

ΜΕΤΑΦΡΑΣΤΕΣ

Αναφορά Ελέγχου

Μέλη ομάδας :

Μανίκας Ελευθέριος Μάριος ,4723

Φωτόπουλος Στέφανος ,4829

Ιωάννινα, 24/05/2023

Contents

Περιγραφή της εργασίας	3
Φάση 1	3
Φάση 2	16
Φάση 3	21

Περιγραφή της εργασίας

Φάση 1

Λεκτικός-Συντακτικός Αναλυτής:

Πραγματοποιήσαμε 5 test.cpy για να διαπιστώσουμε την ορθότητα τόσο του λεκτικού όσο και του συντακτικού αναλυτή που δημιουργήσαμε στο test1.cpy(Το οποίο βρίσκεται παρακάτω) έχουμε την συνάρτηση convergence.Με την οποία ελέγχουμε κατά πόσο ο συντακτικός αναλυτής δουλεύει σωστά για if-else,return,print,while,return.

Όπως βλέπουμε παρακάτω γίνεται successful compilation του κώδικα που είναι άλλωστε και το επιθυμητό αποτέλεσμα μιας και το αρχείο .cpy δεν παραβιάζει τους κανόνες που έχει ορίσει ο συντακτικός.

```
1  #$ ELEFThERIOS MARIOs MANIKAS AM:4723 #$
2  #$ STEFANOS FOTOpOULOS AM:4829 #$
3  def convergence():
4  #{
5      #declare x
6      a = 200;
7      b = 33;
8      if (b > a):
9          print(b);
10     else:
11         print(a);
12
13     while(x>1):
14         #{
15             return (0);
16         #}
17
18     return (1);
19  #}
```

```

PS C:\Users\lefte\PycharmProjects\Metafrastes_1> python cutePy_4723_4829.py test1.cpy
def                                family:keyword                                line:3
convergence                        family:id                                    line:3
(                                  family:groupSymbol                               line:3
)                                  family:groupSymbol                               line:3
:                                  family:delimiter                               line:3
#{                                 family:groupSymbol                               line:4
#declare                           family:keyword                                line:5
x                                  family:id                                    line:5
a                                  family:id                                    line:6
=                                  family:assignment                               line:6
200                                family:number                                    line:6
;                                  family:delimiter                               line:6
b                                  family:id                                    line:7
=                                  family:assignment                               line:7
33                                 family:number                                    line:7
;                                  family:delimiter                               line:7
if                                  family:keyword                                line:8
(                                  family:groupSymbol                               line:8
b                                  family:id                                    line:8
>                                  family:relOperator                               line:8
a                                  family:id                                    line:8
)                                  family:groupSymbol                               line:8
:                                  family:delimiter                               line:8
print                              family:keyword                                line:9
(                                  family:groupSymbol                               line:9
b                                  family:id                                    line:9
)                                  family:groupSymbol                               line:9
;                                  family:delimiter                               line:9

else                                family:keyword                                line:10
(                                  family:groupSymbol                               line:18
1                                  family:number                                    line:18
)                                  family:groupSymbol                               line:18
;                                  family:delimiter                               line:18
#}                                 family:groupSymbol                               line:19
End of Verbal Analysis
Compilation Successfull

```

Αφού πρώτα ο λεκτικός μας επέστρεψε κάθε λεκτική μονάδα του προγράμματος μαζί με την οικογένειά της και την γραμμή στην οποία εμφανίζεται. Έπειτα, έχουμε προσθέσει ένα print για να καταλάβουμε πως έχει έρθει το τέλος του λεκτικού και εισαγάμαστε στον συντακτικό. Τέλος το print μας επιβεβαιώνει ότι το αρχείο που δώσαμε σαν Input τήρει όλους τους συντακτικούς κανόνες και λήγει το πρόγραμμα.

