ΜΥΥ802 Μεταφραστές

Προγραμματιστική εργασία εξαμήνου

Κατασκευή λειτουργικού μεταγλωττιστή για την εκπαιδευτική γλώσσα προγραμματισμού *cpy*

Μέλη ομάδας

Κωνσταντίνος Χριστοδούλου – 3367

Αλέξανδρος Γεωργαλλή – 5135

Λεκτική ανάλυση

Το πρώτο στάδιο της υλοποίησης του μεταγλωττιστή είναι ο λεκτικός αναλυτής. Η συνάρτηση που υλοποιεί τον λεκτικό αναλυτή του μεταγλωττιστή μας είναι η ***next_token(file_name)***. Με όρισμα το όνομα ενός αρχείου της cpy (**.cpy**) παράγει τις λεκτικές μονάδες (tokens). Υλοποιεί το παρακάτω αυτόματο:

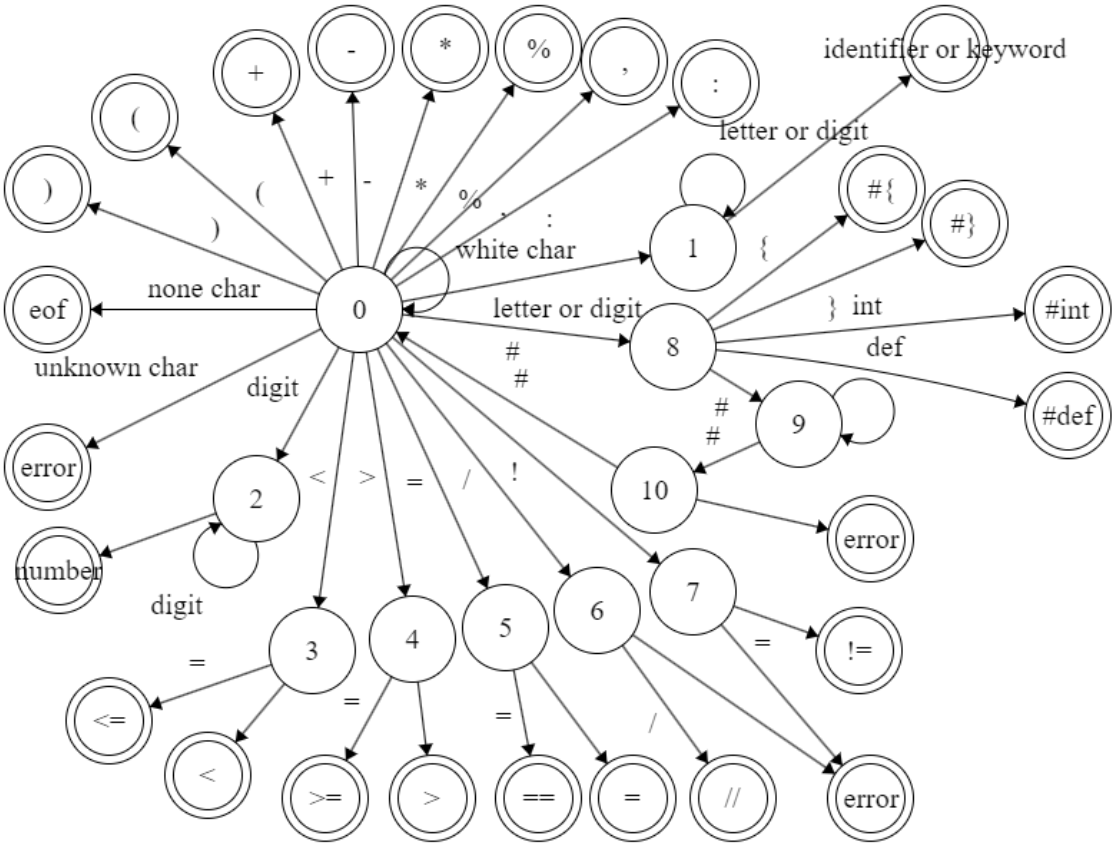


Figure 1: Το αυτόματο καταστάσεων του λεκτικού αναλυτή

Συντακτική ανάλυση

Το δεύτερο στάδιο της υλοποίησης του μεταγλωττιστή είναι ο συντακτικός αναλυτής.

