

Proyecto Final Compiladores



Ruben Cuadra

21 de Noviembre de 2018

Compilador de C-

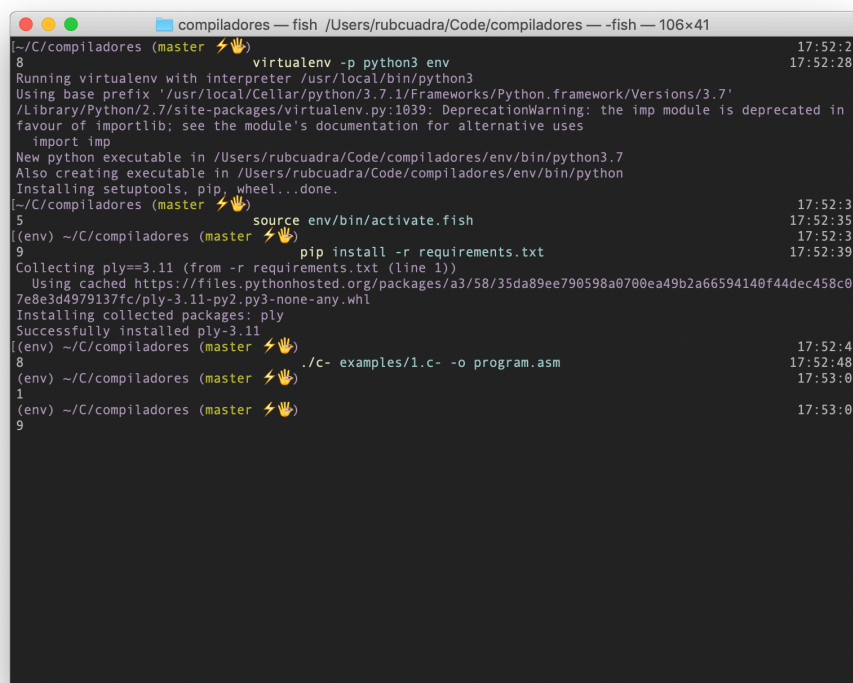
Introducción

Como objetivo de la materia fue requerido crear un compilador en **Python** para código escrito en **C-**, la sintaxis de este lenguaje se puede consultar en la sección apéndices, el compilador creado genera código ensamblador entendible para una maquina con el set de instrucciones de la arquitectura **MIPS I** (32 bits). El código generado se puede correr en una máquina con estas características o bien, en algún simulador, se recomienda usar QtSpim 9.1.20 (Simulador de procesador MIPS R3000) por su compatibilidad con diferentes entornos, buena documentación, su uso fácil y por ser software abierto.

Uso

Para su correcto funcionamiento es necesario tener instalado *Python* ≥ 3.6 y la librería de *ply*

La forma recomendada de ejecución seria creando un entorno virtual, instalando las librerías de requirements.txt y ejecutando la linea **./c- {archivo_a_compilar.c} -o {ejecutable.asm}**. En caso de no usarse un entorno virtual ejecutar con python3 una vez que se hayan instalado las librerías necesarias



```
compiladores — fish /Users/rubcuadra/Code/compiladores — fish — 106x41
[~/C/compiladores (master 🚀)] virtualenv -p python3 env
Running virtualenv with interpreter /usr/local/bin/python3
Using base prefix '/usr/local/Cellar/python/3.7.1/Frameworks/Python.framework/Versions/3.7'
/Library/Python/2.7/site-packages/virtualenv.py:1039: DeprecationWarning: the imp module is deprecated in
favour of importlib; see the module's documentation for alternative uses
  import imp
New python executable in /Users/rubcuadra/Code/compiladores/env/bin/python3.7
Also creating executable in /Users/rubcuadra/Code/compiladores/env/bin/python
Installing setuptools, pip, wheel...done.
[~/C/compiladores (master 🚀)] source env/bin/activate.fish
(env) ~/C/compiladores (master 🚀) pip install -r requirements.txt
Collecting ply==3.11 (from -r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/a3/58/35da89ee790598a070ea49b2a66594140f44dec458c0
7e8e3d4d979137fc/ply-3.11-py2.py3-none-any.whl
Installing collected packages: ply
Successfully installed ply-3.11
(env) ~/C/compiladores (master 🚀) ./c- examples/1.c- -o program.asm
1
(env) ~/C/compiladores (master 🚀)
```