Στο test2.cpy γίνεται έλεγχος για άνοιγμα-κλείσιμο συναρτήσεων, if-else και `x = int(input())`; .Ομοίως με το προηγούμενο περνά σωστά τόσο τον λεκτικό όσο και τον συντακτικό αναλυτή τυπώνοντας τα παρακάτω πράγματα στο terminal.

```

PS C:\Users\lefte\PycharmProjects\Metafrastes_1> python cutePy_4723_4829.py test2.cpy
def                                family:keyword                line:3
test_II                            family:id                      line:3
(                                  family:groupSymbol             line:3
)                                  family:groupSymbol             line:3
:                                  family:delimiter               line:3
#{                                 family:groupSymbol             line:4
#declare                           family:keyword                 line:7
number                             family:id                     line:7
,                                  family:delimiter               line:7
grade                             family:id                     line:7
,                                  family:delimiter               line:7
x                                  family:id                     line:7
number                             family:id                     line:8
=                                  family:assignment              line:8
70                                 family:number                 line:8
;                                  family:delimiter               line:8
x                                  family:id                     line:9
=                                  family:assignment              line:9
int                                family:keyword                 line:9
(                                  family:groupSymbol             line:9
input                              family:keyword                 line:9
(                                  family:groupSymbol             line:9
)                                  family:groupSymbol             line:9
)                                  family:groupSymbol             line:9
;                                  family:delimiter               line:9
if                                  family:keyword                 line:10
(                                  family:groupSymbol             line:10
grade                             family:id                     line:15

```

```

)                                family:groupSymbol             line:15
;                                family:delimiter               line:15
#}                               family:groupSymbol             line:16
End of Verbal Analysis
Compilation Successfull

```

Στο test3.cpy βάλαμε τις συναρτήσεις που βρίσκονται στο αρχείο της cutePy με σκοπό να τεστάρουμε την κώδικά μας για ποικίλους συντακτικούς κανόνες όπως while, return, if else , άνοιγμα-κλείσιμο συναρτήσεων τόσο των main όσο και εμφολευμένων, κάλεσμα των συναρτήσεων κ.α

Όπως βλέπουμε και παρακάτω κάθε λεκτική μονάδα έχει αναγνωριστεί και τηρούνται όλοι οι συντακτικοί κανόνες.

def	family:keyword	line:3
main_factorial	family:id	line:3
(family:groupSymbol	line:3
)	family:groupSymbol	line:3
:	family:delimiter	line:3
{	family:groupSymbol	line:4
#declare	family:keyword	line:6
x	family:id	line:6
#declare	family:keyword	line:7
i	family:id	line:7
,	family:delimiter	line:7
fact	family:id	line:7
x	family:id	line:9
=	family:assignment	line:9
int	family:keyword	line:9
(family:groupSymbol	line:9
input	family:keyword	line:9
(family:groupSymbol	line:9
)	family:groupSymbol	line:9
)	family:groupSymbol	line:9
;	family:delimiter	line:9
fact	family:id	line:10
=	family:assignment	line:10
1	family:number	line:10
;	family:delimiter	line:10
i	family:id	line:11
=	family:assignment	line:11
1	family:number	line:11
;	family:delimiter	line:11

while	family:keyword	line:12
(family:groupSymbol	line:12
i	family:id	line:12
<=	family:relOperator	line:12
x	family:id	line:12
)	family:groupSymbol	line:12
:	family:delimiter	line:12
{	family:groupSymbol	line:13
fact	family:id	line:14
=	family:assignment	line:14
fact	family:id	line:14
*	family:mulOperator	line:14
i	family:id	line:14
;	family:delimiter	line:14
i	family:id	line:15
=	family:assignment	line:15
i	family:id	line:15
+	family:addOperator	line:15
1	family:number	line:15
;	family:delimiter	line:15
}	family:groupSymbol	line:16
print	family:keyword	line:17
(family:groupSymbol	line:17
fact	family:id	line:17
)	family:groupSymbol	line:17
;	family:delimiter	line:17
}	family:groupSymbol	line:18
def	family:keyword	line:19

