήρα του αρχείου εισόδου.
def find_state(char,transition_table)	Βρίσκει την επόμενη κατάσταση βάση της τρέχουσας κατάστασης και του χαρακτήρα που διαβάστηκε.

Το παρακάτω στιγμιότυπο είναι από εκτέλεση για ένα αρχείο το οποίο περιέχει μόνο συνάρτηση για την main:

```
G:\My Drive\CSE_UOI\8thSemester\compilers>python compiler.py test.cpy
Family: keyword_family, String: main, Line Number: 6
Family: identifier_family, String: i, Line Number: 7
Family: assignment_family, String: =, Line Number: 8
Family: identifier_family, String: i, Line Number: 9
Family: keyword_family, String: int, Line Number: 9
Family: keyword_family, String: input, Line Number: 9
Family: group_family, String: ), Line Number: 9
Family: keyword_family, String: print, Line Number: 10
Family: identifier_family, String: i, Line Number: 10
Family: identifier_family, String: i, Line Number: 11
Family: integer_family, String: 1600, Line Number: 11
Family: identifier_family, String: i, Line Number: 12
Family: integer_family, String: 2000, Line Number: 12
Family: group_family, String: #{, Line Number: 13
Family: group_family, String: (, Line Number: 14
Family: group_family, String: (, Line Number: 14
Family: group_family, String: ), Line Number: 14
Family: identifier_family, String: i, Line Number: 15
Family: identifier_family, String: i, Line Number: 15
Family: integer_family, String: 400, Line Number: 15
Family: keyword_family, String: print, Line Number: 17
Family: identifier_family, String: leap, Line Number: 17
Family: integer_family, String: 2023, Line Number: 17
Family: group_family, String: ), Line Number: 17
Family: group_family, String: (, Line Number: 18
Family: group_family, String: (, Line Number: 18
Family: group_family, String: ), Line Number: 18
Family: keyword_family, String: print, Line Number: 19
Family: identifier_family, String: quad, Line Number: 19
Family: integer_family, String: 3, Line Number: 19
Family: group_family, String: ), Line Number: 19
Family: group_family, String: (, Line Number: 20
Family: group_family, String: (, Line Number: 20
Family: group_family, String: ), Line Number: 20
Family: identifier_family, String: i, Line Number: 21
Family: integer_family, String: 1, Line Number: 21
Family: identifier_family, String: i, Line Number: 22
Family: integer_family, String: 12, Line Number: 22
Family: group_family, String: #{, Line Number: 23
Family: group_family, String: (, Line Number: 24
Family: group_family, String: (, Line Number: 24
Family: group_family, String: ), Line Number: 24
Family: identifier_family, String: i, Line Number: 25
Family: identifier_family, String: i, Line Number: 25
Family: integer_family, String: 1, Line Number: 25
Family: keyword_family, String: print, Line Number: 27
Family: identifier_family, String: counterFunctionCalls, Line Number: 27
Family: EOF, String: , Line Number: 28
```

Το παρακάτω στιγμιότυπο είναι από εκτέλεση για ένα αρχείο το οποίο περιέχει μόνο συνάρτηση για την fib:

```
G:\My Drive\CSE_UOI\8thSemester\compilers>python compiler.py test.cpy
Family: identifier_family, String: fib, Line Number: 8
Family: identifier_family, String: x, Line Number: 8
Family: delimiter_family, String: :, Line Number: 8
Family: keyword_family, String: global, Line Number: 10
Family: identifier_family, String: counterFunctionCalls, Line Number: 11
Family: identifier_family, String: counterFunctionCalls, Line Number: 11
Family: integer_family, String: 1, Line Number: 11
Family: identifier_family, String: x, Line Number: 12
Family: integer_family, String: 0, Line Number: 12
Family: keyword_family, String: return, Line Number: 13
Family: integer_family, String: 1, Line Number: 13
Family: identifier_family, String: x, Line Number: 14
Family: integer_family, String: 0, Line Number: 14
Family: identifier_family, String: x, Line Number: 14
Family: integer_family, String: 1, Line Number: 14
Family: keyword_family, String: return, Line Number: 15
Family: keyword_family, String: else, Line Number: 16
Family: keyword_family, String: return, Line Number: 17
Family: group_family, String: (, Line Number: 17
Family: addOper_family, String: -, Line Number: 17
Family: group_family, String: ), Line Number: 17
Family: identifier_family, String: fib, Line Number: 17
Family: identifier_family, String: x, Line Number: 17
Family: integer_family, String: 2, Line Number: 17
Family: group_family, String: #}, Line Number: 18

G:\My Drive\CSE_UOI\8thSemester\compilers>
```

```
PS G:\To Drive μου\UOI\Μεταφραστές> python .\compiler.py .\inputTest5.cpy
Family: identifier_family, String: fib, Line Number: 1
Family: identifier_family, String: x, Line Number: 1
Family: delimiter_family, String: :, Line Number: 1
Family: keyword_family, String: global, Line Number: 3
Family: delimiter_family, String: ,, Line Number: 3
Family: identifier_family, String: counterFunctionCalls, Line Number: 4
Family: identifier_family, String: counterFunctionCalls, Line Number: 4
Family: integer_family, String: 1, Line Number: 4
Family: identifier_family, String: x, Line Number: 5
Family: integer_family, String: 0, Line Number: 5
Family: keyword_family, String: return, Line Number: 6
Family: integer_family, String: 1, Line Number: 6
Family: identifier_family, String: x, Line Number: 7
Family: integer_family, String: 0, Line Number: 7
Family: identifier_family, String: x, Line Number: 7
Family: integer_family, String: 1, Line Number: 7
Family: keyword_family, String: return, Line Number: 8
Family: keyword_family, String: else, Line Number: 9
Family: keyword_family, String: return, Line Number: 10
Family: group_family, String: (, Line Number: 10
Family: addOper_family, String: -, Line Number: 10
Family: group_family, String: ), Line Number: 10
Family: identifier_family, String: fib, Line Number: 10
Family: identifier_family, String: x, Line Number: 10
Family: integer_family, String: 2, Line Number: 10
Family: group_family, String: #}, Line Number: 11
PS G:\To Drive μου\UOI\Μεταφραστές>
```

Ανίχνευση Σφαλμάτων

Οι περιπτώσεις που ο λεκτικός αναλυτής μπορεί να ανιχνεύσει σφάλματα είναι οι παρακάτω περιπτώσεις:

`find_state(...)`: όταν χρησιμοποιείται ένας χαρακτήρας που δεν ανήκει στην γλώσσα.

`lex()`:

- όταν έχει ξεκινήσει η ροή ενός σχολίου και συναντηθεί ο χαρακτήρας EOF πριν το τέλος του σχολίου
- όταν έχει αναγνωριστεί κάποιος τύπος (#γράμματα) και δεν πρόκειται για τις λέξεις κλειδιά `#int` και `#def`
- όταν το τρέχον token είναι ένας ακέραιος αριθμός με τιμή εκτός του διαστήματος `[-32767,32767]`

Σε αυτές τις περιπτώσεις σταματά η εκτέλεση του προγράμματος και τυπώνονται τα αντίστοιχα διαγνωστικά μηνύματα όπως φαίνεται στα παρακάτω παραδείγματα λανθασμένων αρχείων.

test.cpy:

```
##i = int(input())
print(i)

i = 1600
while i<=2000:
    #{
        print(leap(i))
        i = i + 400
    #}
print(leap(2023))
print(leap(2024))
print(quad(3))
print(fib(5))

i=1
while i<=12:
    #{
        print(isPrime(i))
        i = i + 1
    #}

print(counterFunctionCalls)
```

Error:

```
C:\Windows\System32\cmd.e  X  +  v
Family: integer_family, String: 1, Line Number: 88
Family: keyword_family, String: return, Line Number: 90
Family: group_family, String: #}, Line Number: 91
Family: identifier_family, String: leap, Line Number: 92
Family: identifier_family, String: year, Line Number: 92
Family: delimiter_family, String: :, Line Number: 92
Family: keyword_family, String: global, Line Number: 95
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: integer_family, String: 1, Line Number: 96
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 4, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 100, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 400, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: keyword_family, String: return, Line Number: 98
Family: keyword_family, String: else, Line Number: 99
Family: keyword_family, String: return, Line Number: 100
Family: group_family, String: #}, Line Number: 100
Family: keyword_family, String: main, Line Number: 100
Family: identifier_family, String: i, Line Number: 101
Family: assignment_family, String: =, Line Number: 102
Error: '##' delimiter found on line 127.
Expected '##' delimiter before end of file.

G:\My Drive\CSE_UOI\8thSemester\compilers>
```

