Documentación

Wogawg

Proyecto - Compilador en Cabécar

Etapa 4 - Generador de Código

Prof. Kirstein Gätjens S.

Compiladores e Intérpretes GR 2

Instituto Tecnológico de Costa Rica

Sede Central

Erick Kauffmann, c. 2022180244

Samuel Valverde, c. 2022090162

II Semestre 2024

Índice

Definición del lenguaje

Estructura del programa

Estructura del título del programa:

wögawg <id> .

Esta palabra significa “enciende el fuego”, por lo que es una buena forma de abrir el programa, indicando que es el principio del mismo.

Uso: Wögawg Programa .

Apertura de Secciones

Sección constantes:

kará - tieso

Escogimos esta palabra para la apertura de la sección de constantes ya que significa “tieso”, lo cual relacionamos con las constantes ya que estas no cambian, al igual que algo tieso.

Uso: kará

```
$ sección de las constantes del programa
    Ñákjëë Shtáwa Constante1 wáka 27   .
    Ñákjëë ékla nombrewáka 'A'.
    Ñákjëë shéj Constante3 wáka "Hola mundo" .
```

Sección de tipos:

klo - árbol

Se escogió para la sección de tipos la palabra en cabécar “klo” la cual significa árbol

Uso: klo

```
$ sección de los tipos del programa
    Edad :> Shtáwa .
    Garabato :> ékla .
    vector :> Kichána Shtáwa bikö 100 .   $ vector de 100 enteros
    matriz :> Kichána ékla bikö 20 bikö 30 bikö 25 .   $ matriz tridimencional de caracteres
```

Sección de variables:

kaska - cambio

Se utiliza la palabra en cabécar “kaska” para la sección de variables ya que esta significa “cambio” lo que se relaciona con el comportamiento de las variables.

Uso: kaska

```
$ sección de variables
    Wömële Num1 := w|CCCXXV|w.
    Shtáwa X:=1,Y,Z:=5 . $ hay declaraciones múltiples y la inicialización es opcional
```

Sección de prototipos:

wäkiri - patrón

Se utiliza la palabra “wäkiri”, la cual en cabécar significa patrón.

Uso: Wäkiri

```
$ sección de prototipos
    kiana fibonacci ( Shtáwa N ) chani Shtáwa .  $ la lista de parámetros es obligatoria en los prototipos
    wayuä Imprimir3Mensajes ( Shéj msj,msj2 . Shéj msj3 ) .
```

Sección de rutinas:

prá - cuerpo

Se utiliza la palabra en cabecar “prá” la cual se traduce como cuerpo.

Uso: prá

```
$ sección de rutinas
    fibonacci ( Shtáwa N ) chani Shtáwa shä
     ká
        chagö N<2 jéná
            toli 1 .
        shirína
            toli fibonacci(däwá N)+fibonacci(däwá däwá N)..
    tuína .
```

Punto de entrada

Punto de entrada del programa:

Jarree - rechinar de una puerta

Se utiliza la palabra en cabécar “Jarree”, porque simboliza el sonido que hace el rechinar de una puerta.

Uso: Jarree

$resto del programa

Sistemas de asignación

Sistema de asignación de constantes:

Ñákjëë <tipo> id <literal> .

Se escogió la palabra en cabécar “Ñákjëë”, que significa uniforme.

Uso: Ñákjëë ékla Constante2 wáka 'A' .

Sistema de asignación de tipos:

id :> <tipo> .

Se utiliza el símbolo “:>” como una variante del “::=”

Uso: Edad :> Shtáwa .

Sistema de declaración de variables:

<tipo> id := <inicialización> .

Se utilizará el símbolo “:=” para declarar variables.

Uso: Shtáwa X:=1,Y,Z:=5 .

Tipos de datos

Tipo de dato entero:

Shtáwa

Se eligió la palabra en cabécar "shtáwa" porque significa contar números.

Uso: Shtáwa X wáka 1 .

Tipo de dato caracter:

ékla

Se utiliza la palabra en cabécar “ékla” porque significa “uno”, como el tipo de dato caracter, que es un solo caracter.

Uso: ékla X wáka ‘a’ .

Tipo de dato string:

Shéj

Se utiliza la palabra “shéj” que en cabécar significa tira de bejuco

Uso: shéj X wáka “abcdefg” .

Tipo de dato booleano:

YínaKulíwa

Se utiliza la palabra “YínaKulíwa” que es la combinación de las palabras en cabécar “Yína” y “Kulíwa” que significan Verdadero y Falso respectivamente.

Uso: YínaKulíwaX wáka Síwa.

Tipo de dato conjunto:

Járjá

Se escogió la palabra en cabécar “járjá” que significa conjunto.

Uso: Járjá Vocales wáka {: 'A' ,'E','I', 'O','U':} .

Tipo de dato archivo de texto:

älwíí

Se escogió la palabra en cabécar “äliwíí” porque significa libro.

Uso: älwíí Texto wáka {/ "salida.txt" , 'E' }.

Tipo de datos números flotantes:

Gache

Se escogió la palabra en cabécar “gache” porque significa trozo o pedazo de algo.

Uso: gache X wáka 19,14241 .

Tipo de dato creativo:

wömële

Se utiliza la palabra “wömële” ya que esta en cabécar significa enemigo. El dato creativo serán los números romanos.

Uso: Wömële Num1 wáka w|CCCXXV|w.

Tipo de dato arreglos:

Kichána <tipo> id bikö 100

Se utiliza la palabra en cabécar “kichána” que significa “al lado de”.

Uso: Kichána Shéj Cursos bikö 3 wáka ["Compi","Diseño","AP"] .

Tipo de dato registros:

Bók

Se utiliza la palabra en cabecar “bók” que significa mochila.

Uso: Bók Ká

Shtáwa Edad .

Shéj Nombre .

Tuína Juan wáka {*18 , "juan perez" *} .

Literales

Literales booleanas:

Síwa (verdad) - Kóyuwé (mentira)

Uso: yínakulíwa flag wáka síwa .

Literales de conjuntos:

{: ‘a’, ‘e’, ‘i’, ‘o’, ‘u’ :}

Literal predeterminado.

Uso: járjá comidas wáka {: “pinto”, “waffles”, “huevo frito con tostada” :} .

Literales de archivos:

{/ "Archivo.txt" , 'L' /}.

Literal predeterminado. Modos: 'L' 'E' 'A'

Uso: älwíí Indexes wáka {/ "indexes.xlsx" , 'L' }.

Literales de números flotantes:

-3,45

Literal predeterminado. Se utiliza , en vez de .

Uso: gache promedio wáka 88,75 .

Literales de enteros:

-123, 0xF4EC

Literal predeterminado.

Uso: shtáwa id wáka 123456 .

Literales de caracteres:

‘K’

Literal predeterminado.

Uso: ékla inicial wáka ‘s’ .

Literales de strings:

“Hola Mundo”

Literal predeterminado.

Uso: shéj bienvenida wáka “Bienvenidos al programa!” .

Literales de arreglos:

[1, 2, 3]

Literal predeterminado.

Uso: Kichána ékla letras bikö 6 wáka [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’] .

Literales de registros:

{* ”hola”, 8, ‘k’ *}

Literal predeterminado.

Uso: bók Santiago wáka {*24 , "santiago cruz", w| MMXXII |w , 'B' *} .

Sistemas de acceso

Sistema de acceso arreglos:

[2][3][4]

Este sistema permite acceder a elementos específicos dentro de un arreglo utilizando índices.

Uso: Kichána ékla letras bikö 6 wáka [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’] .

ékla[2] . $ ‘c’

Sistema de acceso strings:

"hola" &# 2

Aquí, se accede a un carácter específico en una cadena utilizando un índice, permitiendo realizar operaciones detalladas sobre los elementos individuales dentro del string.