Fig 1: Compilar archivos

En la figura 2 podemos observar el código en C-

```

1  int fibonacci[10];
2
3  int next(int x){
4      return x+1;
5  }
6
7  void main(void)
8  {
9      int l; int r; int i;
10
11     /*Calculate it*/
12     fibonacci[0] = 0; fibonacci[1] = 1;
13     i = 2;
14     while(i < 10){
15         r = fibonacci[i-2];
16         l = fibonacci[i-1];
17         fibonacci[i] = l+r;
18         i = next(i);
19     }
20
21     /*Print it*/
22     i = 0;
23     while(i < 10){
24         output(fibonacci[i]);
25         i = next(i);
26     }
27 }

```

Fig 2: examples/1.c-

El código imprime los primeros 10 números en la serie de Fibonacci. Para ejecutar el programa.asm usaremos **QtSpim**. Cargaremos el archivo .asm y le daremos al botón de play, nos debe imprimir en la consola como se muestra en las imágenes. En la figura 3 podemos observar el código ya compilado listo para ser ejecutado por el procesador.

Ahora lo correremos en el simulador tal y como lo muestran las figuras 4 y 5.

Fig 3: program.asm

```

1  .text
2  .globl main
3  next:
4      lw $a0, 4($sp)
5      sw $a0, 0($sp)
6      addi $sp, $sp, -4
7      li $a0, 1
8      lw $t1, 4($sp)
9      add $a0, $t1, $a0
10     addi $sp, $sp, 4
11     addi $sp, $sp, 0
12     jr $ra
13 main:
14     li $s0, 0
15     sw $s0, 0($sp)
16     addi $sp, $sp, -4
17     li $s0, 0
18     sw $s0, 0($sp)
19     addi $sp, $sp, -4
20     li $s0, 0
21     sw $s0, 0($sp)
22     addi $sp, $sp, -4
23     li $a0, 0
24     move $t5, $a0
25     li $a0, 0
26     li $t1, 4
27     mult $a0, $t1
28     mflo $a0
29     la $a1, fibonacci
30     add $a1, $a1, $a0
31     sw $t5, 0($a1)
32     li $a0, 1
33     move $t5, $a0
34     li $a0, 1
35     li $t1, 4
36     mult $a0, $t1
37     mflo $a0
38     la $a1, fibonacci
39     add $a1, $a1, $a0
40     sw $t5, 0($a1)
41     li $a0, 2
42     sw $a0, 4($sp)
43 stwhile1:
44     lw $a0, 4($sp)
45     sw $a0, 0($sp)
46     addi $sp, $sp, -4
47     li $a0, 10
48     lw $t1, 4($sp)
49     addi $sp, $sp, 4
50     blt $t1, $a0, ifwhile1
51     j endwhile1
52 ifwhile1:
53     lw $a0, 4($sp)
54     sw $a0, 0($sp)
55     addi $sp, $sp, -4
56     li $a0, 2
57     lw $t1, 4($sp)
58     sub $a0, $t1, $a0
59     addi $sp, $sp, 4
60     li $t1, 4
61     mult $a0, $t1
62     mflo $a0
63     la $a1, fibonacci
64     add $a1, $a1, $a0
65     lw $a0, 0($a1)
66     sw $a0, 0($sp)
67     lw $a0, 4($sp)
68     sw $a0, 0($sp)
69     addi $sp, $sp, -4
70     li $a0, 1
71     lw $t1, 4($sp)
72     sub $a0, $t1, $a0
73     addi $sp, $sp, 4
74     li $t1, 4
75     mult $a0, $t1
76     mflo $a0
77     la $a1, fibonacci
78     add $a1, $a1, $a0
79     lw $a0, 0($a1)
80     sw $a0, 12($sp)
81     lw $a0, 12($sp)
82     sw $a0, 0($sp)
83     addi $sp, $sp, -4
84     lw $a0, 12($sp)
85     lw $t1, 4($sp)
86     add $a0, $t1, $a0
87     addi $sp, $sp, 4
88     move $t5, $a0
89     lw $a0, 4($sp)
90     li $t1, 4
91     mult $a0, $t1
92     mflo $a0
93     la $a1, fibonacci
94     add $a1, $a1, $a0
95     sw $t5, 0($a1)
96     lw $a0, 4($sp)
97     addi $sp, $sp, -0
98     sw $a0, 0($sp)
99     addi $sp, $sp, 0
100    addi $sp, $sp, -4
101    jal next
102    addi $sp, $sp, 4
103    sw $a0, 4($sp)
104    j stwhile1
105 endwhile1:
106    li $a0, 0
107    sw $a0, 4($sp)
108 stwhile2:
109    lw $a0, 4($sp)
110    sw $a0, 0($sp)
111    addi $sp, $sp, -4
112    li $a0, 10
113    lw $t1, 4($sp)
114    addi $sp, $sp, 4
115    blt $t1, $a0, ifwhile2
116    j endwhile2
117 ifwhile2:
118    lw $a0, 4($sp)
119    li $t1, 4
120    mult $a0, $t1
121    mflo $a0
122    la $a1, fibonacci
123    add $a1, $a1, $a0
124    lw $a0, 0($a1)
125    li $s0, 1
126    syscall
127    move $t8, $a0
128    addi $a0, $0, 0xA
129    addi $v0, $0, 0xB
130    syscall
131    move $a0, $t8
132    lw $a0, 4($sp)
133    addi $sp, $sp, -0
134    sw $a0, 0($sp)
135    addi $sp, $sp, 0
136    addi $sp, $sp, -4
137    jal next
138    addi $sp, $sp, 4
139    sw $a0, 4($sp)
140    j stwhile2
141 endwhile2:
142    li $s0, 10
143    syscall
144
145 .data
146 fibonacci: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
147