test.cpy:

```
$i = int(input())
print(i)

i = 1600
while i<=2000:
#{
    print(leap(i))
    i = i + 400
#}
```

```

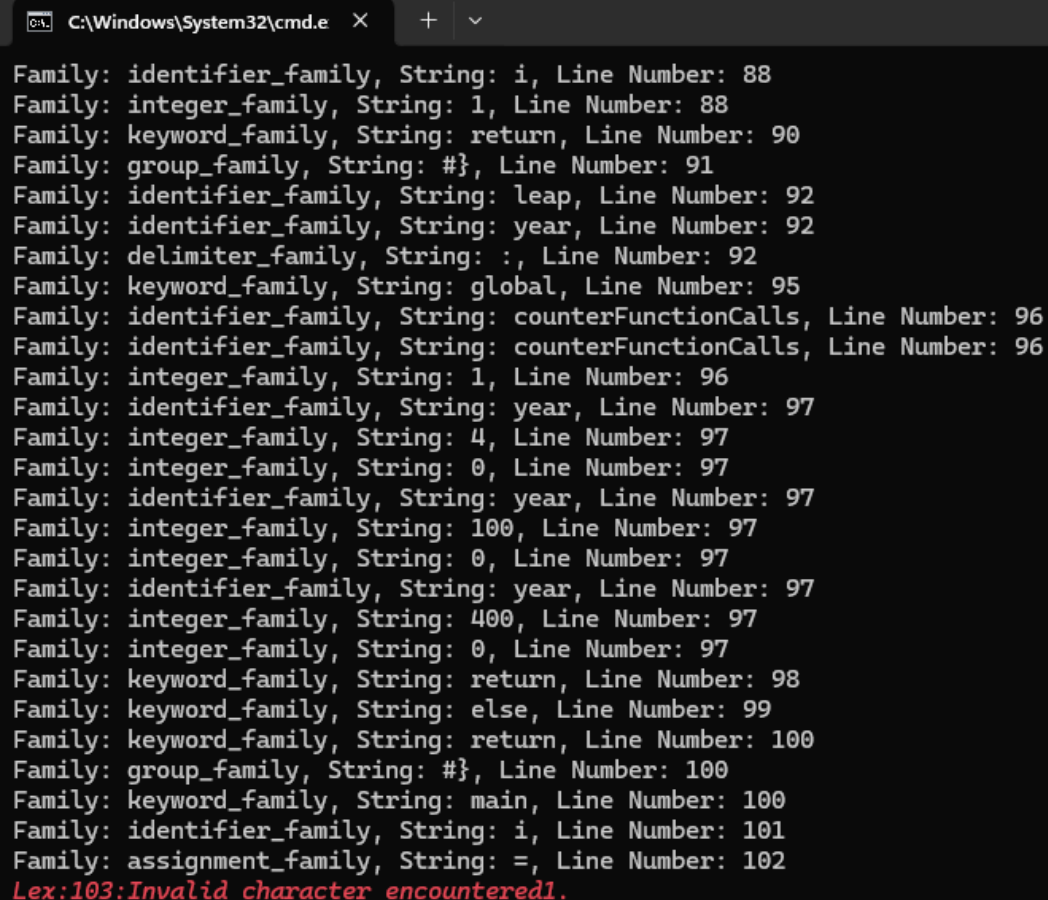
print( leap(2023) )
print( leap(2024) )
print( quad(3) )
print( fib(5) )

i=1
while i<=12:
#{
    print(isPrime(i))
    i = i + 1
#}

print(counterFunctionCalls)

```

Error:



```

C:\Windows\System32\cmd.e  X  +  v
Family: identifier_family, String: i, Line Number: 88
Family: integer_family, String: 1, Line Number: 88
Family: keyword_family, String: return, Line Number: 90
Family: group_family, String: #}, Line Number: 91
Family: identifier_family, String: leap, Line Number: 92
Family: identifier_family, String: year, Line Number: 92
Family: delimiter_family, String: :, Line Number: 92
Family: keyword_family, String: global, Line Number: 95
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: integer_family, String: 1, Line Number: 96
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 4, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 100, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 400, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: keyword_family, String: return, Line Number: 98
Family: keyword_family, String: else, Line Number: 99
Family: keyword_family, String: return, Line Number: 100
Family: group_family, String: #}, Line Number: 100
Family: keyword_family, String: main, Line Number: 100
Family: identifier_family, String: i, Line Number: 101
Family: assignment_family, String: =, Line Number: 102
Lex:103:Invalid character encountered1.

```

```

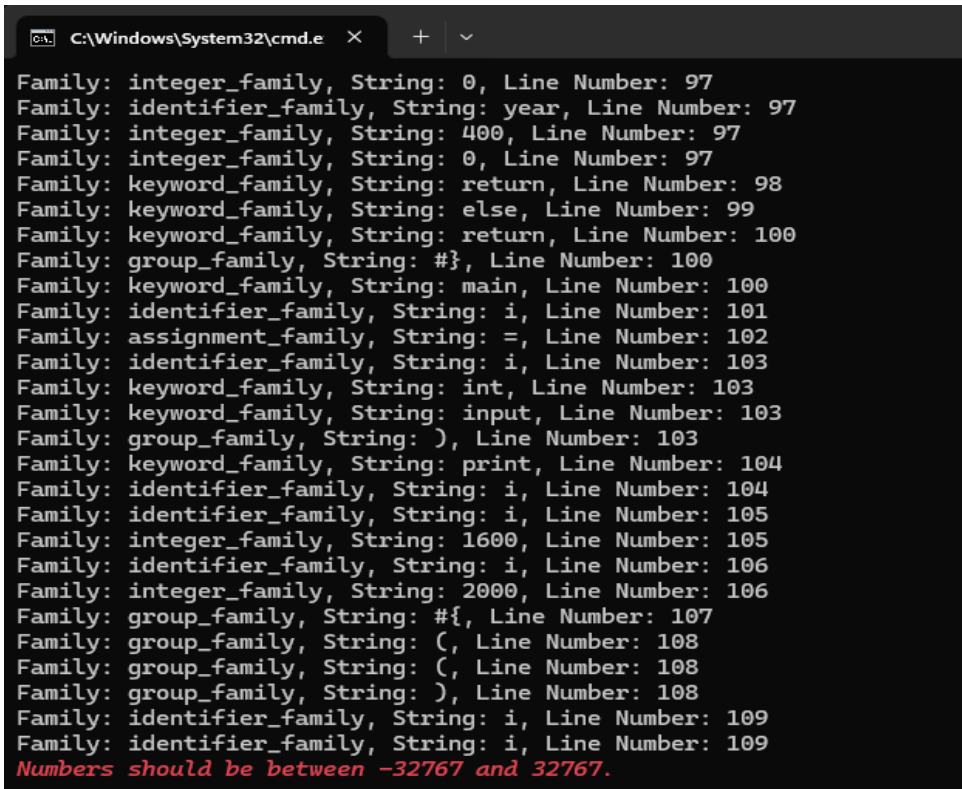
i = 1600
while i<=2000:
#{
    print(leap(i))
    i = i + 32768
#}
print(leap(2023))
print(leap(2024))
print(quad(3))
print(fib(5))

i=1
while i<=12:
#{
    print(isPrime(i))
    i = i + 1
#}

print(counterFunctionCalls)

```

Error:



```

C:\Windows\System32\cmd.e  X  +  v
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 400, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: keyword_family, String: return, Line Number: 98
Family: keyword_family, String: else, Line Number: 99
Family: keyword_family, String: return, Line Number: 100
Family: group_family, String: #}, Line Number: 100
Family: keyword_family, String: main, Line Number: 100
Family: identifier_family, String: i, Line Number: 101
Family: assignment_family, String: =, Line Number: 102
Family: identifier_family, String: i, Line Number: 103
Family: keyword_family, String: int, Line Number: 103
Family: keyword_family, String: input, Line Number: 103
Family: group_family, String: ), Line Number: 103
Family: keyword_family, String: print, Line Number: 104
Family: identifier_family, String: i, Line Number: 104
Family: identifier_family, String: i, Line Number: 105
Family: integer_family, String: 1600, Line Number: 105
Family: identifier_family, String: i, Line Number: 106
Family: integer_family, String: 2000, Line Number: 106
Family: group_family, String: #{, Line Number: 107
Family: group_family, String: (, Line Number: 108
Family: group_family, String: (, Line Number: 108
Family: group_family, String: ), Line Number: 108
Family: identifier_family, String: i, Line Number: 109
Family: identifier_family, String: i, Line Number: 109
Numbers should be between -32767 and 32767.

```

test.cpy:

```
#ind i
counterFunctionCalls = 0

i = int(input())
print(i)


i = 1600
while i<=2000:
    #{
        print(leap(i))
        i = i + 400
    #}
print(leap(2023))
print(leap(2024))
print(quad(3))
print(fib(5))


i=1
while i<=12:
    #{
        print(isPrime(i))
        i = i + 1
    #}

print(counterFunctionCalls)
```

Error:

```
C:\Windows\System32\cmd.e X + v
Family: keyword_family, String: return, Line Number: 87
Family: identifier_family, String: i, Line Number: 88
Family: identifier_family, String: i, Line Number: 88
Family: integer_family, String: 1, Line Number: 88
Family: keyword_family, String: return, Line Number: 90
Family: group_family, String: #}, Line Number: 91
Family: identifier_family, String: leap, Line Number: 92
Family: identifier_family, String: year, Line Number: 92
Family: delimiter_family, String: :, Line Number: 92
Family: keyword_family, String: global, Line Number: 95
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: identifier_family, String: counterFunctionCalls, Line Number: 96
Family: integer_family, String: 1, Line Number: 96
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 4, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 100, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: identifier_family, String: year, Line Number: 97
Family: integer_family, String: 400, Line Number: 97
Family: integer_family, String: 0, Line Number: 97
Family: keyword_family, String: return, Line Number: 98
Family: keyword_family, String: else, Line Number: 99
Family: keyword_family, String: return, Line Number: 100
Family: group_family, String: #}, Line Number: 100
Family: keyword_family, String: main, Line Number: 100
Found invalid type token #ind in line 101. Try valid tokens : 'int' and 'def'.
```

Συντακτικός Αναλυτής

Η υλοποίηση του συντακτικού αναλυτή ακολουθεί την παρακάτω γραμματική:

Γραμματική

Για τον σκοπό αυτό δημιουργήθηκαν οι συναρτήσεις:

- `syntax_analyzer()`

- `start_rule()`
- `def_main_part()`
- `call_main_part()`
- `def_function()`
- `code_blocks()`
- `code_block()`
- `simple_code_block()`
- `assignment_code_block()`
- `expression()`
- `optional_sign()`
- `term()`
- `factor()`
- `idtail()`
- `actual_param_list()`
- `print_code_block()`
- `return_code_block()`
- `structured_code_block()`
- `if_code_block()`
- `condition()`
- `bool_term()`
- `bool_factor()`
- `while_code_block()`
- `id_list()`
- `declarations()`
- `declaration_line()`
- `globals_()`
- `globals_line()`

`syntax_analyzer()`: Καλείται η μέθοδος `start_rule()` και τυπώνεται το μήνυμα επιτυχούς μετάφρασης, σε περίπτωση που δεν έχει προκύψει κάποιο συντακτικό σφάλμα.