main_fibonacci	family:id	line:19
(family:groupSymbol	line:19
)	family:groupSymbol	line:19
:	family:delimiter	line:19
{	family:groupSymbol	line:20
#declare	family:keyword	line:21
x	family:id	line:21
def	family:keyword	line:22
fibonacci	family:id	line:22
(family:groupSymbol	line:22
x	family:id	line:22
)	family:groupSymbol	line:22
:	family:delimiter	line:22
{	family:groupSymbol	line:23
if	family:keyword	line:24
(family:groupSymbol	line:24
x	family:id	line:24
<=	family:relOperator	line:24
1	family:number	line:24
)	family:groupSymbol	line:24
:	family:delimiter	line:24
return	family:keyword	line:25
(family:groupSymbol	line:25
x	family:id	line:25
)	family:groupSymbol	line:25
;	family:delimiter	line:25
else	family:keyword	line:26
:	family:delimiter	line:26
return	family:keyword	line:27

(family:groupSymbol	line:27
fibonacci	family:id	line:27
(family:groupSymbol	line:27
x	family:id	line:27
-	family:addOperator	line:27
1	family:number	line:27
)	family:groupSymbol	line:27
+	family:addOperator	line:27
fibonacci	family:id	line:27
(family:groupSymbol	line:27
x	family:id	line:27
-	family:addOperator	line:27
2	family:number	line:27
)	family:groupSymbol	line:27
)	family:groupSymbol	line:27
;	family:delimiter	line:27
#}	family:groupSymbol	line:28
x	family:id	line:29
=	family:assignment	line:29
int	family:keyword	line:29
(family:groupSymbol	line:29
input	family:keyword	line:29
(family:groupSymbol	line:29
)	family:groupSymbol	line:29
)	family:groupSymbol	line:29
;	family:delimiter	line:29
print	family:keyword	line:30
(family:groupSymbol	line:30
fibonacci	family:id	line:30

(family:groupSymbol	line:30
x	family:id	line:30
)	family:groupSymbol	line:30
)	family:groupSymbol	line:30
;	family:delimiter	line:30
#}	family:groupSymbol	line:31
def	family:keyword	line:32
main_countdigits	family:id	line:32
(family:groupSymbol	line:32
)	family:groupSymbol	line:32
:	family:delimiter	line:32
#{	family:groupSymbol	line:33
#declare	family:keyword	line:34
x	family:id	line:34
,	family:delimiter	line:34
count	family:id	line:34
x	family:id	line:35
=	family:assignment	line:35
int	family:keyword	line:35
(family:groupSymbol	line:35
input	family:keyword	line:35
(family:groupSymbol	line:35
)	family:groupSymbol	line:35
)	family:groupSymbol	line:35
;	family:delimiter	line:35
count	family:id	line:36
=	family:assignment	line:36
0	family:number	line:36
;	family:delimiter	line:36

while	family:keyword	line:37
(family:groupSymbol	line:37
x	family:id	line:37
>	family:relOperator	line:37
0	family:number	line:37
)	family:groupSymbol	line:37
:	family:delimiter	line:37
{	family:groupSymbol	line:38
x	family:id	line:39
=	family:assignment	line:39
x	family:id	line:39
//	family:mulOperator	line:39
10	family:number	line:39
;	family:delimiter	line:39
count	family:id	line:40
=	family:assignment	line:40
count	family:id	line:40
+	family:addOperator	line:40
1	family:number	line:40
;	family:delimiter	line:40
}	family:groupSymbol	line:41
print	family:keyword	line:42
(family:groupSymbol	line:42
count	family:id	line:42
)	family:groupSymbol	line:42
;	family:delimiter	line:42
}	family:groupSymbol	line:43
def	family:keyword	line:44
main_primes	family:id	line:44