```

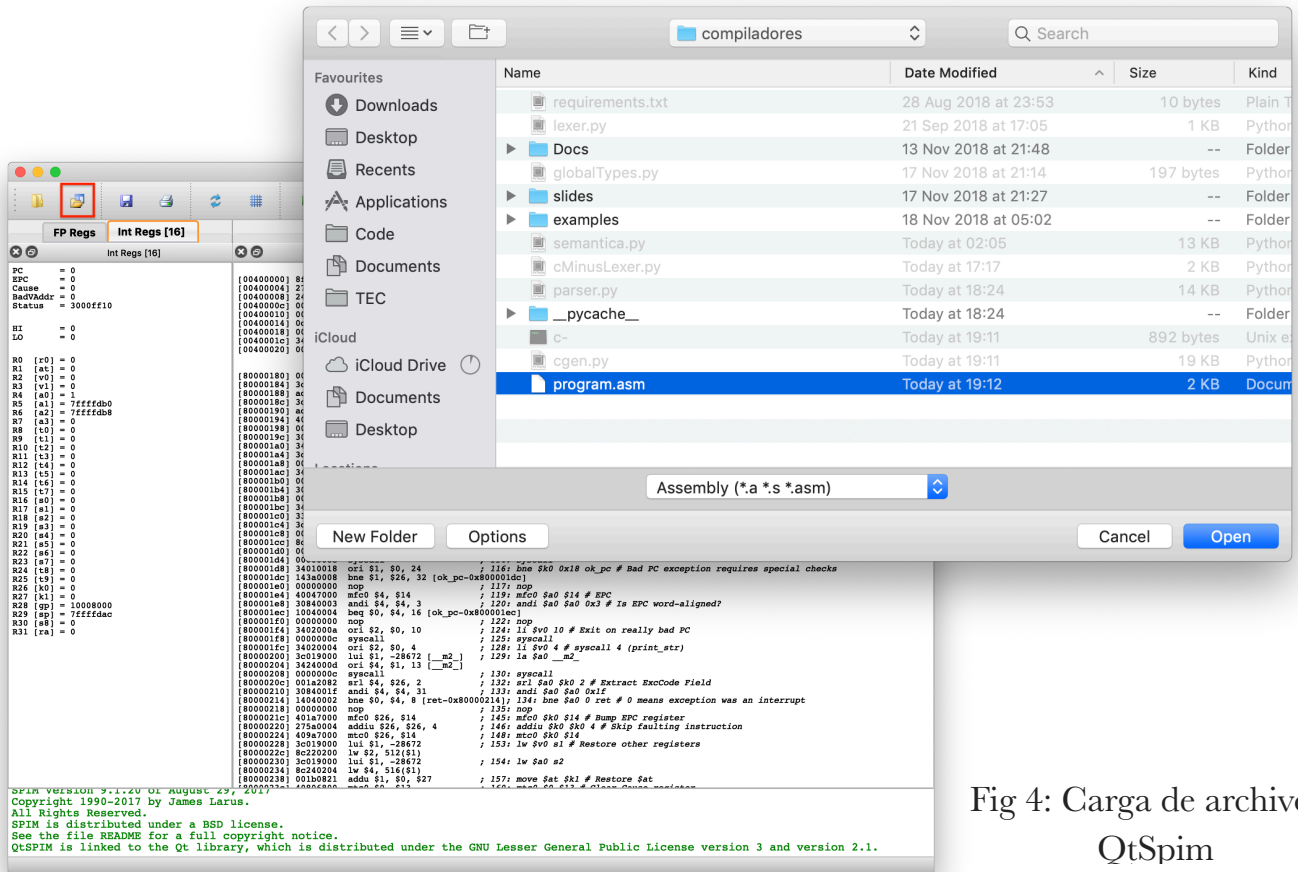


Fig 4: Carga de archivo a QtSpim

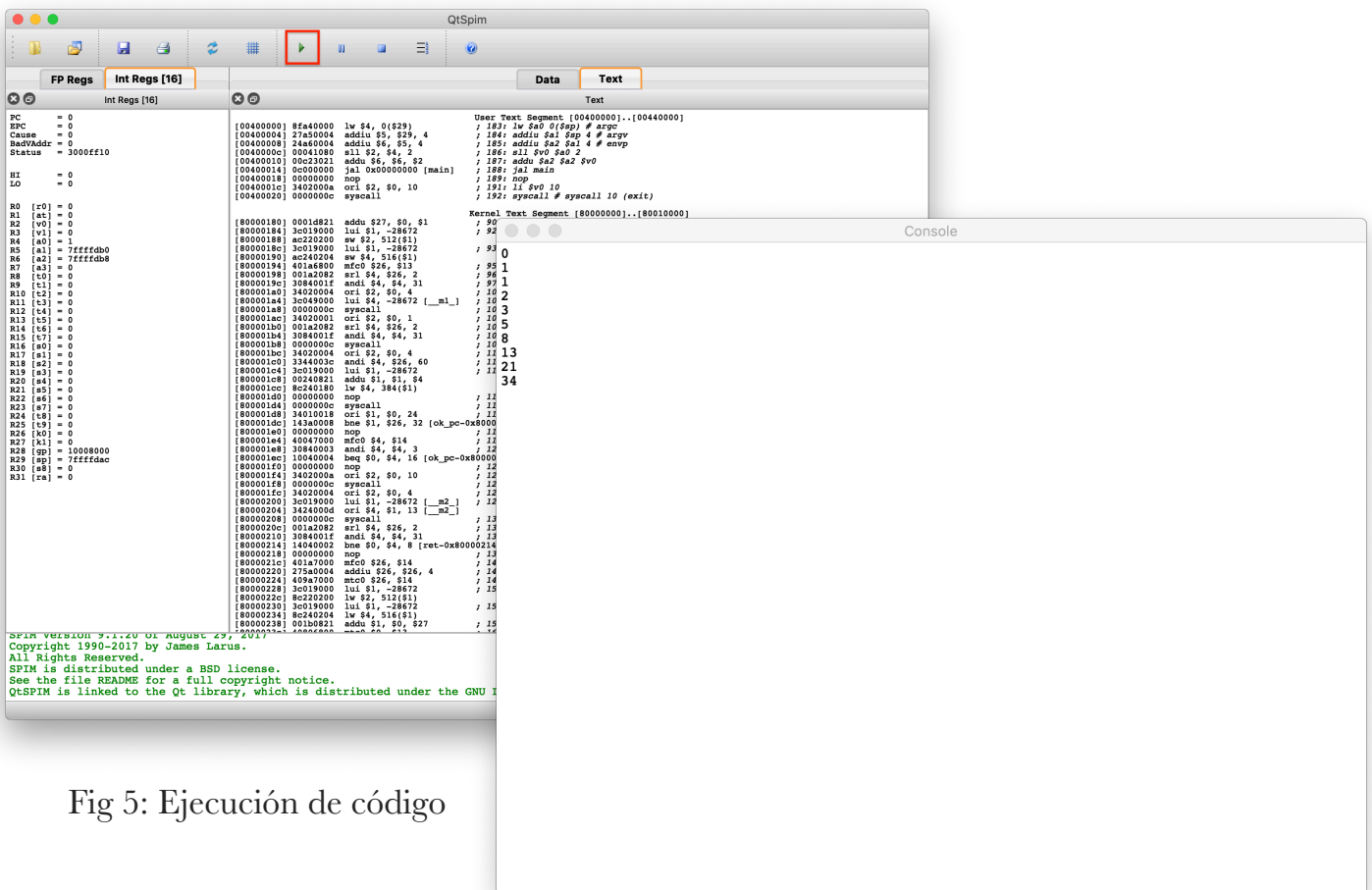


Fig 5: Ejecución de código

Apéndice:

El compilador esta conformado por multiples scripts que se fueron desarrollando a lo largo del semestre, al final de este documento se adjuntan los documentos entregado con cada script, de manera general el código se dividió en 4 entregables:

Analizador Léxico:	lexer.py
Parser (Tokens => Abstract Syntax Tree)	parser.py
Analizador Semantico	semantica.py
Generador de código	cgen.py