Ακολουθεί η **γραμματική** της **cpy**, η οποία δίνει και την ακριβή περιγραφή της γλώσσας:

```
program         :   declarations functions main
                ;

declarations    :   ( '#int' id_list )*
                ;

id_list         :   ID ( ',' ID )*
                ;

glob_decl       :   ( 'global' id_list )*
                ;

functions       :   function*
                ;

function        :   'def' ID '(' id_list ')' ':'
                    '#{'
                        declarations
                        functions
                        glob_decl
                        code_block
                    '#}'
                ;

main            :   '#def' 'main'
                        declarations
                        code_block
                ;

code_block      :   statement+
                ;

statement       :   assignment_stat
                |   print_stat
                |   return_stat
                |   if_stat
                |   while_stat
                ;

assignment_stat :   ID '=' ( expression
                           | 'int' '(' 'input' '(' ')' ')'
                           )
                ;


print_stat      : 'print' '(' expression ')'
                ;

return_stat     : 'return' expression
                ;

statement_or_block
                : statement
                | '#{' statement+ '#}'
                ;

if_stat         :   'if' condition ':'
```

```
                    statement_or_block
                ('elif' condition':'
                    statement_or_block )*
                (   'else' ':'
                    statement_or_block )?
            ;

while_stat      :   'while' condition ':'
                    statement_or_block
                   ;

expression      :   optional_sign term ( add_oper term )*
                ;

term            :   factor ( mul_oper factor )*
                ;

factor          :   INTEGER
                |   '(' expression ')'
                |   ID idtail
                ;

idtail          :   '(' actual_par_list ')'
                |
                ;


actual_par_list :   expression ( ',' expression )*
                |
                ;

condition       :   bool_term ( 'or' bool_term )*
                ;

bool_term       :   bool_factor ( 'and' bool_factor )*
                ;

bool_factor     :   expression rel_op expression
                |   'not' expression rel_op expression
                ;

optional_sign   :   add_oper
                |
                ;

add_oper        :   '+' | '-'
                ;

mul_oper        :   '*' | '//' | '%'
                ;

rel_op          :   '>' | '<' | '>=' | '<=' | '==' | '!='
                ;
```

Η συνάρτηση που τον υλοποιεί είναι η **syntax_analyzer(tokens)**, η οποία παίρνει ως είσοδο την έξοδο του λεκτικού αναλυτή. Στην υλοποίησή μας μια λίστα L από υπολίστες καθεμία από τις οποίες είναι ένα token. Κάθε κανόνας της γραμματικής είναι μια συνάρτηση στην syntax_analyzer προκειμένου να ελεγχθεί το κατά πόσο το πρόγραμμα είναι συντακτικό ορθό και επακόλουθα το κατά πόσο ανήκει στη γλώσσα

της cpy. Σε περίπτωση που υπάρχουν λάθη, εμφανίζει ο μεταγλωττιστής τα ανάλογα μηνύματα λάθους και τερματίζεται η μεταγλώττιση με sys.exit().

Παραγωγή ενδιάμεσου κώδικα

Το ενδεικτικό πρόγραμμα test.cpy που μας δόθηκε:

```
#int counterFunctionCalls


def max3(x,y,z):
#{
   #int m
   global counterFunctionCalls
   counterFunctionCalls = counterFunctionCalls + 1
   if x>y and x>z:
      m = x
   elif y>x and y>z:
      m = y
   else:
      m = z
   return m
#}



def fib(x):
#{
   global counterFunctionCalls
   counterFunctionCalls = counterFunctionCalls + 1
   if x<0:
      return -1
   elif x==0 or x==1:
      return 1
   else:
```

```
        return fib(x-1)+fib(x-2)
#}


def isPrime(x):
#{
    ## declarations for isPrime ##
    #int i

    def divides(x,y):
    #{
        ## body of divides ##
        global counterFunctionCalls
        counterFunctionCalls = counterFunctionCalls + 1
        if y == (y//x)*x:
            return 1
        else:
            return 0
    #}

    ## body of isPrime ##
    global counterFunctionCalls
    counterFunctionCalls = counterFunctionCalls + 1
    i = 2
    while i < x:
    #{
        if divides(i,x) == 1:
            return 0
        i = i + 1
    #}
    return 1
#}
```

```
def quad(x):
#{
   #int y

   ## nested function sqr ##
   def sqr(x):
   #{
      ## body of sqr ##
      global counterFunctionCalls
      counterFunctionCalls = counterFunctionCalls + 1
      return x*x
   #}

   ## body of quad ##
   global counterFunctionCalls
   counterFunctionCalls = counterFunctionCalls + 1
   y = sqr(x)*sqr(x)
   return y
#}


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
```

```
#}



#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)
```

