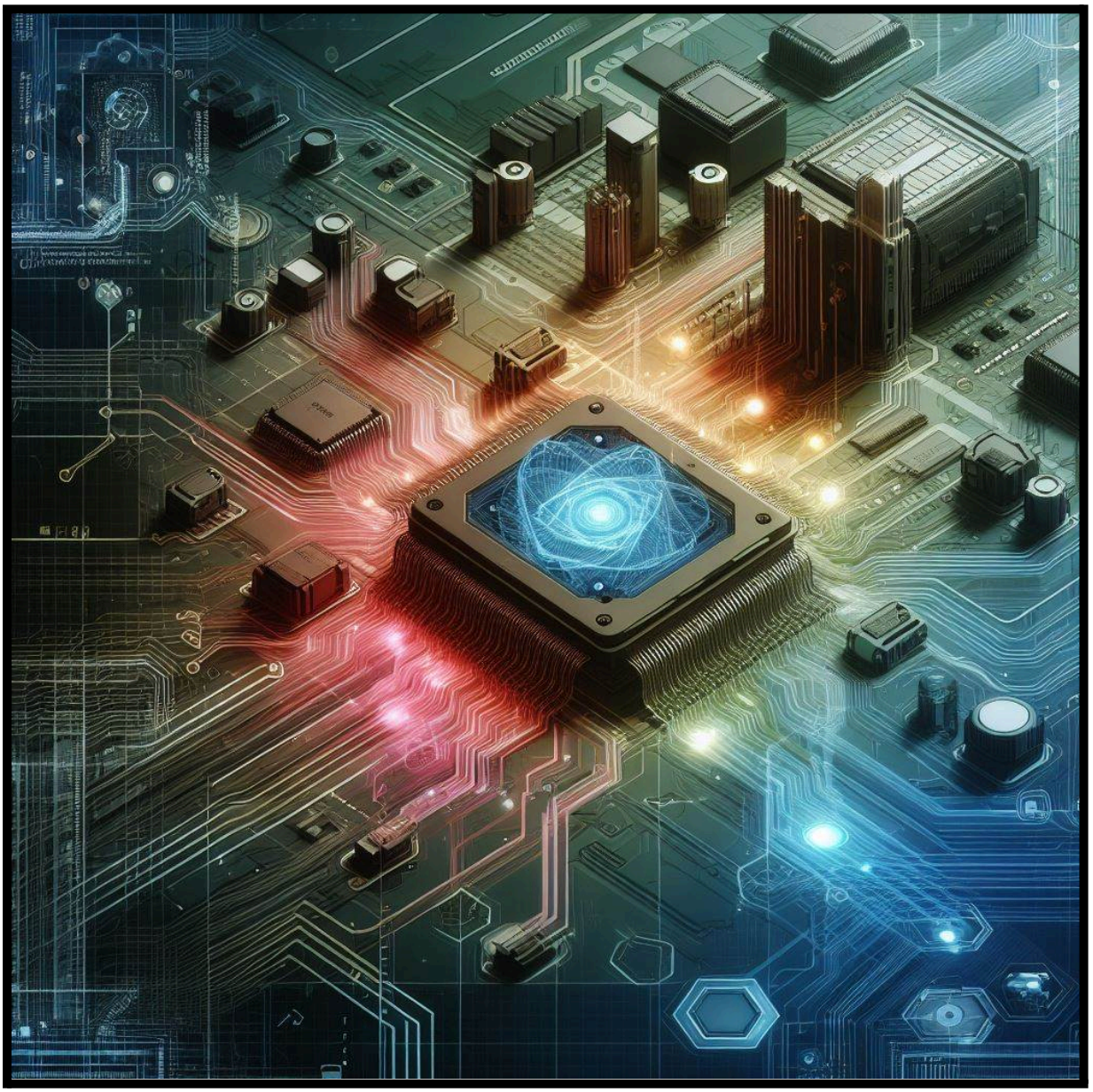
Memoria práctica 1:

Procesadores de Lenguajes:

Integrantes: Suhuai Chen, Daniel Pizarro, David Starry y Jiahui You



Índice:

0. Apéndice A

Tiny(0) es un subconjunto de **Tiny** que incluye únicamente las siguientes características:

- Declaraciones de variables, con tipos básicos **int**, **real** y **bool**.
- Únicamente instrucciones básicas. Las expresiones asociadas incluyen únicamente:
  - Literales enteros, reales, y booleanos (**true** y **false**).
  - Variables.
  - Operadores aritméticos +,-,* y /, menos unario (-), los operadores lógicos **and**, **or** y **not**, los operadores relacionales (<,>,<=,>=,==,!=), y el operador de asignación (=).
- Las prioridades y asociatividades de estos operadores son las de **Tiny**. Como en **Tiny**, pueden utilizarse paréntesis para alterarlas. Ejemplo de programa Tiny(0)

1. Tiny(0)

1.1 Especificación léxica:

1.1.1 (Paso 1) Identificar las **clases léxicas**

- Identificador (el nombre de un variable)
- Literal entero (número entero positivo o negativo)
- Literal real (consta de una parte entera seguido de una parte fraccionaria, exponencial, o fraccionaria y exponencial)
- Booleano (verdadero o falso)
- Una clase léxica por cada **operador**: suma, resta, multiplicación, división, módulo entero, menos unario, and, or, not, operadores relacionales (<,>,<=,>=,==,!=).
- Una clase léxica por cada **símbolo de puntuación**: paréntesis de apertura, paréntesis de cierre, igual { } && @ ;
- Una clase léxica por cada **palabra reservada**: true, false, and, or, not.

1.1.2 (Paso 2) Identificar cadenas ignorables

- Espacio en blanco, retroceso (\b), retorno de carro (\r), tabulador (\t), salto de línea (\n), fin de declaración

1.1.3 (Paso 3) **Especificación formal** mediante definiciones regulares

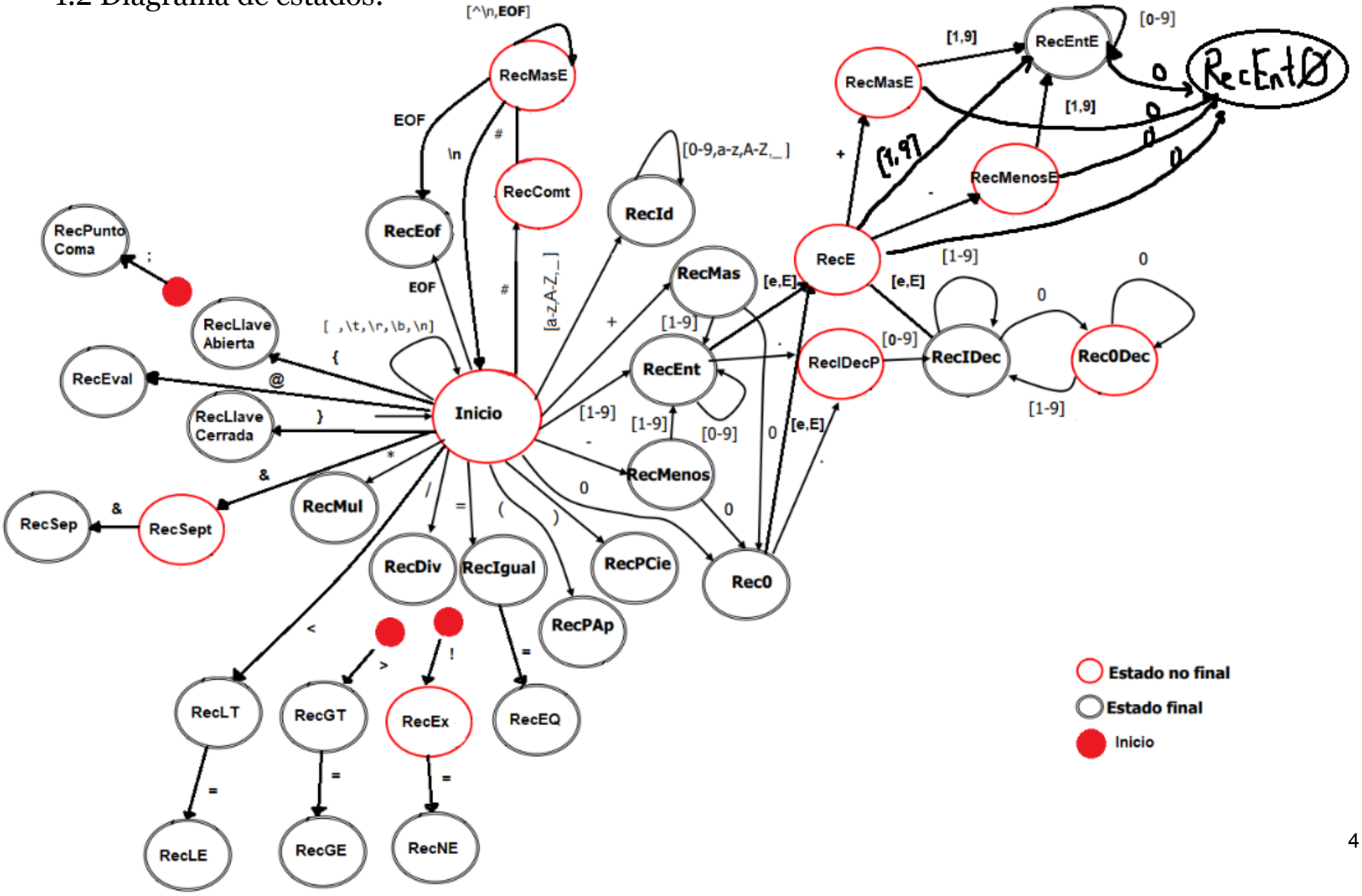
Definiciones auxiliares
letra ≡ [a-z,A-Z,_]
digitoPositivo ≡ [1-9]
digito ≡ {digitoPositivo} | 0
parteEntera ≡ {digitoPositivo} {digito}* | 0
parteDecimal ≡ {digito}* {digitoPositivo} | 0
parteExponential ≡ {e | E} {\+ | \-} {parteEntera}