`start_rule()`: Αρχικά ελέγχει τις ενδεχόμενες δηλώσεις που υπάρχουν στην κορυφή του προγράμματος, καλεί την συνάρτηση `def_main_part()`, η οποία είναι υπεύθυνη για την μετάφραση όλων των συναρτήσεων εκτός της `main`. Την μετάφραση της `main` την αναλαμβάνει η συνάρτηση `call_main_part()`, που καλείται στο τέλος.

`def_main_part()`: Στην συγκεκριμένη συνάρτηση γίνεται η μετάφραση όλων των συναρτήσεων. Κάθε φορά που ολοκληρώνεται η μετάφραση κάποιας συνάρτησης ελέγχεται το επόμενο token και αν είναι το keyword “def” τότε ακολουθεί η μετάφραση της επόμενης συνάρτησης.

`call_main_part()`: Σε αυτή τη μέθοδο πραγματοποιείται η μετάφραση της `main` που προϋποθέτει την ανάγνωση του token “#def” που ακολουθείται από το token “main”. Αν ισχύει αυτή η προϋπόθεση τότε καλούνται οι συναρτήσεις `declarations()`, `globals_()` και `code_blocks()`.

`def_function()`: Σε αυτή τη μέθοδο γίνεται η μετάφραση κάποιας συνάρτησης. Για να ξεκινήσει η μετάφραση πρέπει να αναγνωριστούν τα εξής:

1. το token “def”
2. ένα identifier token
3. άνοιγμα παρένθεσης
4. (προαιρετικά) κάποια ορίσματα

5. κλείσιμο παρένθεσης
6. το token “:”
7. το token “#{”

Σημείωση: Για την αναγνώριση των προαιρετικών ορισμάτων καλείται η μέθοδος `id_list()`.

Ακολουθεί η μετάφραση των δηλώσεων, των εμφωλευμένων συναρτήσεων(αναδρομικά), των καθολικών μεταβλητών και των `code_blocks()`. Τέλος για επιτυχή μετάφραση αναμένεται να υπάρχει και το token ομαδοποίησης “#”.

`code_blocks()`: Στην συγκεκριμένη μέθοδο μεταφράζονται όλα τα `code_block()` που μπορεί να περιέχει μία συνάρτηση τα οποία είναι τα εξής:

- while-code-block
- return-code-block
- print-code-block
- if-code-block

ή να ακολουθεί ένα `identifier-token`.

`code_block()`: Ανάλογα με το τρέχον token θα κληθεί και η αντίστοιχη μέθοδος μεταξύ των `simple_code_block()` και `structured_code_block()`.

`simple_code_block()`: Ανάλογα με το τρέχον token θα κληθεί και η αντίστοιχη μέθοδος μεταξύ των `assignment_code_block()`, `print_code_block()` και `return_code_block()`.

`structured_code_block()`: Ανάλογα με το τρέχον `token` θα κληθεί και η αντίστοιχη μέθοδος μεταξύ των `if_code_block()` και `while_code_block()`.

`assignment_code_block()`: Αρχικά ελέγχεται αν το τρέχον `token` είναι το σύμβολο `"="`. Σε αυτή την περίπτωση μπορεί να ακολουθεί η μετάφραση μιας εκχώρησης ενός `expression`(μέθοδος `expression()`) ή η εκχώρηση ενός `input`.

`expression()`: Στην αρχή ενός `expression` ενδέχεται να υπάρχει ένα προαιρετικό πρόσημο , που μεταφράζεται από την συνάρτηση `optional_sign()`, το οποίο ακολουθείται από έναν όρο. Όσο συναντάται ένα `token` που ανήκει στην οικογένεια `"addOper_family"` απαιτείται να ακολουθεί κάποιος όρος.

`optional_sign()`: Όπως προαναφέρθηκε καταναλώνει ένα `token` που ανήκει στην οικογένεια `"addOper_family"`.

`term()`: Κατά την μετάφραση ενός όρου πρέπει να αναγνωριστεί ένα `"factor"` το οποίο μπορεί να ακολουθείται από οσοδήποτε ζευγάρια ενός `token` της κατηγορίας `"mulOper_family"` και κάποιο άλλο `"factor"`.

`factor()`: Για την μετάφραση ενός παράγοντα υπάρχουν τρεις περιπτώσεις. Η πρώτη είναι να ανήκει στην οικογένεια `"integer_family"`, η δεύτερη να είναι της μορφής `"(expression)"` και η τρίτη να ανήκει στην οικογένεια `"identifier_family"`. Στην τελευταία περίπτωση θα κληθεί η συνάρτηση `"idtail()"` που θα αναλυθεί παρακάτω.

`idtail()`: Στην πράξη η συγκεκριμένη μέθοδος μεταφράζει την κλήση μίας συνάρτησης και την αναγνώριση των παραμέτρων της, για αυτό καλείται η μέθοδος `“actual_param_list()`”.

`actual_param_list()`: Σε αυτή τη μέθοδο αναμένεται ότι θα βρεθεί ένα `“expression”` το οποίο μπορεί να ακολουθείται από ζευγάρια της μορφής `“, expression”`. Με αυτόν τον τρόπο αναγνωρίζει τις παραμέτρους κατά την κλήση μίας συνάρτησης.

`print_code_block()`: Έχοντας σίγουρα αναγνωρίσει το keyword `“print”` περιμένουμε να διαβάσουμε το άνοιγμα και κλείσιμο των παρενθέσεων και των ορισμάτων που περικλείουν.

`return_code_block()`: Έχοντας αναγνωρίσει το keyword `“return”` σε περίπτωση που δεν υπάρχει κάποια τιμή επιστροφής θα βρεθεί το token `“#”` ενώ διαφορετικά θα αναγνωρισθεί ένα `“expression”`.

`if_code_block()`: Στην συγκεκριμένη μέθοδο η αναγνώριση του `if-code-block` γίνεται αναδρομικά. Την πρώτη φορά θα διαβαστεί το keyword `“if”`, το `“condition”`, το token `“:”` , το token ομαδοποίησης `“#{”` (στην περίπτωση που το περιεχόμενο του `if`(αντίστοιχα `elif`) είναι παραπάνω από μία γραμμή) και θα κληθεί αναδρομικά η συνάρτηση. Μετά την πρώτη αναδρομική κλήση μπορούν να αναγνωριστούν είτε `elif-code-block` είτε `else-code-block` που ακολουθούν την ίδια λογική με το `if-code-block` που αναφέρθηκε προηγουμένως. Η βασική περίπτωση της αναδρομής είναι

η αναγνώριση ενός token διαφορετικό από τα keywords “elif” και “else”.

condition(): Κατά την μετάφραση μιας συνθήκης πρέπει να αναγνωρισθεί ένα “bool_term” το οποίο μπορεί να ακολουθείται από οσοδήποτε ζευγάρια token “or” και “bool_term”.

bool_term(): Για την μετάφραση ενός “bool_term” πρέπει να αναγνωρισθεί ένα “bool_factor” το οποίο μπορεί να ακολουθείται από οσοδήποτε ζευγάρια token “and” και “bool_factor”.

bool_factor(): Επειδή η γραμματική είναι LL(1) η αναγνώριση ή όχι του keyword “not” θα καθορίσει τον τρόπο μετάφρασης του εκάστοτε “bool_factor”.

while_code_block(): Έχοντας διαβάσει το keyword “while” αναμένεται η μετάφραση του “condition” , το token “:” , το token ομαδοποίησης “#{” (στην περίπτωση που το περιεχόμενο while είναι παραπάνω από μία γραμμή) και καλείται αντιστοίχως είτε η code_blocks() είτε η code_block().

id_list(): Η συγκεκριμένη μέθοδος χρησιμοποιείται για την αναγνώριση των τυπικών παραμέτρων στον ορισμό μιας συνάρτησης, στην δήλωση καθολικών ή μη καθολικών μεταβλητών. Πρόκειται για οσοδήποτε αριθμό ζευγαριών “,identifier” εξαιρώντας το πρώτο “identifier” token.

declarations(): Όσο διαβάζεται το keyword “#int” καλείται η μέθοδος “declaration_line()” για την αναγνώριση των δηλώσεων μεταβλητών που ακολουθούν.

declaration_line(): Στην συγκεκριμένη μέθοδο αναγνωρίζονται δηλώσεις μεταβλητών, που ενδεχομένως να είναι χωρισμένες με το token “,”. Για αυτό τον λόγο αξιοποιείται και η μέθοδος “id_list()” που εξηγήθηκε παραπάνω.

globals_(): Όσο διαβάζεται το keyword “global” καλείται η μέθοδος “global_line()” για την αναγνώριση των καθολικών μεταβλητών που ακολουθούν.

globals_line(): Στην συγκεκριμένη μέθοδο αναγνωρίζονται δηλώσεις καθολικών μεταβλητών, που ενδεχομένως να είναι χωρισμένες με το token “,”. Για αυτό τον λόγο αξιοποιείται και η μέθοδος “id_list()” που εξηγήθηκε παραπάνω.

Error Detection:

test.cpy:

```
def max3(x,y,z):
#{
    #int m,z,e

    def divides(x,y):
        #{
            ## body of divides ##

            global counterFunctionCalls
            counterFunctionCalls = counterFunctionCalls + 1
            if y == (y//x)*x and x<z:
                return 1
            else:
                return 0

global counterFunctionCallsQWEQEWQEQWEQWEQ
global vc
global mp
id = int(input())
counterFunctionCalls = vc + temp(1,1)
if x == 1:
    #{
        return
    #}
print(1)
if x>y and x>z:
    m = x
elif y>x and y>z:
    m = y
else:
    m = z
return m
#}
```

Λείπει η παρένθεση για την λήξη της “divides”.