Το παραγόμενο από αυτό .int αρχείο ενδιάμεσου κώδικα:

100: begin_block, program, _, _

101: begin_block, max3, _, _

102: +, counterFunctionCalls, 1, T_1

103: =, T_1, _, counterFunctionCalls

104: >, x, y, 106

105: jump, _, _, 110

106: >, x, z, 108

107: jump, _, _, 110

108: =, x, _, m

109: jump, _, _, 117

110: >, y, x, 112

111: jump, _, _, 116

112: >, y, z, 114

113: jump, _, _, 116

114: =, y, _, m

115: jump, _, _, 117

116: =, z, _, m

117: retv, m, _, _

118: end_block, max3, _, _

119: begin_block, fib, _, _

120: +, counterFunctionCalls, 1, T_2

121: =, T_2, _, counterFunctionCalls

122: <, x, 0, 124

123: jump, _, _, 127

124: -, 0, 1, T_3

125: retv, T_3, _, _

126: jump, _, _, 143

127: ==, x, 0, 131

128: jump, _, _, 129

129: ==, x, 1, 131

130: jump, _, _, 133

131: retv, 1, _, _

132: jump, _, _, 143

133: -, x, 1, T_5

134: par, T_5, CV, _

135: par, T_4, RET, _

136: call, fib, _, _

137: -, x, 2, T_7

138: par, T_7, CV, _

139: par, T_6, RET, _

140: call, fib, _, _

141: +, T_4, T_6, T_8

142: retv, T_8, _, _

143: end_block, fib, _, _

144: begin_block, divides, _, _

145: +, counterFunctionCalls, 1, T_9

146: =, T_9, _, counterFunctionCalls

147: //, y, x, T_10

148: *, T_10, x, T_11

149: ==, y, T_11, 151

150: jump, _, _, 153

151: retv, 1, _, _

152: jump, _, _, 154

153: retv, 0, _, _

154: end_block, divides, _, _

155: begin_block, isPrime, _, _

156: +, counterFunctionCalls, 1, T_12

157: =, T_12, _, counterFunctionCalls

158: =, 2, _, i

159: <, i, x, 161

160: jump, _, _, 172

161: par, i, CV, _

162: par, x, CV, _

163: par, T_13, RET, _

164: call, divides, _, _

165: ==, T_13, 1, 167

166: jump, _, _, 169

167: retv, 0, _, _

168: jump, _, _, 169

169: +, i, 1, T_14

170: =, T_14, _, i

171: jump, _, _, 159

172: retv, 1, _, _

173: end_block, isPrime, _, _

174: begin_block, sqr, _, _

175: +, counterFunctionCalls, 1, T_15

176: =, T_15, _, counterFunctionCalls

177: *, x, x, T_16

178: retv, T_16, _, _

179: end_block, sqr, _, _

180: begin_block, quad, _, _

181: +, counterFunctionCalls, 1, T_17

182: =, T_17, _, counterFunctionCalls

183: par, x, CV, _

184: par, T_18, RET, _

185: call, sqr, _, _

186: par, x, CV, _

187: par, T_19, RET, _

188: call, sqr, _, _

189: *, T_18, T_19, T_20

190: =, T_20, _, y

191: retv, y, _, _

192: end_block, quad, _, _

193: begin_block, leap, _, _

194: +, counterFunctionCalls, 1, T_21

195: =, T_21, _, counterFunctionCalls

196: %, year, 4, T_22