Definiciones léxicas
true ≡ {T|t} {R|r} {U|u} {E|e}
false ≡ {F|f} {A|a} {L|l} {S|s} {E|e}
and ≡ {A|a}{N|n}{D|d}
or ≡ {N|n}{R|r}
not ≡ {N|n}{O|o}{T|t}
bool ≡ {B|b}{O|o}{O|o}{L|l}
real ≡ {R|r}{E|e}{A|a}{L|l}
int ≡ {I|i}{N|n}{T|t}
identificador ≡ {letra}({letra}|{digito})*
literalEntero ≡ [\+,\-]?{parteEntera}
literalReal ≡
{literalEntero}{(.{parteDecimal})|{parteExponential}|(.{parteDecimal}){parteExponential}}

suma ≡ \+
resta ≡ \-
mul ≡ *
div ≡ \/
mayor ≡ >
mayorIgual ≡ >=
menor ≡ <
menorIgual ≡ <=
igual ≡ ==
distinto ≡ !=
asignar ≡ =
puntoycoma ≡ ;
parentesisApertura ≡ \(
parentesisCierre ≡ \)
llaveApertura ≡ {
llaveCierre ≡ }
eval ≡ @
separador ≡ &&

Definiciones de cadenas ignorables
separador [,\t,\r,\b,\n]
comentario ##([^\n,EOF])*

1.2 Diagrama de estados:



2. Tiny

El lenguaje **Tiny** es un pequeño lenguaje de programación imperativo, cuyo propósito es ejercitar las distintas técnicas descritas en la asignatura de "Procesadores de Lenguajes". A continuación, se realiza una descripción completa, por niveles, de este lenguaje. También se muestra un programa de ejemplo, que lee una serie de nombres en un 'array', forma un árbol de búsqueda con dichos nombres, y utiliza dicho árbol para escribir los nombres ordenados y sin duplicados.