(family:groupSymbol	line:44
)	family:groupSymbol	line:44
:	family:delimiter	line:44
{	family:groupSymbol	line:45
#declare	family:keyword	line:46
i	family:id	line:46
def	family:keyword	line:47
isPrime	family:id	line:47
(family:groupSymbol	line:47
x	family:id	line:47
)	family:groupSymbol	line:47
:	family:delimiter	line:47
{	family:groupSymbol	line:48
#declare	family:keyword	line:49
i	family:id	line:49
def	family:keyword	line:50
divides	family:id	line:50
(family:groupSymbol	line:50
x	family:id	line:50
,	family:delimiter	line:50
y	family:id	line:50
)	family:groupSymbol	line:50
:	family:delimiter	line:50
{	family:groupSymbol	line:51
if	family:keyword	line:52
(family:groupSymbol	line:52
y	family:id	line:52
==	family:relOperator	line:52
(family:groupSymbol	line:52

y	family:id	line:52
//	family:mulOperator	line:52
x	family:id	line:52
)	family:groupSymbol	line:52
*	family:mulOperator	line:52
x	family:id	line:52
)	family:groupSymbol	line:52
:	family:delimiter	line:52
return	family:keyword	line:53
(family:groupSymbol	line:53
1	family:number	line:53
)	family:groupSymbol	line:53
;	family:delimiter	line:53
else	family:keyword	line:54
:	family:delimiter	line:54
return	family:keyword	line:55
(family:groupSymbol	line:55
0	family:number	line:55
)	family:groupSymbol	line:55
;	family:delimiter	line:55
#}	family:groupSymbol	line:56
i	family:id	line:57
=	family:assignment	line:57
2	family:number	line:57
;	family:delimiter	line:57
while	family:keyword	line:58
(family:groupSymbol	line:58
i	family:id	line:58
<	family:relOperator	line:58

x	family:id	line:58
)	family:groupSymbol	line:58
:	family:delimiter	line:58
{	family:groupSymbol	line:59
if	family:keyword	line:60
(family:groupSymbol	line:60
divides	family:id	line:60
(family:groupSymbol	line:60
i	family:id	line:60
,	family:delimiter	line:60
x	family:id	line:60
)	family:groupSymbol	line:60
==	family:relOperator	line:60
1	family:number	line:60
)	family:groupSymbol	line:60
:	family:delimiter	line:60
return	family:keyword	line:61
(family:groupSymbol	line:61
0	family:number	line:61
)	family:groupSymbol	line:61
;	family:delimiter	line:61
i	family:id	line:62
=	family:assignment	line:62
i	family:id	line:62
+	family:addOperator	line:62
1	family:number	line:62
;	family:delimiter	line:62
}	family:groupSymbol	line:63
return	family:keyword	line:64

(family:groupSymbol	line:64
1	family:number	line:64
)	family:groupSymbol	line:64
;	family:delimiter	line:64
#}	family:groupSymbol	line:65
i	family:id	line:67
=	family:assignment	line:67
2	family:number	line:67
;	family:delimiter	line:67
while	family:keyword	line:68
(family:groupSymbol	line:68
i	family:id	line:68
<=	family:relOperator	line:68
30	family:number	line:68
)	family:groupSymbol	line:68
:	family:delimiter	line:68
if	family:keyword	line:69
(family:groupSymbol	line:69
isPrime	family:id	line:69
(family:groupSymbol	line:69
i	family:id	line:69
)	family:groupSymbol	line:69
==	family:relOperator	line:69
1	family:number	line:69
)	family:groupSymbol	line:69
:	family:delimiter	line:69
print	family:keyword	line:70
(family:groupSymbol	line:70
i	family:id	line:70

)	family:groupSymbol	line:70
;	family:delimiter	line:70
i	family:id	line:71
=	family:assignment	line:71
i	family:id	line:71
+	family:addOperator	line:71
main_primes	family:id	line:78
(family:groupSymbol	line:78
)	family:groupSymbol	line:78
;	family:delimiter	line:78
End of Verbal Analysis		
Compilation Succeeded		

Στο test4.cpy έχουμε την συνάρτηση main_primes. Με την οποία ελέγχουμε κατά πόσο ο συντακτικός αναλυτής δουλεύει σωστά για εμφολευμένες

συναρτήσεις, if-else, return, print, διαφορετικά conditions μέσα στην if κ.α.

Όπως βλέπουμε στο τερματικό γίνεται successful compilation του κώδικα που είναι άλλωστε και το επιθυμητό αποτέλεσμα μιας και το αρχείο .cpy δεν παραβιάζει τους κανόνες που έχει ορίσει ο συντακτικός.

