GDInt

Introducción breve al proyecto

GDint es un compilador que busca reducir la brecha entre la lengua natural y los lenguajes de programación. GDint busca facilitar el proceso de aprendizaje para jóvenes que tienen interés en la programación y no saben por dónde empezar. Este lenguaje se diferencia de otros lenguajes pseudocódigo porque tiene implementado funciones para acceder a APIs.

Primera versión de la gramática

Simbología

<NO TERMINAL>

[terminal]

<ESPACIO> PRODUCE [espacio]<ESPACIO>

<ESPACIO> PRODUCE [espacio]

<CADENA> PRODUCE [<CADENA>|∈Σ-']

<CADENA> PRODUCE [∈Σ-'] excepto ‘, cualquier caracter.

<CONSTANTE> PRODUCE [constante]

<CONSTANTE> PRODUCE [‘]<CADENA>[‘]

<OPERADOR> PRODUCE [+]|[-]|[/]|[*]

<EXP> PRODUCE <EXP><OPERADOR><EXP>

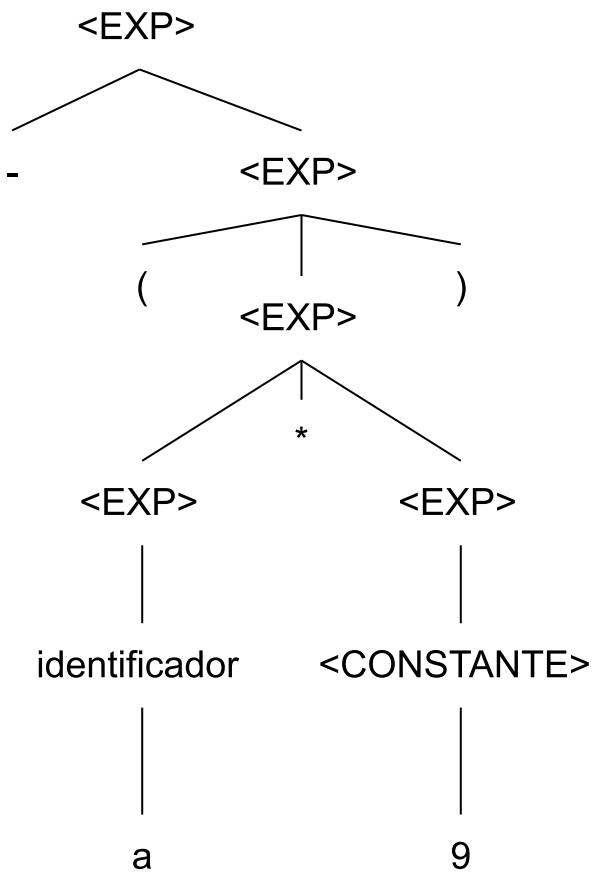
<EXP> PRODUCE [-]<EXP>

<EXP> PRODUCE [(]<EXP>[)]

<EXP> PRODUCE [identificador]

<EXP> PRODUCE <CONSTANTE>

- (a * 9)



variable a = 0

<DECLARACIÓN> PRODUCE [keyword variable]<ESPACIO>[identificador]

a = 3+2

<ASIGNACIÓN> PRODUCE [identificador]<ESPACIO>[=]<ESPACIO><EXP>

<NOMBRE DE FUNCIÓN> PRODUCE [keyword obtener]

<NOMBRE DE FUNCIÓN> PRODUCE [keyword publicar]

<NOMBRE DE FUNCIÓN> PRODUCE [keyword cambiar]

<NOMBRE DE FUNCIÓN> PRODUCE [keyword borrar]

<PARÁMETROS> PRODUCE <PARÁMETROS><ESPACIO><CONSTANTE>

<PARÁMETROS> PRODUCE <ESPACIO><CONSTANTE>

obtener ‘https://pokeapi.co/api/v2/pokemon/ditto’