Error:

```
Did not find token '#}'
```

```
G:\My Drive\CSE_UOI\8thSemester\compilers>
```

test.cpy:

```
def leap(year):  
    ## returns 1 if year is a leap year, otherwise it  
    returns 0 ##  
    #{  
        global counterFunctionCalls  
        counterFunctionCalls = counterFunctionCalls +  
1  
        if year%4==0 and year%100!=0 or year%400==0  
            return 1  
        else:  
            return 0  
    #}
```

Λείπει “:” μετά το `year%400==0`:

```
Syntax:219:Expected ':'
```

```
G:\My Drive\CSE_UOI\8thSemester\compilers>
```

test.cpy:

```
def leap(year:  
    ## returns 1 if year is a leap year, otherwise it  
    returns 0 ##
```

```
# {  
    global counterFunctionCalls  
    counterFunctionCalls = counterFunctionCalls +  
1  
    if year%4==0 and year%100!=0 or year%400==0:  
        return 1  
    else:  
        return 0  
# }
```

Λείπει η παρένθεση στο `leap(year:`

Error:

Did not find token ')'

G:\My Drive\CSE_UOI\8thSemester\compilers>|

Ενδιάμεσος κώδικας

Για την υλοποίηση του ενδιάμεσου κώδικα δημιουργήθηκε η κλάση “Quad”(τα quads αποθηκεύονται σε μια λίστα και η αρίθμηση γίνεται με βάση την μεταβλητή quad_counter), η οποία έχει ως πεδία τον αριθμό της τετράδας, έναν τελεστή και τρία τελούμενα, και οι παρακάτω συναρτήσεις:

def nextquad()	Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
def genquad(op, x, y, z)	Δημιουργεί την επόμενη τετράδα (op, x, y, z)
def emptylist()	Δημιουργεί μία κενή λίστα ετικετών τετράδων
def makelist(x)	Δημιουργεί μία λίστα ετικετών τετράδων που περιέχει μόνο το x
def merge(list1,list2)	Δημιουργεί μία λίστα ετικετών τετράδων από τη συνένωση των λιστών list1, list2
def backpatch(lst,z)	-η λίστα list αποτελείται από δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο -η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z

Αριθμητικές Παραστάσεις

Δομή expression:

$E \rightarrow T^1 (+ T^2 \{P_1\}) * \{P_2\}$

$\{P_1\}$: `w = newTemp()`
 `genquad("+",T1.place,T2.place,w)`
 `T1.place=w`

$\{P_2\}$: `E.place=T1.place`

```
def expression():    #Consumes 1 token
    global token

    optional_sign()
    t1_place = term()

    while token.family == 'addOper_family':
        if token.family == 'addOper_family':
            sign = token.current_string
            token = lex()

        t2_place = term()

        w = newtemp()
        genquad(sign,t1_place,t2_place,w)
        t1_place = w

    return t1_place
```

Μεταβλητή w : Η συγκεκριμένη προσωρινή μεταβλητή χρησιμοποιείται για την αποθήκευση του τρέχοντος αποτελέσματος.

Παραγωγή τετράδας: Το προσωρινό αποτέλεσμα θα προστεθεί στην τιμή της μεταβλητής $t2_place$.

$t1_place = w$: Το προσωρινό αποτέλεσμα αποθηκεύεται στην μεταβλητή $t1_place$ έτσι ώστε να μπορεί να αξιοποιηθεί σε περίπτωση που υπάρχει και άλλο $t2_place$.

Τελικά επιστρέφεται το αποτέλεσμα $t1_place$ καθώς υπάρχει περίπτωση να μην υπάρχει κάποιο $t2_place$.

Δομή term:

$T \rightarrow F^1 (\times F^2 \{P_1\})^* \{P_2\}$

$\{P_1\}$: $w = \text{newTemp}()$

$\text{genquad}(\text{"\times"}, F^1.\text{place}, F^2.\text{place}, w)$

$F^1.\text{place} = w$

$\{P_2\}$: $T.\text{place} = F^1.\text{place}$

```

def term():
    global token

    f1_place = factor()

    while token.family == 'mulOper_family':
        if token.family == 'mulOper_family':
            token = lex()

        f2_place = factor()

        w = newtemp()
        genquad('*', f1_place, f2_place, w)
        f1_place = w

    return f1_place

```

Με παρόμοιο τρόπο γίνεται και η παραγωγή του ενδιάμεσου κώδικα για την δομή term. Όπως και προηγουμένως εξασφαλίζεται και η περίπτωση που υπάρχει μόνο f1_place χωρίς κάποιο mulOper και f2_place να ακολουθεί.

Μεταβλητή w: Η συγκεκριμένη προσωρινή μεταβλητή χρησιμοποιείται για την αποθήκευση του τρέχοντος αποτελέσματος.

Παραγωγή τετράδας: Το προσωρινό αποτέλεσμα θα προστεθεί στην τιμή της μεταβλητής f2_place.

f1_place = w: Το προσωρινό αποτέλεσμα αποθηκεύεται στην μεταβλητή f1_place έτσι ώστε να μπορεί να αξιοποιηθεί σε περίπτωση που υπάρχει και άλλο f2_place.

Τελικά επιστρέφεται το αποτέλεσμα f1_place καθώς υπάρχει περίπτωση να μην υπάρχει κάποιο f2_place.

Δομή factor:

F -> (E) {P₁}

{P₁}: F.place=E.place

F -> id {P₁}

{P₁}: F.place=id.place

```
def factor():
    global token

    f_place = ""

    if token.family == 'integer_family':
        f_place = token.current_string
        token = lex()
    elif token.current_string == '(':
        token = lex()
        f_place = expression()
        if token.current_string == ')':
            token = lex()
    else:
```

```

        print(f"\033[3;31mSyntax:{token.line_num}:Expected
') '\033[0m")
        exit(0)
    elif token.family == 'identifier_family': # fib(x+1)
        factor_string = token.current_string
        token = lex()
        f_place = idtail(factor_string)
    else:
        print(f"\033[3;31mSyntax:{token.line_num}:Expected
factor\033[0m")

        exit(0)

    return f_place

```

Στην συγκεκριμένη δομή ο ενδιαμέσος κώδικας που πρόκειται να παραχθεί προέρχεται από τις δομές expression, idtail ή από κάποιον identifier. Σε κάθε περίπτωση γίνεται απλή μεταφορά του εκάστοτε περιεχομένου(place) στο f_place.

Δομή idtail:

```

def idtail(factor_string):
    global token

    if token.current_string == '(':
        token = lex()
        actual_param_list()

        w = newtemp()
        genquad('par',w,'ret','_')
        genquad('call',factor_string,'_','_')

    if token.current_string == ')':

        token = lex()
        return w

```

```

else:

    print(f"\033[3;31mSyntax:{token.line_num}:Expected ' '\033[0m")
    exit(0)
return factor_string

```

Στην δομή “idtail” αφού παραχθεί ο ενδιάμεσος κώδικας από την μέθοδο “actual_param_list()” δημιουργείται ενδιάμεσος κώδικας για τον ορισμό της παραμέτρου επιστροφής και την κλήση της συνάρτησης.

Δομή actual_param_list:

```

def actual_param_list():
    global token

    ident_var=Variable(token.current_string,calculate_offset()+4)
    add_entity(ident_var)

    e_place = expression()
    genquad('par',e_place,'cv','_')

    while token.current_string == ',':
        token = lex()

        ident_var=Variable(token.current_string,calculate_offset()+4)
        add_entity(ident_var)

    e_place = expression()
    genquad('par',e_place,'cv','_')

```

Στην παραπάνω δομή παράγεται ενδιάμεσος κώδικας για της πραγματικές παραμέτρους κατά την κλήση κάποιας συνάρτησης. Όλες οι μεταβλητές περνάνε με τιμή (cv) ,όπως έχει οριστεί από την εκφώνηση της άσκησης.

Λογικές παραστάσεις

Λογικές Παραστάσεις - OR

$B \rightarrow Q^1 \{P_1\} (\text{ or } \{P_2\} Q^2 \{P_3\})^*$

$\{P_1\}$: $B.\text{true} = Q^1.\text{true}$

$B.\text{false} = Q^1.\text{false}$

$\{P_2\}$: $\text{backpatch}(B.\text{false}, \text{nextquad}())$

$\{P_3\}$: $B.\text{true} = \text{merge}(B.\text{true}, Q^2.\text{true})$

$B.\text{false} = Q^2.\text{false}$

```
def condition():
    global token

    b1_place = bool_term()

    c_true = b1_place[0]
    c_false = b1_place[1]

    while token.current_string == 'or':
        if token.current_string == 'or':
            token = lex()
            backpatch(c_false, nextquad())

        b2_place = bool_term()
        c_true = merge(c_true, b2_place[0])
        c_false = b2_place[1]

    return c_true, c_false
```


Το αποτέλεσμα αυτής της δομής είναι δύο λίστες οι `c_true` και η `c_false`. Επειδή ο λογικός τελεστής είναι το λογικό “ή” η λίστα που περιέχει τις τετράδες που αποτιμήθηκαν ως μη αληθής `c_false` εκτελεί `backpatch` στην επόμενη τετράδα καθώς ο προορισμός του επόμενου άλματος θα καθοριστεί από τα αποτελέσματα των επόμενων “quad”(μόνο μία συνθήκη αρκεί να αποτιμηθεί ως αληθής για να ισχύει η λογική πράξη). Σε περίπτωση που η τετράδα αντιστοιχεί σε αληθή αποτίμηση θα προστεθεί στην λίστα `c_true`. Η συμπλήρωση τους θα πραγματοποιηθεί σε επόμενη φάση όταν ο προορισμός του επόμενου άλματος γνωστοποιηθεί. Οι λίστες `c_true` και `c_false` θα μεταφερθούν σε παραπάνω κανόνες.