197: ==, T_22, 0, 199

198: jump, _, _, 202

199: %, year, 100, T_23

200: !=, T_23, 0, 205

201: jump, _, _, 202

202: %, year, 400, T_24

203: ==, T_24, 0, 205

204: jump, _, _, 207

205: retv, 1, _, _

206: jump, _, _, 208

207: retv, 0, _, _

208: end_block, leap, _, _

209: begin_block, main, _, _

210: =, 0, _, counterFunctionCalls

211: inp, T_25, _, _

212: =, T_25, _, i

213: out, i, _, _

214: =, 1600, _, i

215: <=, i, 2000, 217

216: jump, _, _, 224

217: par, i, CV, _

218: par, T_26, RET, _

219: call, leap, _, _

220: out, T_26, _, _

221: +, i, 400, T_27

222: =, T_27, _, i

223: jump, _, _, 215

224: par, 2023, CV, _

225: par, T_28, RET, _

226: call, leap, _, _

227: out, T_28, _, _

228: par, 2024, CV, _

229: par, T_29, RET, _

230: call, leap, _, _

231: out, T_29, _, _

232: par, 3, CV, _

233: par, T_30, RET, _

234: call, quad, _, _

235: out, T_30, _, _

236: par, 5, CV, _

237: par, T_31, RET, _

238: call, fib, _, _

239: out, T_31, _, _

240: =, 1, _, i

241: <=, i, 12, 243

242: jump, _, _, 250

243: par, i, CV, _

244: par, T_32, RET, _

245: call, isPrime, _, _

246: out, T_32, _, _

247: +, i, 1, T_33

248: =, T_33, _, i

249: jump, _, _, 241

250: out, counterFunctionCalls, _, _

251: halt, _, _, _

252: end_block, main, _, _

253: end_block, program, _, _

Παραγωγή τελικού κώδικα

Η παραγωγή του τελικού κώδικα είναι το τελευταίο στάδιο της υλοποίησης του μεταγλωττιστή και αφορά την υλοποίηση του οπίσθιου τμήματος του μεταγλωττιστή μας. Υλοποιήσαμε τις βοηθητικές συναρτήσεις gnlvcode(v), που μεταφέρει στον καταχωρητή t0 τη διεύθυνση μιας μη τοπικής μεταβλητής, loadvr(v, reg) που

φορτώνει τη μεταβλητή v στον καταχωρητή reg και storerv(reg, v) που αποθηκεύει τα δεδομένα ενός καταχωρητή σε μια μεταβλητή v. Επίσης, υλοποιήσαμε τη gfnlcode(quad) από το generate final code, η οποία με όρισμα ένα quad από την γλώσσα της ενδιάμεσης αναπαράστασης παράγει τις αντίστοιχες γραμμές τελικού κώδικα για την αρχιτεκτονική RISC-V.

Ενδεικτικό αρχείο εισόδου = sqr.cpy

```
def sqr(x):
#{
   return x*x
#}

def division(x,y):
#{
   return x // y
#}

def fib(x):
#{

   if x<=1:
      return x
   else:
      return fib(x-1)+fib(x-2)
#}

#def main
#int i, j, k
i = int(input())
j = int(input())
print(i)
print(j)
k = i + j
```

```
print(k)
w = 1600
while w<=2000:
#{
    w = w + 100
    print(w)
#}

print(sqr(i))
print(sqr(j))
print(division(12,5))
print(division(24,6))
print(division(11,3))
print(fib(6))
print(fib(7))
print(fib(8))
```