<LLAMADA> PRODUCE <NOMBRE DE FUNCIÓN><PARÁMETROS>

<INSTRUCCIÓN> PRODUCE <DECLARACIÓN>
<INSTRUCCIÓN> PRODUCE <ASIGNACIÓN>
<INSTRUCCIÓN> PRODUCE <LLAMADA>
<INSTRUCCIÓN> PRODUCE ε

#obtener ‘https://pokeapi.co/api/v2/pokemon/ditto’
<COMENTARIO> PRODUCE [#]<INSTRUCCIÓN>

si a < 4 && b ==-3 entonces
obtener ‘https://pokeapi.co/api/v2/pokemon/ditto’
sino
obtener ‘https://pokeapi.co/api/v2/pokemon/incineroar’
finsi

<BLOQUE> PRODUCE <INSTRUCCIÓN>[\n]<BLOQUE>
<BLOQUE> PRODUCE <INSTRUCCIÓN>[\n]

<FACTOR> PRODUCE [identificador] | [!][identificador]
<FACTOR> PRODUCE <CONSTANTE>
<COMPARADOR> PRODUCE [==] | [<] | [>]
<COMPARACIÓN> PRODUCE <FACTOR><COMPARADOR><FACTOR>
<CONECTOR> PRODUCE [&&] | [||]
<CONDICIÓN> PRODUCE <COMPARACIÓN><CONECTOR><CONDICIÓN>
<CONDICIÓN> PRODUCE <COMPARACIÓN>
<CONDICIONAL> PRODUCE [keyword si]<ESPACIO><CONDICIÓN><ESPACIO>[keyword entonces][\n]<BLOQUE>[keyword finsi][\n] | [keyword si]<ESPACIO><CONDICIÓN><ESPACIO>[keyword entonces][\n]<BLOQUE>[keyword sino][\n]<BLOQUE>[keyword finsi][\n]

<BLOQUE> PRODUCE <CONDICIONAL>
<CÓDIGO> PRODUCE <BLOQUE><CÓDIGO>
<CÓDIGO> PRODUCE <BLOQUE>

Código fuente de ejemplo completo
variable a = 0
a = 3 + 2
si a < 4 && a ==-3 entonces

obtener ‘https://pokeapi.co/api/v2/pokemon/ditto’
sino
obtener ‘https://pokeapi.co/api/v2/pokemon/incineroar’
finsi
#obtener ‘https://pokeapi.co/api/v2/pokemon/ditto’
variable body = ‘
doe: "a deer, a female deer"
 ray: "a drop of golden sun"
 pi: 3.14159
 xmas: true
 french-hens: 3
 calling-birds:
   - huey
   - dewey
   - louie
   - fred
 xmas-fifth-day:
   calling-birds: four
   french-hens: 3
   golden-rings: 5
   partridges:
     count: 1
     location: "a pear tree"
   turtle-doves: two
‘
publicar ‘https://pokeapi.co/api/v2/pokemon/ditto’

Eliminar cualquier recursión izquierda posible
CODIGO -> BLOQUE CODIGO'
CODIGO' -> BLOQUE CODIGO' | ε
BLOQUE -> CONDICIONAL BLOQUE' | INSTRUCCION BLOQUE'
BLOQUE' -> \n INSTRUCCION BLOQUE' | \n CONDICIONAL BLOQUE' | ε
CONDICIONAL -> si ESPACIO CONDICION ESPACIO entonces \n BLOQUE \n SINO finsi \n
SINO -> sino \n BLOQUE \n | ε
CONDICION -> COMPARACION CONDICION'
CONDICION' -> && COMPARACION CONDICION' | || COMPARACION CONDICION' | ε

