Campus Tecnológico Central de Cartago
Compiladores e Intérpretes

Etapa 2: Analizador Sintáctico

Prof. Kirstein Gätjens Soto

Daniel Sequeira Retana - 2019039641
Kenneth Palacios Molina - 2020035407

Fecha de entrega: 13 de mayo

I Semestre

2024

índice

índice 2
Propuesta: Concurso de nombres 4
Definición del lenguaje 5
Autómata 10
Autómata de la letra A 11
Autómata de la letra C 12
Autómata de la letra E 13
Autómata de la letra F 14
Autómata de la letra I 15
Autómata de la letra J 16
Autómata de la letra L 18
Autómata de la letra M 19
Autómata de la letra N 20
Autómata de la letra O 21
Autómata de la letra P 22
Autómata de la letra Q 23
Autómata de la letra S 24
Autómata de la letra T 25
Autómata de la letra Y 26
Índice de pruebas 27
Propuesta gramatical 27
Estructura del título del programa nhequéquequé 27
Apertura de sección de constantes octará 27
Apertura de sección de tipos aíca 28
Apertura de sección de variables tilhtilh 28
Apertura de sección de rutinas purú 28
Apertura de sección de prototipos chá 28
Sistema de asignación de constantes arrá 29
Sistema de asignación de tipos putú 29
Sistema de asignación de variables nicó 29
Sistema de asignación de booleanos afá 30
Sistema de asignación de registros poréteca … turé 30
Tipos de datos 31
Literales de conjuntos 31
Literales de archivos 31
Literales de números flotantes 32
Literales de enteros 32
Literales de caracteres 33
Literales de strings 34
Literales de arreglos 34

Literales de registros....35
Sistema de acceso a arreglos....35
Sistema de acceso a strings....35
Sistema de acceso a registros....35
Asignación con: := += -= *= %= /=....36
Operaciones aritméticas básicas de enteros: + - * % /....36
Decremento e incremento de variables: jené y taqué....37
Operaciones básicas sobre caracteres: >> << &? #?....37
Operaciones lógicas: ta yú pucá emé....38
Operaciones de strings: Concatenar: catáqui / Length: lharátesufá / cortar: joné / recortar: nhalh / encontrar: cochéqui....39
Operaciones con conjuntos: Agregar: erronh / Borrar:quené / Union: fuinhye / Interseccion: sulí / PerteneceA: pcalúfa / Vacío?: culíte....41
Operaciones con archivos: Abrir: quijí / Cerrar: eloc / Escribir: lhioc / Leer: lhaic / Crear: quijérri / Asociar: láte....42
Operaciones con números flotantes:|++ ++| |+| +|+....42
Operaciones de comparación: < > = >= <= ><....43
Manejo de bloques de más de una instrucción: cojcoj … fuchí....44
Instrucción while....44
Instrucción if....45
Instrucción if-then-else....45
Instrucción switch....45
Instrucción repeat-until....46
Instrucción for....46
Instrucciones break, continue y halt....46
Encabezado de funciones....47
Encabezado de procedimientos....47
Manejo de parámetros formales....47
Manejo de parámetros reales....47
Instrucción return....47
Operación sizeof....47
Sistema de coerción de tipos....47
Salida Estándar....48
Entrada Estándar....48
Pruebas de análisis léxico....48
Pruebas de análisis sintáctico....48
Test de comentarios....48
Test de fors....48
Test de ifs....48
Test de inicio incorrecto....48
Test de repeat until....48
Test de sección de constantes....48
Test de falta de inicio....49
Test de sección de prototipos....49
Test de sección de rutinas....49

Test de sección de tipos..... 49
Test de sección de variables..... 49
Test de while.....49
Funcionamiento del scanner..... 49

Propuesta: Concurso de nombres

Nombre propuesto: Táfac

Significado: Diablo celestial de los que tienen como función dar a los condenados su dosis diaria de tormento, que consiste en que los mastican, se los tragan y luego los expulsan, convertidos en bagazo, al defecar, orinar o vomitar

Logotipo propuesto:

Extensión para los archivos propuesta: .taf

Definición del lenguaje

Estructura o parte del lenguaje	Elemento en el lenguaje	Traducciones del Malecu	Comentarios / Aclaraciones
Estructura del título del programa FC	nhequéquequé	Sonido onomatopéyico del chirrido de una puerta.	
Apertura sección constantes	octará	Traducción de piedra	escrito en piedra
Apertura sección de tipos	aíca	Traducción de varios	tipos equivale a que hay variedad
Apertura de la sección de variables	tí	Traducción de agua, en contraste a piedra	
Apertura de la sección de rutinas	urúma	Traducción de costumbre	
Apertura de la sección de prototipos	chóje	Traducción de proteger, un prototipo protege de un error declarando la función antes de su definición	
Punto de entrada del programa	cojcoj	Tocar la puerta para poder ingresar	
Sistema de asignación de constantes	caralh	Sonido de una piedra chocando con otra	
Sistema de asignación de tipos	putú	Traducción de cielo	Hay muchos tipos de cielo
Sistema de declaración de variables	sulí	Traducción de Cambio	
Tipo de dato entero	chátené	Entero o completo	
Tipo de dato carácter	filhílhi	Representación	

Tipo de dato string	purucpuruc	Cuerda	
Tipo de dato booleano	cochója	Doblado	
Tipo de dato conjunto	apúchá	Conjunto	
Tipo de dato archivo de texto	puríri	Contener	
Tipo de datos números flotantes	jané	Flotar	
Tipo de dato arreglos	cóye	Guardado/ Tener guardado	Los arreglos son sin límite de dimensión y son homogéneos y de cualquier tipo del lenguaje
Tipo de dato registros	lhará	Seña	
Tipo de datos atómico creativo	Número imaginario: úji	Traducción de imaginarse	
Literales booleanas	True: suírra / False: epéme	Traducción literal de verdadero y parcial de falso	
Literales de conjuntos	{ 'X1', 'X2',….,'Xn' }		Usa las literales de caracteres, separados por comas
Literales de archivos	{@ "c:\akw\tarea1.txt" , L @}		La primer literal de string es la ruta y nombre con extensión del archivo físico. La letra indica si es de lectura (L), escritura (S), ambas (D)
Literales de números flotantes	1.223		
Literales de enteros	-123 0xF4EC		Las de C, exactamente las de C.
Literales de caracteres	Con comilla simple: 'A' '\n' '\666'		Las de C, exactamente las de C.

Literales de strings	"Hola \n"		Las de C, exactamente las de C.
Literales del tipo de datos atómico creativo	3i un numero entero finalizando con una i		
Literales de arreglos	<[1,2,3,4]>		Llaves con corchetes
Literales de registros	úpalá [[ ]]	Traducción de casa	Brackets dobles
Sistema de acceso arreglos	<[]>		
Sistema de acceso strings	¿?		
Sistema de acceso registros	-->		
Asignación y Familia	óri / <operación>óri	Dar órdenes a alguien	Escoger un símbolo o palabra para la asignación y una sintaxis para las asignaciones combinadas.
Operaciones aritméticas básicas de enteros	+ - * % /		Suma Resta Multiplicación Residuo y División Entera
Incremento y Decremento	fay+ / tij-	Aumentar: fayátaqué / Disminuir: tíjye	Solo en prefijo
2 operaciones de enteros creativas	curíjurí para logaritmo / caquí: suma del piso x en el triángulo de pascal	Traducción de ritual / traducción de pilar	
Operaciones básicas sobre caracteres	Pasar a mayúsculas: taq / Pasar a minúsculas: jen / is alpha: mal / is digit: jat	Subir: taqué / Bajar: jené / Letra: malhióca / Numeroso: jatájatá	Pasar a mayúsculas, a minúsculas, is alpha, is digit, todas son unarias y en prefijo.
Operaciones lógicas solicitadas	and: muri / or: cóqui / xor: tiátiá / not: emé	Traducción de and y Traducción de suceder (un or es esperar que algo suceda). Xor: Traducción de	

		exactamente Not: Traducción de no	
Operaciones de Strings solicitadas	Concatenar: catáqui / Length: lharátesufá / cortar: joné / recortar: nhalh / encontrar: cochéqui	Adherir: catáqui / Cosa larga: lharátesufá / Cortar: joné / Ideófono del sonido del cortar de un solo golpe un objeto: nhalh / Encontrar: cochéqui	Concatenar length cortar recortar find
Operaciones de conjuntos solicitadas	Agregar: erronh / Borrar: quené / Union: fuinhye / Interseccion: sulí / PerteneceA: pcalúfa / Vacío?: culíte	Añadir: erronh / Borrar: quené / Mezclar: fuinhye / Cruzar: sulí / Miembro: pcalúfa / Ser vaciado: culíte	Agregar Borrar Union Interseccion PerteneceA Vacío?
Operaciones de archivos solicitadas	Abrir: quijí / Cerrar: eloc / Escribir: lhioc / Leer: lhaic / Crear: quijérri / Asociar: láte	Abrir: quijí / Cerrar: eloc / Escribir: lhioc / Leer: lhaic / Crear: quijérri / Relación: láte	abrir cerrar escribir leer crear asociar
Operaciones de números flotantes	Suma: erroc / Resta: sírru / Multiplicación-> fují / División: palé	Añadir: erroc / Quitar: sírru / Acumulación: fují / Dividir: palé	suma, resta, multiplicación, división (binarias)
Operaciones del tipo de datos atómico creativo	i+ i* i/ i-		Suma, multiplicación, división, resta
Operaciones de comparación solicitadas	> < = >= <= ><		Se comparan enteros y retornan booleanos
Manejo de Bloques de más de una instrucción	CojCoj		Hay que pensar el de cierre. O sugerir un par diferentes si encuentran algo más divertido.
Instrucción while	pují <exp> sóta / pují (<exp>)	Dedicar el tiempo a algo: pují / Hacer: sóta	while <exp> do while (<exp>)
Instrucción if-then-else	if: arú / else: ótacá		Como el ifTrue de Smalltalk!! then y else opcionales y en cualquier orden.
Instrucción switch	Switch: lhifíji / case: palhá / default: quírra	Cambiar: lhifíji / Indica que llego el momento de hacer algo: palhá / Origen: quírra	switch case default Como el de C.