Λογικές Παραστάσεις - AND

$Q \rightarrow R^1 \{P_1\} (\text{and } \{P_2\} R^2 \{P_3\})^*$

$\{P_1\}$: `Q.true = R1.true`

`Q.false = R1.false`

$\{P_2\}$: `backpatch(Q.true, nextquad())`

$\{P_3\}$: `Q.false = merge(Q.false, R2.false)`

`Q.true = R2.true`

```

def bool_term():
    global token

    bf1_place = bool_factor()

    b_true = bf1_place[0]
    b_false = bf1_place[1]

    while token.current_string == 'and':
        if token.current_string == 'and':
            token = lex()
            backpatch(b_true,nextquad())

        bf2_place = bool_factor()
        b_false = merge(b_false,bf2_place[1])
        b_true = bf2_place[0]

    return b_true,b_false

```

Το αποτέλεσμα αυτής της δομής είναι δύο λίστες οι `b_true` και η `b_false`. Επειδή ο λογικός τελεστής είναι το λογικό “και” η λίστα που περιέχει τις τετράδες που αποτιμήθηκαν ως μη αληθής `b_true` εκτελεί `backpatch` στην επόμενη τετράδα καθώς ο προορισμός του επόμενου άλματος θα καθοριστεί από τα αποτελέσματα των επόμενων “quad”(πρέπει όλες οι συνθήκες να αποτιμηθούν ως αληθής για να ισχύει η λογική πράξη). Σε περίπτωση που η τετράδα αντιστοιχεί σε μη αληθή αποτίμηση θα προστεθεί στην λίστα `b_false` και θα γίνει `merge` με τις υπόλοιπες τετράδες. Η συμπλήρωση τους θα πραγματοποιηθεί σε επόμενη φάση όταν ο προορισμός του επόμενου άλματος γνωστοποιηθεί. Οι λίστες `b_true` και `b_false` θα μεταφερθούν σε παραπάνω κανόνες.

Απλό πέρασμα αποτελεσμάτων

$R \rightarrow (B) \{P_1\}$

$\{P_1\}: \quad R.true=B.true$

$R.false=B.false$

Στην συγκεκριμένη περίπτωση τα αποτελέσματα b_true και b_false απλά προωθούνται στον παραπάνω κανόνα.

Δομή NOT

$R \rightarrow \text{not } (B) \{P_1\}$

$\{P_1\}: \quad R.true=B.false$

$R.false=B.true$

Σε αυτή τη δομή γίνεται αντιστροφή των λιστών b_true και b_false και προωθούνται στον παραπάνω κανόνα.

Δομή bool_factor

$R \rightarrow E^1 \text{ relop } E^2 \{P_1\}$

$\{P_1\}$: $R.\text{true} = \text{makelist}(\text{nextquad}())$
 $\text{genQuad}(\text{relOp}, E^1.\text{place}, E^2.\text{place}, \text{"_"})$
 $R.\text{false} = \text{makelist}(\text{nextquad}())$
 $\text{genQuad}(\text{"jump"}, \text{"_"}, \text{"_"}, \text{"_"})$

```
def bool_factor():
    global token

    if token.current_string == 'not':
        e1_place = expression()

        if token.family == 'relOper_family':
            token = lex()
            e2_place = expression()

            r_true = makelist(nextquad())
            genquad('rel_op', e1_place, e2_place, '_')
            r_false = makelist(nextquad())
            genquad('jump', '_', '_', '_')

            return r_true, r_false
        else:
            e1_place = expression()

            if token.family == 'relOper_family':
                token = lex()
                e2_place = expression()

                r_true = makelist(nextquad())
```

```

        genquad('rel_op',e1_place,e2_place,'_')
        r_false = makelist(nextquad())
        genquad('jump','_','_','_')

        return r_true, r_false
    else:
        print("\033[3;31mSyntax:{}:Expected expression relOper
expression.\033[0m {}".format(token.line_num, format.string))
        exit(0)

```

Η λίστα `r_true` περιέχει την μη συμπληρωμένη τετράδα που αντιστοιχεί στην αληθή αποτίμηση της “relor”. Η λίστα `r_false` περιέχει την μη συμπληρωμένη τετράδα που αντιστοιχεί στην μη αληθή αποτίμηση της “relor”.

Στην περίπτωση του “not” τα περιεχόμενα των δύο λιστών αντιστρέφονται.

Δομή return

S -> return (E) {P1}

{P1}: genquad(“retv”,E.place,”_”,”_”)

```

genquad('retv',e_place,'_','_')

```

Δομή assignment

$S \rightarrow id := E \{P1\};$

$\{P1\} : \quad \text{genQuad}(":=", E.\text{place}, _ , id)$

```
genquad('=', e_place, '_', ident)
```

Δομές εισόδου-εξόδου

$S \rightarrow \text{input}(id) \{P1\}$

$\{P1\} : \quad \text{genquad}(\text{"inp"}, id.\text{place}, _ , _)$

$S \rightarrow \text{print}(E) \{P2\}$

$\{P2\} : \quad \text{genquad}(\text{"out"}, E.\text{place}, _ , _)$

```
def print_code_block():  
    global token  
  
    token = lex()  
  
    if token.current_string == '(':  
        token = lex()
```

```

    e_place = expression()
    genquad('out',e_place,'_','_')

    if token.current_string == ')':
        token = lex()
    else:
print(f"\033[3;31mSyntax:{token.line_num}:Expected') '\033[0m")
        exit(0)

def return_code_block():
    global token
    token = lex()

    if token.current_string == '#}':
        return

    e_place = expression()
    genquad('retv',e_place,'_','_')

```

Δομές if/while

Δομή if

$S \rightarrow \text{if } B \text{ then } \{P1\} S^1 \{P2\} \text{TAIL } \{P3\}$

$\{P1\}$: `backpatch(B.true,nextquad())`

$\{P2\}$: `ifList=makelist(nextquad())`

`genquad("jump","_","_","_")`

`backpatch(B.false,nextquad())`

$\{P3\}$: `backpatch(ifList,nextquad())`

$\text{TAIL} \rightarrow \text{else } S^2 \mid \text{TAIL} \rightarrow \epsilon$

Σημείωση: Ο αντίστοιχος κώδικας βρίσκεται στην γραμμή 1021

Σε αυτή τη φάση οι τετράδες `b_true` μπορούν να συμπληρωθούν με τον αριθμό της επόμενης τετράδας. Για να εξασφαλίσουμε ότι η εκτέλεση του `if-code-block` δεν θα συνεχιστεί και στο `else-code-block` δημιουργούμε μία λίστα `"if_list"` με ένα ασυμπλήρωτο `jump` με επιθυμητό προορισμό αμέσως μετά το `else-code-block`. Κατά την εκτέλεση του `if-code-block` η επόμενη τετράδα αντιστοιχεί στο `else-code-block` στο οποίο πλέον μπορούμε να κάνουμε `backpatch` των τετράδων της λίστας `b_false`. Τελικά όταν βρεθούμε εκτός και από το `else-code-block` γνωρίζουμε πλέον και τον προορισμό του `jump` που δημιουργήσαμε προηγουμένως και σε αυτό το σημείο εκτελούμε ένα ακόμα `backpatch` για την συμπλήρωση του.

Δομή **while**

S -> while {P1} B do {P2} S¹ {P3}

{P1}: Bquad:=nextquad()

{P2}: backpatch(B.true,nextquad())

**{P3}: genquad("jump","_","_",Bquad)
 backpatch(B.false,nextquad())**

```
def while_code_block():  
    global token
```



```

first_quad = nextquad()

token = lex()

b_place = condition()

if token.current_string == ':':
    backpatch(b_place[0],nextquad())

    token = lex()

    if token.current_string == '#{':
        token = lex()
        code_blocks()
        if token.current_string == '#}':
            token = lex()
        else:
            print("\033[3;31mSyntax:{}:Expected '#{'}'.\033[0m".format(token.line_num, format.string))
            exit(0)
    else:
        code_block()

    genquad('jump','_','_',first_quad)
    backpatch(b_place[1],nextquad())

    return b_place[0],b_place[1]

else:
    print("\033[3;31mSyntax:{}:Expected ':'.\033[0m".format(token.line_num, format.string))
    exit(0)

```

Αρχικά αποθηκεύουμε την πρώτη τετράδα (“first_quad”) και λαμβάνουμε από το παρακάτω επίπεδο τις λίστες b_true και b_false. Σε αυτή τη φάση γνωρίζουμε τον προσορισμό των τετράδων που είναι αποθηκευμένες στην λίστα b_true και εκτελούμε backpatch στο nextquad. Για να επαναφέρουμε την ροή εκτέλεσης στην αρχή του while-code-block και να επαναληφθεί ο σχετικός έλεγχος παράγουμε μία τετράδα

jump στην διεύθυνση της τετράδας που αποθηκεύσαμε στην μεταβλητή “first_quad”. Τελικά στην περίπτωση των μη αληθών τετράδων b_false εκτελούμε backpatch στο nextquad. Στην δομή while έχει σημασία η σειρά με την οποία γίνεται η παραγωγή της τετράδας jump και του backpatch των τετράδων b_false καθώς διαφορετικά η ροή εκτέλεσης θα επέστρεφε κάθε φορά στο σημείο ελέγχου του while-code-block.