Το παραγόμενο αρχείο .asm από αυτό:

```
.data
str_nl: .asciz "\n"
.text

j L_125

L_100:
sw ra, 0(sp)

L_101:
sw ra, 0(sp)

L_102:
```

```
        lw t1, -12(sp)
        lw t2, -12(sp)
mul t1, t1, t2
        sw t1, -16(sp)

L_103:
        lw t1, -16(sp)
lw t0, -8(sp)
sw t1, 0(t0)
lw ra, 0(sp)
jr ra

L_104:
lw ra, 0(sp)
jr ra

L_105:
sw ra, 0(sp)

L_106:
        lw t1, -12(sp)
        lw t2, -16(sp)
div t1, t1, t2
        sw t1, -20(sp)

L_107:
        lw t1, -20(sp)
lw t0, -8(sp)
sw t1, 0(t0)
lw ra, 0(sp)
jr ra
```

```
L_108:
lw ra, 0(sp)
jr ra

L_109:
sw ra, 0(sp)

L_110:
        lw t1, -12(sp)
        li t2, 1
ble t1, t2, L_112
L_111:
j L_114
L_112:
        lw t1, -12(sp)
lw t0, -8(sp)
sw t1, 0(t0)
lw ra, 0(sp)
jr ra

L_113:
j L_124
L_114:
        lw t1, -12(sp)
        li t2, 1
sub t1, t1, t2
        sw t1, -20(sp)

L_115:
addi fp, sp, 36
        lw t0, -20(sp)
sw t0, -12(fp)
```

```
L_116:
addi t0, sp, -16
sw t0, -8(fp)

L_117:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 36
jal L_109
addi sp, sp, -36

L_118:
        lw t1, -12(sp)
        li t2, 2
sub t1, t1, t2
        sw t1, -28(sp)

L_119:
addi fp, sp, 36
        lw t0, -28(sp)
sw t0, -12(fp)

L_120:
addi t0, sp, -24
sw t0, -8(fp)

L_121:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 36
jal L_109
```

```
addi sp, sp, -36

L_122:
        lw t1, -16(sp)
        lw t2, -24(sp)
add t1, t1, t2
        sw t1, -32(sp)

L_123:
        lw t1, -32(sp)
lw t0, -8(sp)
sw t1, 0(t0)
lw ra, 0(sp)
jr ra

L_124:
lw ra, 0(sp)
jr ra

L_125:
addi sp, sp, 80
mv gp, sp

L_126:
li a7, 5
ecall
        sw a0, -24(sp)

L_127:
        lw t1, -24(sp)
        sw t1, -12(sp)
```

```
L_128:
li a7, 5
ecall
        sw a0, -28(sp)

L_129:
        lw t1, -28(sp)
        sw t1, -16(sp)

L_130:
        lw a0, -12(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_131:
        lw a0, -16(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_132:
        lw t1, -12(sp)
        lw t2, -16(sp)
add t1, t1, t2
        sw t1, -36(sp)

L_133:
```

```
        lw t1, -36(sp)
        sw t1, -20(sp)

L_134:
        lw a0, -20(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_135:
        li t1, 1600
        sw t1, -40(sp)

L_136:
        lw t1, -40(sp)
        li t2, 2000
ble t1, t2, L_138
L_137:
j L_142
L_138:
        lw t1, -40(sp)
        li t2, 100
add t1, t1, t2
        sw t1, -44(sp)

L_139:
        lw t1, -44(sp)
        sw t1, -40(sp)

L_140:
```

```
        lw a0, -40(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_141:
j L_136
L_142:
addi fp, sp, 20
        lw t0, -12(sp)
sw t0, -12(fp)

L_143:
addi t0, sp, -48
sw t0, -8(fp)

L_144:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 20
jal L_101
addi sp, sp, -20

L_145:
        lw a0, -48(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall
```

```
L_146:
addi fp, sp, 20
        lw t0, -16(sp)
sw t0, -12(fp)

L_147:
addi t0, sp, -52
sw t0, -8(fp)

L_148:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 20
jal L_101
addi sp, sp, -20

L_149:
        lw a0, -52(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_150:
addi fp, sp, 24
        li t0, 12
sw t0, -12(fp)

L_151:
        li t0, 5
```

```
sw t0, -16(fp)

L_152:
addi t0, sp, -56
sw t0, -8(fp)

L_153:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 24
jal L_105
addi sp, sp, -24

L_154:
        lw a0, -56(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_155:
addi fp, sp, 24
        li t0, 24
sw t0, -12(fp)

L_156:
        li t0, 6
sw t0, -16(fp)

L_157:
addi t0, sp, -60
```