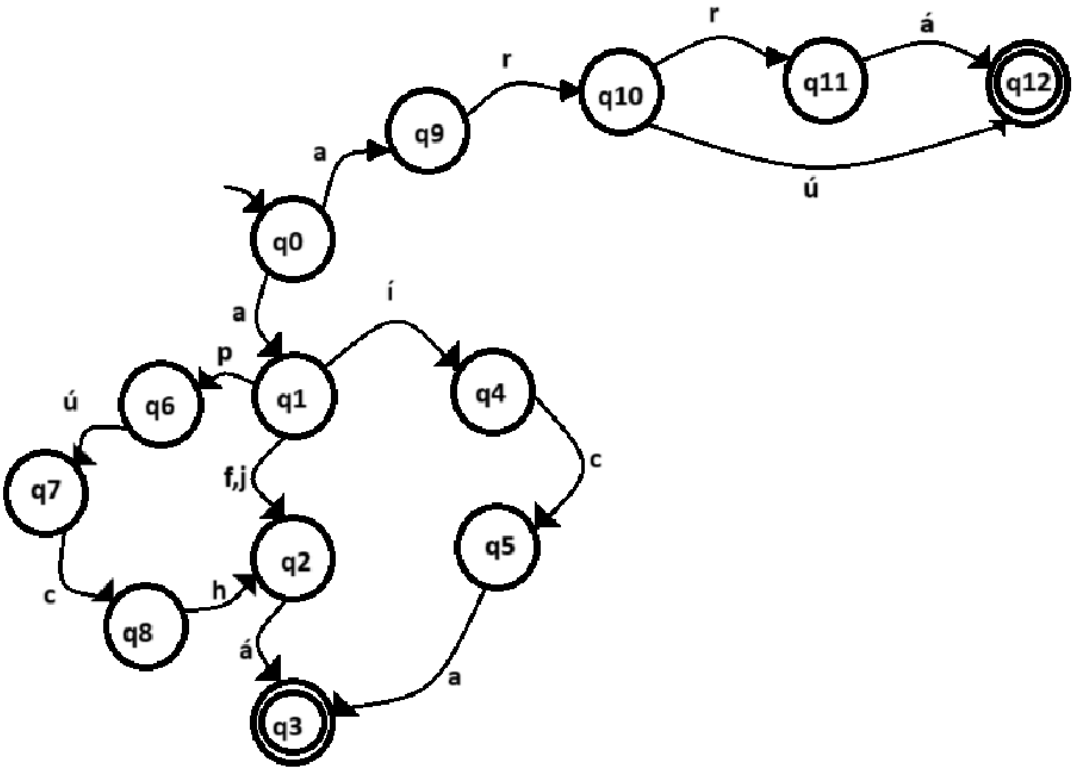
Instrucción Repeat-until	paípaí / yája	Repetidamente: paípaí / Hasta: yája	como el repeat until de pascal.
Instrucción For	ajá	Para: ajá	como el for de pascal no de C puede ser como el de python
Instrucción With	tiní	Con: tiní	como el whith de pascal
2 Instrucciones creativas	Tarráni: hacer una pausa en la ejecución hasta que se reciba una entrada en el teclado / Chequéle: equivalente a un cls, limpiar consola	Traducción de tiempo	
Instrucción break	foróforóye	Romperle la entrada de la cueva a una taltuza	como el de C
Instrucción continue	taráne	Seguir la línea de algo como se determinó previamente	como el de C
Instrucción Halt	lauc	Terminar	Detiene la ejecución del programa completamente.
Encabezado de funciones	lufá id (...) : tipo	Actividad	buscar palabra melecu
Encabezado de procedimientos	ónhaónha id (...)	Conducta	buscar palabra melecu
Manejo de parámetros formales	x(.a, .b, .c)		Deben ser consistentes con la declaración de variables. Hay que pensar la forma de decir que un parámetro va por referencia y no por valor.
Manejo de parámetros reales	x(a, b, c)		
Instrucción return	jalac	Devolver	Hay dos trabajos diferentes que hace el return de C. ¿Los separamos? Sí, proponemos que

			jalac no cumpla la función de terminar el programa
Operación de size of	níca	De este tamaño	tamaño ya sea de un tipo o de una expresión.
Sistema de coerción de tipos	ü(nombre_del_tipo)		expresión (pensar un operador en malecu) tipo
Manejo de la entrada estándar	tioctipo	Entrar: tioc	Los seis tipos atómicos
Manejo de la salida estándar	pítetipo	Salir: píte	Los seis tipos atómicos
Terminador	;		Es el punto y coma y es un terminador

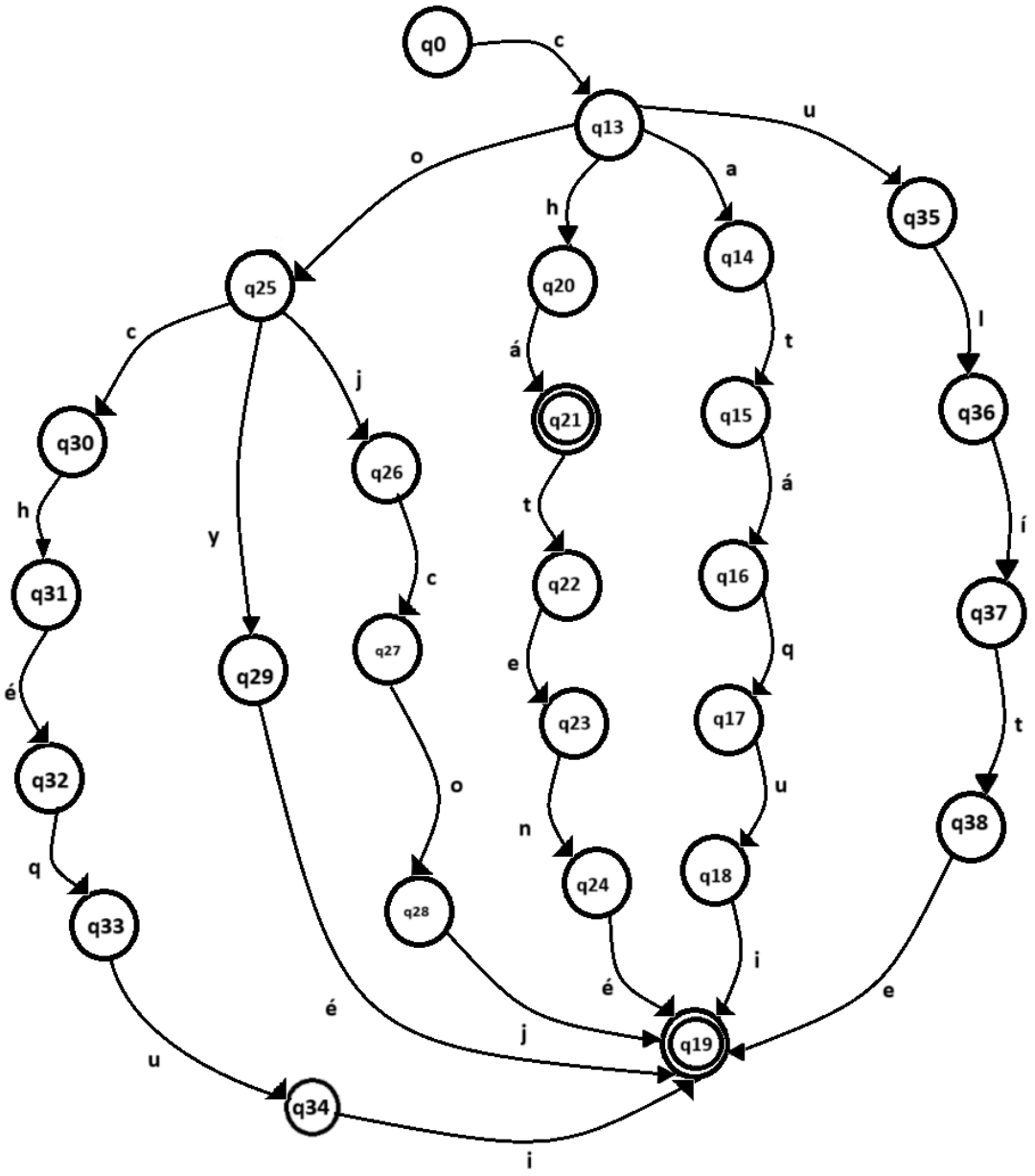
Autómata

El autómata del proyecto fue desarrollado por partes debido a su alta complejidad. Aún así, se hizo de manera que sea simple de entender y cada “diagrama individual” comienza en el estado q0 y tiene conexiones hacia otros estados dependiendo del token que esté formando. Los caminos no definidos para cada estado se enciclan, es decir, no son aceptados por el autómata.