Uso:

shéj bienvenida wáka “Bienvenidos al programa!” .
bienvenida&#4 . $ “v”

Sistema de acceso registros:

registro@campo

Este método de acceso permite extraer un elemento particular dentro de un registro, utilizando índices para identificar y operar sobre los datos almacenados en la estructura de registros.

Uso:

bók Estudiante wáka {*18 , "Samuel Valverde", w| MMXXII |w , 'F' *} .
bók@3 . $ ‘F’

Asignación

Asignación y Familia:

wáka

Utilizaremos la palabra “wáka” que significa “transformar”.

Uso: Ñákjëë ékla nombreArchivo wáka “cabecarCompiler”.

Operaciones

Operaciones aritméticas básicas de enteros:

+ - * % /

Operaciones predeterminadas.

Incremento y Decremento:

däká - däwá

La palabra en cabécar “däká” se escogió para el incremento ya que significa preguntar, y “däwá” para el decremento porque significa bajar.

Uso: däwá VariableName .

däká VariableName .

Operaciones básicas sobre caracteres:

Tölö - Jélá - Júru - Kári

Estas operaciones permiten manipular y comparar caracteres de manera sencilla y directa.

1. Tölö se escogió para isdigit?
2. Jélá se escogió para isalpha?
3. Júru se escogió para mayúscula
4. Kári se escogió para minúscula

Uso: ékla X wáka ‘a’ .

X jélá .

Operaciones lógicas solicitadas

Irä - irälé - kái - jébä

Se escogieron las siguientes palabras en cabécar para las operaciones lógicas:

1. Irä se escogió para “y”
2. Irälé se escogió para “o”
3. Kái se escogió para “no”
4. Jébä se escogió para “xor”

Uso: ká

```
    chagö N<10 irä K<100 jéná
        $instrucciones
    shirína
        $instrucciones
tuína .
```

Operaciones de Strings:

bawa - matsíí - tiä - kjätiä - kua

1. Concatenar/Yuxtaponer: se escogió la palabra en cabécar bawa
2. Largo: se escogió la palabra en cabécar matsíí
3. Cortar: se escogió la palabra en cabécar tiä
4. Recortar: se escogió la palabra en cabécar kjätiä
5. Encontrar: se escogió la palabra en cabécar kua

Uso: shéj X wáka "mariposa"
X tiä 2 tiä 3 -> "rip".

Operaciones de Conjuntos:

wóbogo - shénawa - ñawötkö - pakakirei - dodwa - járebo

1. agregar (añadir): se escogió la palabra en cabécar wóbogo
2. borrar: se escogió la palabra en cabécar shénawa
3. unión (reunirse): se escogió la palabra en cabécar ñawötkö
4. intersección (parte interna): se escogió la palabra en cabécar pakakirei
5. pertenecer: se escogió la palabra en cabécar dodwa
6. vacío: se escogió la palabra en cabécar járebo

Uso: Járjá Vocales wáka {: 'A' ,'E','I', 'O':} .
Vocales wóbogo ‘U’ .

Operaciones de Archivos:

Jówa - Itana - äyështä - washä - yuä - tapáwa

1. abrir: se escogió la palabra en cabécar Jówa
2. cerrar: se escogió la palabra en cabécar Itana
3. escribir: se escogió la palabra en cabécar äyështä
4. leer: se escogió la palabra en cabécar washä
5. crear: se escogió la palabra en cabécar yuä
6. asociar(juntar): se escogió la palabra en cabécar tapáwa

Uso: älwíí Texto wáka {/ "salida.txt" , 'E' }.
Jówa Texto .

Operaciones de números flotantes:

+ , - , / , * , %

Las operaciones aritméticas básicas para números flotantes, como sumar, restar, multiplicar y dividir, siguen la misma lógica de las operaciones aritméticas de enteros, adaptadas para manejar decimales.

Operaciones de comparación solicitadas:

>, <, =, >=, <=, ><

Estas operaciones permiten comparar valores numéricos o caracteres para determinar relaciones de orden o igualdad.

- Mayor que (>): Verifica si un valor es mayor que otro.
- Menor que (<): Verifica si un valor es menor que otro.
- Igual a (=): Verifica si dos valores son iguales.
- Mayor o igual que (>=): Verifica si un valor es mayor o igual a otro.
- Menor o igual que (<=): Verifica si un valor es menor o igual a otro.
- Diferente (><): Verifica si dos valores son distintos.

Uso: YínaKulíwa resultado1 wáka (10 > 5) . $ resultado1 es True
YínaKulíwa resultado2 wáka (5 >< 9) . $resultado2 es True

Manejo de Bloques de más de una instrucción:

ká - tuína

Se utilizan las palabras en cabécar “ká” y “tuína” para abrir y cerrar un bloque de instrucciones ya que significan “día” y “noche” respectivamente.

Uso: ká
$cuerpo del bloque de instrucciones
tuína .

Instrucciones

Instrucción while:

daleika shä

"Se utilizan las palabras en cabécar “daleika shä” que significan “mientras hacer”.

Uso:

```
daleika tiempo <= 100 shä
ká
        BákShtáwa(tiempo) .
        däká tiempo .
tuína .
```

Instrucción if-then-else:

chagö jéná shirína

Se utilizan las palabras en cabécar “chagö” para el if, ya que significa preguntar, seguidamente va la expresión y luego “jéná” que representa el then y significa entonces, y finalmente para el else se utiliza “shirína” que significa equivocarse.

Uso:

```
chagö (x > 10) jéná
        y wáka 5 .
shirína
        y wáka 0 .
```

Instrucción switch:

manéwa <exp> shä nuí tsú

Se utiliza “manéwa” que significa cambiar.

Uso:

```
manéwa x shä
        1: y wáka 100 .
        2: y wáka 200 .
        nuí tsú
        y wáka 0 .
```

Instrucción Repeat-until:

shäni kana

Se utilizan las palabras “shämi” y “kana” que significan repetir hasta.

Uso: shäni

x wáka x + 1 .

kana (x >= 10) .

Instrucción For:

tuna id := <exp> katso <exp> nai <exp> shä

Se escogió la palabra “tuna” que significa correr, “katso” que significa cerro y “nai” que significa caballo.

Uso: tuna i := 1 katso 1 nai 5 shä

y wáka y + i .

Instrucción With:

ta shä

Se escogió la palabra “ta” que significa “con”, y como de costumbre el “shä” que es hacer

Uso: ta registro shä

wáka campo wáka valor .

Instrucción break:

tuluwo

Se escogió la palabra en cabécar “tuluwo” que significa retiro.

Uso: tuna i wáka 1 katso 1 nai 10 shä

chagö (i = 5) jéná tuluwo .

Instrucción continue:

Tsëré

Se escogió la palabra “tsëré” que significa continuar

Uso: tuna i wáka 1 katso 1 nai 10 shä

chagö (i wáka 3) jéná tsëré .

y wáka y + i .

Instrucción Halt:

shána

Se escogió la palabra en cabécar “shána”, que significa interrumpir.

Uso: shána .

Instrucción return:

toli

Se escogió la palabra “toli” que significa rastro.

Uso: kiana suma (Shtáwa a , Shtáwa b) chani Shtáwa shä

ká

c wáka a + b .

toli c .

tuína .

Operación de size of:

Bikö

Se escogió la palabra “bikö” ya que significa tamaño.

Uso: Kichána Shéj Colores bikö 5 wáka ["Azul","Rojo","Amarillo","Verde", “Morado”] .

Sistema de coerción de tipos:

kuklë

Se utiliza la palabra “kuklë” que significa cruz.

Uso: y wáka kuklë Shtáwa (3.14) .