Ruben Cuadra

Dr. Victor de la Cueva

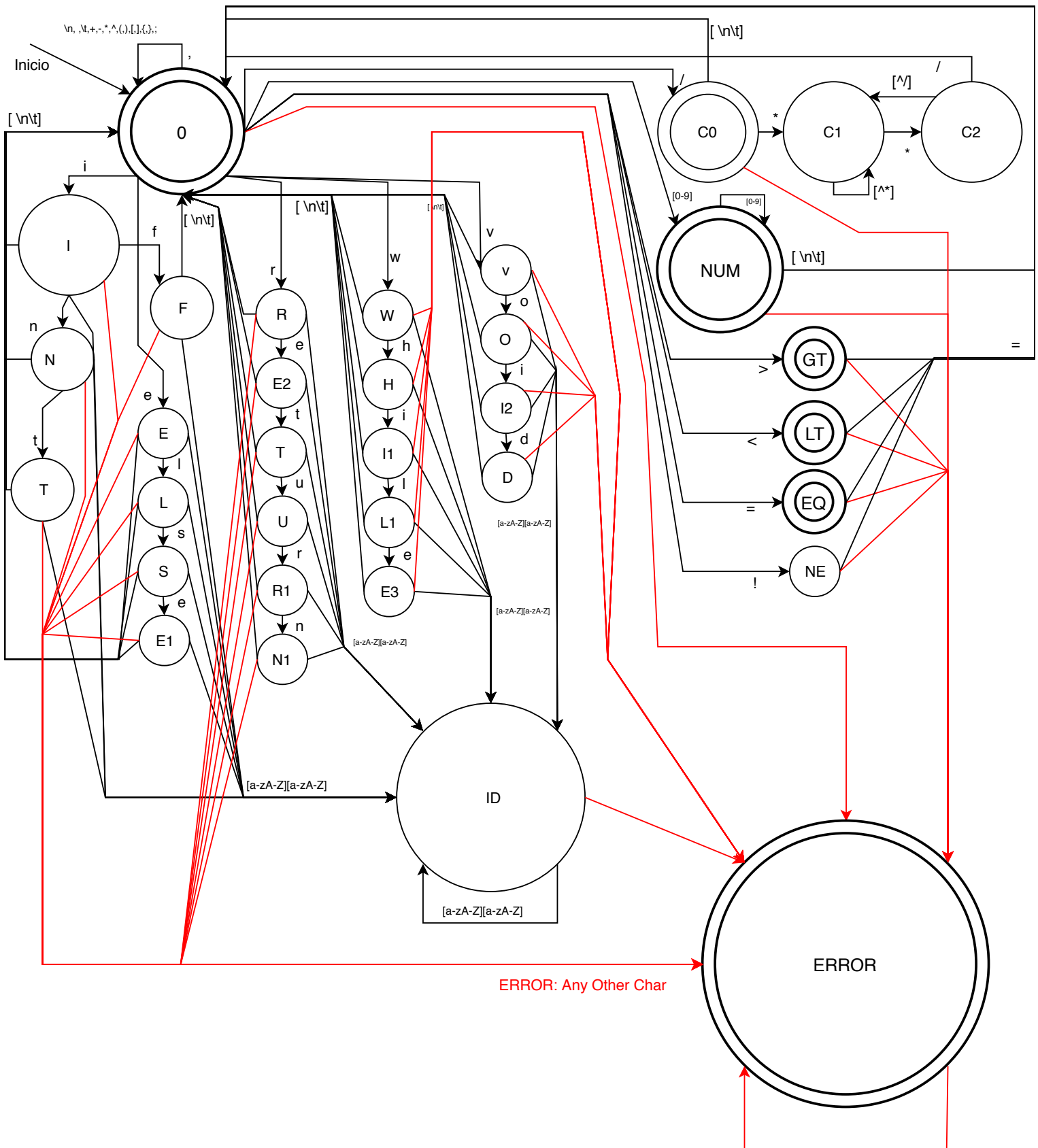
Lexer C-

1 September 2018

Lexer para C- (Menos)

Expresiones regulares, cada una representa un token diferente

else	/	{
if	^	}
int	<	,
return	<=	;
void	>=	(\n \t)
while	==	\d+
=	!=	(\^(. \\n)*?\\^/)
+	([a-zA-Z]+
-)	
*	[
+]	



Ruben Cuadra

Dr. Victor de la Cueva

Proyecto 2

21 September 2018

Gramática C-

1. program : declaration_list
2. declaration_list : declaration {declaration}
3. declaration : type_specifier **ID** ;
 | type_specifier **ID** [**INTEGER**] ;
 | type_specifier **ID** (params) compound_stmt
4. type_specifier : **int** | **void**
5. compound_stmt : { local_declarations_list statement_list }
6. local_declarations_list : var_declarations { local_declarations_list }
7. statement_list : statement { statement_list }
8. params : param_list
 | **void**
9. param_list : param { , param_list }
10. param : type_specifier **ID**
 | type_specifier **ID** []
11. statement : expression_stmt
 | compound_stmt
 | selection_stmt
 | iteration_stmt
 | return_stmt

12.iteration_stmt : **while** (expression) statement

13.selection_stmt : **if** (expression) statement

| **if** (expression) statement **else** statement

14.return_stmt : **return** ;

| **return** expression ;

15.expression_stmt : expression ;

| ;