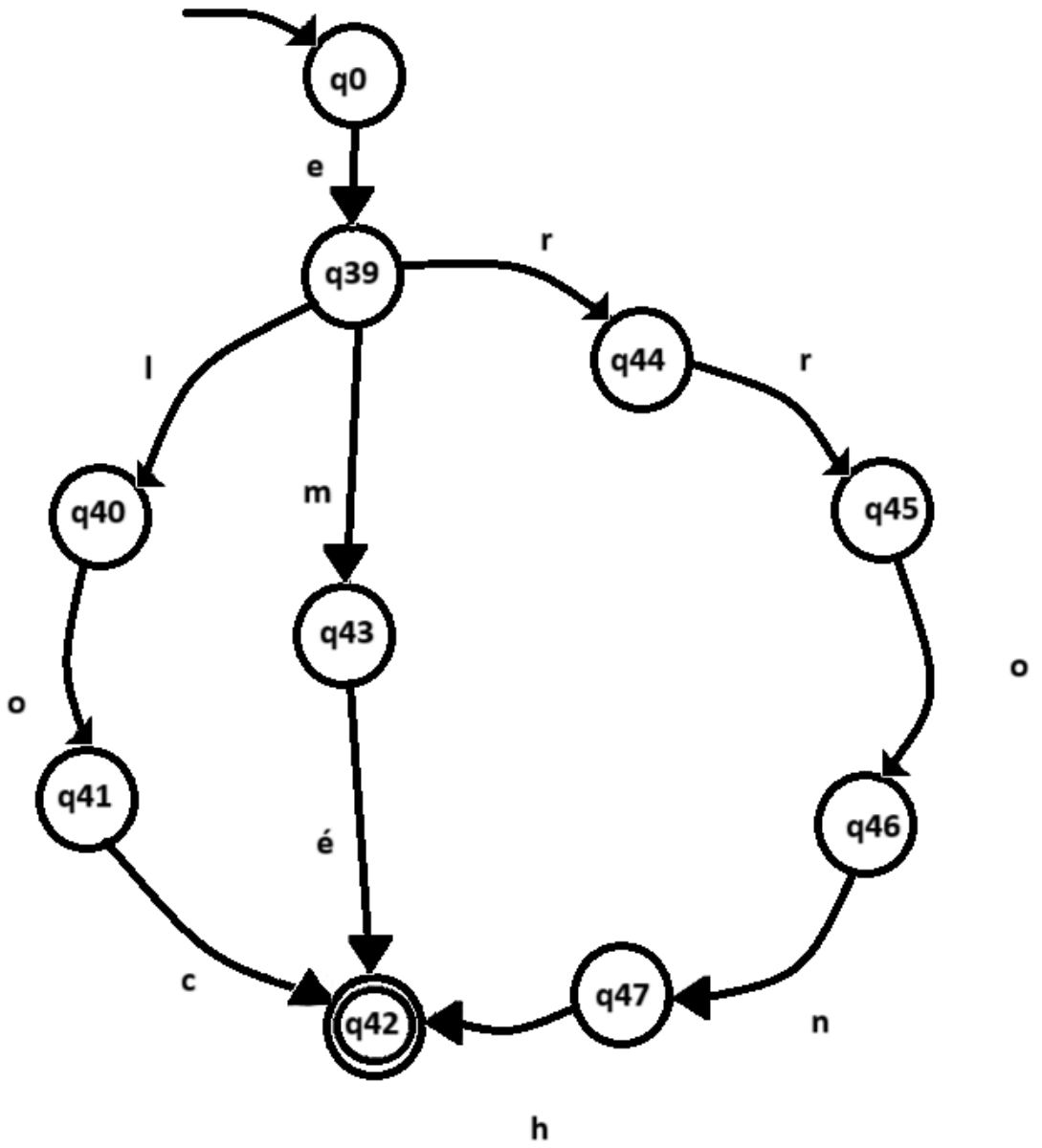
Autómata de la letra A



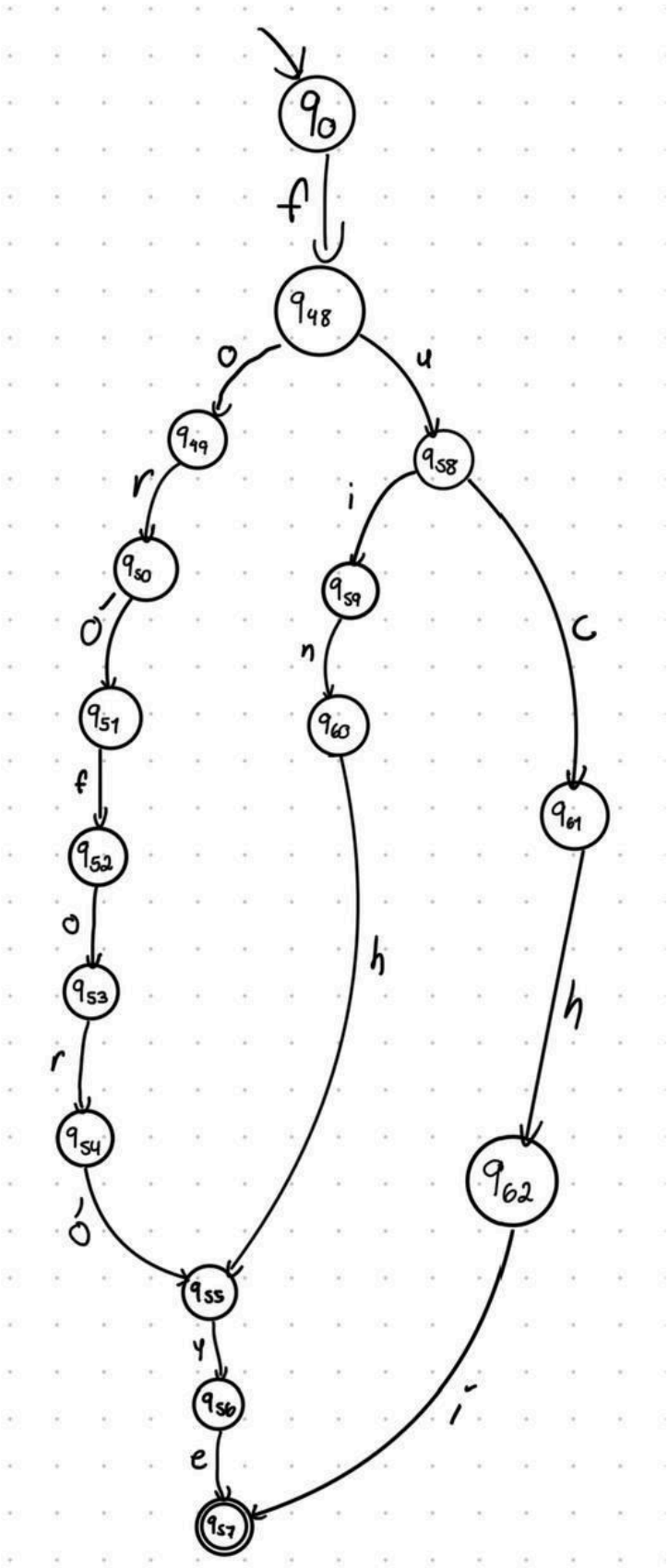
Autómata de la letra C



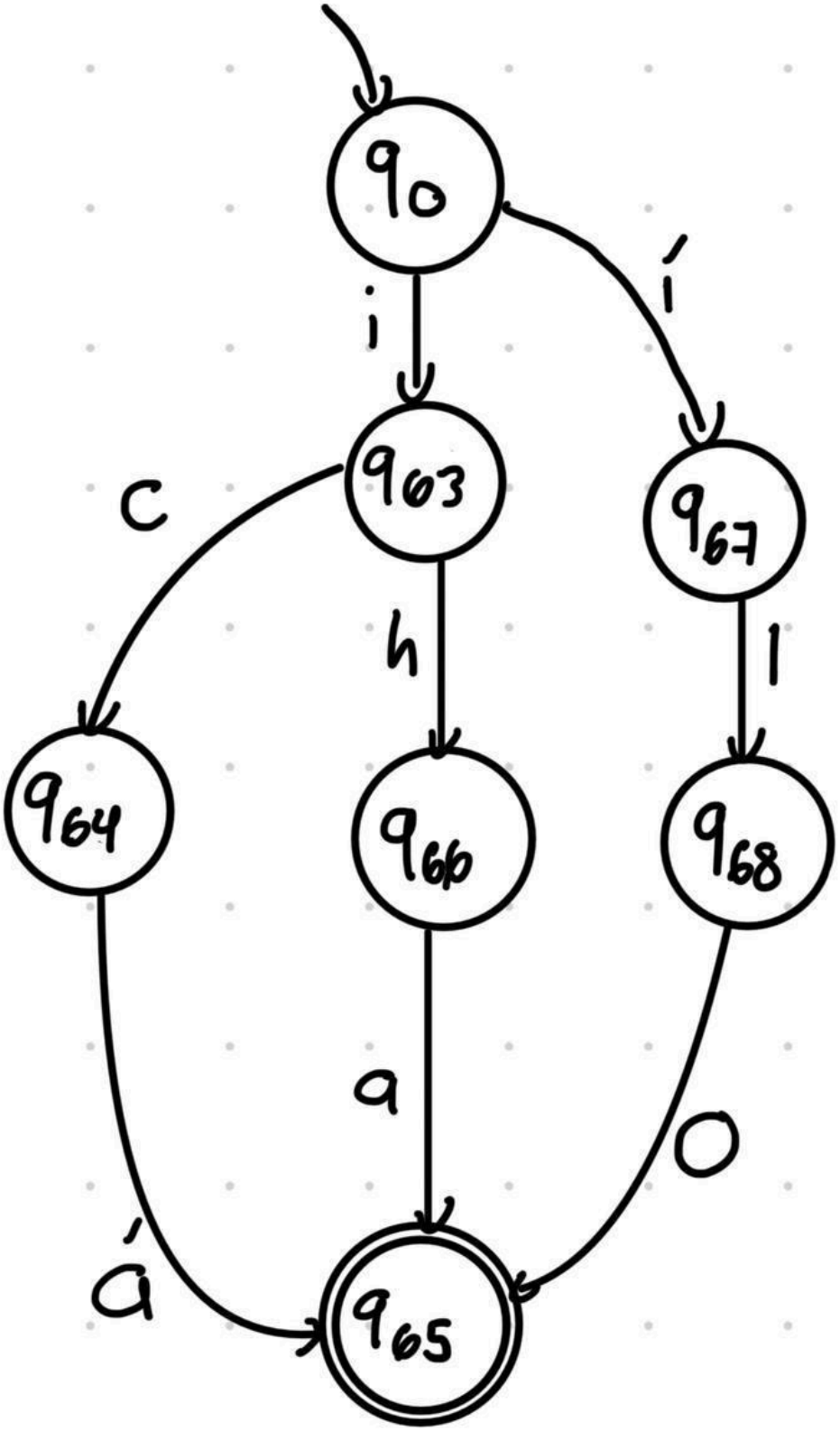
Autómata de la letra E



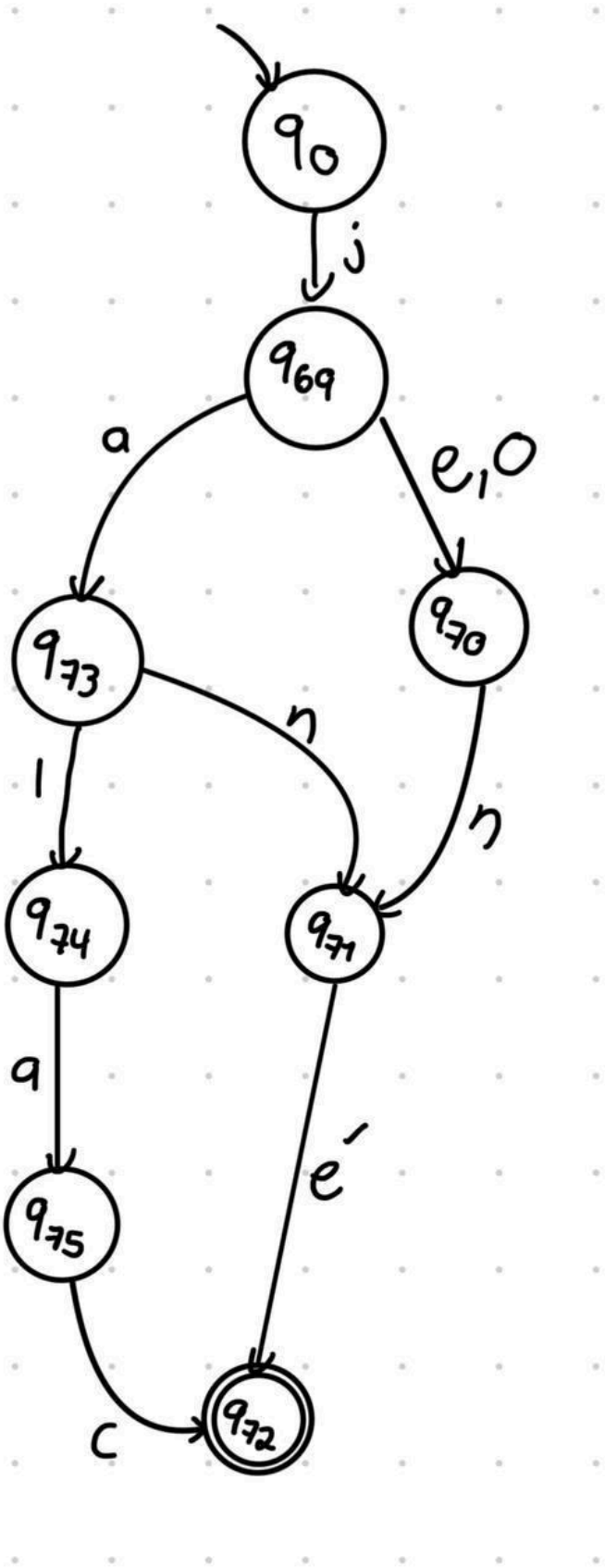
Autómata de la letra F



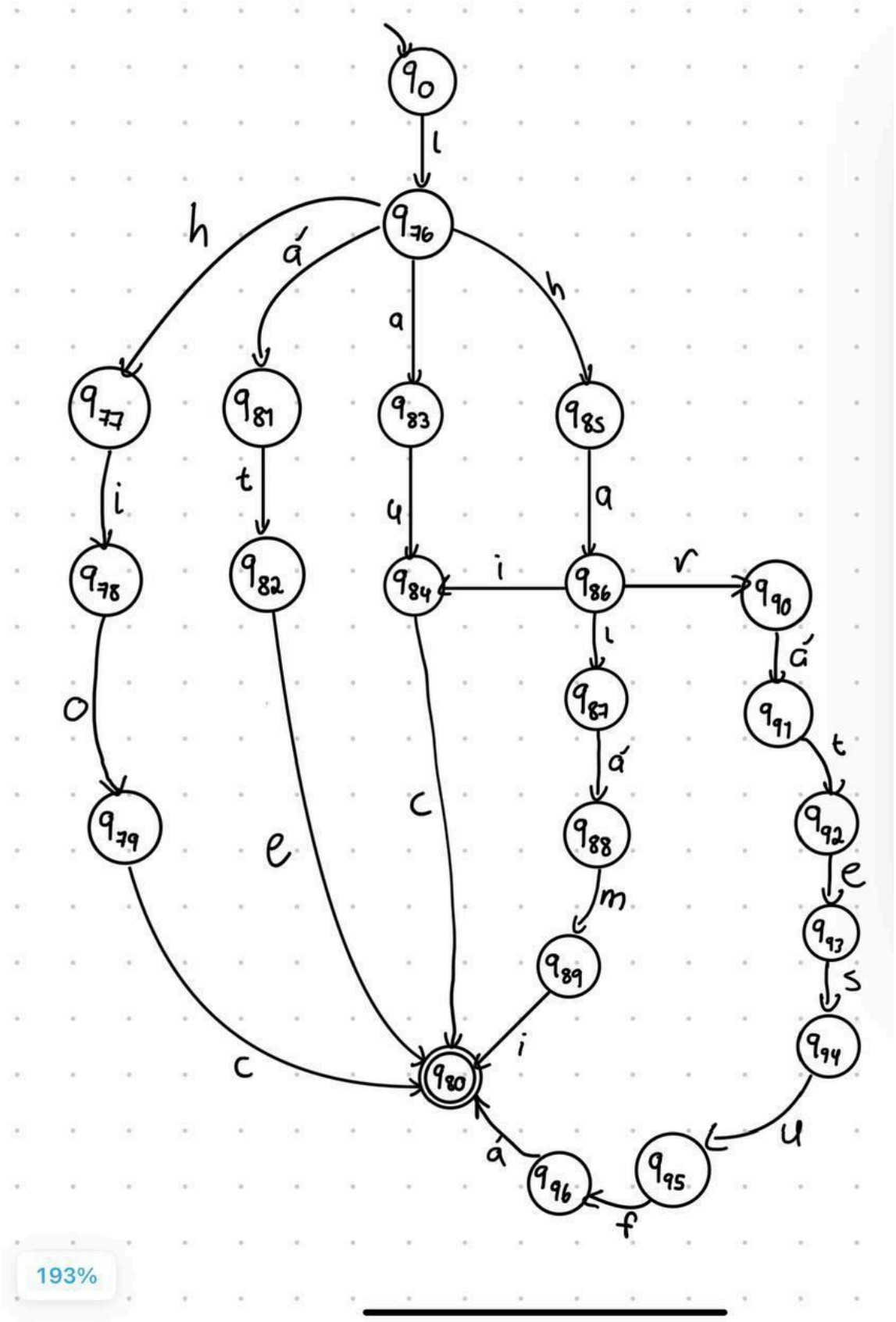
Autómata de la letra l



Autómata de la letra J

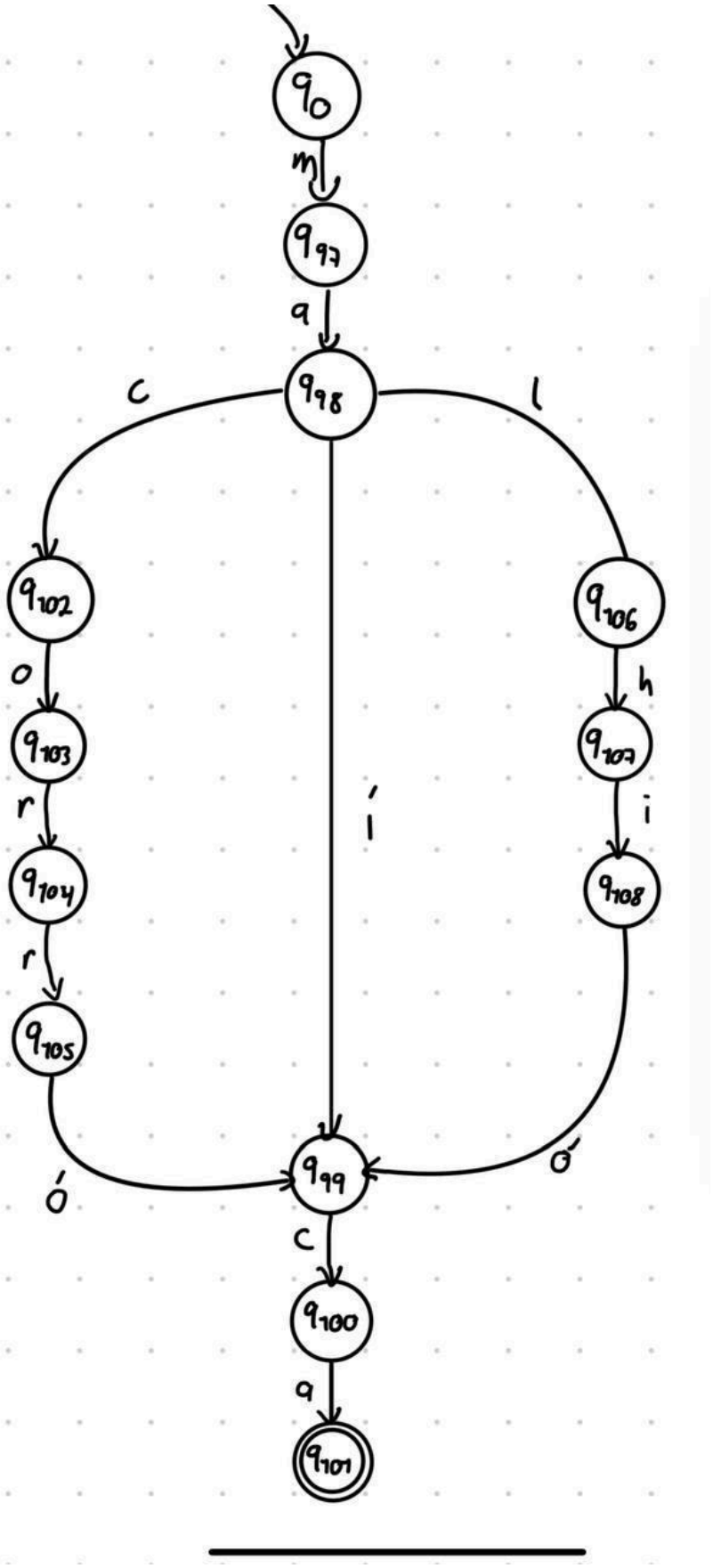


Autómata de la letra L

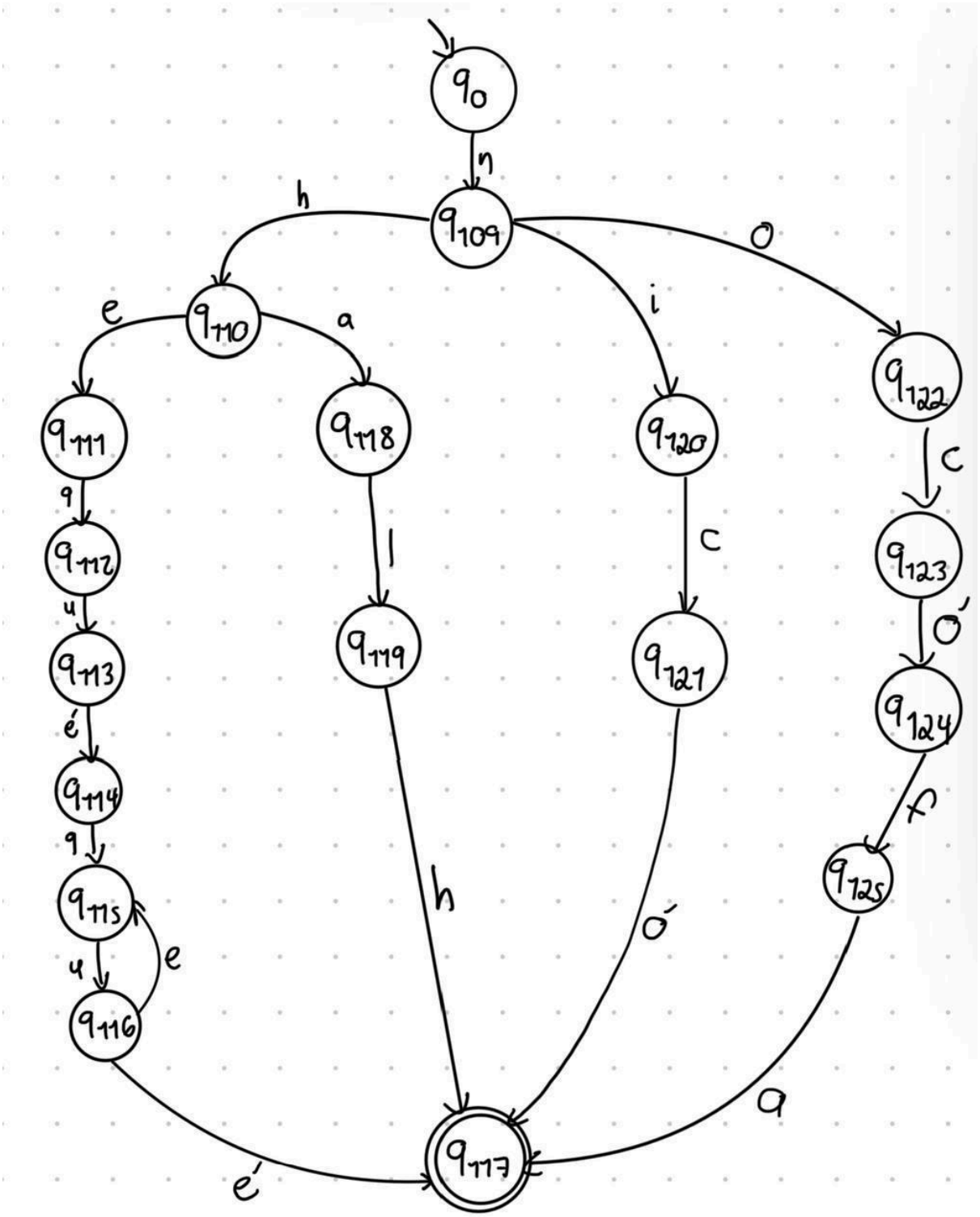


193%

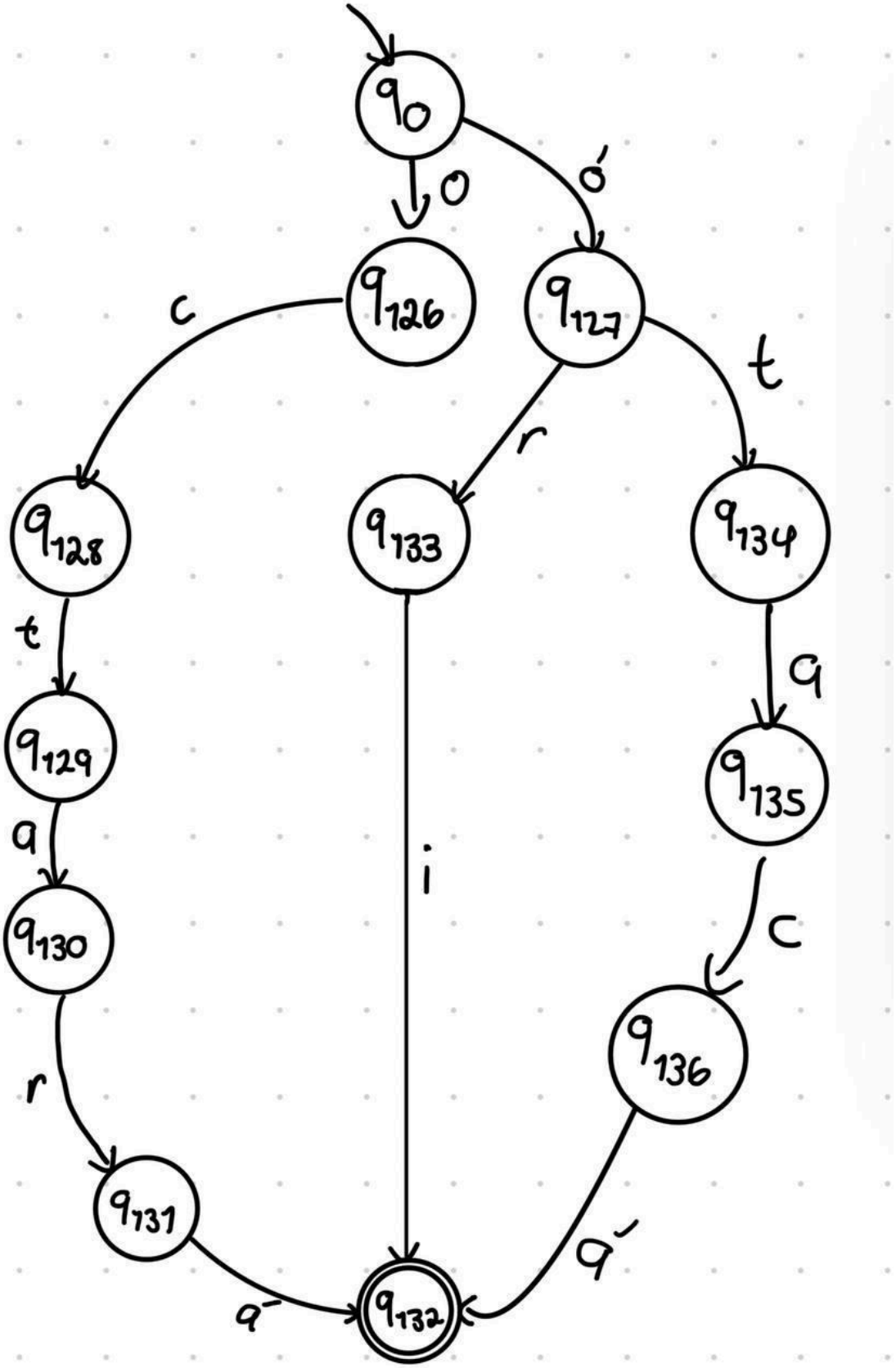
Autómata de la letra M



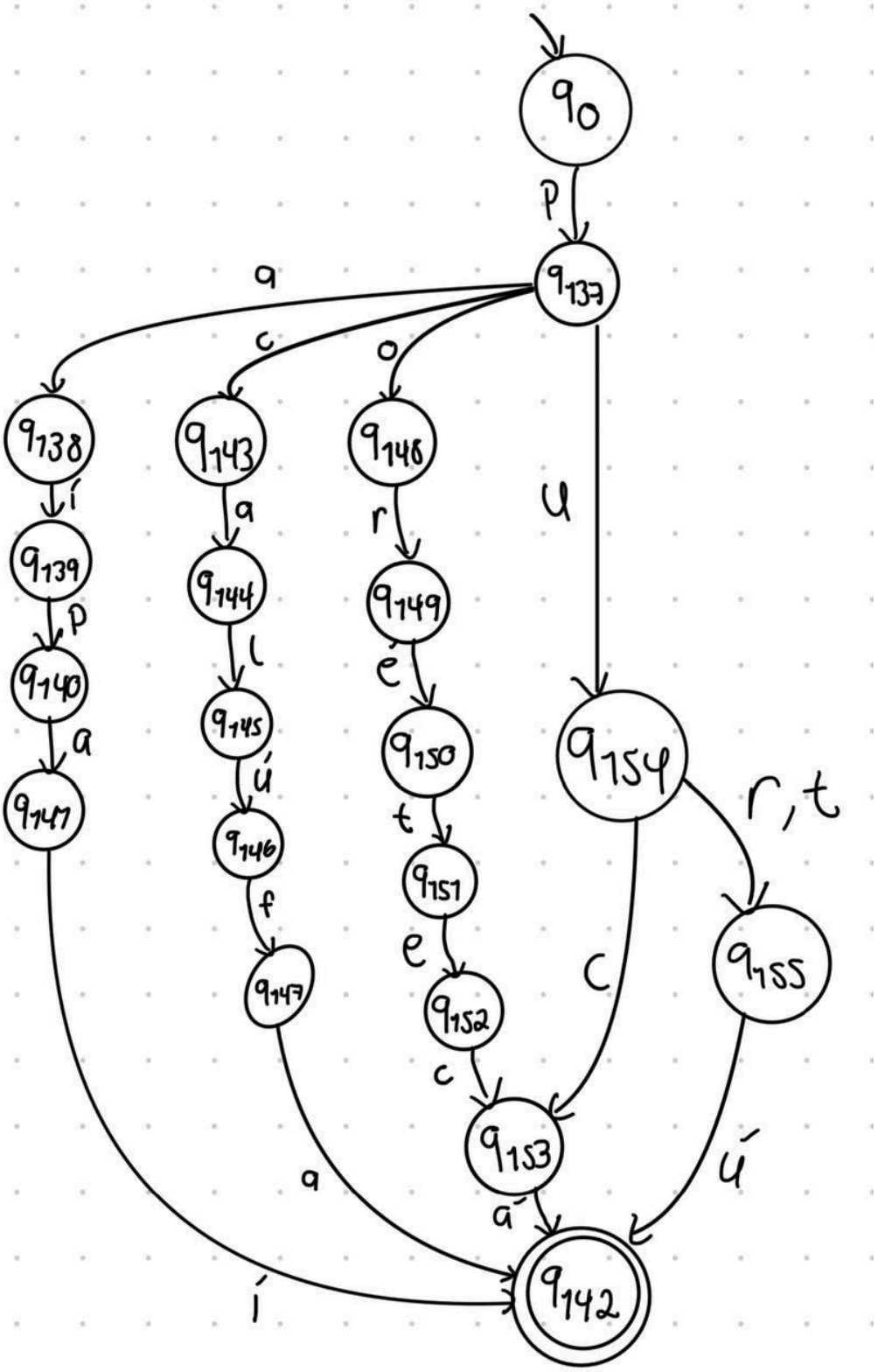
Autómata de la letra N



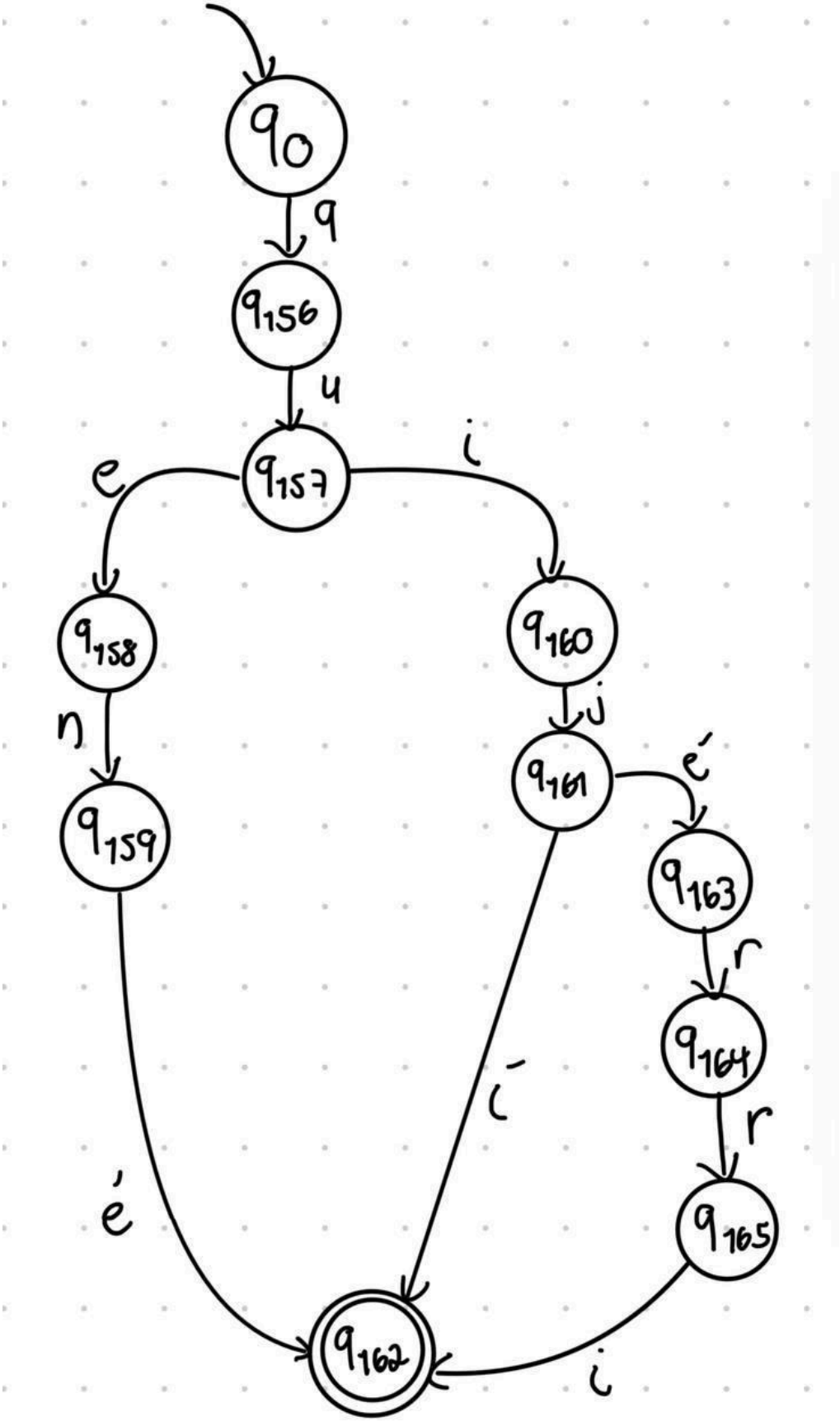
Autómata de la letra O



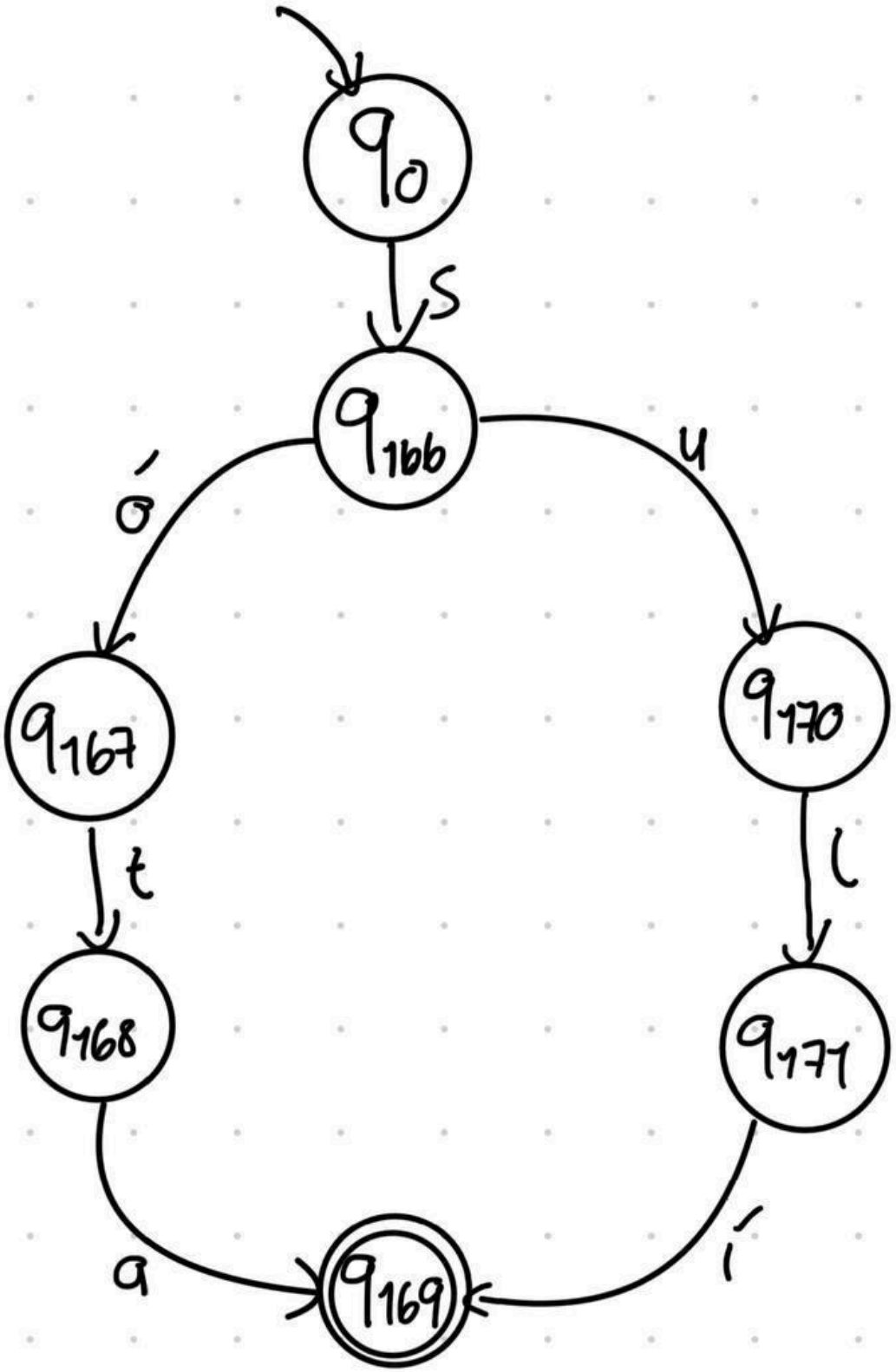
Autómata de la letra P



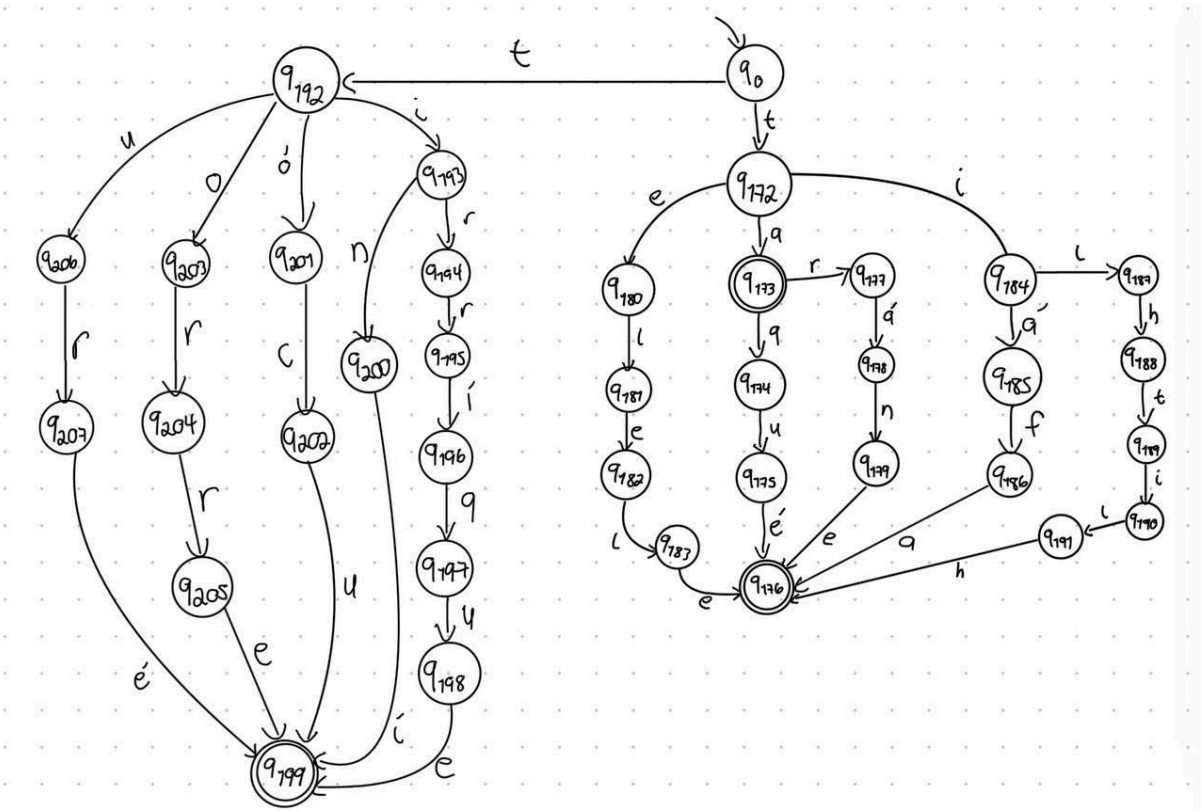
Autómata de la letra Q



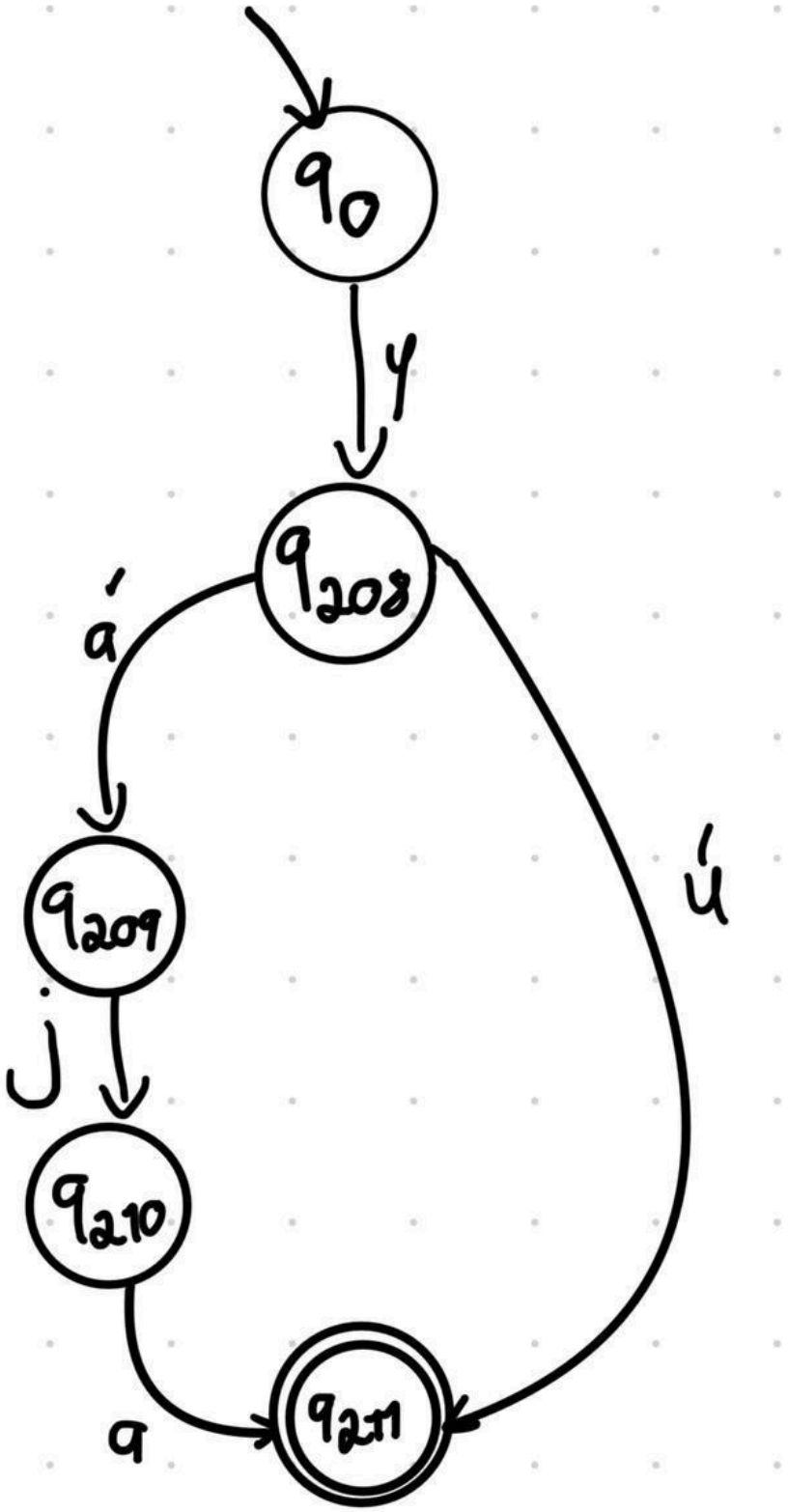
Autómata de la letra S



Autómata de la letra T


t
u
o
ó
i
r
r
r
r
c
n
í
q
u
u
é
e
í
e
t
e
i
r
á
á
l
h
t
i
l
e
a
h
l
e
e
a
f
l
e
q
u
é
q
r
l
e
l
h
t
i
q
u
é
e
e
e

Autómata de la letra Y



Índice de pruebas

Asignaciones.txt

- Contiene ejemplos de las asignaciones

instrucciones.txt

- Contiene ejemplos de las instrucciones

Operaciones.txt

- Contiene ejemplos de las instrucciones

Tipos-de-datos.txt

- Contiene ejemplos de los tipos de datos