2.1 Especificación léxica:

2.1.1 (Paso 1) Identificar las **clases léxicas**

- identificador: el nombre de un variable.
- literal entero: número entero positivo o negativo.
- literal real: consta de una parte entera seguido de una parte fraccionaria, exponencial, o fraccionaria y exponencial.
- Literal Cadena: una secuencia de caracteres que no sean ", EOF.
- booleano: verdadero o falso.
- Una clase léxica por cada **operador**: suma, resta, multiplicación, división, menos unario, and, or, not, operadores relacionales (<,>,<=,>=,==,!=).
- Una clase léxica por cada **símbolo de puntuación**: paréntesis de apertura, paréntesis de cierre, igual, { } && @ ; [ ] . , ^ &
- Las siguientes palabras reservadas: **int**, **real**, **bool**, **string**, **and**, **or**, **not**, **null**, **true**, **false**, **proc**, **if**, **else**, **while**, **struct**, **new**, **delete**, **read**, **write**, **nl**, **type**, **call**.

2.1.2 (Paso 2) Identificar cadenas ignorables

- Espacio en blanco, retroceso (\b), retorno de carro (\r), tabulador (\t), salto de línea (\n)
- Comentarios: Son comentarios *de línea*. Comienzan por **##**, seguido de una secuencia de 0 o más caracteres, a excepción del salto de línea.

2.1.3 (Paso 3) **Especificación formal** mediante definiciones regulares

Definiciones auxiliares

letra ≡ [a-z,A-Z,_]
digitoPositivo ≡ [1-9]
digito ≡ {digitoPositivo} | 0
parteEntera ≡ {digitoPositivo} {digito}* | 0
parteDecimal ≡ {digito}* {digitoPositivo} | 0
parteExponential ≡ {e | E} {\+ | \-} {parteEntera}

Definiciones léxicas

true ≡ {T|t} {R|r} {U|u} {E|e}
false ≡ {F|f} {A|a} {L|l} {S|s} {E|e}
and ≡ {A|a}{N|n}{D|d}
or ≡ {N|n}{R|r}
not ≡ {N|n}{O|o}{T|t}
bool ≡ {B|b}{O|o}{O|o}{L|l}
real ≡ {R|r}{E|e}{A|a}{L|l}
int ≡ {I|i}{N|n}{T|t}
string ≡ {S|s}{T|t}{R|r}{I|i}{N|n}{G|g}
null ≡ {N|n}{U|u}{L|l}{L|l}
proc ≡ {P|p}{R|r}{O|o}{C|c}
if ≡ {I|i}{F|f}
else ≡ {E|e}{L|l}{S|s}{E|e}
while ≡ {W|w}{H|h}{I|i}{L|l}{E|e}
struct ≡ {S|s}{T|t}{R|r}{U|u}{C|c}{T|t}
new ≡ {N|n}{E|e}{W|w}
delete ≡ {D|d}{E|e}{L|l}{E|e}{T|t}{E|e}
read ≡ {R|r}{E|e}{A|a}{D|d}
write ≡ {W|w}{R|r}{I|i}{T|t}{E|e}
nl ≡ {N|n}{L|l}
type ≡ {T|t}{Y|y}{P|p}{E|e}
call ≡ {C|c}{A|a}{L|l}{L|l}
identificador ≡ {letra}({letra}|{digito})*
literalEntero ≡ [\+,\-]?{parteEntera}
literalReal = {literalEntero}({parteExponencial}|(\.{parteDecimal})|(\.{parteDecimal}{parteExponencial))
literalCadena ≡ \"(^[",EOF])*\"
suma ≡ \+
resta ≡ \-
mul ≡ *
div ≡ \/
mayor ≡ >
mayorIgual ≡ >=
menor ≡ <
menorIgual ≡ <=
igual ≡ ==
distinto ≡ !=

asignar ≡ =
puntoycoma ≡ ;
parentesisApertura ≡ \(
parentesisCierre ≡ \)
llaveApertura ≡ {
llaveCierre ≡ }
eval ≡ @
modulo ≡ %
corcheteApertura ≡ \[
corcheteCierre ≡ \]
punto ≡ \.
coma ≡ \,
circunflejo ≡ \^
ampersand ≡ &
separador ≡ &&

Definiciones de cadenas ignorables
separador [,\t,\r,\b,\n]
comentario #([^\n,EOF])*