Universidad de Guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
DIVISIÓN DE TECNOLOGÍAS PARA LA INTEGRACIÓN CIBER-HUMANA
INGENIERÍA EN COMPUTACIÓN

Gramática

Nombre: Dixie Yamile Aguilar Cervantes

Código: 218448769

Materia: Compiladores

Sección: D08

Guadalajara, Jal., 09 de septiembre de 2025

Bases de la gramática

Elementos Léxicos Fundamentales

Los componentes básicos del lenguaje se definen de la siguiente manera:

- **Identificadores**: Nombres asignados a variables, constantes, tipos y funciones. Deben comenzar con un carácter alfabético (<letra>) y pueden ser seguidos por una secuencia de caracteres alfanuméricos.
- **Literales**: Representaciones de valores fijos en el código fuente.
    - **Numéricos**: *Gem* para enteros y *Shimmer* para valores de punto flotante.
    - **Booleanos**: *sparkle_on* para verdadero y *sparkle_off* para falso.
    - **Carácter**: Un único carácter delimitado por comillas simples (ejemplo *'a'*).
    - **Cadena**: Una secuencia de caracteres delimitada por comillas dobles (ejemplo *"Hello, World!"*).

Tipos de Datos

Primitivos

Son los tipos de datos fundamentales incorporados en el lenguaje.

- *Gem*: Representa números enteros de 32 o 64 bits.
- *Shimmer*: Representa números de punto flotante de precisión simple o doble.
- *Truth_potion*: Tipo de dato booleano.
- *Letter*: Representa un único carácter, conforme al estándar Unicode.

Compuestos

Son tipos construidos a partir de los tipos primitivos.

- *Story*: Una secuencia inmutable de caracteres (String).
- *Collection<Tipo>*: Un tipo de dato genérico que representa un arreglo o lista de elementos de un Tipo homogéneo.
- *Esemble*: Una estructura de datos que agrupa un conjunto de valores con nombre, similar a un *struct* en C o un *record*.

Definidos por el Usuario

Permiten la abstracción de datos y la programación orientada a objetos.

- *design:* Define una clase, el plano para la creación de objetos. Encapsula estado (variables miembro) y comportamiento (funciones miembro).
- *blueprint*: Define una interfaz, un contrato que especifica un conjunto de métodos que una clase debe implementar.

Declaraciones

Las declaraciones introducen nuevos identificadores en el programa.

- **Declaración de Variables**: Se utiliza la palabra clave *let* para declarar una variable mutable. La especificación de tipo es obligatoria. Ejemplo: *let contador be a Gem = 0;*
- **Declaración de Constantes**: Se utiliza la palabra clave *forever* para declarar una constante inmutable, cuyo valor no puede ser modificado después de la inicialización. Ejemplo: *forever MAX_USERS = 100;*
- **Asignación**: Se utiliza el operador = para asignar un nuevo valor a una variable previamente declarada. Ejemplo: *contador = 1;*

Operadores y Expresiones

Las expresiones se construyen a partir de literales, variables y operadores. La gramática define una jerarquía de precedencia para resolver expresiones sin ambigüedad.

- **Operadores Aritméticos**: *+, -, *, /, %* (módulo).
- **Operadores Relacionales**: *is* (igualdad), *is_not* (desigualdad*), >, <, >=, <=.*
- **Operadores Lógicos**: *and* (conjunción lógica), *or* (disyunción lógica), *not* (negación lógica).

Estructuras de Control

Controlan el flujo de ejecución del programa.

- **Estructuras Condicionales**: Permiten la ejecución selectiva de código.
    - *if*: Ejecuta un bloque si la condición es verdadera.
    - *or_if*: Cláusula *else if* para evaluar condiciones adicionales.
    - *otherwise*: Cláusula *else* que se ejecuta si ninguna condición anterior es verdadera.

- **Estructuras Iterativas (Bucles)**: Permiten la ejecución repetida de un bloque de código.
    - *as_long_as*: Bucle *while*. El bloque se ejecuta mientras la condición sea verdadera.
    - *for_every*: Bucle *for*. Incluye una sección de inicialización, una condición de continuación y una expresión de incremento.
    - *dream { ... } while*: Bucle *do-while*. El bloque se ejecuta al menos una vez antes de que se evalúe la condición.
- **Sentencias de Control de Flujo**:
    - *break_free*: Termina la ejecución del bucle más interno (sentencia *break*).
    - *next_please*: Omite el resto del cuerpo del bucle y pasa a la siguiente iteración (sentencia *continue*).
    - *give_back*: Retorna un valor desde una función (sentencia *return*).