Propuesta gramatical

Estructura del título del programa nhequéquequé

Código:

```
## esto va al inicio del programa
## y declara el nombre del mismo
nhequéquequé miPrograma;
```

<apertura> ::= nhequéquequé id ;

Apertura de sección de constantes octará

En código:

```
##apertura de sección de constantes: octará
octará

    ## arrá instrucción prefija que declara x1
    arrá x1 45;
    arrá x2 'd';
    arrá x3 maíca; ##maica es False
```

<const> ::= octará <expresión>

<expresión> ::= ... <expresión>

<expresión> ::= ε

Apertura de sección de tipos aíca

En código:

```
##apertura de sección de tipos: aíca
aíca

    ## putú es una instrucción prefija
    putú Edad chátené;
```

<tipoSec> ::= aíca <expresión>

Apertura de sección de variables tilhtilh

En código:

```
##apertura de sección de variables: tilhtilh
tilhtilh

    ##chátené es tipo entero y nicó su asignación,
    chátené x4 nicó 23;
    coyé chátené x5 ta 10;                ## esto de
    ílo x6 nicó "Esto es un string";      ## ílo es
```

<variableSec> ::= tilhtilh <expresión>

Apertura de sección de rutinas purú

En código:

```
## sección de rutinas (funciones y procedimientos): purú
purú

macorróca afá esUnaVocal (malhióca Letra)
cojcoj                    ## se puede interpretar como {