16.expression : **ID** = expression

| **ID** [expression] = expression

| **ID** [expression] {operations}

| **ID** [expression] {operations} conditional {operations}

| factor {operations}

| factor {operations} conditional {operations}

17.conditional : relop factor

18.operations : multis

| multis sumres

| sumres

19.sumres : { addop factor multis }

20.multis : { mulop factor }

21.factor : (expression)

| **NUM**

| **ID**

| **ID** [expression]

| **ID** (args)

22.mulop : * | /

23.addop : + | -

24.relop : < | <= | > | >= | == | !=

Analizador semantico

Reglas de inferencia

La tabla semántica generada es en realidad un **árbol** que contiene 1 nodo padre (o ninguno en la raíz) y multiples hijos (0 en los nodos hoja). Cada nodo cuenta con una variable llamada **scope** la cual es una **tabla hash** donde la **llave** es un **identificador** y el **valor** es una **tupla** con *tipo_de_identificador* (int | void), *estructura_identificador* (arreglo/función/variable) y lo siguiente es el *tamaño_del_arreglo*, *valor_de_variable* o *parametros_de_funcion*

Reglas

Operadores	Tipo
Int * Int	Int
Int / Int	Int
Int + Int	Int
Int - Int	Int
Int < Int	Int
Int <= Int	Int
Int > Int	Int
Int >= Int	Int
Int == Int	Int
Int != Int	Int
Int = Int	Int
void = void	Void
int[int]	Int
Int	Int
Void	Void