Funciones y Programación Orientada a Objetos

- **Funciones (charm)**: Bloques de código reutilizables definidos con la palabra clave charm. Soportan parámetros tipados y un tipo de retorno explícito.
- **Clases (design)**: El elemento central de la POO en *Pixie*.
    - **Herencia**: Se implementa con la palabra clave *inspired_by*, permitiendo que una clase herede de otra.
    - **Modificadores de Acceso**:
        - *for_everyone*: Miembro público.
        - *my_secrets*: Miembro privado.
        - *for_my_circle*: Miembro protegido.

Manejo de Excepciones

El lenguaje provee un mecanismo para manejar errores en tiempo de ejecución.

- *oopsie { ... } recover_with (error) { ... }:* Corresponde a un bloque *try-catch*. El código propenso a errores se coloca en el bloque *oopsie*, y el manejo del error se realiza en el bloque *recover_with*.
- *panic with:* Lanza una excepción (sentencia *throw*).

Modularidad y Espacios de Nombres

El lenguaje soporta la organización del código en unidades lógicas separadas.

- *get_magic_from*: Sentencia de importación para incluir módulos externos.
- *share:* Sentencia de exportación para hacer que las definiciones (clases, funciones) sean visibles para otros módulos.
- *magic_closet:* Define un espacio de nombres (namespace) para evitar colisiones de nombres.

Backus-Naur Form

Elementos Básicos

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

<letra> ::= <letra_mayuscula> | <letra_minuscula>

<letra_mayuscula> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

<letra_minuscula> ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

<identificador> ::= <letra> <resto_identificador>

<resto_identificador> ::= <letra> <resto_identificador> | <digito> <resto_identificador> | ε

<simbolo> ::= "!" | "@" | "#" | "$" | "%" | "^" | "&" | "*" | "(" | ")" | "-" | "_" | "=" | "+" | "[" | "]" | "{" | "}" | "|" | "\" | ";" | ":" | "," | "." | "<" | ">" | "/" | "?" | "~" | "`"

Literales y Valores

<literal> ::= <entero> | <flotante> | <booleano> | <caracter> | <cadena>

<entero> ::= "-" <numero_natural> | <numero_natural>

<numero_natural> ::= <digito> <numero_natural> | <digito>

<flotante> ::= <entero> "." <numero_natural>

<booleano> ::= "sparkle_on" | "sparkle_off"

<caracter> ::= "'" <contenido_caracter> "'"

<contenido_caracter> ::= <letra> | <digito> | <simbolo>

<cadena> ::= "" <secuencia_caracteres> ""

<secuencia_caracteres> ::= <contenido_caracter> <secuencia_caracteres> | ε

Tipos de Datos

<tipo> ::= <tipo_primitivo> | <tipo_compuesto> | <identificador>

<tipo_primitivo> ::= "Gem" | "Shimmer" | "Truth_potion" | "Letter"

<tipo_compuesto> ::= "Story" | <tipo_coleccion> | "Esemble"

<tipo_coleccion> ::= "Collection" "<" <tipo> ">"

Expresiones

<expresion> ::= <expresion_logica>

<expresion_logica> ::= <termino_logico> <expresion_logica_cola>

<expresion_logica_cola> ::= "or" <termino_logico> <expresion_logica_cola> | ε

<termino_logico> ::= <factor_logico> <termino_logico_cola>

<termino_logico_cola> ::= "and" <factor_logico> <termino_logico_cola> | ε

<factor_logico> ::= "not" <expresion_relacional> | <expresion_relacional>

<expresion_relacional> ::= <expresion_aritmetica> <expresion_relacional_cola>

<expresion_relacional_cola> ::= <op_relacional> <expresion_aritmetica> | ε

<op_relacional> ::= "<" | ">" | "<=" | ">=" | "Is" | "is_not"

<expresion_aritmetica> ::= <termino> <expresion_aritmetica_cola>

<expresion_aritmetica_cola> ::= <op_suma> <termino> <expresion_aritmetica_cola> | ε

<op_suma> ::= "+" | "-"

<termino> ::= <factor> <termino_cola>

<termino_cola> ::= <op_multi> <factor> <termino_cola> | ε

<op_multi> ::= "*" | "/" | "%"

<factor> ::= <primario> | <op_unario> <factor>

<op_unario> ::= "+" | "-"

<primario> ::= <literal> | <identificador> | <llamada_funcion> | "(" <expresion> ")"