Encabezados

Encabezado de funciones:

kiana id (<params>) chani <tipo> shä

La palabra en cabécar “kiana” significa ser útil y “chani” es devolver.

Uso: kiana sumar (Shtáwa x , Shtáwa y) chani Shtáwa shä.

Encabezado de procedimientos:

wayuä id (<params>) shä

Se escogió la palabra “wayuä” que significa procesar.

Uso: wayuä imprimirResultado (Shéj mensaje) shä.

Manejo de parámetros

Manejo de la entrada estándar:

äwé**tipo**

La palabra en cabécar “äwé” significa recibir, por lo que se utiliza para la entrada estándar.

Uso: äwéShtáwa(x) .

Manejo de la salida estándar:

Bák**tipo**

La palabra en cabécar “bák” significa enviar, por lo que se utiliza para la salida estándar.

Uso: BákShtáwa(x) .

Manejo de parámetros formales:

(<tipo> id ,id . <tipo> iáwa id)

La palabra “iáwa” significa en cabécar transformar

Uso: kiana multiplica (Shtáwa x , Shtáwa y) chani Shtáwa shä

resultado wáka x * y .

Manejo de parámetros reales:

(5, A, 4, B)

Aquí se muestra cómo se pasan los valores reales (argumentos) a una función durante su invocación. Los elementos 5, A, 4, B representan los datos que se proporcionan a la función, alineándose con los parámetros formales definidos anteriormente, asegurando que la función reciba la información necesaria para operar.

Uso: resultado wáka multiplica (3 , 5) .

Terminador

Terminador o separador de instrucciones - Instrucción nula:

.

Utilizaremos el símbolo . como terminador o separador de instrucciones.

Uso: Residuo wáka 0 .

ká

$más instrucciones

tuína .

Todo programa se debe cerrar con un:

wëikä

Utilizaremos la palabra en cabécar wëikä para cerrar el programa ya que significa destruir.

Uso: wëikä .

Comentario de Bloque:

$* comentarios anidados *$

Utilizaremos el símbolo “$* *$” para comentar bloques de comentarios.

Uso: $* comentario1
Lsodlknla
Lskofiepnopfjp *$

Comentario de Línea:

$ comentarios

Utilizaremos el símbolo “$” para comentar una línea.

Uso: $ sección de lo que sea

Creativo

Tipo de dato creativo:

wömële

Se utiliza la palabra “wömële” ya que en cabécar significa enemigo. El dato creativo serán los números romanos.

Uso: Wömële Num1 := w|CCCXXV|w.

Literales de tipo creativo:

w| X X V III |w

El tipo de dato creativo al ser números romanos, evidentemente se utilizan los caracteres permitidos para formar un número romano, pero encerrados por “w| |w”

Uso: wömële anno wáka w| XIV |w .

Operaciones de números romanos:

wö+, wö-, wö*, wö/, wö%

Las operaciones aritméticas básicas para números romanos, en orden son la suma, resta, multiplicación, división y residuo.

Uso: id wáka X wö+ V .

id wáka X wö- V .

id wáka X wö* II .

id wáka X wö/ II .

id wáka X wö% III .

Operación 1: Potencia

Kalwá - caballo

La palabra “kalwá” en cabécar significa caballo y se utilizará para representar la potencia en números romanos.

Uso: id wáka kalwá X II .

Operación 2: Máximo Común Divisor

Saká - pariente

La palabra “saká” en cabécar significa pariente y se utilizará para representar la operación de máximo común divisor en números romanos.

Uso: id wáka saká X V .

Operación 3: Bidireccional

töka <exp> shä <instrucción> - reverso

Esta operación hace el reverso de un número romano.

Uso: töka X > V shä wáka id wö+ X .

Operación 4: Switch random

Nitsósi shä nuí - suerte

Esta operación cambia el orden el los símbolos del número romano al azar.

Uso: Nitsósi shä nuí wáka wörandom .

Índice de Pruebas

1. inputCorrecto-1.wgw
2. inputIncorrecto-1.wgw
3. Prueba_Gordita.wgw

Recuperación de Errores

Para la recuperación de errores, el Parses al no lograr hacer match, o encontrar alguna instrucción no esperada desde la que se encontraba anteriormente, empieza a consumir instrucciones del programa que se está parseando, hasta encontrar una instrucción esperada según la tabla de follows. Si ya esperaba un token y no logra hacer el match, va a consumir tokens hasta encontrarlo.

Algoritmos de Conversión

Tipo Shtáwa (entero)

shtáwa ➔ ékla (carácter): El carácter es el dado por el código ASCII que se obtiene de la operación entero módulo 256.

Ejemplo: 65 -> 'A'

shtáwa ➔ shéj (string): Es el mismo número pero representado en un string, carácter por carácter, incluyendo el símbolo negativo de ser necesario.

Ejemplo: -456 -> “-456”

shtáwa ➔ yínakulíwa (booleano): Si el entero es 0, entonces el booleano sería kóyuwé, osea falso. Con cualquier otro valor el booleano sería síwa, verdadero.

Ejemplo: 0 -> kóyuwé
1 -> síwa

shtáwa ➔ járjá (conjunto): El número entero se convierte en un conjunto de caracteres que representan los dígitos del número.

Ejemplo: 123 -> {: '1', '2', '3' :}

shtáwa ➔ wömële (número romano): El número entero se convierte a su representación en números romanos. Los números romanos sólo son válidos para enteros positivos, por lo que si el número es negativo, se podría retornar un mensaje de error o usar un valor por defecto (como 0).

Ejemplo: 49 -> XLIX

Tipo Ékla (carácter)

ékla ➔ shtáwa (entero): El valor numérico del carácter se obtiene directamente de su código ASCII.

Ejemplo: 'A' -> 65

ékla ➔ shéj (string): El carácter se convierte en un string de un solo carácter.

Ejemplo: 'A' -> "A"

ékla ➔ yínakulíwa (booleano): Si el carácter es un espacio o '0', el booleano sería kóyuwé (falso). Cualquier otro carácter se convierte en síwa (verdadero).

Ejemplo: 'A' -> síwa

ékla ➔ járjá (conjunto): El carácter se convierte en un conjunto que contiene solo ese carácter.

Ejemplo: 'A' -> {: 'A' :}

ékla ➔ wömële (número romano): El carácter se verifica si es un número romano válido. Si es uno de los símbolos válidos (I,V,X,L,C,D,M), se convierte a su valor numérico. Si no, se retorna un error o un valor por defecto.

Ejemplo: 'X' -> 10

Tipo Shéj (string)

shéj ➔ shtáwa (entero): Si el string representa un número válido (incluyendo el signo negativo), se convierte en ese número entero. Si no es un número válido, se podría retornar un error o un valor por defecto como 0.

Ejemplo: "123" -> 123
"-456" -> -456

shéj ➔ ékla (carácter): Si el string tiene una longitud de un solo carácter, se convierte en ese carácter. Si el string es más largo, se toma el primer carácter del string.

Ejemplo: "A" -> 'A'
"Hola" -> 'H'

shéj ➔ yínakulíwa (booleano): Si el string es "0", se convierte en kóyuwé (falso). Si el string es cualquier otro valor distinto de "0", se convierte en síwa (verdadero).

Ejemplo: "0" -> kóyuwé
"123" -> síwa

shéj ➔ járjá (conjunto): El string se convierte en un conjunto donde cada carácter del string es un elemento del conjunto.

Ejemplo: "Hola" -> {: 'H', 'o', 'l', 'a' :}

shéj ➔ wömële (número romano): Si el string representa un número romano válido, se convierte en su valor numérico. Si no es un número romano válido, se retorna un error o un valor por defecto.