```

<rutinasSec> ::= purú <expresión>

Apertura de sección de prototipos chá

En código:

```
## sección de prototipos (se refiere a la declar
chá
    ##macorróca es el encabezado de la función,
    macorróca chátené fibonacci (chátené N);
    óri factorial (chátené N, suí chátené Resp);
    macorróca afá esUnaVocal (malhióca Letra);
```

<prototiposSec> ::= chá <expresion>

Sistema de asignación de constantes arrá

En código:

```
## arrá instrucción prefija que
arrá x1 45;
arrá x2 'd';
arrá x3 maíca; ##maica es False
```

<asigConst> ::= arrá id <literalX> ;

Sistema de asignación de tipos putú

En código:

```
## putú es una instr
putú Edad chátené;
```

<asigTipo> ::= putú id <tipoDato> ;

Sistema de asignación de variables nicó

En código:

```
##chátené es tipo entero y nicó su
chátené x4 nicó 23;
coyé chátené x5 ta 10;
ílo x6 nicó "Esto es un string";
afá x7 nicó tócu;
```

<asigVar>::= <tiposDeDatos> id nicó <literalX> ;

Sistema de asignación de booleanos afá

En código:

```
afá x7 nicó tócu;
```

<booleanoTipo> ::= afá id <asignacionX> ;

<asignacionX> ::= nicó <booleano>

<asignacionX> ::= ε

<booleano> ::= tócu

<booleano> ::= maíca

Sistema de asignación de registros poréteca … turé

En código:

```
##Esto es el equivalente
poréteca
    chátené campo1;