Στο test5.cpy έχουμε την συνάρτηση main_fibonacci. Με την οποία ελέγχουμε κατά πόσο ο συντακτικός αναλυτής δουλεύει σωστά για εμφολευμένες συναρτήσεις, if-else, return και print.

Όπως βλέπουμε στο τερματικό γίνεται successful compilation του κώδικα που είναι άλλωστε και το επιθυμητό αποτέλεσμα μιας και το αρχείο .cpy δεν παραβιάζει τους κανόνες που έχει ορίσει ο συντακτικός.

Φάση 2

Ενδιάμεσος -Πίνακας Συμβόλων:

Τα αρχεία που χρησιμοποιήθηκαν για testing σε αυτήν την φάση είναι τα εξής
cp.cpy, small.cpy, ifwhile.cpy, symbol.cpy, mainprimes.cpy.

Το αρχείο ifwhile.cpy παρόμοιο αρχείο με αυτό των σημειώσεων

```
0: jump _ _ main
1: begin_block main _ _
2: = 1 _ a
3: + a b %1
4: < %1 1 6
5: jump _ _ 28
6: < b 5 8
7: jump _ _ 28
8: == t 1 10
9: jump _ _ 12
10: = 2 _ c
11: jump _ _ 17
12: == t 2 14
13: jump _ _ 16
14: = 4 _ c
15: jump _ _ 17
16: = 0 _ c
17: < a 1 19
18: jump _ _ 27
19: == a 2 21
20: jump _ _ 26
21: == b 1 23
22: jump _ _ 25
23: = 2 _ c
24: jump _ _ 21
25: jump _ _ 26
26: jump _ _ 17
27: jump _ _ 3
28: ret c _ _
29: halt _ _ _
30: end_block main _ _

1 def ifWhile():
2   #{
3     #declare c,a,b,t
4     a=1
5     while (a+b<1 and b<5):
6       #{
7         if (t==1):
8           c=2
9         else:
10          if (t==2):
11            c=4
12          else:
13            c=0
14          while (a<1):
15            if (a==2):
16              while(b==1):
17                c=2
18          #}
19      return(c)
20   #}
21   if __name__ == "__main__":
22     #$ call of main functions #$
23     ifWhile()
24
25
26
27
```

0,1: Παρατηρούμε πως υπάρχουν τα labels 0,1 που προκύπτουν στην αρχή κάθε συνάρτησης

2: γίνεται η εκχώρηση του 1 στο α

3: πρόσθεση α με β και δημιουργία newTemp

4-7: τα condition του while και τα αντίστοιχα jump έξω σε περίπτωση που δεν ισχύουν

8-11: έλεγχος του if εάν δεν ισχύει jump στο else ενώ εάν ισχύει jump στο while

12-16: Παρόμοια λογική με το επάνω εάν το if ισχύει γίνεται η εκχώρηση και jump στο while αλλιώς πηγαίνει στο else

17,18: στην περίπτωση που η συνθήκη του while ισχύει συνεχίζει στο if αλλιώς jump στην 27 με σκοπό να ξαναγυρίσουμε στο 1^ο while

19,20: αν δεν ισχύει το if πηγαίνουμε στο 26 για να ξαναγυρίσουμε στο 2^ο while

21-23: εάν η συνθήκη του 3^{ου} while δεν ισχύει μεταφερόμαστε στο 25 που μας κατευθύνει στο 26 δηλαδή στο 2^ο while αλλιώς γίνεται κανονικά η εκχώρηση

28,29,30: βλέπουμε την τετράδα του return το απαραίτητο halt και το end block της main

Αυτό το τεστ έγινε για να παρατηρήσουμε εάν πραγματοποιούνται σωστά τα jumps στον ενδιάμεσο, δεν θα γίνει ανάλυση για τον πίνακα συμβόλων αυτού του τεστ καθώς έχει απλά ένα level