Ejemplo: "XLIX" -> 49

Tipo YínaKulíwa (booleano)

yínakulíwa ➔ shtáwa (entero): El valor kóyuwé (falso) se convierte en 0, mientras que síwa (verdadero) se convierte en 1.

Ejemplo: kóyuwé -> 0
síwa -> 1

yínakulíwa ➔ ékla (carácter): kóyuwé (falso) se convierte en el carácter '0', mientras que síwa (verdadero) se convierte en el carácter '1'.

Ejemplo: kóyuwé -> '0'
síwa -> '1'

yínakulíwa ➔ shéj (string): kóyuwé (falso) se convierte en el string "false", mientras que síwa (verdadero) se convierte en el string "true".

Ejemplo: kóyuwé -> "false"
síwa -> "true"

yínakulíwa ➔ járjá (conjunto): Si el valor booleano es kóyuwé (falso), se crea un conjunto vacío. Si el valor es síwa (verdadero), se crea un conjunto con un solo elemento, como 'T' (True).

Ejemplo: kóyuwé -> {: :}
síwa -> {: 'T' :}

yínakulíwa ➔ wömële (número romano): kóyuwé (falso) podría retornar 0 (que no tiene representación en números romanos), mientras que síwa (verdadero) se convertiría en I (1 en números romanos).

Ejemplo: kóyuwé -> 0 (sin representación romana)
síwa -> I

Tipo Járjá (conjunto)

járjá ➔ shtáwa (entero): El conjunto se convierte en un número entero sumando el valor ASCII de cada carácter en el conjunto. Si el conjunto está vacío, el resultado será 0.

Ejemplo: {: '1', '2', '3' :} -> 150 (suma de los valores ASCII de '1', '2', y '3')

járjá ➔ ékla (carácter): Si el conjunto contiene un solo carácter, este se convierte en el valor correspondiente. Si el conjunto tiene más de un carácter, se selecciona el primero. Si el conjunto está vacío, se puede retornar un valor por defecto como '$'.

Ejemplo: {: 'A' :} -> 'A'
{: 'B', 'C' :} -> 'B'
{: :} -> '$'

járjá ➔ shéj (string): El conjunto se convierte en un string concatenando todos los caracteres en orden. Si el conjunto está vacío, el string será una cadena vacía.

Ejemplo: {: 'H', 'o', 'l', 'a' :} -> "Hola"
{: :} -> "" (string vacío)

járjá ➔ yínakulíwa (booleano): El conjunto se convierte en síwa (verdadero) si contiene al menos un elemento, y en kóyuwé (falso) si está vacío.

Ejemplo:{: 'A', 'B' :} -> síwa (verdadero)
{: :} -> kóyuwé (falso)

járjá ➔ wömële (número romano): Si el conjunto contiene solo caracteres que representan números romanos válidos (I, V, X, L, C, D, M), se convierte en el número romano correspondiente. Si hay caracteres no válidos o el conjunto está vacío, se podría retornar un valor por defecto como 0 o un mensaje de error.

Ejemplo: {: 'X', 'I', 'V' :} -> 14
{: :} -> 0 (sin representación romana)

Tipo Wömële (número romano)

wömële ➔ shtáwa (entero): El número romano se convierte a su valor entero. Si el número romano no es válido, se podría retornar un error o un valor por defecto como 0.

Ejemplo: XLIX -> 49

wömële ➔ ékla (carácter): El número romano se convierte en el carácter equivalente al valor entero en código ASCII, donde el entero es el valor en número romano mod 256.

Ejemplo: I (1) -> '' (código ASCII 1)
XLIX (49) -> '1' (código ASCII 49)

wömële ➔ shéj (string): El número romano se convierte en un string que representa el número en formato romano.

Ejemplo: XLIX -> "XLIX"

wömële ➔ yínakulíwa (booleano): Si el número romano es 0, se convierte en kóyuwé (falso). Cualquier otro número romano válido se convierte en síwa (verdadero).

Ejemplo: I -> síwa
0 (sin representación romana) -> kóyuwé

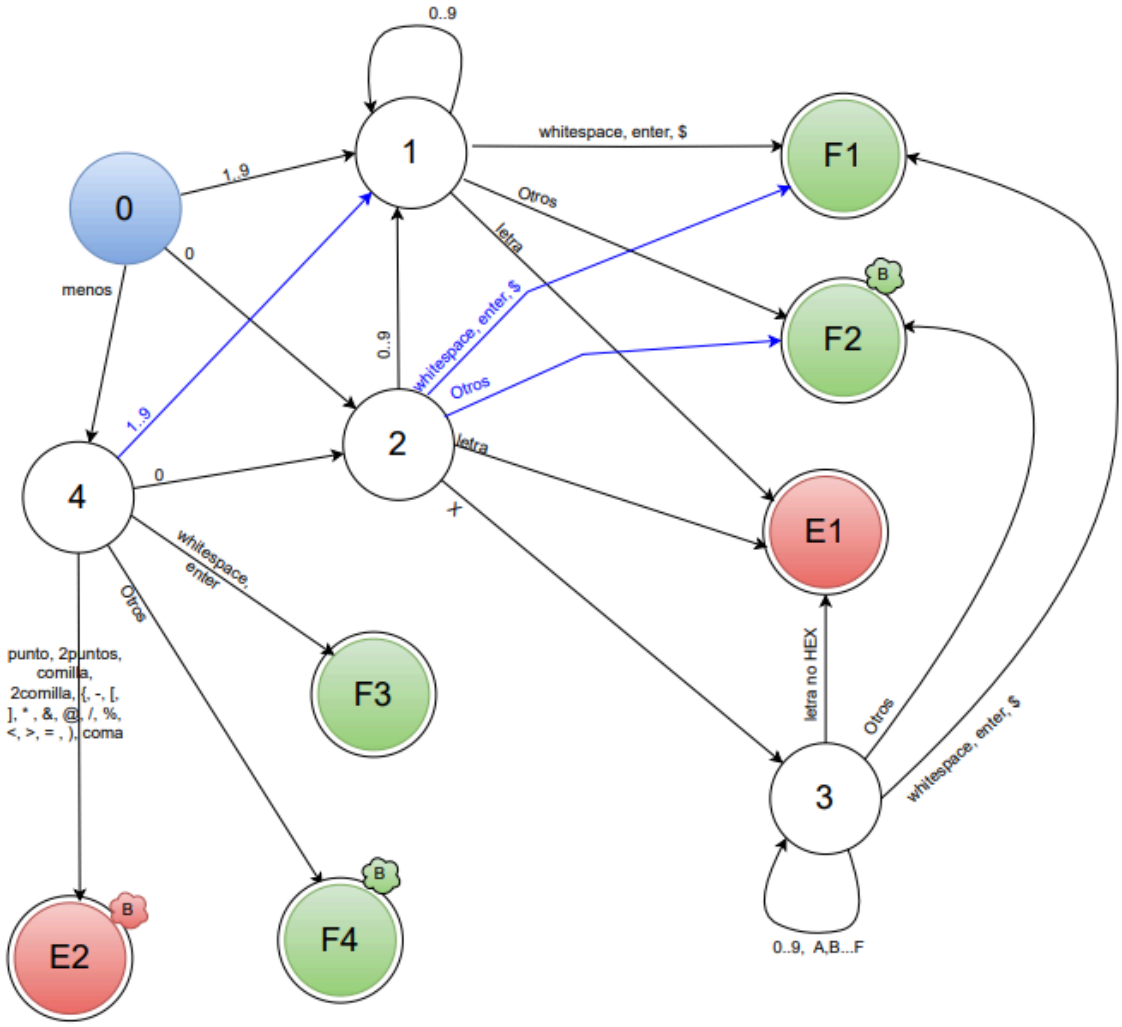
wömële ➔ járjá (conjunto): Si el número romano es válido, se convierte en un conjunto de caracteres que representan los dígitos del valor numérico.