    malhióca campo2;
    ílo campo3;
    afá campo4;
turé X nicó .... ;
```

<reg> ::= poréteca <listaDatos> turé id <asignacionX> ;

<listaDatos> ::= <nuevoDato>

<listaDatos> ::= <nuevoDato> <listaDatos>

<asignacionX> ::= ε

Tipos de datos

En código:

```
chátené campo1;
malhióca campo2;
ílo campo3;
afá campo4;
```

<tiposDeDatos> ::= <tipoDato> id <asignacionX> ;

<asignacionX> ::= ε

<tipoDato> ::= chátené

<tipoDato> ::= malhióca

<tipoDato> ::= ílo

<tipoDato> ::= afá

<tipoDato> ::= apúchá

<tipoDato> ::= Ihalámi

<tipoDato> ::= jané

<tipoDato> ::= coyé <tipoDato> id ta

Literales de conjuntos

En código:

```
apúchá x8 nicó {x4,x5,x6,x7};
```

<S> ::= apuchá id <asignacionX> ;

<asignacionX> ::= nicó { <conjunto> } | ε

<conjunto> ::= id

<conjunto> ::= id, <conjunto>

Literales de archivos

En código:

```
lhalámi x9 nicó {{ "c:\akw\tarea1.txt" , L }} ;
```

<S> ::= lhalámi id <asignacionX> ;

<asignacionX> ::= nicó {{ <stringRuta>, <accionArchivo> }} | ε

<accionArchivo> ::= L

<accionArchivo> ::= S

<accionArchivo> ::= D

Literales de números flotantes

En código:

```
jané x10 nicó 1.89;
```

Es importante resaltar que los literales de los flotantes en Táfac son iguales a los del lenguaje C, por lo que la gramática de C influenció fuertemente nuestra propuesta.