COMPARACION -> BOOL COMPARACION'
COMPARACION' -> == BOOL COMPARACION' | < BOOL COMPARACION' | > BOOL COMPARACION' | ε
BOOL -> identificador | ! identificador | CONSTANTE
COMENTARIO -> # INSTRUCCION
INSTRUCCION -> DECLARACION | ASIGNACION | LLAMADA
LLAMADA -> FUNCION PARAMETROS
PARAMETROS -> CONSTANTE PARAMETROS'
PARAMETROS' -> ESPACIO CONSTANTE PARAMETROS' | ε
FUNCION -> obtener | publicar | cambiar | borrar
DECLARACION -> variable ESPACIO identificador
ASIGNACION -> identificador ESPACIO = ESPACIO EXP
EXP -> TERM EXP’
EXP’ -> + TERM EXP’ | - TERM EXP’ | ε
TERM -> FACTOR TERM’
TERM’ -> * FACTOR TERM’ | / FACTOR TERM’ | ε
FACTOR -> identificador | CONSTANTE | (EXP)
ESPACIO -> espacio ESPACIO'
ESPACIO' -> espacio ESPACIO' | ε
CONSTANTE -> constante | ‘ CADENA ‘
CADENA -> ∈Σ-’ CADENA'
CADENA' -> ∈Σ-’ CADENA' | ε

Eliminar mismo first

CODIGO -> BLOQUE CODIGO'
CODIGO' -> BLOQUE CODIGO' | ε
BLOQUE -> ACCION BLOQUE'
BLOQUE' -> \n ACCION BLOQUE' | ε
ACCION -> CONDICIONAL | INSTRUCCION | COMENTARIO
CONDICIONAL -> si ESPACIO CONDICION ESPACIO entonces \n BLOQUE \n SINO finsi \n
SINO -> sino \n BLOQUE \n | ε
CONDICION -> COMPARACION CONDICION'
CONDICION' -> && COMPARACION CONDICION' | || COMPARACION CONDICION' | ε
COMPARACION -> BOOL COMPARACION'
COMPARACION' -> == BOOL COMPARACION' | < BOOL COMPARACION' | > BOOL COMPARACION' | ε

BOOL -> identificador | ! identificador | CONSTANTE

COMENTARIO -> # INSTRUCCION

INSTRUCCION -> DECLARACION | ASIGNACION | LLAMADA | ε

LLAMADA -> FUNCION ESPACIO PARAMETROS

PARAMETROS -> CONSTANTE PARAMETROS'

PARAMETROS' -> ESPACIO CONSTANTE PARAMETROS' | ε

FUNCION -> obtener | publicar | cambiar | borrar

DECLARACION -> variable ESPACIO identificador

ASIGNACION -> identificador ESPACIO = ESPACIO EXP

EXP -> TERM EXP’

EXP’ -> + TERM EXP’ | - TERM EXP’ | ε

TERM -> FACTOR TERM’

TERM’ -> * FACTOR TERM’ | / FACTOR TERM’ | ε

FACTOR -> identificador | CONSTANTE | (EXP)

ESPACIO -> espacio ESPACIO'

ESPACIO' -> espacio ESPACIO' | ε

CONSTANTE -> constante | ‘ CADENA ‘

CADENA -> ∈Σ-’ CADENA'