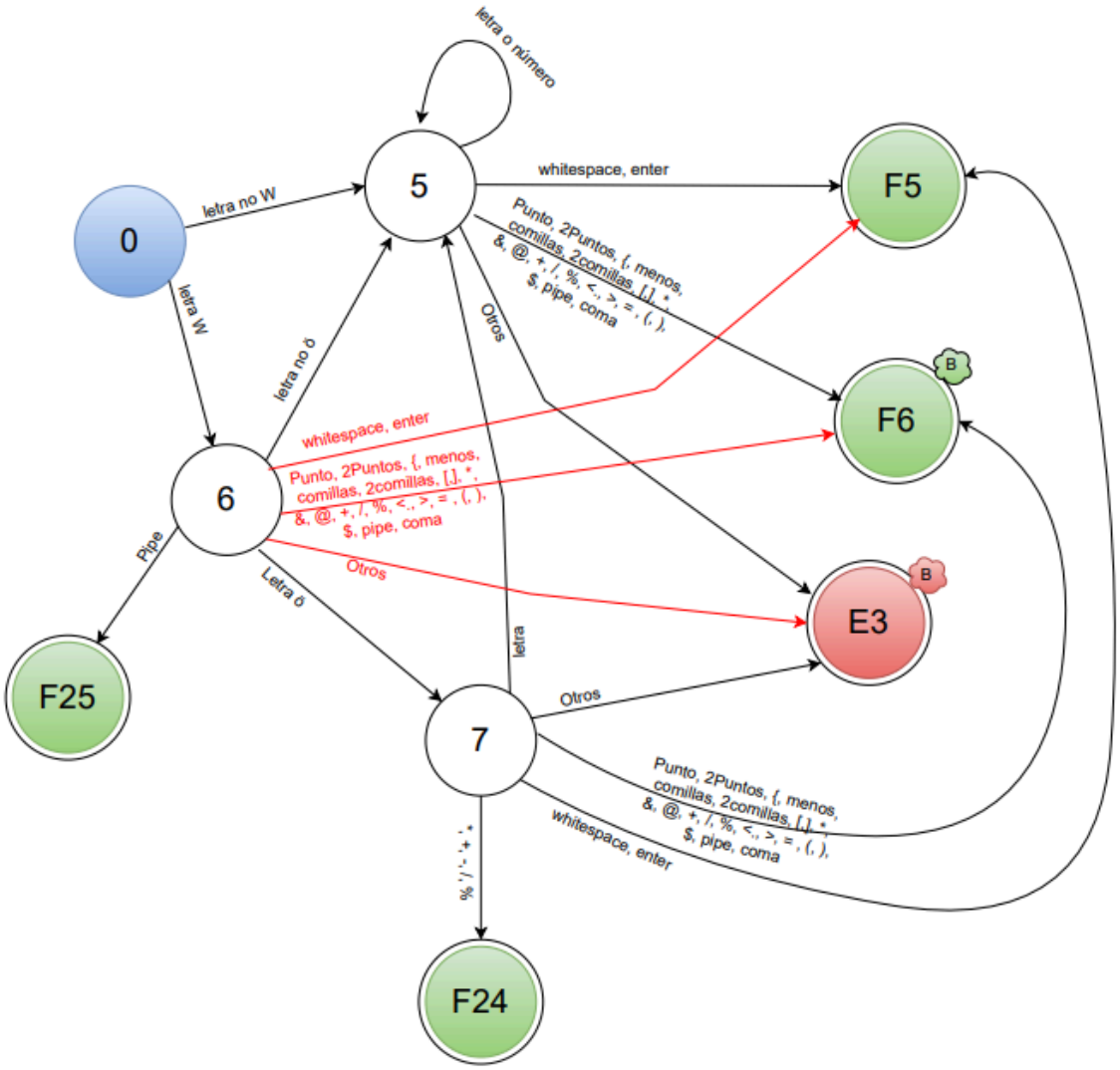
Ejemplo: XLIX -> {: '4', '9' :}

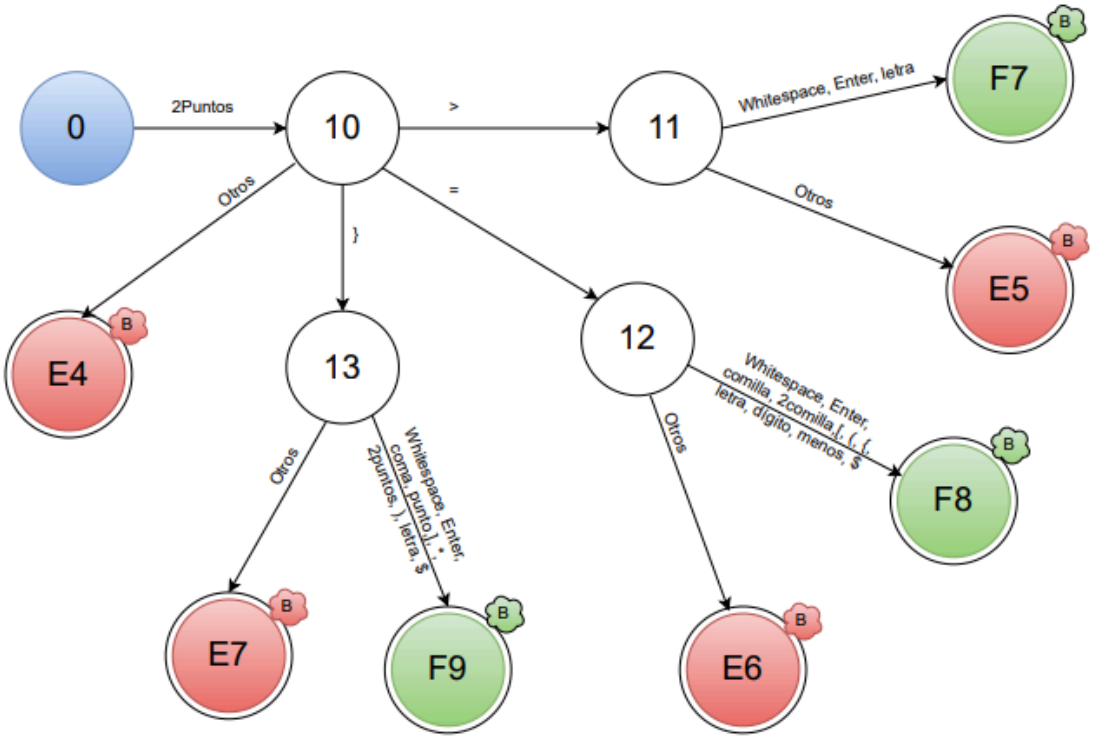
Autómata

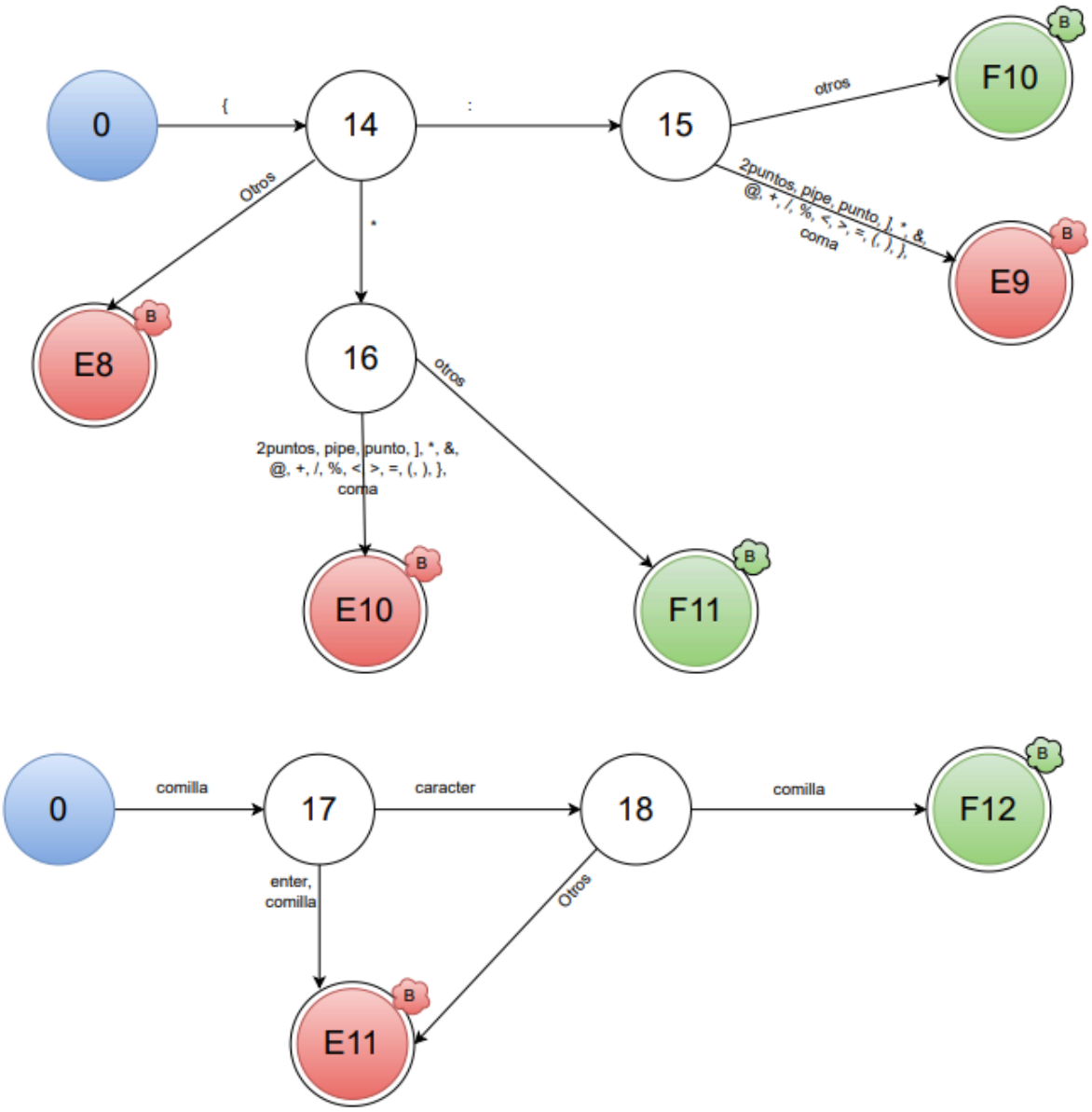
(se pueden hacer los dibujos por partes para no tener una “araña” poco legible. Eso sí, todas las partes deben estar conectadas al símbolo inicial)

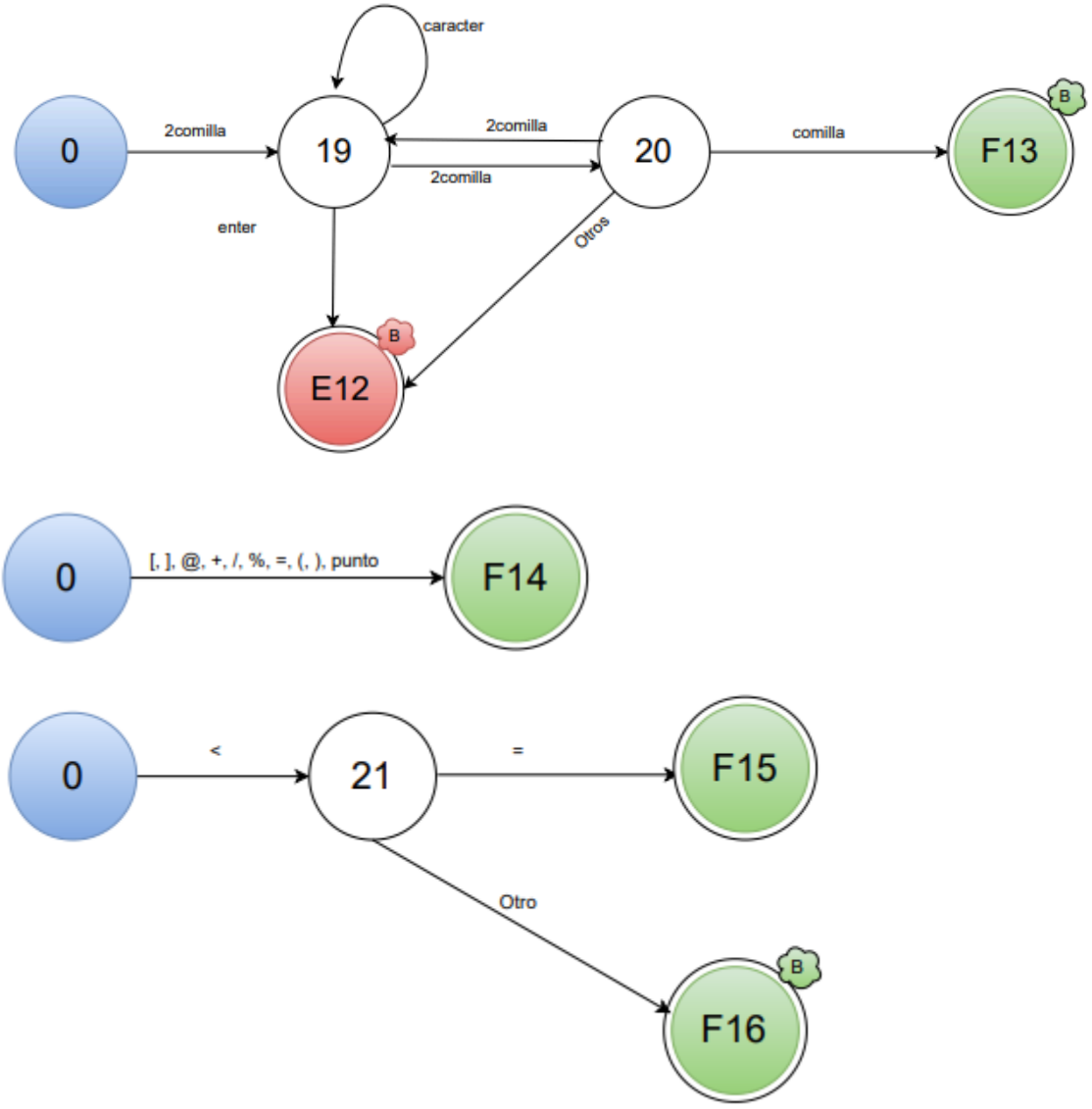
Los autómatas también vienen en un archivo llamado “Autómatas Scanner” en el la carpeta Documentación