Το αρχείο mainprimes.cry παρόμοιο αρχείο με αυτό των σημειώσεων

<pre>Current Scope: main_primes Level: 0 [i/12] <- [isPrime] Current Scope: isPrime Level: 0 [i/12] <- [isPrime] Level: 1 [x/12/cv] <- [i/16] <- [divides] Current Scope: divides Level: 0 [i/12] <- [isPrime] Level: 1 [x/12/cv] <- [i/16] <- [divides/28/2] Level: 2 [x/12/cv] <- [y/16/cv] <- [%1/20] <- [%2/24] <- Current Scope: isPrime Level: 0 [i/12] <- [isPrime/36/11] Level: 1 [x/12/cv] <- [i/16] <- [divides/28/2] [%3/32] <- Current Scope: main_primes Level: 0 [i/12] <- [isPrime/36/11] [%5/40] <- [main_primes/44/28]</pre>	<pre>1 0: jump _ _ main 2 1: begin_block main_divides _ _ 3 2: // y x %1 4 3: * %1 x %2 5 4: == y %2 6 6 5: jump _ _ 8 7 6: ret y _ _ 8 7: jump _ _ 9 9 8: ret y _ _ 10 9: end_block main_divides _ _ 11 10: begin_block main_isPrime _ _ 12 11: = 2 _ 1 13 12: < i x 14 14 13: jump _ _ 25 15 14: par i cv divides 16 15: par x cv divides 17 16: par %3 ret divides 18 17: call divides _ _ 19 18: == %3 1 20 20 19: jump _ _ 22 21 20: ret i _ _ 22 21: jump _ _ 22 23 22: + i 1 %4 24 23: = %4 _ 1 25 24: jump _ _ 12 26 25: ret i _ _ 27 26: end_block main_isPrime _ _ 28 27: begin_block main _ _ 29 28: = 2 _ 1 30 29: <= i 30 31 31 30: jump _ _ 39 32 31: par i cv isPrime 33 32: par %5 ret isPrime 34 33: call isPrime _ _ 35 34: == %5 1 36 36 35: jump _ _ 38 37 36: out i _ _ 38 37: jump _ _ 38 39 38: jump _ _ 29 40 39: + i 1 %6 41 40: = %6 _ 1 42 41: ret i _ _</pre>	<pre>1 def main_primes(): 2 #{ 3 #declare i 4 def isPrime(x): 5 #{ 6 #declare i 7 def divides(x,y): 8 #{ 9 if (y == (y//x) * x): 10 return (y) 11 else: 12 return (y) 13 } 14 i = 2 15 while (i<x): 16 #{ 17 if (divides(i,x)==1): 18 return (i) 19 i = i + 1 20 } 21 } 22 return (i) 23 } 24 # \$ body of main_primes \$ 25 i = 2 26 while (i<=30): 27 if (isPrime(i)==1): 28 print(i) 29 i = i + 1 30 } 31 } 32 if __name__ == "__main__": 33 # \$ call of main functions \$ 34 main_primes()</pre>
--	---	--

Ανάλυση Ενδιάμεσου:

0,1: Παρατηρούμε πως υπάρχουν τα labels 0,1 που προκύπτουν στην αρχή κάθε συνάρτησης

2-5: Διάρθρωση του x με y και αποθήκευση σε temp έπειτα πολ/σμος x με temp και καταχώρηση του αποτελέσματος σε νέο temp και jump στο else αν δεν ισχύει η συνθήκη

Θα μεταβούμε στην 14 καθώς τα υπόλοιπα είναι κάποια return και begin block που εξηγήσαμε πως δουλεύουν και στο προηγούμενο test

14-17: έχουμε τα ορίσματα i, x της divides που περνιούνται με τιμή(cv) και το αποτέλεσμα της το κρατάμε σε μια νέα temp

Με παρόμοιο τρόπο γίνεται και για την isPrime με την μόνη διαφορά εκεί να βρίσκουμε ένα print που μεταφράζεται σε out στον ενδιαμέσο
Ανάλυση Πίνακα συμβόλων:

Για το scope divides:

Level 2: Έχουμε το x,y που είναι τα ορίσματά της συνάρτησης(με cv) και οι temps που δημιουργήθηκαν μέσα στην συνθήκη του if

Level 1: Έχουμε το x,i από το scope της isPrime με ενημερωμένο πλέον το framelength της divides και την πρώτη quad της συνάρτησης

Level 0: Έχουμε το scope της main_primes

Για το scope isPrime:

Level 1: Έχουμε το x,i από το scope της isPrime με ενημερωμένο πλέον το framelength της divides και την πρώτη quad της συνάρτησης είναι τα ορίσματά της συνάρτησης(με cv) και η temp που κρατά το αποτέλεσμα της divides

Level 0: Έχουμε το scope της main_primes με ενημερωμένο framelength και την πρώτη quad της συνάρτησης isPrime

Για το scope main_primes:

Level 0: Έχουμε το scope της main_primes με ενημερωμένο framelength και την πρώτη quad της συνάρτησης main_primes μαζί με την μεταβλητή που κρατά το αποτέλεσμα της isPrime

Το αρχείο symbol.cry παρόμοιο αρχείο με αυτό των σημειώσεων

```
Current Scope: symbol
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1]

Current Scope: procedure_P1
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1]
Level: 1
[x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11]

Current Scope: function_F11
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1]
Level: 1
[x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11/24/2]
Level: 2
[g/12/cv] <- [a/16] <- [%1/20] <-

Current Scope: procedure_P1
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1]
Level: 1
[x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11/24/2] [function_F12]

Current Scope: function_F12
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1]
Level: 1
[x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11/24/2] [function_F12/20/11]
Level: 2
[x/12/cv] <- [%2/16] <-

Current Scope: procedure_P1
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18]
Level: 1
[x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11/24/2] [function_F12/20/11]

Current Scope: symbol
Level: 0
[a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18] [procedure_P2]

1 0: jump _ _ main
2 1: begin_block main_function_F11 _ _
3 2: = a _ b
4 3: = x _ a
5 4: par x cv function_F11
6 5: par %1 ret function_F11
7 6: call function_F11 _ _
8 7: = %1 _ c
9 8: ret c _ _
10 9: end_block main_function_F11 _ _
11 10: begin_block main_function_F12 _ _
12 11: par x cv function_F11
13 12: par %2 ret function_F11
14 13: call function_F11 _ _
15 14: = %2 _ c
16 15: ret c _ _
17 16: end_block main_function_F12 _ _
18 17: begin_block main_procedure_P1 _ _
19 18: = x _ y
20 19: ret y _ _
21 20: end_block main_procedure_P1 _ _
22 21: begin_block main_procedure_P2 _ _
23 22: = 1 _ y
24 23: ret y _ _
25 24: end_block main_procedure_P2 _ _
26 25: begin_block main _ _
27 26: halt _ _ _
28 27: end_block main _ _
29

1 def symbol():
2 #{
3 #declare a,b,c
4 def procedure_P1(x,y):
5 #{
6 #declare a
7 def function_F11(g):
8 #{
9 #declare a
10 b = a
11 a = x
12 c = function_F11(x)
13 return (c)
14 #}
15 def function_F12(x):
16 #{
17 c = function_F11(x)
18 return (c)
19 #}
20 y = x
21 return(y)
22 #}
23 def procedure_P2(y):
24 #{
25 #declare x
26 y = 1
27 return(y)
28 #}
29 #}
30 if __name__ == "__main__":
31 # $ call of main functions $
32
33 symbol()
```

```
33 Current Scope: procedure_P1
34 Level: 0
35 [a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18]
36 Level: 1
37 [x/12/cv] <- [y/16/cv] <- [a/20] <- [function_F11/24/2] [function_F12/20/11]
38
39 Current Scope: symbol
40 Level: 0
41 [a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18] [procedure_P2]
42
43 Current Scope: procedure_P2
44 Level: 0
45 [a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18] [procedure_P2/20/22]
46 Level: 1
47 [y/12/cv] <- [x/16] <-
48
49 Current Scope: symbol
50 Level: 0
51 [a/12] <- [b/16] <- [c/20] <- [procedure_P1/24/18] [procedure_P2/20/22] [symbol/24/26]
52
53
```

