ΜΥΥ802 - Μεταφραστές

Κωνσταντινος Γκιουλης, ΑΜ:4654
Ευάγγελος Μπαλλος, ΑΜ:4739
E-mails: cs04654@uoi.gr
cs04739@uoi.gr

Στο report θα περιγραφεί και θα εξηγηθεί η παραγωγή των αρχείων λεκτικου, ενδιάμεσου, πίνακα συμβόλων και τελικού κώδικα από το τεστ με την χρήση του μεταφραστη καθώς και η ορθότητα τους.

Το τεστ αποτελείται από τα παραδείγματα που βρήκαμε στις σημειώσεις.
Δεν θα περιγραφεί ολο το τεστ αλλά επιλεγμένα κομμάτια του για λογους έκτασης.

Αρχικα θα εξετάσουμε το πρώτο function του τεστ ως προς το κομματι της λεκτικης και συντακτικης ανάλυσης
Η main_factorial() στο test1.cpy:

```
def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact
    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
         fact = fact * i;
         i = i + 1;
    #}
    print(fact);
#}
```

Η main_factorial() μετά την εκτέλεση στο lex.out :

```
recognized string       |  family       |  line number
'def'                   |  idk          |  1
'main_factorial'        |  idk          |  1
'('                     |  group        |  1
')'                     |  group        |  1
':'                     |  delimiter    |  1
'#{'                    |  group        |  2
'#declare'              |  idk          |  4
'x'                     |  idk          |  4
'#declare'              |  idk          |  5
'i'                     |  idk          |  5
','                     |  delimiter    |  5
'fact'                  |  idk          |  5
'x'                     |  idk          |  7
'='                     |  assignment   |  7
'int'                   |  idk          |  7
'('                     |  group        |  7
'input'                 |  idk          |  7
'('                     |  group        |  7
')'                     |  group        |  7
')'                     |  group        |  7
```

```
';'                     |  delimiter    |  7
'fact'                  |  idk          |  8
'='                     |  assignment   |  8
'1'                     |  number       |  8
';'                     |  delimiter    |  8
'i'                     |  idk          |  9
'='                     |  assignment   |  9
'1'                     |  number       |  9
';'                     |  delimiter    |  9
'while'                 |  idk          |  10
'('                     |  group        |  10
'i'                     |  idk          |  10
'<='                    |  relOperator  |  10
'x'                     |  idk          |  10
')'                     |  group        |  10
':'                     |  delimiter    |  10
'#{'                    |  group        |  11
'fact'                  |  idk          |  12
'='                     |  assignment   |  12
'fact'                  |  idk          |  12
'*'                     |  mulOperator  |  12
'i'                     |  idk          |  12
';'                     |  delimiter    |  12
'i'                     |  idk          |  13
'='                     |  assignment   |  13
'i'                     |  idk          |  13
'+'                     |  addOperator  |  13
'1'                     |  number       |  13
';'                     |  delimiter    |  13
'#}'                    |  group        |  14
'print'                 |  idk          |  15
'('                     |  group        |  15
'fact'                  |  idk          |  15
')'                     |  group        |  15
';'                     |  delimiter    |  15
'#}'                    |  group        |  16
```

Παρατηρούμε οτι καθε token αναγνωριζεται σωστά απο τον λεκτικό αναλυτή.(πχ το def αναγνωρίζεται ως ανεξάρτητο token και μπαίνει στη σωστή κατηγορία idk και στη σωστή σειρά 1).

Στη συνέχεια παρατηρούμε πως ο συντακτικός αναλυτής δεν εντοπίζει λάθη, όμως εάν κάνουμε κάποιο σκόπιμο λάθος το εντοπίζει και μας ενημερώνει αντίστοιχα. Εάν δεν υπάρχουν σφάλματα μας ενημερώνει πως πέρασε τον έλεγχο και προχωράει στην παραγωγή ενδιάμεσου κώδικα.

Παρακάτω θα εξετάσουμε την ορθότητα του ενδιάμεσου κώδικα

Η main_factorial() μετά την εκτέλεση στο test1.int :

```
label    operator       op1            op2   op3
0        begin_block    main_factorial         _       _
1        in             x                      _       _
2        =              fact                   _       1
3        =              i                      _       1
4        <=             i                      x       6
5        jump           _                      _       11
6        *              fact                   i       T%0
7        =              fact                   _       T%0
```

```
8       +               i                   1       T%1
9       =               i                   _       T%1
10      jump            _                   _       4
11      out             fact                _       _
12      end_block       main_factorial      _       _
13      begin_block     main                _       _
14      call            main_factorial      _       _
15      halt            _                   _       _
16      end_block       main
```

Παρατηρούμε τα quad begin_block και end_block στην αρχή και τέλος του ενδιάμεσου κώδικα αντίστοιχα. Έγιναν οι παρακάτω μεταφράσεις:

Αρχικός	Ενδιάμεσος
x = int(input());	1, in, x, _, _
fact = 1;	2, =, fact, _, 1
i = 1;	3, =, i, _, 1
while (i<=x):	4, <=, i, x, 6 5, jump, _, _, 11 …while loop… 10, jump, _, _, 4
fact = fact * i;	6, *, fact, i, T%0 7, =, i, _, T%0
i = i + 1;	8, +, i, 1, T%1 9, =, i, _, T%1
print(fact);	11, out, fact, _, _

Παρακάτω θα εξετάσουμε την ορθότητα του πίνακα συμβόλων

Η main_factorial() μετά την εκτέλεση στο test1.scp

```
main_factorial
0 | x/12,    i/16,    fact/20,    T%0/24,    T%1/28
```

Παρατηρούμε πως έχει δεσμευτεί επαρκής μνήμη και με τα σωστά offset.