Παράδειγμα εκτέλεσης:

```
#def main
#int i
counterFunctionCalls = 0

i = int(input())
print(i)

i = 1600
while i<=2000:
#{
|   print(leap(i))
|   i = i + 400
#}
print(leap(2023))
print(leap(2024))
print(quad(3))
print(fib(5))

i=1
while i<=12:
#{
|   print(isPrime(i))
|   i = i + 1
#}

print(counterFunctionCalls)
```

```

(Quad number: 0 ,op: begin_block ,first operand: main ,second operand: _ ,result: _)
(Quad number: 1 ,op: = ,first operand: 0 ,second operand: _ ,result: counterFunctionCalls)
(Quad number: 2 ,op: inp ,first operand: i ,second operand: _ ,result: _)
(Quad number: 3 ,op: out ,first operand: i ,second operand: _ ,result: _)
(Quad number: 4 ,op: = ,first operand: 1600 ,second operand: _ ,result: i)
(Quad number: 5 ,op: rel_op ,first operand: i ,second operand: 2000 ,result: 7)
(Quad number: 6 ,op: jump ,first operand: _ ,second operand: _ ,result: 14)
(Quad number: 7 ,op: par ,first operand: i ,second operand: cv ,result: _)
(Quad number: 8 ,op: par ,first operand: T_8 ,second operand: ret ,result: _)
(Quad number: 9 ,op: call ,first operand: leap ,second operand: _ ,result: _)
(Quad number: 10 ,op: out ,first operand: T_8 ,second operand: _ ,result: _)
(Quad number: 11 ,op: + ,first operand: i ,second operand: 400 ,result: T_9)
(Quad number: 12 ,op: = ,first operand: T_9 ,second operand: _ ,result: i)
(Quad number: 13 ,op: jump ,first operand: _ ,second operand: _ ,result: 5)
(Quad number: 14 ,op: par ,first operand: 2023 ,second operand: cv ,result: _)
(Quad number: 15 ,op: par ,first operand: T_10 ,second operand: ret ,result: _)
(Quad number: 16 ,op: call ,first operand: leap ,second operand: _ ,result: _)
(Quad number: 17 ,op: out ,first operand: T_10 ,second operand: _ ,result: _)
(Quad number: 18 ,op: par ,first operand: 2024 ,second operand: cv ,result: _)
(Quad number: 19 ,op: par ,first operand: T_11 ,second operand: ret ,result: _)
(Quad number: 20 ,op: call ,first operand: leap ,second operand: _ ,result: _)
(Quad number: 21 ,op: out ,first operand: T_11 ,second operand: _ ,result: _)
(Quad number: 22 ,op: par ,first operand: 3 ,second operand: cv ,result: _)
(Quad number: 23 ,op: par ,first operand: T_12 ,second operand: ret ,result: _)
(Quad number: 24 ,op: call ,first operand: quad ,second operand: _ ,result: _)
(Quad number: 25 ,op: out ,first operand: T_12 ,second operand: _ ,result: _)
(Quad number: 26 ,op: par ,first operand: 5 ,second operand: cv ,result: _)
(Quad number: 27 ,op: par ,first operand: T_13 ,second operand: ret ,result: _)
(Quad number: 28 ,op: call ,first operand: fib ,second operand: _ ,result: _)
(Quad number: 29 ,op: out ,first operand: T_13 ,second operand: _ ,result: _)
(Quad number: 30 ,op: = ,first operand: 1 ,second operand: _ ,result: i)
(Quad number: 31 ,op: rel_op ,first operand: i ,second operand: 12 ,result: 33)
(Quad number: 32 ,op: jump ,first operand: _ ,second operand: _ ,result: 40)
(Quad number: 33 ,op: par ,first operand: i ,second operand: cv ,result: _)
(Quad number: 34 ,op: par ,first operand: T_14 ,second operand: ret ,result: _)
(Quad number: 35 ,op: call ,first operand: isPrime ,second operand: _ ,result: _)
(Quad number: 36 ,op: out ,first operand: T_14 ,second operand: _ ,result: _)
(Quad number: 37 ,op: + ,first operand: i ,second operand: 1 ,result: T_15)
(Quad number: 38 ,op: = ,first operand: T_15 ,second operand: _ ,result: i)
(Quad number: 39 ,op: jump ,first operand: _ ,second operand: _ ,result: 31)
(Quad number: 40 ,op: out ,first operand: counterFunctionCalls ,second operand: _ ,result: _)
(Quad number: 41 ,op: halt ,first operand: main ,second operand: _ ,result: _)
(Quad number: 42 ,op: end_block ,first operand: main ,second operand: _ ,result: _)

```

Όπως φαίνεται παραπάνω στην αρχή κάθε συνάρτησης δημιουργείται η τετράδα:

(begin_block,<function_name>, _, _)

και στο τέλος η τετράδα:

(end_block,<function_name>, _, _).

Επιπρόσθετα για την main δημιουργείται και ακριβώς πριν την τελευταία τετράδα η:

(halt, main, _, _)

Πίνακας Συμβόλων

Για την δημιουργία του πίνακα συμβόλων χρησιμοποιήθηκαν οι κλάσεις Entity, Variable, Function, Constant, Parameter, TempVariable, Scope και Argument. Οι κλάσεις Variable, Function, Constant και Parameter κληρονομούν από την κλάση Entity.

Για την κλάση Entity και τις κλάσεις που κληρονομούν από αυτήν έχει οριστεί constructor και to string μέθοδος. Για κάθε κλάση που έχει σχέση κληρονομικότητας, στον constructor γίνεται αρχικοποίηση του super και ορίζονται οι μεταβλητές της.

Αντίστοιχα για το Scope και το Argument, έχουμε constructor που ορίζει τις μεταβλητές και μια to string μέθοδο.

Επομένως έχουμε:

Entity:

- name

Variable:

- name(super)
- offset

Function:

- name(super)
- type
- start_quad
- arguments

-frame_length(σε bytes)

Constant:

-name(super)

-value

Parameter:

-name(super)

-par_mode

-offset

TempVariable:

-name(super)

-offset

Scope:

-entities

-nesting_level

Argument:

-par_mode

Δημιουργήθηκαν οι παρακάτω συναρτήσεις για την διαχείριση των Entities, Scopes και Arguments:

def add_scope(scope)	για την προσθήκη ενός scope στην λίστα scopes
def delete_scope()	για την διαγραφή ενός scope από την λίστα scopes
def calculate offset()	για τον υπολογισμό του offset του επόμενου entity που θα προστεθεί στην

	λίστα ,η οποία αντιπροσωπεύει την στοίβα
def add_entity(entity)	για την προσθήκη ενός entity στην λίστα με τα entities του εκάστοτε scope
def search_by_name(name)	για την αναζήτηση ενός entity, με βάση το όνομα του, στις λίστες με τα entities

Δημιουργήθηκαν επίσης οι παρακάτω μεταβλητές:

```
par_length = []
framelength = []
nesting_level = 0
record_arguments = []
scopes = []
made_main = False
```

οι οποίες έχουν οριστεί σαν global.

Ενέργειες στον Πίνακα Συμβόλων

Προσθήκη νέου Scope γίνεται όταν ξεκινά η μετάφραση μιας νέας συνάρτησης. Επομένως στην μέθοδο def_function():

-δημιουργούνται:

```
par_length.append(0)
framelength.append(12)
```

```
record_arguments.append([])
```

έπειτα:

```
entinties = []  
current_scope = Scope(entinties, nesting_level)  
add_scope(current_scope)
```

για να μην χρησιμοποιηθούν arguments που δεν ανήκουν στο entity:

```
temp_record_args = record_arguments[-1][:]  
record_arguments.pop()  
record_arguments.append([])
```

στην συνέχεια:

```
temp_args = par_length[-1]
```

και αναλόγως το f_type ίσως αλλάξει:

```
f_type = 1  
if (temp_args == 0):  
    f_type = 0
```

Τέλος δημιουργείται το entity function και γίνονται pop των πεδίων του και διαγραφή του scope του:

```
function_entity=  
Function(function_name, f_type, start_quad, temp_reco  
rd_args, framelength[-1])  
add_entity(function_entity)  
par_length.pop()  
framelength.pop()  
record_arguments.pop()
```



```
f_type = 1
delete_scope()
```

Η δημιουργία Variable συμβαίνει στην actual_param_list():

```
def actual_param_list():
    global token

    ident_var=Variable(token.current_string,calculate_offset()+4)
    add_entity(ident_var)
    e_place = expression()
    genquad('par',e_place,'cv','_')

    while token.current_string == ',':
        token = lex()
        ident_var = Variable(token.current_string,calculate_offset()+4)
        add_entity(ident_var)

    e_place = expression()
    genquad('par',e_place,'cv','_')
```

Στην μέθοδο id_list δημιουργούνται Parameter που αντιστοιχούν στις τυπικές παραμέτρους μιας συνάρτησης, είτε σε δηλώσεις καθολικές και μη. Επίσης δημιουργούνται και Arguments για τον τύπο των τυπικών παραμέτρων της συνάρτησης. Το πιο βασικό κομμάτι στην id_list():

```
if token.family == 'identifier_family'

    par_var = Parameter(token.current_string,"cv",calculate_offset()+4)
    add_entity(par_var)

    if(len(par_length) > 0):
        par_length[-1] += 1
    else:
        par_length.append(0)
        par_length[-1] += 1

    record_arg = Argument("cv")
    if(len(record_arguments) > 0):
        record_arguments[-1].append(record_arg.__str__())
    else:
        record_arguments[0].append(record_arg.__str__())
```

Την δημιουργία TempVariable αναλαμβάνει η newtemp():

```
def newtemp():
    global temp_counter

    return_string = "T_" + str(temp_counter)
    temp_counter += 1

    temp_var = TempVariable(return_string, calculate_offset()+4)
    add_entity(temp_var)

    return return_string
```

Για να μην δημιουργηθεί δύο φορές ο πίνακας συμβόλων για την main, αλλά το scope της να φαίνεται και στις υπόλοιπες συναρτήσεις στην συνάρτηση start_rule() καλείται πρώτα η μέθοδος για την μετάφραση της main. Με αυτόν τον τρόπο επιτυγχάνεται η δημιουργία του πίνακα συμβόλων για αυτήν. Έπειτα ο δείκτης αρχείου επιστρέφει στον πρώτο χαρακτήρα του αρχείου και μετά από αυτό αρχικοποιείται η quad_list ώστε να μην περιέχει quads από την main πριν μεταφραστούν οι υπόλοιπες συναρτήσεις.

```
call_main_part()
inputFile.seek(0)

token = lex()

quad_counter = 0
quad_list = []
```

Αμέσως μετά μεταφράζονται οι συναρτήσεις του προγράμματος, η λίστα με τα scopes “αδειάζει”(για να μην δημιουργηθεί ξανά πίνακας συμβόλων της main) και καλείται πάλι η μέθοδος για μετάφραση της main, όπου αυτήν την φορά θα δημιουργηθούν quads.

```

def_main_part()

framelength = [main_framelength]
temp_scopes = scopes[:]
temp_main_framelength = framelength[0]
delete_scope()
scopes = temp_scopes

call_main_part()
main_framelength = temp_main_framelength
make_final_code()