<S> ::= jané id <asignacionX> ;

<asignacionX> ::= nicó <flotante> | ε

<flotante> ::= <entero>. <parte_fraccionaria> <exponente>

<exponente> ::= E + <entero>

<exponente> ::= E - <entero>

<exponente> ::= E <entero>

<exponente> ::= €

<entero> ::= <digito>

<entero> ::= <digito> <entero>

<entero> ::= €

<parte_fraccionaria> ::= <digito>

<parte_fraccionaria> ::=< digito> <parte_fraccionaria>

<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Literales de enteros

En código:

```
chátené x4 nicó 23;
```

Es importante resaltar que los literales de los enteros en Táfac son iguales a los del lenguaje C, por lo que la gramática de C influenció fuertemente nuestra propuesta.

<S> ::= chátené id <asignacionx> ;

<asignacionX> ::= nicó <entero> | ε

<entero> ::=< entero_decimal>

<entero> ::=< entero_octal>

<entero> ::=< entero_hexadecimal>

<entero_decimal> ::= <digito_no_cero>

<entero_decimal> ::=< digito_no_cero><digitos_decimales>

<entero_decimal> ::= 0

<entero_octal> ::= 0

<entero_octal> ::= 0 <digitos_octales>

<entero_hexadecimal> ::= 0 X<digitos_hexadecimales>

<digitos_decimales> ::=< digito_decimal>

<digitos_decimales> ::=< digito_decimal><digitos_decimales>

<digitos_octales> ::=< digito_octal>

<digitos_octales> ::=< digito_octal><digitos_octales>

<digitos_hexadecimales> ::= <digito_hexadecimal>

<digitos_hexadecimales> ::=< digito_hexadecimal><digitos_hexadecimales>

<digito_no_cero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<digito_decimal> ::= 0 | <digito_no_cero>

<digito_octal> ::= 0 | 1 | 2 |3|4|5|6|7

<digito_hexadecimal> ::=< digito_decimal> | A | B | C | D | E|F

Literales de caracteres

En código:

```
malhióca x11 nicó ':';
```

Es importante resaltar que los literales de los caracteres en Táfac son iguales a los del lenguaje C, por lo que la gramática de C influenció fuertemente nuestra propuesta. Se evita

escribir todos los posibles caracteres por comodidad visual, pero se entiende que son los mismos que se aceptarían en C.

<S> ::= malhióca id <asignacionX> ;

<asignacionX> ::= nicó <caracter> | ε

<caracter> ::= '<caracter_literal>'

<caracter> ::= '<escape_secuencia>'

<caracter_literal> ::= cualquier carácter visible excepto' y

<escape_secuencia> ::=< combinacionesValidasC>

Literales de strings

En código:

```
ílo x6 nicó "Esto es un string";
```

Es importante resaltar que los literales de los strings en Táfac son iguales a los del lenguaje C, por lo que la gramática de C influenció fuertemente nuestra propuesta. Se evita escribir todos los posibles caracteres por comodidad visual, pero se entiende que son los mismos que se aceptarían en C.

<S> ::= ílo id <asignacionX> ;

<asignacionX> ::= nicó <string> | ε

<string> ::= "<string_literal> <escape_secuencia>"

<string_literal> ::= <combinacionesStringsC>

<string_literal> ::= €

<escape_secuencia> ::=< combinacionesValidasC>

<escape_secuencia> ::= €

Literales de arreglos

En código:

```
coyé chátené x12 nicó {[ 1,2,3,4 ]};
```

<S> ::= coyé <tipoDato> id <asignacionX> ;

<asignacionX> ::= nicó <arreglo> | ε

<arreglo> ::= { [<datosDeTipoDato>] }

<datosDeTipoDato> ::= <dato>

<datosDeTipoDato> ::=< dato>, <datosDeTipoDato>

<dato> ::= id

<dato> ::= <datoDeTipoDato>

Literales de registros

En código:

```
porétecá                                    ## con
    chátené campo1;                         ## Los
    malhióca campo2;                        ## lue
    ilo campo3;
    afá campo4;
turé X nicó {@ 1, 'a', "string", x7 @} ;
```

<S> ::= porétecá <listaDatos> turé id <asignacionX> ;

<asignacionX> ::= nicó <registro> | ε

<registro> ::= {@ <datosCorrespondientes> @ }

<datosCorrespondientes> ::= <dato>

<datosCorrespondientes> ::= <dato>, <datosDeTipoDato>

<dato> ::= id

<dato> ::= <datoCorrespondiente>

Sistema de acceso a arreglos

En código:

```
x5[1] := 6;
```

<S> ::= id [<accesador>] :=< dato> ;

<accesador> ::= id

<accesador> ::= <entero>

Sistema de acceso a strings

En código:

```
x6 $$ 3;
```

<S> ::= id $$ <entero> ;

Sistema de acceso a registros

En código:

```
campo1@A := 25;
```

<S> ::= idDato @ idRegistro :=< datoTipoDato> ;

Asignación con: := += -= *= %= /=

En código:

```
chátené x4 nicó 23;
campo1@A := 25;

x4 += 2;
x4 -= 1;
x5 *= 10;
x6 %= 2;
x7 /= 3;
```

<asiganciones> ::= idDato <asignacion> <datoTipoDato> ;

<asignacion> ::= :=

<asignacion> ::= +=

<asignacion> ::= -=

<asignacion> ::= *=

<asignacion> ::= %=

<asignacion> ::= /=

Operaciones aritméticas básicas de enteros: + - * % /

En código:

```
chátené x4 nicó 23;
chátené x5;

x5 := x4 + 5;
x5 := x4 - 3;
x5 := x4 * 4;
x5 := x4 / 2;
x5 := x4 % 6;
```

<S> ::=< dato> <operacionEntero> <dato> ;

<dato> ::= id

<dato> ::= <entero>

<operacionEntero> ::= *

<operacionEntero> ::= /

<operacionEntero> ::= %

<operacionEntero> ::= +

<operacionEntero> ::= -

Decremento e incremento de variables: jené y taqué

En código:

```
chatené x nicó 23;
x jené;
```

<DecIncremento> ::= id <cremento>

<cremento> jené

<cremento> taqué

Operaciones básicas sobre caracteres: >> << &? #?

En código:

```
##definimos el caracter
malhióca char1 nicó 'f';

##Mayuscula
>> char1;
→ 'F'

##minúscula
<< char1;
→ 'f'

##is Alpha (o sea letra)
&? char1;
→ tócu

##is digit (numero)
#? char1;
→ maíca
```

<operacionesCaracteres> ::= <operadorCaracter> id

<operadorCaracter> ::= >>

<operadorCaracter> ::=<<

<operadorCaracter> ::= &?

<operadorCaracter> ::= #?

Operaciones lógicas: ta yú pucá emé

En código:

```
##Definimos el booleano
afá b1 nicó tócu;
afá b2 nicó maíca;

## and
b1 ta b2;
→ maíca

## or
b1 yú b2;
→ tócu

## xor
b1 pucá b2;
→ tócu

## not
b1 emé
→ maíca
```

<operacionesLogicas> ::= id <operadorBooleano> id ;

<operadorBooleano> ::= ta

<operadorBooleano> ::= yú

<operadorBooleano> ::= pucá

<operadorBooleano> ::= emé

Operaciones de strings: Concatenar: catáqui / Length: lharátesufá / cortar: joné / recortar: nhalh / encontrar: cochéqui

En código:

```
##Definir las variables
ilo string1 nicó "Esto es un string";
ilo string2 nicó "el veloz murciélago";

##concatenar
string1 catáqui string2
→ "Esto es un stringel veloz murciélago"

## lenght
lharátesufá string1
→ 17

## cortar
joné[0:5] string1
→ "Esto e"

##recortar
nhalh[0:5] string2
→ "oz murciélago"

##encontrar
cochéqui["un"] string1
→ 8
```

<operacionesStrings> ::= id catáqui id

<operacionesStrings> ::= lharátesufá id

<operacionesStrings> ::= <operadorManejoString> [<literalEntera> : <literalEntera>] id

<operadorManejoString> ::= joné

<operadorManejoString> ::= nhalh

<S> ::= cochéqui [<literalString>] id

Operaciones con conjuntos: Agregar: erronh / Borrar:quené / Union: fuinhye / Interseccion: sulí / PerteneceA: pcalúfa / Vacío?: culíte

En código:

```
## agregar
erronh con1 'x5';
→ {'a1','x1','x2','x3','x4','x5'}

## borrar
quené con1 'x1';
→ {'a1','x2','x3','x4'}

## Union
fuinhye con1 con2
→ {'a1','x1','x2','x3','x4','a1','y1','y2','y3','y4'}

## intersección
sulí con1 con2
→ {'a1'}

## pertenece
pcalúfa con1 'a1'
→ tócu

## vacio
culíte con1
→ maíca
```

<operacionesConjuntos> ::= <operadorConjunto> id <literalX>

<operadorConjunto> ::= erronh

<operadorConjunto> ::= quemé

<operadorConjunto> ::= pcalúfa

<operacionesConjuntos> ::= <operadoConjunto> id id

<operadorConjunto> ::= sulí

<operadorConjunto> ::= fuinhye

<operacionesConjuntos> ::= <operadorConjunto> id

<operadorConjunto> ::= culíte>

Operaciones con archivos: Abrir: quijí / Cerrar: eloc / Escribir: lhioc / Leer: lhaic / Crear: quijérri / Asociar: láte

En código:

```
lhalámi cheto nicó {{ "c:\akw\tarea1.txt" , L }}

## abrir
quijí cheto;

##cerrar
eloc cheto;

## escribir
lhioc cheto "texto a escribir";

## crear
quijérri muffin;

## asociar
láte muffin cheto;
```

<manejoArchivo> ::= <operadorArchivo> id

<manejoArchivo> ::= <operadorArchivo> id id

<manejoArchivo> ::= <operadorArchivo> id <literalString>

<operadorArchivo> ::= quijí <operadorArchivo> ::= eloc

<operadorArchivo> ::= lhioc

<operadorArchivo> ::= quijérri

<operadorArchivo> ::= láte

Operaciones con números flotantes:|++ ++| |+| +|+

En código:

```
jané f1 nicó 1.23;
jané f2 nicó 41.77;

## suma
f1 |++ f|
→ 43.000

## resta
f1 ++| f2
→ -40.54

## multiplicacion
f1 |+| f2
→ 51.3771

## division
f1 +|+ f2
→ 0.00294469
```

<operacionesFlotantes> ::= id <operadorFlotante> id

<operacionesFlotantes> ::= <literalFlotante> <operadorFlotante> id

<operacionesFlotantes> ::= id <operadorFlotante> <literalFlotante>

<operadorFlotante> ::= |++

<operadorFlotante> ::= ++|

<operadorFlotante> ::= |+|

<operadorFlotante> ::= +|+

Operaciones de comparación: < > = >= <= ><

En código:

<operaracionesComparacion> ::= id <operadorComparacion> id

<operaracionesComparacion> ::= id <operadorComparacion> <literalX>

<operaracionesComparacion> ::= <literalX> <operadorComparacion> id

<operadorComparacion> ::= <

<operadorComparacion> ::= >

<operadorComparacion> ::= =

<operadorComparacion> ::= >=

<operadorComparacion> ::=<=

<operadorComparacion> ::= ><

Manejo de bloques de más de una instrucción: cojcoj … fuchí

En código:

```
cojcoj
    pítechátené i;
    íca i;
fuchí
```

<bloqueInstrucciones> ::= cojcoj <loop-instrucciones> fuchí

<loop-instrucciones> ::= ε

<loop-instrucciones> ::= <instruccion> <loop-instrucciones>

<instruccion> ::= <asigConst> | <asigTipo> | <asigVar> | <booleanoTipo> | <reg> | <llamada-rutina> | halt | <while> | <for> | <repeat-until> | <if-then-else>

Instrucción while

En código:

```
##ejemplo de un while
ótacá contador >= 0 sóta ## while(contador >= 5) do
cojcoj
    contador jené;       ## esto es un contador--
    aumentador taqué;    ## esto es un aumentador++
    taráne               ## continue (no es obligatorio)
fuchí
```

<bloque-while> ::= ótacá <expresión-booleana> sóta <loop-instrucciones>

Instrucción if

En código: similar a if-then-else pero sin el segundo bloque. En la siguiente se puede ver un ejemplo.