<llamada_funcion> ::= <identificador> "(" <lista_expresiones_opcional> ")"

<lista_expresiones_opcional> ::= <lista_expresiones> | ε

<lista_expresiones> ::= <expresion> <lista_expresiones_cola>

<lista_expresiones_cola> ::= "," <expresion> <lista_expresiones_cola> | ε

Declaraciones y Sentencias

<sentencia> ::= <declaracion_variable> | <declaracion_constante> | <asignacion> | <llamada_funcion_sentencia> | <condicional> | <bucle> | <control_flujo> | <manejo_errores> | <lanzamiento_excepcion> | <bloque>

<declaracion_variable> ::= "let" <identificador> "be" "a" <tipo> ("=" <expresion>)? ";"

<declaracion_constante> ::= "forever" <identificador> ("=" <expresion>)? ";"

<asignacion> ::= <identificador> "=" <expresion> ";"

<llamada_funcion_sentencia> ::= <llamada_funcion> ";"

Estructuras de Control

<condicional> ::= "if" "(" <expresion> ")" <bloque> <lista_or_if> | "if" "(" <expresion> ")" <bloque> <lista_or_if> <parte_else>

<lista_or_if> ::= "or_if" "(" <expresion> ")" <bloque> <lista_or_if> | ε

<parte_else> ::= "otherwise" <bloque> | ε

<bucle> ::= <bucle_while> | <bucle_for> | <bucle_do_while>

<bucle_while> ::= "as_long_as" "(" <expresion> ")" <bloque>

<bucle_for> ::= "for_every" "(" <for_inicializacion> ";" <expresion> ";" <for_incremento> ")" <bloque>

<for_inicializacion> ::= <declaracion_variable_sin_punto_y_coma> | <asignacion_sin_punto_y_coma>

<declaracion_variable_sin_punto_y_coma> ::= "let" <identificador> "be" "a" <tipo> ("=" <expresion>)?

<asignacion_sin_punto_y_coma> ::= <identificador> "=" <expresion>

<for_incremento> ::= <asignacion_sin_punto_y_coma>

<bucle_do_while> ::= "dream" <bloque> "while" "(" <expresion> ")" ";"

<control_flujo> ::= "break_free" ";" | "next_please" ";" | "give_back" <expresion>? ";"

Funciones, Clases, Errores y Módulos

<declaracion_funcion> ::= "charm" <identificador> "(" <lista_parametros_opcional> ")" ("returns" <tipo>)? <bloque>

<lista_parametros_opcional> ::= <lista_parametros> | ε

<lista_parametros> ::= <parametro> <lista_parametros_cola>

<lista_parametros_cola> ::= "," <parametro> <lista_parametros_cola> | ε

<parametro> ::= <identificador> "be" "a" <tipo>

<declaracion_clase> ::= "design" <identificador> ("inspired_by" <identificador>)? ("follows_blueprint" <identificador>)? "{" <lista_miembros_clase> "}"

<lista_miembros_clase> ::= <miembro_clase> <lista_miembros_clase> | ε

<miembro_clase> ::= <seccion_acceso> ":" (<declaracion_variable> | <declaracion_funcion>)

<seccion_acceso> ::= "for_everyone" | "my_secrets" | "for_my_circle"

<declaracion_interfaz> ::= "blueprint" <identificador> <bloque_interfaz>

<bloque_interfaz> ::= "{" <lista_declaraciones_funcion> "}"

<lista_declaraciones_funcion> ::= <declaracion_funcion> <lista_declaraciones_funcion> | ε

<manejo_errores> ::= "oopsie" <bloque> "recover_with" "(" <identificador> ")" <bloque>

<lanzamiento_excepcion> ::= "panic" "with" <expresion> ";"

<importacion> ::= "get_magic_from" <cadena> ";"

<exportacion> ::= "share" (<declaracion_funcion> | <declaracion_clase> | <declaracion_interfaz>)

<namespace> ::= "magic_closet" <identificador> <bloque>

Programa Principal y Bloques

<programa> ::= <lista_elementos_globales>

<lista_elementos_globales> ::= <elemento_global> <lista_elementos_globales> | ε

<elemento_global> ::= <importacion> | <declaracion_funcion> | <declaracion_clase> | <declaracion_interfaz> | <namespace> | <declaracion_variable> | <declaracion_constante>

<bloque> ::= "{" <lista_sentencias> "}"

<lista_sentencias> ::= <sentencia> <lista_sentencias> | ε