```

Για να λειτουργήσει αυτός ο μηχανισμός χρησιμοποιείται και η boolean `made_main` μέσα στην `call_main_part()` η οποία ελέγχει αν έχει ήδη δημιουργηθεί ο πίνακας συμβόλων της `main` και αν ισχύει τότε δεν δημιουργεί `scope`.

```

if not made_main:
    add_scope(current_scope)
    made_main = True
else:
    file.write("Main      framelength:      " +
str(main_framelength))

```

ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

<code>search_scope(name)</code>	επιστρέφει το <code>entity</code> και το <code>index</code> του <code>scope</code> στο οποίο βρίσκεται
---------------------------------	--

gnvlcode(name)	μεταφέρει στον t0 την διεύθυνση μιας μη τοπικής μεταβλητής
loadvr(v, r)	μεταφέρει δεδομένα στον καταχωρητή r
store(r, v)	μεταφέρει δεδομένα από τον καταχωρητή r στην μνήμη(μεταβλητή v)
make_final_code()	είναι η μέθοδος που χρησιμοποιείται για την παραγωγή τελικού κώδικα

Συνάρτηση search_scope(name)

Η συγκεκριμένη συνάρτηση παίρνει σαν όρισμα το όνομα ενός entity και προσπελάσει τα scopes ξεκινώντας από το τέλος και επιστρέφει το index του scope, στον πίνακα scopes, με την πρώτη εμφάνιση του entity με αυτό το όνομα.

```
def search_scope(name):
    global scopes

    scope_counter = len(scopes) - 1

    for scope in reversed(scopes):
        for entity in scope.entities:
            if entity.name == name:
                return (entity, scope_counter)
        scope_counter -= 1
```

Συνάρτηση `genvlcode(name)`:

Αρχικά φορτώνεται η διεύθυνση της στοίβας του γονέα(μέσω του συνδέσμου προσπέλασης):

```
lw t0,-4(sp)
```

```
global assembly_file
    global scopes

    assembly_file.write("lw t0,-4(sp)\n")
```

Στην συνέχεια , αν χρειαστεί , θα βρεθεί ο πρόγονος στον οποίο βρίσκεται η μεταβλητή οπότε καλείται η `search_scope` με όρισμα το όνομα της μεταβλητής που αναζητείται:

```
search_result = search_scope(name)
```

Σε περίπτωση που βρεθεί το αντίστοιχο entity:

όσες φορές χρειαστεί:

```
lw t0,-4(t0)
```

```
if search_result is not None:
    level_difference = scopes[len(scopes)-1].nesting_level -
        scopes[search_result[1]].nesting_level

    level_difference -= 1
```

```
for i in range(level_difference):  
    assembly_file.write("lw t0,-4(t0)\n")
```

Ελέγχεται αν το entity είναι μεταβλητή ή παράμετρος και παράγεται ο τελικός κώδικας:

```
addi t0,t0,-offset
```

```
if isinstance(search_result[0],Variable) or  
isinstance(search_result[0],Parameter):  
  
    temp_offset = search_result[0].offset  
    assembly_file.write("addi t0,t0,-"+str(temp_offset)+"\n")
```

Συνάρτηση loavr(v, r):

Η συνάρτηση loadnr μεταφέρει την μεταβλητή στον καταχωρητή r.

Διακρίνονται οι παρακάτω περιπτώσεις με βάση τον τύπο της μεταβλητής:

αν v είναι σταθερά

```
li tr,v
```

```
if v.lstrip('-').isdigit():  
    assembly_file.write("li "+str(r)+", "+str(v) +"\n")
```

αν ν είναι **καθολική μεταβλητή** – δηλαδή ανήκει στο κυρίως πρόγραμμα

lw tr,-offset(gp)

```
if current_scope.nesting_level == 0 and
(instance(search_result[0],Variable) or
instance(search_result[0],TempVariable)):
    assembly_file.write("lw
"+str(r)+",-"+str(search_result[0].offset)+"(gp)\n")
```

αν η ν έχει δηλωθεί στη
συνάρτηση που αυτή τη στιγμή
εκτελείται και είναι **τοπική
μεταβλητή**, ή **τυπική
παράμετρος** που περνάει με
τιμή, ή **προσωρινή μεταβλητή**

lw tr,-offset(sp)

```
elif current_scope.nesting_level ==
scopes[len(scopes)-1].nesting_level and
(instance(search_result[0],Variable) or
instance(search_result[0],TempVariable) or
instance(search_result[0],Parameter)):

assembly_file.write("lw"+str(r)+",-"+str(search_result[0].offset)+
"(sp)\n")
```

αν η ν έχει δηλωθεί σε κάποιο πρόγονο και εκεί είναι **τοπική μεταβλητή**, ή **τυπική
παράμετρος** που περνάει με τιμή

gnlvcode()

lw tr,(t0)

```
elif current_scope.nesting_level <
scopes[len(scopes)-1].nesting_level and
```

```
(isinstance(search_result[0], Variable) or
isinstance(search_result[0], Parameter)):
    gnvocode(v)
    assembly_file.write("lw " + str(r) +
", " + "0(t0)" + "\n")
```

Σε κάθε μία από τις παραπάνω περιπτώσεις (εκτός της πρώτης περίπτωσης που η μεταβλητή είναι σταθερά) χρειάστηκε ο εντοπισμός του scope που βρισκόταν η μεταβλητή *v*, μέσω της κλήσης `search_scopes(v)`.

Συνάρτηση `storerv(r, v)`:

Χρησιμοποιείται για να μεταφέρει δεδομένα από τον καταχωρητή *r* στην μνήμη(μεταβλητή *v*).

Όπως και προηγουμένως διακρίνονται οι παρακάτω περιπτώσεις:

αν *v* είναι καθολική μεταβλητή – δηλαδή ανήκει στο κυρίως πρόγραμμα

```
sw tr,-offset(gp)
```

```
if current_scope.nesting_level == 0 and
(isinstance(search_result[0], Variable) or
isinstance(search_result[0], TempVariable)):
    assembly_file.write("sw
"+str(r)+",-"+str(search_result[0].offset)+"(gp)\n
")
```


αν ν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή

```
sw tr,-offset(sp)
```

```
elif current_scope.nesting_level ==  
scopes[len(scopes)-1].nesting_level and  
(isinstance(search_result[0],Variable) or  
isinstance(search_result[0],TempVariable) or  
isinstance(search_result[0],Parameter)):  
    assembly_file.write("sw  
"+str(r)+",-"+str(search_result[0].offset)+"(sp)\n  
")
```

αν ν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον

```
gnlvcde(v)
```

```
sw tr,(t0)
```

```
elif current_scope.nesting_level <  
scopes[len(scopes)-1].nesting_level and  
(isinstance(search_result[0],Variable) or  
isinstance(search_result[0],Parameter)):  
    gnlvcde(v)  
    assembly_file.write("sw "+str(r)+" "+  
"0(t0)"+"\\n")
```

Αντίστοιχα με την συνάρτηση loadnr σε κάθε μία από τις παραπάνω περιπτώσεις (εκτός της πρώτης περίπτωσης που η μεταβλητή είναι σταθερά) χρειάστηκε ο εντοπισμός

του scope που βρισκόταν η μεταβλητή *v*, μέσω της κλήσης `search_scopes(v)`

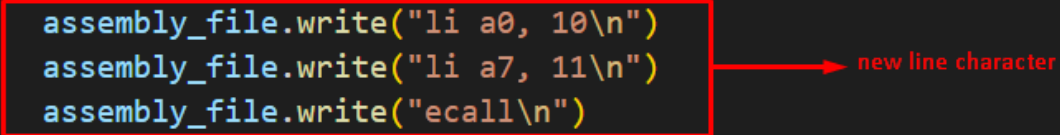
Συνάρτηση `make_final_code()`:

Για την παραγωγή του τελικού κώδικα διαβάζονται οι τετράδες που έχουν παραχθεί μέχρι εκείνη την στιγμή και ανάλογα με το περιεχόμενο της τετράδας προκύπτει ο αντίστοιχος κώδικας σε assembly.

Είσοδος και έξοδος δεδομένων:

```
if quad.quad[1] == 'inp':
    assembly_file.write("li a7,63\n")
    assembly_file.write("ecall\n")
    storerv("a0", quad.quad[2])

elif quad.quad[1] == 'out':
    loadvr(quad.quad[2], "a0")
    assembly_file.write("li a7,1\n")
    assembly_file.write("ecall\n")
    assembly_file.write("li a0, 10\n")
    assembly_file.write("li a7, 11\n")
    assembly_file.write("ecall\n")
```



Για την είσοδο δεδομένων χρησιμοποιείται το system call “li a7, 63” και το αποτέλεσμα τοποθετείται στον register a0.

Για την έξοδο δεδομένων χρησιμοποιείται το system call “li a7, 1” αφού έχουν φορτωθεί στον a0 τα δεδομένα. Στην συνέχεια τυπώνεται new line character, όπως φαίνεται και παραπάνω.