Η δεύτερη φωτογραφία έχει τον υπολειπόμενο πίνακα συμβόλων

Για το scope function_F11:

Level 2: Έχουμε το g από το όρισμα της συνάρτησης, το α από το declare και το temp που

κρατά το αποτέλεσμα στην κλήση της συνάρτησης function_F11

Level 1: Έχουμε το x,y από το όρισμα της συνάρτησης, το α από το declare με ενημερωμένο

πλέον το framelength της function_F11 και την πρώτη quad της συνάρτησης

Level 0: Έχουμε το scope της symbol

Για το scope function_F12:

Level 2: Έχουμε το x από το όρισμα της συνάρτησης, και το temp που

κρατά το αποτέλεσμα στην κλήση της συνάρτησης function_F11

Level 1: Έχουμε το x,y από το όρισμα της συνάρτησης, το α από το declare με ενημερωμένο

πλέον τα framelength των function_F11, function_F12 και τις πρώτες quad των συναρτήσεων

Level 0: Έχουμε το scope της symbol

Για το scope procedure_P1:

Level 1: Έχουμε το x,y από το όρισμα της συνάρτησης, το α από το declare με ενημερωμένο

πλέον τα framelength των function_F11, function_F12 και τις πρώτες quad των συναρτήσεων

Level 0: Έχουμε το scope της symbol με ενημερωμένο πλέον το framelength της

procedure_P1 και την πρώτη quad της συνάρτησης

Για το scope procedure_P2:

Level 1: Έχουμε το y από το όρισμα της συνάρτησης, το x από το declare

Level 0: Έχουμε το scope της symbol με ενημερωμένο πλέον το framelength της

procedure_P2 και την πρώτη quad της συνάρτησης

Για το scope symbol:

Level 0: Έχουμε το scope της symbol με ενημερωμένο πλέον το framelength της

Symbol και την πρώτη quad της συνάρτησης


Το παρών τεστ έγινε με σκοπό να δούμε κατά πόσο φορτώνονται σωστά οι τιμές στον πίνακα συμβόλων όταν στο πρόγραμμα μας έχουμε φωλιασμένες συναρτήσεις και να ελέγξουμε εάν τα level μας είναι τα επιθυμητά.

Το υπολειπόμενο τεστ είναι της small ένα πρόγραμμα παρεμφερές αυτού της symbol δημιουργήθηκε επίσης για τον πίνακα συμβόλων με λειτουργίες τις οποίες καλύψαμε και στο αρχείο symbol.

Φάση 3

Τελικός κώδικας:

Για τα τεστ του τελικού κώδικα είχαμε το `assembly_test.cpy` το οποίο δημιουργεί ένα απλό πρόγραμμα σε assembly όπου τυπώνεται το αποτέλεσμα μια πρόσθεσης στο τερματικό. Δυστυχώς, δεν καταφέραμε να υλοποιήσουμε την εντολή `return` επιτυχώς οπότε ο `riscv` μας πετάει ένα `error`. Ωστόσο εάν αφαιρέσουμε το `return` θα παρατηρήσουμε ότι το πρόγραμμα περνάει το `compilation` και το `expected output` τυπώνεται στο `terminal`.

Το ίδιο πρόβλημα αντιμετωπίσαμε και στα υπόλοιπα αρχεία όπως το `finalCodeExample.cpy` που πάρθηκε από τις σημειώσεις και εδώ παρατηρούμε ότι μέχρι ένα σημείο η `assembly` λειτουργεί όπως θα έπρεπε αλλά όταν συναντήσει την αντίστοιχη εντολή για την `return` θα τυπωθεί το ίδιο σφάλμα. Οι υπόλοιπες εντολές λειτουργούν άψογα αυτό επιβεβαιώνεται αν τρέξουμε το αρχείο με `run one step at a time`  οι πράξεις εκτελούνται και τα αποτελέσματα αυτών αποθηκεύονται στους καταχωρητές που έχουμε ορίσει.