Παρακάτω θα εξετάσουμε την ορθότητα του τελικού κώδικα

Η main_factorial() μετά την εκτέλεση στο test1.s

```
.data
str_nl: .asciz "\n"
j Lmain
```

```
L0:
#0 begin_block main_factorial _ _
sw ra, 0(sp)

L1:
#1 in x _ _
li a7, 5
ecall
lw t0, -4(sp)
addi t0, t0, -12
sw a7, 0(t0)

L2:
#2 = fact _ 1
lw t0, -4(sp)
addi t0, t0, -20
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -20
sw t0, 0(t0)

L3:
#3 = i _ 1
lw t0, -4(sp)
addi t0, t0, -16
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -16
sw t0, 0(t0)

L4:
#4 <= i x 6
lw t0, -4(sp)
addi t0, t0, -16
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -12
lw t1, 0(t0)
ble t0, t1, 6

L5:
#5 jump _ _ 11
j L11

L6:
#6 * fact i T%0
lw t0, -4(sp)
addi t0, t0, -20
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -16
lw t1, 0(t0)
mul t0, t1, t0
sw t0, -24(sp)
```

```
L7:
#7 = fact _ T%0
lw t0, -4(sp)
addi t0, t0, -20
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -20
sw t0, 0(t0)

L8:
#8 + i 1 T%1
lw t0, -4(sp)
addi t0, t0, -16
lw t0, 0(t0)
li t1, 1
add t0, t1, t0
sw t0, -28(sp)

L9:
#9 = i _ T%1
lw t0, -4(sp)
addi t0, t0, -16
lw t0, 0(t0)
lw t0, -4(sp)
addi t0, t0, -16
sw t0, 0(t0)

L10:
#10 jump _ _ 4
j L4

L11:
#11 out fact _ _
lw t0, -4(sp)
addi t0, t0, -20
lw a0, 0(t0)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L12:
#12 end_block main_factorial _ _
lw ra, 0(sp)
jr ra

Lmain:
sw sp, -4(fp)
addi sp, sp, 32
jal L0
addi sp, sp, -32
li a0, 0
li a7, 93
ecall
```

Παρατηρούμε πως ακολουθεί τους κανόνες μετάφρασης ενδιάμεσου κώδικα σε τελικού.(πχ)
Έγιναν οι παρακάτω μεταφράσεις:

Αρχικός	Ενδιάμεσος	Τελικός

x = int(input());	1, in, x, _, _	li a7, 5
		ecall
		lw t0, -4(sp)
		addi t0, t0, -12
		sw a7, 0(t0)

fact = 1;	2, =, fact, _, 1	lw t0, -4(sp)
		addi t0, t0, -20
		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -20
		sw t0, 0(t0)

i = 1;	3, =, i, _, 1	lw t0, -4(sp)
		addi t0, t0, -16
		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -16
		sw t0, 0(t0)

while (i<=x):	4, <=, i, x, 6	lw t0, -4(sp)
		addi t0, t0, -16
		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -12
		lw t1, 0(t0)
		ble t0, t1, 6

	5, jump, _, _, 11	j L11
	…while loop…	
	10, jump, _, _, 4	j L4

fact = fact * i;	6, *, fact, i, T%0	lw t0, -4(sp)
		addi t0, t0, -20
		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -16
		lw t1, 0(t0)
		mul t0, t1, t0
		sw t0, -24(sp)

	7, =, i, _, T%0	lw t0, -4(sp) addi t0, t0, -20

		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -20
		sw t0, 0(t0)

i = i + 1;	8, +, i, 1, T%1	lw t0, -4(sp)
		addi t0, t0, -16
		lw t0, 0(t0)
		li t1, 1
		add t0, t1, t0
		sw t0, -28(sp)

	9, =, i, _, T%1	lw t0, -4(sp)
		addi t0, t0, -16
		lw t0, 0(t0)
		lw t0, -4(sp)
		addi t0, t0, -16
		sw t0, 0(t0)

print(fact);	11, out, fact, _, _	lw t0, -4(sp)
		addi t0, t0, -20
		lw a0, 0(t0)
		li a7, 1
		ecall
		la a0, str_nl
		li a7, 4
		ecall

Ο κώδικας που παράγεται **δεν** είναι σωστός. Δυστυχώς δε καταφέραμε να βρούμε το σφάλμα μας, ώστε να το διορθώσουμε. Επί το πλείστον φαίνεται σωστός ο τελικός κώδικας, όμως δεν εκτελείται σωστα.

Παρακάτω θα παρουσιάσουμε ένα ακόμα παράδειγμα για τον πίνακα συμβόλων, ώστε να καταστήσουμε σαφή την ορθότητα του:

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

```
        #}
        return (1);
    #}
    #$ body of main_primes #$
    i = 2;
    while (i<=30):
    #{
        if (isPrime(i)==1):
            print(i);
        i = i + 1;
    #}
#}
```

Στον παραπάνω κώδικα, κατά τη μετάφραση της divides προκύπτει το εξής στιγμιότυπο του πίνακα συμβόλων:

```
divides
2 | x/12,    y/16,    T%10/20,    T%11/24
1 | x/12,    ibasic/16,    divides<x,y>
0 | i/12,    isPrime<x>
```