Τερματισμός προγράμματος:

```
elif quad.quad[1] == 'halt':  
    assembly_file.write("li a0,0\n")  
    assembly_file.write("li a7,93\n")  
    assembly_file.write("ecall\n")
```

Όταν βρεθεί η τετράδα που περιέχει το halt, τότε γίνεται έξοδος από το πρόγραμμα με τιμή 0(όπως φαίνεται παραπάνω, η τιμή αυτή φορτώνεται στον register a0)

Εντολές αλμάτων:

```
elif quad.quad[1] == 'jump':  
    assembly_file.write("j label"+str(quad.quad[4])+"\n")
```

Όταν βρεθεί κάποια τετράδα που περιέχει το jump δημιουργείται ο κώδικας assembly που θα μεταφέρει το Program Counter στο αντίστοιχο label.

```

elif quad.quad[1] == ">":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("bgt,t1,t2,label"+str(quad.quad[4])+"\n")

elif quad.quad[1] == ">=":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("bge,t1,t2,label"+str(quad.quad[4])+"\n")

elif quad.quad[1] == "<":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("blt,t1,t2,label"+str(quad.quad[4])+"\n")

elif quad.quad[1] == "<=":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("ble,t1,t2,label"+str(quad.quad[4])+"\n")

elif quad.quad[1] == "!=":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("bne,t1,t2,label"+str(quad.quad[4])+"\n")

elif quad.quad[1] == "==":
    loadvr(quad.quad[2], "t1")
    loadvr(quad.quad[3], "t2")
    assembly_file.write("beq,t1,t2,label"+str(quad.quad[4])+"\n")

```

Όλα τα conditionals (beq: "=", bne: "!=", bgt: ">", blt: "<", bge: ">=", ble: "<=") ακολουθούν παρόμοια λογική. Αν ισχύει η σχέση $t1 \text{ rel_op } t2$ τότε ακολουθείται το branch που δείχνει το label.

Εκχώρηση:

```
elif quad.quad[1] == '=':  
    loadvr(quad.quad[2], "t1")  
    storerv("t1", quad.quad[4])
```

Αρχικά φορτώνεται η τιμή που πρόκειται να εκχωρηθεί στον t1 και στην συνέχεια αποθηκεύεται στην μνήμη(quad.quad[4]).

Εντολές Αριθμητικών Πράξεων:

```
elif quad.quad[1] == "//":  
    loadvr(quad.quad[2], "t1")  
    loadvr(quad.quad[3], "t2")  
    assembly_file.write("div t1,t1,t2"+"\\n")  
    storerv("t1", quad.quad[4])  
  
elif quad.quad[1] == "*":  
    loadvr(quad.quad[2], "t1")  
    loadvr(quad.quad[3], "t2")  
    assembly_file.write("mul t1,t1,t2"+"\\n")  
    storerv("t1", quad.quad[4])  
  
elif quad.quad[1] == "-":  
    loadvr(quad.quad[2], "t1")  
    loadvr(quad.quad[3], "t2")  
    assembly_file.write("sub t1,t1,t2"+"\\n")  
    storerv("t1", quad.quad[4])  
  
elif quad.quad[1] == "+":  
    loadvr(quad.quad[2], "t1")  
    loadvr(quad.quad[3], "t2")  
    assembly_file.write("add t1,t1,t2"+"\\n")  
    storerv("t1", quad.quad[4])  
  
elif quad.quad[1] == "%":  
    loadvr(quad.quad[2], "t1")  
    loadvr(quad.quad[3], "t2")  
    assembly_file.write("rem t1,t1,t2"+"\\n")  
    storerv("t1", quad.quad[4])
```

Όλα τα arithmetic operations (div:“/”, mul:“*” , sub:“-” , add:“+”, rem: “%”) ακολουθούν παρόμοια λογική. Δημιουργείται κώδικας assembly που εκτελεί την αντίστοιχη πράξη χρησιμοποιώντας τους t1 και t2, στους οποίους

έχουν φορτωθεί τα δεδομένα, χρησιμοποιώντας την συνάρτηση `loadvr`.

(Για τον αριθμητικό τελεστή “%” χρησιμοποιήθηκε το `rem`)

Επιστροφή Τιμής Συνάρτησης:

```
elif quad.quad[1] == "retv":  
    loadvr(quad.quad[2], "t1")  
    assembly_file.write("lw t0,-8(sp)\n")  
    assembly_file.write("sw t1,0(t0)\n")  
    assembly_file.write("lw ra,0(sp)\n")  
    assembly_file.write("jr ra\n")
```

Αρχικά φορτώνεται η τιμή επιστροφής στον καταχωρητή `t1`, αποθηκεύεται η διεύθυνση που είναι αποθηκευμένη στην 3η θέση του εγγραφήματος δραστηριοποίησης στον καταχωρητή `t0` και τοποθετείται το περιεχόμενο του `t1` σε αυτή τη διεύθυνση(`t0`). Τελικά αποθηκεύεται η διεύθυνση επιστροφής από την πρώτη θέση του εγγραφήματος δραστηριοποίησης στον καταχωρητή επιστροφής `ra` και γίνεται το κατάλληλο `jump register(jr ra)`.

Παράμετροι Συνάρτησης:

```
parameter_counter = -1  
parameter_f = True  
function_name = ""
```

Όταν βρεθεί μια παράμετρος, αναζητείται η τετράδα call που ακολουθεί την λήξη των τετράδων των παραμέτρων.

```
elif quad.quad[1] == "par":  
  
    if parameter_f == False:  
        temp_index = i  
        temp_quad = quad_list[temp_index]  
  
        while temp_quad.quad[1] != "call":  
            temp_index += 1  
            temp_quad = quad_list[temp_index]  
  
        function_name = temp_quad.quad[2]  
  
        search_result = search_scope(function_name)  
        assembly_file.write("addi s0,sp,"+str(search_result[0].frame_length)+"\n")  
  
    if quad.quad[3] == "cv":  
        parameter_f = True  
  
        if parameter_f == True:  
            parameter_counter += 1  
  
        loadvr(quad.quad[2],"t0")  
        assembly_file.write("sw t0,-"+str(12+4*parameter_counter)+"(s0)\n")  
        continue  
  
    elif quad.quad[3] == "ret":  
        search_result = search_scope(quad.quad[2])  
        assembly_file.write("addi t0, sp, -" + str(search_result[0].offset) + "\n")  
        assembly_file.write("sw t0, -8(s0) \n")
```

Στην περίπτωση που η τετράδα παραμέτρου είναι του τύπου "cv":

Για τον υπολογισμό της θέσης που θα αποθηκευτεί η εκάστοτε παράμετρος χρησιμοποιείται η μεταβλητή

parameter_counter με βάσει τον τύπο “-(12+4*parameter_counter)(s0)”.

Στην περίπτωση που η τετράδα παραμέτρου περιέχει το “ret”:

Γεμίζουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή.

```
parameter_f = False
parameter_counter = -1
function_name = ""
```

Κλήση Συνάρτησης:

```
elif quad.quad[1] == "call":

    search_result = search_scope(quad.quad[2])

    current_scope = scopes[search_result[1]]

    if scopes[len(scopes) - 1].nesting_level == current_scope.nesting_level:
        assembly_file.write("lw t0,-4(sp)\n")
        assembly_file.write("sw t0,-4(s0)\n")
    else:
        assembly_file.write("sw sp,-4(s0)\n")

    assembly_file.write("addi sp,sp,"+str(search_result[0].frame_length)+"\n")
    assembly_file.write("jal"+" label"+str(search_result[0].start_quad)+"\n")
    assembly_file.write("addi sp,sp,-"+str(search_result[0].frame_length)+"\n")
```

Αρχικά, συμπληρώνουμε το δεύτερο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τον σύνδεσμο προσπέλασης, χρησιμοποιώντας τη διεύθυνση του εγγραφήματος δραστηριοποίησης της γονικής συνάρτησης. Έτσι, η

κληθείσα συνάρτηση θα γνωρίζει πού να αναζητήσει μια μεταβλητή που έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει άμεσα. Διακρίνονται δύο περιπτώσεις:

- αν η καλούσα και η κληθείσα συνάρτηση έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα. Η κληθείσα συνάρτηση αντιγράφει την διεύθυνση του γονέα από τον σύνδεσμο προσπέλασης της καλούσας.
- αν η καλούσα και η κληθείσα συνάρτηση έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας. Σε αυτή την περίπτωση αποθηκεύεται στον σύνδεσμο προσπέλασης της κληθείσας η διεύθυνση του γονέα(καλούσα).

Στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα, καλούμε τη συνάρτηση και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα.

Έναρξη και λήξη block:

```
elif quad.quad[1] == 'begin_block':  
  
    if quad.quad[2] == "main":  
        assembly_file.seek(0)  
        assembly_file.write("j label"+str(quad.quad[0])+"\n")  
        assembly_file.seek(0, os.SEEK_END)  
        assembly_file.write("addi sp,sp,"+str(main_framelength)+"\n")  
        assembly_file.write("mv gp,sp\n")  
    else:  
        assembly_file.write("sw ra,0(sp)\n")  
        print("Im here")  
  
elif (quad.quad[1] == "end_block"):  
    if(quad.quad[2] != "main"):  
        assembly_file.write("lw ra,0(sp)\n")  
        assembly_file.write("jr ra\n")
```

Μέσα στην κληθείσα:

Στην αρχή κάθε συνάρτησης αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον ra η jal:

```
sw ra,(sp)
```

Στην περίπτωση που ξεκινά το block της main πρέπει να δεσμευτεί χώρος(κατεβαίνει ο Stack Pointer κατά το framelength της main) και δημιουργείται το label στην πρώτη γραμμή του αρχείου “.asm”. Γίνεται αντιγραφή της τιμής του sp στον gp.

Στο τέλος κάθε συνάρτησης κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον ra. Μέσω του ra επιστρέφουμε στην καλούσα:

```
lw ra,(sp)
```

```
jr ra
```

Τελικά γίνεται εγγραφή των τετράδων στο αρχείο “.int” και έπειτα γίνεται εκκαθάριση της λίστας με τις τετράδες.

Η make_final_code() καλείται στην def_function πριν την διαγραφή του εκάστοτε scope και στην start_rule μετά την παραγωγή του ενδιάμεσου κώδικα της main.

Το αποτέλεσμα του τελικού κώδικα για το αρχείο test.cpy:

Source code Input type: ☒ Assembly ☐ C

718 label146:
719 j label138
720
721 label147:
722 lw a0,-48(gp)
723 li a7,1
724 ecall
725 li a0, 10
726 li a7, 11
727 ecall
728
729 label148:
730 li a0,0
731 li a7,93
732 ecall
733
734 label149:
735

Console

0
1
1
0
1
81
5
1
1
1
0
1
0
1
0
0
0
0
1
0
59

Program exited with code: 0