CADENA' -> ∈Σ-’ CADENA' | ε

Análisis first and follow

First

first(CADENA’) = {ε ∈Σ-’}

first(CADENA) = {∈Σ-’}

first(CONSTANTE) = {constante ‘}

first(ESPACIO’) = {espacio ε}

first(ESPACIO) = {espacio}

first(FACTOR) = {identificador constante ‘ (}

first(TERM’) = {* / ε}

first(TERM) = {identificador constante ‘ (}

first(EXP’) = {+ - ε}

first(EXP) = {identificador constante ‘ (}

first(ASIGNACION) = {identificador}

first(DECLARACION) = {variable}

first(FUNCION) = {obtener publicar cambiar borrar}

first(PARAMETROS’) = {espacio ε}

first(PARAMETROS) = { constante ‘}

first(LLAMADA) = {obtener publicar cambiar borrar}

first(INSTRUCCION) = {variable identificador obtener publicar cambiar borrar}

first(COMENTARIO) = {#}

first(BOOL) = {identificador ! constante ‘}

first(COMPARACION’) = {== < > ε}

first(COMPARACION) = {identificador ! constante ‘}

first(CONDICION’) = {&& || ε}

first(CONDICION) = {identificador ! constante ‘}

first(SINO) = {sino ε}

first(CONDICIONAL) = {si}

first(ACCION) = {si variable identificador obtener publicar cambiar borrar #}

first(BLOQUE’) = {\n ε}

first(BLOQUE) = {si variable identificador obtener publicar cambiar borrar #}

first(CODIGO’) = {si variable identificador obtener publicar cambiar borrar ε #}

first(CODIGO) = {si variable identificador obtener publicar cambiar borrar #}

Follow

follow(CODIGO) = { EOF }

follow(CODIGO') = { EOF }

follow(BLOQUE) = { \n # EOF borrar cambiar identificador obtener publicar si variable }

follow(BLOQUE') = { \n # EOF borrar cambiar identificador obtener publicar si variable }

follow(ACCION) = { \n # EOF borrar cambiar identificador obtener publicar si variable }

follow(CONDICIONAL) = { \n # EOF borrar cambiar identificador obtener publicar si variable}

follow(SINO) = { finsi }

follow(CONDICION) = { space }

follow(CONDICION') = { space }

follow(COMPARACION) = { space && || }

follow(COMPARACION') = { space && || }

follow(BOOL) = { space && < == > || }

follow(COMENTARIO) = { \n # EOF borrar cambiar identificador obtener publicar si variable }

follow(INSTRUCCION) = { \n # EOF borrar cambiar identificador obtener publicar si variable}

follow(LLAMADA) = { \n # EOF borrar cambiar identificador obtener publicar si variable }

follow(PARAMETROS) = { \n # EOF borrar cambiar identificador obtener publicar si variable}

follow(PARAMETROS') = { \n # EOF borrar cambiar identificador obtener publicar si variable}

follow(FUNCION) = { constante ‘ }

follow(DECLARACION) = { \n # EOF borrar cambiar identificador obtener publicar si variable}
follow(ASIGNACION) = { \n # EOF borrar cambiar identificador obtener publicar si variable }
follow(EXP) = { \n #) EOF borrar cambiar identificador obtener publicar si variable }
follow(EXP') = { \n #) EOF borrar cambiar identificador obtener publicar si variable }
follow(TERM) = { \n #) + - EOF borrar cambiar identificador obtener publicar si variable }
follow(TERM') = { \n #) + - EOF borrar cambiar identificador obtener publicar si variable }
follow(FACTOR) = { \n #) * + - / EOF borrar cambiar identificador obtener publicar si variable}
follow(ESPACIO) = { ! = constante identificador ‘ }
follow(ESPACIO') = { ! = constante identificador ‘ }
follow(CONSTANTE) = { \n space # &&) * + - / < == > EOF borrar cambiar identificador obtener publicar si variable || }
follow(CADENA) = { ‘ }
follow(CADENA') = { ‘ }

Retos y dudas

La gramática planteada al inicio no respetaba jerarquía de signos, por lo que se tuvo que alterar y dividir EXP a TERM y FACTOR.

La simbología nos dificultó la expresión de ciertos caracteres necesarios para el lenguaje como el salto de línea y el espacio, aunque se pudo solucionar con otra representación y conversión mediante condicionales para la lectura de las reglas en el código.

Al momento de hacer el análisis first nos dimos cuenta de que dos reglas de la misma producción empezaban con el mismo símbolo, por lo que se tuvo que cambiar para que la gramática pueda ser considerada LL(1).

Como dudas, se encuentra la construcción del árbol sintáctico, ya que en la revisión del algoritmo LL(1) que se revisó no se consideraba ese producto, sino la validación de la gramática. El follow que se presenta en dicha revisión parece ser diferente al visto en clase.

Notas y comentarios

Se creó un lector de gramática para el compilador y un método para eliminar la recursión a la izquierda, es por eso que el formato de las reglas cambió.

La primera versión de la gramática contiene errores mixtos corregidos en la versión más reciente.

El análisis first y follow se realizó con la gramática no LL(1) y fue modificado cuando se cambió ese detalle de la gramática.

Lista de Referencias

No aplica.