<bloque-if> ::= arú <expresión-booleana> nocófa <bloque-código>

Instrucción if-then-else

En código:

```
##ejemplo de if-then-else
arú aumentador > 5 nocófa    ## if (aumentador > 5) then
cojcoj
    aumentador := 0;
    contador := 0;
fuchí
tiáfa                        ## else
cojcoj
    aumentador := 5;
    contador := 5;
fuchí
```

<bloque-if-else> ::= arú <expresión-booleana> nocófa <bloque-código> tiáfa <bloque-código>

Instrucción switch

En código:

```
torré Letra      ## torré es switch
cojcoj
    ári 'a' :    ## ári es el case
    ári 'e' :
    ári 'i' :
    ári 'o' :
    ári 'u' :
        jalac tócu; ## jalac es return, o
        foróforóye; ## este es el break
    mailhíco:       ## este es el default
        jalac maíca;## return false
fuchí
```

<bloque-switch> ::= torré <identificador> cojcoj <loop-casos> fuchí

<loop-casos> ::= ári <literal> : <loop-instrucciones> <loop-casos>

<loop-casos> ::= ári <literal> : <loop-casos>

<loop-casos> ::= mailhíco : <loop-instrucciones>

Instrucción repeat-until

En código:

```
## Ejemplo de repeat-until
paípaí  ## do
    aumentador += 1;
yája aumentador < 10; ## while
```

<bloque-repeat-until> ::= paípaí <loop-instrucciones> yája <expresión-booleana>

Instrucción for

En código:

```
##ejemplo de un for
ajá chátené i := 5 Iha i >= 0 telele i jené sóta ## fo
cojcoj              ## el cojcoj al igual que el {} en
    pítechátené i;  ## píte es salida estandar o sea e
    íca i;          ## sizeof de i
fuchí
```

<for> ::= ajá <tiposDeDatos> <asiganciones> Iha <operaracionesComparacion> telele id

<DecIncremento> sóta <instruccion>

Instrucciones break, continue y halt

En código:

```
chátené contador nicó 10;
chátené aumentador nicó 0;

ótacá(contador >= 0) sóta
cojcoj
    contador jené;
    aumentador taqué;
    arú (aumentador = 5) nocófa
        foróforóye
    taráne
fuchí

lauc
```

<break> ::= foróforóye

<continue> ::= taráne

<halt> ::= lauc

Encabezado de funciones

<encabezadoFun> ::= macorróca <tipo> id ( )

Encabezado de procedimientos

<encabezadoProc> ::= óri id ( )

Manejo de parámetros formales

<parametroFormal> ::= (<tiposDeDatos> id, <tiposDeDatos> id, sui <tiposDeDatos> id)

Manejo de parámetros reales

<parámetrosReales> ::= id (a, b, c)

Instrucción return

<return> ::= jalac id;

<return> ::= jalac <literal> ;

<return> ::= jalac ;

Operación sizeof

<sizeof> ::= icá id;

Sistema de coerción de tipos

<coerción> ::= tirríque (<tiposDeDatos>) id ;

Salida Estándar

<salidaES> ::= lac<tiposDeDatos> id

Entrada Estándar

<entradaES> ::= píte<tiposDeDatos> id

Pruebas de análisis léxico

Test de comentarios

Nombre del archivo: test_comentarios.taf

Objetivo: probar la correcta detección de comentarios en las distintas partes del programa

```
--- exec:3.1.0:exec (default-cli) @ Lexico ---



Tipo: EOF  Lexema:   Linea:5  Columna:1

Compilacion correcta.
```

Este archivo solo genera el token de End Of File, ya que los comentarios son ignorados por el Scanner

Test de Seccion de constantes

Nombre del archivo: test_seccion_constantes.taf

Objetivo: probar la correcta detección de comentarios en las distintas partes del programa

```
Tipo: SECCION  Lexema: octar□  Linea:2  Columna:1
Tipo: ASIGNACION  Lexema: arr□  Linea:4  Columna:2
Tipo: IDENTIFICADOR  Lexema: x1  Linea:4  Columna:7
Tipo: LITERAL_ENTERO  Lexema: 45  Linea:4  Columna:10
Tipo: TERMINADOR  Lexema: ;  Linea:4  Columna:12
Tipo: ASIGNACION  Lexema: arr□  Linea:5  Columna:2
Tipo: IDENTIFICADOR  Lexema: x2  Linea:5  Columna:7
Tipo: LITERAL_FLOTANTE  Lexema: 1.5  Linea:5  Columna:10
```

genera los tokens correctamente (el símbolo □ es porque la consola de java no soporta tildes)

Muro de ladrillos del ejemplo:

Tokens

Lexema	Fila	Columna	Tipo
octará	2	1	SECCION
arrá	4	2	ASIGNACION
x1	4	7	IDENTIFICADOR
45	4	10	LITERAL_ENTERO
;	4	12	TERMINADOR
arrá	5	2	ASIGNACION
x2	5	7	IDENTIFICADOR
1.5	5	10	LITERAL_FLOTANTE
;	5	13	ERROR
	7	1	EOF

Contador de Tokens

- TERMINADOR: 1
- LITERAL_FLOTANTE: 1
- LITERAL_ENTERO: 1
- SECCION: 1
- ERROR: 1
- ASIGNACION: 2
- IDENTIFICADOR: 2
- EOF: 1

Pruebas de análisis sintáctico

Se crearon una serie de archivos .taf con el fin de probar el análisis de la estructura sintáctica de los programas en Táfac. En esta sección se detallará el contenido de cada uno y su intención

Test de comentarios

Nombre del archivo: test_comentarios.taf

Objetivo: probar la correcta detección de comentarios en las distintas partes del programa

Test de fors

Nombre del archivo: test_for.taf

Objetivo: probar tanto la correcta detección de estructuras de ciclos for en el programa, así como sus posibles errores

Test de ifs

Nombre del archivo: test_ifs.taf

Objetivo: probar la correcta detección de estructuras de if then else dentro del programa, también sus posibles errores.

Test de inicio incorrecto

Nombre del archivo: test_mal_inicio.taf

Objetivo: probar la correcta iniciación del programa. Si el programa no tiene el iniciador seguido explícitamente de un nombre, da error.

Test de repeat until

Nombre del archivo: test_repeat_until.taf

Objetivo: probar la correcta detección de la estructura repeat until y sus posibles errores como falta de expresión booleana.

Test de sección de constantes

Nombre del archivo: test_sin_constantes.taf

Objetivo:Al ser la sección de constantes opcional, esta prueba tiene el fin de probar el correcto funcionamiento en el caso de que esta sección no esté definida, es decir, que el programa no encuentre errores sintácticos.

Test de falta de inicio

Nombre del archivo: test_sin_inicio.taf

Objetivo: probar la correcta detección de la falta de iniciación del programa, es decir, no existe la cláusula que inicia el programa con su nombre.

Test de sección de prototipos

Nombre del archivo: test_sin_prototipos.taf

Objetivo: Al ser la sección de prototipos opcional, esta prueba tiene el fin de probar el correcto funcionamiento del caso de que esta sección no esté definida, es decir, que el programa no encuetre errores sintácticos.

Test de sección de rutinas

Nombre del archivo: test_sin_rutinas.taf

Objetivo: Al ser la sección de rutinas opcional, esta prueba tiene el fin de probar el funcionamiento del caso de que esta sección no esté definida, es decir, que el programa no encuentre errores sintácticos.

Test de sección de tipos

Nombre del archivo: test_sin_tipos.taf

Objetivo: Al ser la sección de tipos opcional, esta prueba tiene el fin de probar el funcionamiento del caso de que esta sección no esté definida, es decir, que el programa no encuentre errores sintácticos.

Test de sección de variables

Nombre del archivo:test_sin_variables.taf

Objetivo: Al ser la sección de variables opcional, esta prueba tiene el fin de probar el funcionamiento del caso de que esta sección no esté definida, es decir, que el programa no encuentre errores sintácticos.

Test de while

Nombre del archivo: test_while.taf

Objetivo: Probar la correcta detección de la estructura while a lo largo del programa, así como sus posibles errores sintácticos.

Funcionamiento del scanner

El Scanner es una clase de Java con distintos métodos para conseguir un análisis de cada token del código a compilar. Los distintos métodos poseen manejo de errores mediante las excepciones proveídas por Java. El método principal de este componente es llamado scan(). Esta función crea una lista de tokens y la va llenando con los tokens que saca del archivo de texto mediante el método demeToken(). Este método utiliza los contadores de la clase Scanner para saber por qué parte del código fuente se va escaneando. Primero verifica que no se encuentre al final del archivo, de ser así envía directamente el token de EOF o “End Of File” para indicar la terminación del escaneo. Si no se encuentra en el fin del archivo, revisa si el caracter actual es una letra. De ser así, entra en un ciclo donde se guarda el caracter actual y se pide el siguiente hasta que el caracter actual no sea ni un número ni una letra. Finalmente, una vez concluído el ciclo, se tiene un token formado, por lo que se puede proceder a clasificarlo por tipo de token. En caso de no ser una letra, revisa si es un número. Si es un número, pide caracteres en ciclo revisando que sean números o caracteres pertenecientes a las literales numéricas. De no ser así, retorna un error. Por el contrario, continúa hasta terminar de leer el número completo y retornarlo. Por otra parte, si el token actual no es ni un número ni una letra, analiza si es un token válido, como un comentario por ejemplo. Finalmente, con el token armado, procede a detectar si es un operador válido o, a retornar un error de caracter monstruo.

A lo largo de este método se usan otras funciones como obtenerLineaError(), demeCaracter(), reportarError(), obtenerLineaError() que poseen funcionamientos más sencillos y cuyo nombre es completamente autodescriptivo.

Funcionamiento del parser

El Parser es una clase de Java con disintos métodos para conseguir un análisis sintáctico de la lista de tokens obtenida del paso anterior. En este caso se utilizan las estructuras de datos proporcionadas por el lenguaje de programación para la tabla hash. Al crear una instancia de la clase Parser, se crea a su vez una única instancia de la tabla de parseo para asegurar la integridad de los datos. Existen a su vez, métodos para calcular el First, Follow, Predict para realizar el parseo. La clase posee un método llamado parse(tokens, table, grammar) que utiliza las estructuras de datos definidas, lista de tokens provenientes del analizador morfológico y la lista que define la gramática para analizar si el orden de los tokens es sintácticamente correcto. Utiliza un ciclo para recorrer toda la lista de tokens y a su vez revisar que no haya errores gramaticales mediante la pila de parseo y las producciones obtenidas. La lista “grammar” guarda las reglas gramaticales en las que se basará el análisis sintáctico.