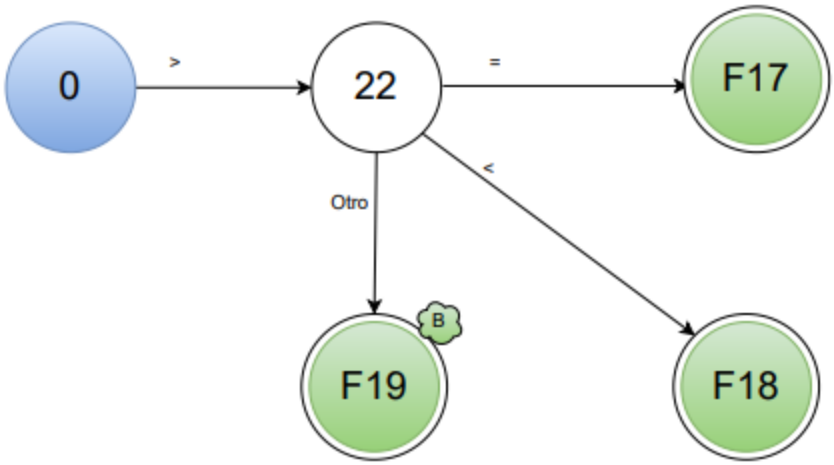


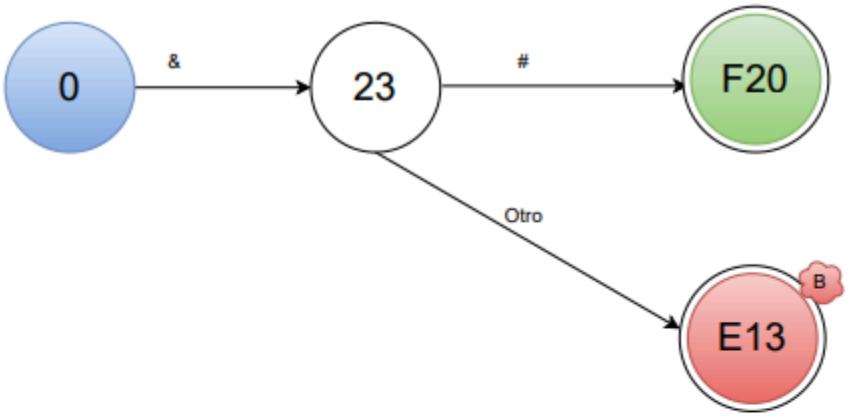

letra o número
0
letra no W
5
whitespace, enter
F5
Punto, 2Puntos, {, menos, comillas, 2comillas, [,], *, &, @, +, /, %, <, >, =, (,), $, pipe, coma
letra W
letra no ó
Otros
B
F6
whitespace, enter
Punto, 2Puntos, {, menos, comillas, 2comillas, [,], *, &, @, +, /, %, <, >, =, (,), $, pipe, coma
6
Pipe
Otros
B
E3
Letra ó
letra
F25
Otros
7
Punto, 2Puntos, {, menos, comillas, 2comillas, [,], *, &, @, +, /, %, <, >, =, (,), $, pipe, coma
whitespace, enter
% ' / - . + *
F24

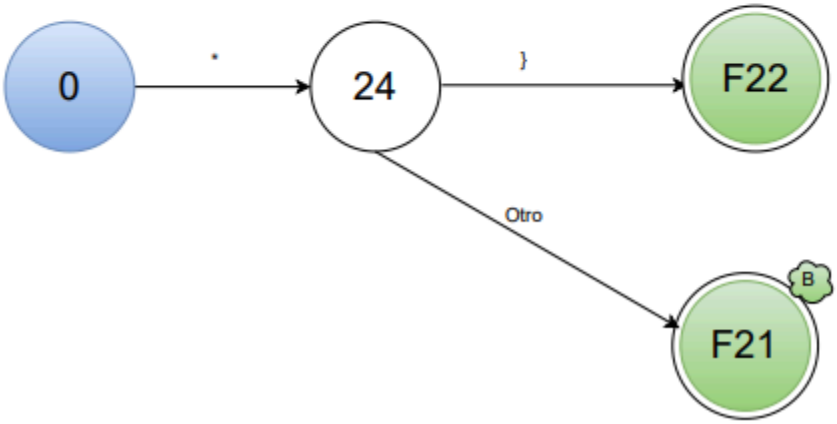

0
2Puntos
10
>
11
Whitespace, Enter, letra
F7
B
Otros
Otros
E5
B
=
}
E4
B
13
12
Whitespace, Enter, comilla, 2comilla, (,), letra, digito, menos, $
F8
B
Otros
Whitespace, Enter, coma, punto, (, *, 2puntos,), letra, $
Otros
E7
B
F9
B
E6
B

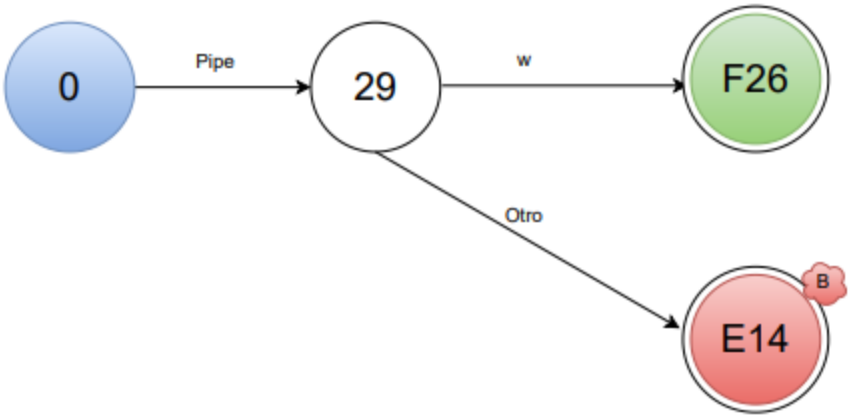

0
{
14
:
15
otros
F10
B
Otros
*
2puntos, pipe, punto,], *, &, @, +, /, %, <, >, =, (,), }, coma
E9
B
E8
B
16
otros
2puntos, pipe, punto,], *, &, @, +, /, %, <, >, =, (,), }, coma
E10
B
F11
B
0
comilla
17
caracter
18
comilla
F12
B
enter, comilla
Otros
E11
B

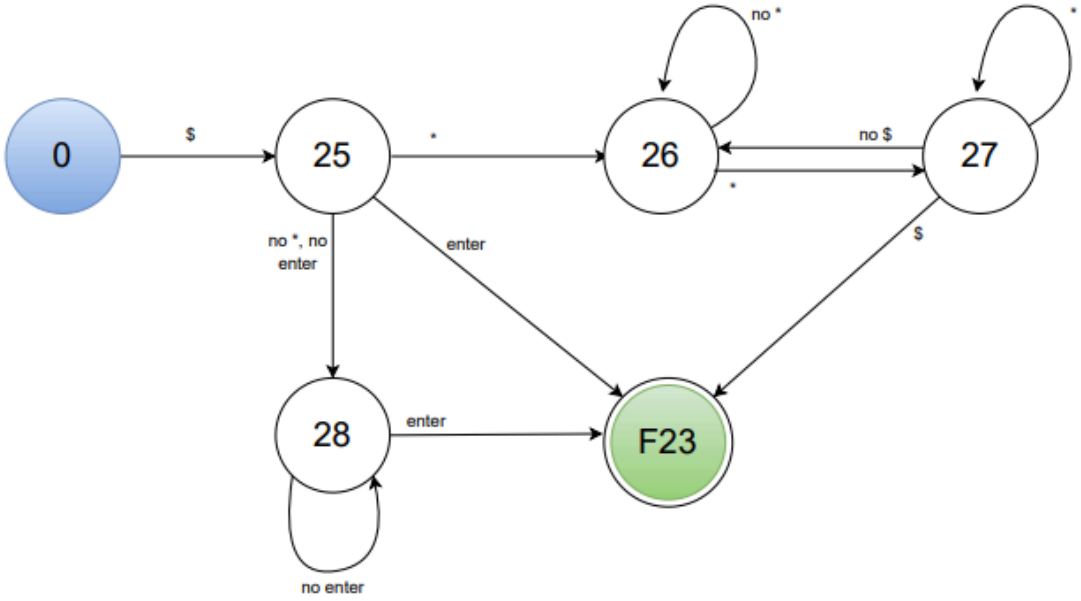

caracter
0
2comilla
19
2comilla
2comilla
20
comilla
F13
B
enter
Otros
B
E12
0
[,], @, +, /, %, =, (,), punto
F14
0
<
21
=
F15
Otro
B
F16