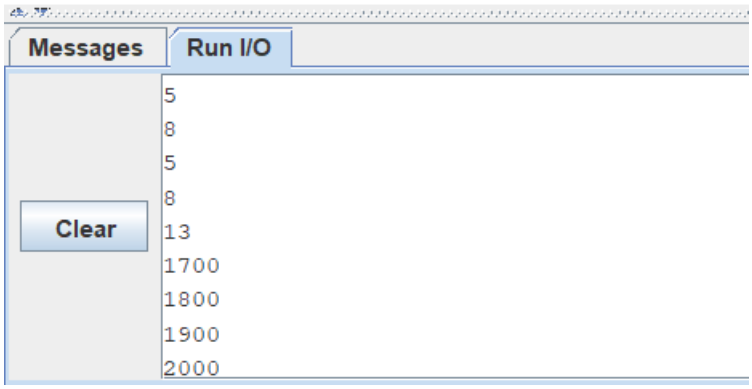
```
sw t0, -8(fp)

L_158:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 24
jal L_105
addi sp, sp, -24

L_159:
        lw a0, -60(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_160:
addi fp, sp, 24
        li t0, 11
sw t0, -12(fp)

L_161:
        li t0, 3
sw t0, -16(fp)

L_162:
addi t0, sp, -64
sw t0, -8(fp)

L_163:
lw t0, -4(fp)
```

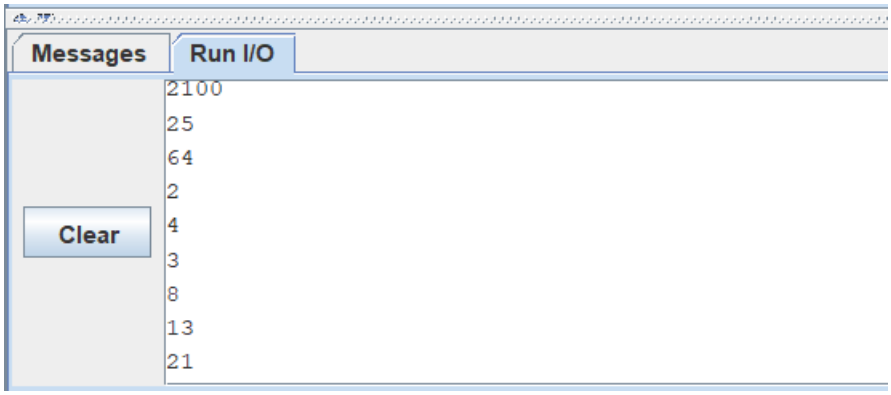
```
sw t0, -4(sp)
addi sp, sp, 24
jal L_105
addi sp, sp, -24

L_164:
        lw a0, -64(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_165:
addi fp, sp, 36
        li t0, 6
sw t0, -12(fp)

L_166:
addi t0, sp, -68
sw t0, -8(fp)

L_167:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 36
jal L_109
addi sp, sp, -36

L_168:
        lw a0, -68(sp)
li a7, 1
```

```
ecall
la a0, str_nl
li a7, 4
ecall

L_169:
addi fp, sp, 36
        li t0, 7
sw t0, -12(fp)

L_170:
addi t0, sp, -72
sw t0, -8(fp)

L_171:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 36
jal L_109
addi sp, sp, -36

L_172:
        lw a0, -72(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_173:
addi fp, sp, 36
        li t0, 8
```

```
sw t0, -12(fp)

L_174:
addi t0, sp, -76
sw t0, -8(fp)

L_175:
lw t0, -4(fp)
sw t0, -4(sp)
addi sp, sp, 36
jal L_109
addi sp, sp, -36

L_176:
        lw a0, -76(sp)
li a7, 1
ecall
la a0, str_nl
li a7, 4
ecall

L_177:
li a0, 0
li a7, 93
ecall
L_178:
lw ra, 0(sp)
jr ra

L_179:
lw ra, 0(sp)
jr ra
```

Τρέξιμο στον RARS:





Τρέξιμο στην Python:

```
5
8
5
8
13
1700
1800
1900
2000
2100
25
64
2
4
3
8
13
21
>>>
```