0
>
22
=
F17
Otro
<
B
F19
F18


0
&
23
#
F20
Otro
B
E13


0
*
24
}
F22
Otro
B
F21


0
Pipe
29
w
F26
Otro
B
E14


0
$
25
*
26
no *
no $
27
*
*
$
no *, no enter
enter
28
enter
F23
no enter

Listado de Errores

(Deben ir numerados)

1. TOKEN_UNKNOWN,
2. TOKEN_ERR_INT_B,
3. TOKEN_ERR_MENOS_B,
4. TOKEN_ERR_TOKEN_STRING,
5. TOKEN_ERR_CHAR_PIPE_B,
6. TOKEN_ERR_TOKEN_ID_B,
7. TOKEN_ERR_2PUNTO_B,
8. TOKEN_ERR_TIPO_B,
9. TOKEN_ERR_INICIA_B,
10. TOKEN_ERR_CERRCONJ_B,
11. TOKEN_ERR_LLAVE_B,
12. TOKEN_ERR_OPENCONJ_B,
13. TOKEN_ERR_OPENREG_B,
14. TOKEN_ERR_TOKEN_CHAR_B,
15. TOKEN_ERR_CHAR_AMPER_B,
16. CANT_TOKEN

Diccionario de Símbolos Semánticos

```
 /* rutinas semánticas */

#define RS1001 188
#define RS1002 189
#define RS1017 190
#define RS1003 191
#define RS1005 192
#define RS1004 193
#define RS1014 194
#define RS1006 195
#define RS1009 196
#define RS1013 197
#define RS1015 198
#define RS1012 199
#define RS1016 200
#define RS1018 201
#define RS1019 202
#define RS1007 203
#define RS1008 204
#define RS1020 205
#define RS1023 206
#define RS1024 207
#define RS1022 208
#define RS1021 209
#define RS1031 210
#define RS1025 211
#define RS1027 212
#define RS1028 213
#define RS1026 214
#define RS1029 215
#define RS1030 216
#define RS1040 217
#define RS1039 218
#define RS1038 219
#define RS1036 220
#define RS1035 221
#define RS1037 222
#define RS1032 223
#define RS1033 224
#define RS1034 225
#define RS1011 226
#define RS1010 227
```

Errores Semánticos

"Error semántico: No coincide el tipo de dato. La constante se definió de tipo "
"Error semántico: Está definiendo una variable o constante que ya fue definida anteriormente.”
"Error semántico: Error inesperado insertando símbolo en el diccionario de datos"
"Error semántico:Del lado izquierdo de una asignación, solamente puede ir una variable."
"Error semantico: Id no definido.”
"Error semántico: Se espera que llame a una función o rutina.”
"Error semántico: No son compatibles el tipo de la variable al lado izquierdo y la expresión del lado derecho”
"Error semántico: estructura inesperada. No encontró la variable en la pila"
"Error semántico: estructura inesperada. Falta Expresion en la asignación"
"Error semántico:Inesperado. Se esperaba la finalización de una expresión."
"Error semántico: La cantidad de parámetros formales y funcionales no coincide en X. Se esperan Y parámetros y se usaron Z”
"Error semántico:Se esperaba una función o subrutina para analizar pero no llegó ninguna."
"Error semántico: Identificador no definido."
"Error semántico: Los operadores para concatenación deben ser STRING"
"Error semántico: estructura inesperada. Falta segundo operador"
"Error semántico: Los operadores para cortar deben ser INT"
"Error semántico: estructura inesperada. Faltan operadores adecuados en el cortar o recortar strings"
"Error semantico: estructura inesperada. Faltan operadores adecuados en operación aritmética"
"Error semántico: Los operadores dos operadores aritméticos no son compatibles."
"Error semántico: Los operadores dos operadores infijos no son compatibles."
"Error semántico: estructura inesperada. Faltan operadores adecuados en operacion infija"
"Error semántico: estructura inesperada. Faltan operadores adecuados en operación de comparación"
"Error semantico: estructura inesperada. Faltan operadores adecuados en operación aritmética"

Generación de Código

Diccionario Símbolos

#define GC0001: Genera pila y code segments
#define GC0002: Genera el main e Inicio en el code segment
#define GC0003: Genera final de asm
#define GC0004: Guardar el puntero base y establecer el nuevo marco de pila
#define GC0005: Return de una función
#define GC0009: Asignación de variables
#define GC0020: Llamado a Procedimiento o función
#define GC0010: Inicia IF después de evaluar expresión
#define GC0011: Finaliza parte verdadera del if e inicia el else del if
#define GC0012: Finaliza else del if
#define GC0013: Inicio del while
#define GC0014: Resultado de la evaluación de la expresión booleana de un while
#define GC0015: Reiniciar el ciclo del while nuevamente y fin del while
#define GC0016
#define GC0017
#define GC0018
#define GC0019: Fin de expresion parametro
#define GC0008: Genera operadores (sumas, restas, etc)
#define GC0007: Sección constantes (tipos)
#define GC0006

Runtime Library

Propósito

RUNTIME.asm es una biblioteca de rutinas en ensamblador organizada para gestionar operaciones de Entrada/Salida (IO), Operaciones (Op) y Conversiones de Tipos para el conjunto de datos básicos y avanzados del lenguaje, específicamente: enteros, caracteres, strings, booleanos, conjuntos y números romanos.

Estructura de la Biblioteca

RUNTIME.asm se divide en tres secciones principales de rutinas para cada tipo de dato. A continuación, se describen estas secciones:

Rutinas de Entrada/Salida (IO): Estas rutinas son responsables de gestionar la entrada y salida de cada tipo de dato. Cada rutina está prefijada con el formato ***<tipo>_io_***, donde <tipo> es el identificador del tipo de dato.
Ejemplo: int_io_leer e int_io_imprimir para enteros.

Rutinas de Operaciones (Op): Las operaciones abarcan acciones comunes que se realizan con los tipos de datos, como comparaciones, cálculos matemáticos y manipulaciones de cadenas o caracteres. Cada rutina en esta sección sigue el formato ***<tipo>_op_***.
Ejemplo: str_op_bawa para concatenación de cadenas y rom_op_suma para sumar dos números romanos.

Rutinas de Algoritmos de Conversión: Estas rutinas son responsables de convertir datos de un tipo a otro, permitiendo compatibilidad y reutilización de datos entre diferentes operaciones. Cada rutina de conversión sigue el formato ***<tipoOrigen>_conv_<tipoDestino>***.
Ejemplo: int_conv_char para convertir un entero a un carácter ASCII y rom_conv_int para convertir un número romano a un entero.

Estructura de Datos

Cada tipo de dato tiene un segmento ***DATOS_<TIPO>*** dedicado, que se utiliza para almacenar variables específicas de entrada y salida. Esto ayuda a reducir los conflictos y facilita el acceso organizado a las variables.

Convenciones de Nombres y Prefijos

Tipos de datos:

- `int_` para enteros (**Shtáwa**)
- `char_` para caracteres (**ékla**)
- `str_` para cadenas (**Shéj**)
- `bool_` para booleanos (**YínaKulíwa**)
- `set_` para conjuntos (**Járjá**)
- `rom_` para números romanos (**wömële**)

Subcategorías:

- `_io` para Entrada/Salida
- `_op` para Operaciones
- `_conv` para Conversiones