Gramática

CompScript

Diego Abraham Robles Meza
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Organización de lenguajes y compiladores 1
Sección “B”

Terminales:

“true” = {TRUE}

“false” = {FALSE}

“print” = {PRINT}

“println” = {PRINTLN}

“typeof” = {TYPE_OF}

“toString” = {TO_STRING}

“toCharArray” = {TO_CHAR_ARRAY}

“length” = {LENGTH}

“toLower” = {TO_LOWER}

“toUpper” = {TO_UPPER}

“round” = {ROUND}

“if” = {IF}

“else” = {ELSE}

“do” = {DO}

“while” = {WHILE}

“switch” = {SWITCH}

“case” = {CASE}

“default” = {DEFAULT}

“for” = {FOR}

“break” = {BREAK}

“continue” = {CONTINUE}

“return” = {RETURN}

“new” = {NEW}

“void” = {VOID}

“run” = {RUN}

“int” = {INT]

“double” = {DOUBLE}

“char” = {CHAR}

“boolean” = {BOOLEAN}

“string” = {STRING}

“vector” = {VECTOR}

\d+\.\d+ = {DECIMAL}

\d+ = {ENTERO}

\'.\' = {CARACTER}

(\"[^\"]*\") = {CADENA}

([a-zA-Z])[a-zA-Z0-9_]* = {IDENTIFICADOR}

“(“ = {PAR_ABRE}

“)” = {PAR_CIERRA}

“{“ = {LL_ABRE}

“}” = {LL_CIERRA}

“[“ = {COR_ABRE}

“]” = {COR_CIERRA}

“==” = {IGUAL_IGUAL}

“<=” = {MENOR_IGUAL}

“<” = {MENOR}

“>=” = {MAYOR_IGUAL}

“>” = {MAYOR}

“!=” = {DIFERENTE}

“=” = {IGUAL}

“?” = {QUESTION}

“||” = {OR}

“&&” = {AND}

“!” = {NOT}

“,” = {COMA}

“:” = {DOS_PUNTOS}

“;” = {PUNTO_Y_COMA}

“++” = {MAS_MAS}

“+” = {MAS}

“- -“ = {MENOS_MENOS}

“-“ = {MENOS}

“*” = {POR}

“/” = {DIVIDIR}

“^” = {POTENCIA}

“%” = {MODULO}

No terminales:

ini

instrucciones

inicio

declaración

declaracion_multiple

asignacion

print

listaExpr

if

else

switch

lista_case

case

statement_case

default

while

for

do_while

statement

function

tipo_dato

parametros

tipos

llamada_funcion

return

incremento_decremento

ternario

casteos

declaracion_vectores

vector_una_dimension

vector_dos_dimensiones

acceso_vectores

modificacion_vectores

type_of

to_string

to_char_array

length

to_lower

to_upper

round

expresion

Producciones:

Producción inicial:

ini ::= **instrucciones** EOF

Producción de instrucciones:

Esta producción es una de las principales, ya que puede contener una o muchas instrucciones de otros tipos, sirve para encapsular varias sentencias dentro de algún bloque como un “if”, “else” o el cuerpo de alguna función.

instrucciones ::= **instrucciones inicio**

| **inicio**

Producción de inicio:

Esta producción es la que se encarga de recibir una instrucción, la producción de instrucciones depende de esta y de esta derivan todas las demás producciones.

inicio ::= **asignacion** PUNTO_Y_COMA

| **declaracion** PUNTO_Y_COMA

| **print** PUNTO_Y_COMA

| **if**

| **while**

| **do_while**

| **for**

| **switch**

| **function**

| **llamada_funcion** PUNTO_Y_COMA

| **incremento_decremento** PUNTO_Y_COMA

| BREAK PUNTO_Y_COMA

| CONTINUE PUNTO_Y_COMA

| **return** PUNTO_Y_COMA

| **declaracion_vectores** PUNTO_Y_COMA

| **modificacion_vectores** PUNTO_Y_COMA

Producción de declaración:

Esta producción es la que se encarga de recibir una entrada como las siguientes:

tipo_dato id

tipo_dato lista_ids

tipo_dato id = valor

tipo_dato lista_ids = valor

tipo_dato id = casteo

tipo_dato lista_ids = casteo

en donde: tipo_dato puede ser int, double, char, boolean, string.

declaracion ::= INT **declaracion_multiple**

| DOUBLE **declaracion_multiple**

| CHAR **declaracion_multiple**

| BOOLEAN **declaracion_multiple**

| STRING **declaracion_multiple**

| INT **declaracion_multiple** IGUAL **expresion**

| DOUBLE **declaracion_multiple** IGUAL **expresion**

| CHAR **declaracion_multiple** IGUAL **expresion**

| BOOLEAN **declaracion_multiple** IGUAL **expresion**

| STRING **declaracion_multiple** IGUAL **expresion**

| INT **declaracion_multiple** IGUAL **casteos**

| DOUBLE **declaracion_multiple** IGUAL **casteos**

| CHAR **declaracion_multiple** IGUAL **casteos**

| BOOLEAN **declaracion_multiple** IGUAL **casteos**

| STRING **declaracion_multiple** IGUAL **casteos**

declaracion_multiple ::= **declaracion_multiple** COMA IDENTIFICADOR

| IDENTIFICADOR

Producción de asignación:

Esta producción se encarga de reconocer una entrada como las siguientes:

Id = valor

asignacion ::= IDENTIFICADOR IGUAL **expresion**

| IDENTIFICADOR

Producción de Print y Println:

Esta producción reconoce una entrada como la siguiente:

Print (valor)

Print(lista_valores)

Println (valor)

Println(lista_valores)

print ::= PRINT PAR_ABRE **listaExpr** PAR_CIERRA

| PRINTLN PAR_ABRE **listaExpr** PAR_CIERRA

listaExpr ::= **listaExpr** COMA **expresion**

| **expresion**

Producción de IF y Else:

Esta producción recibe una entrada como convencionalmente viene un if, teniendo las siguientes posibilidades de ocurrencia:

If (condición) { statement } else { statement }

If (condición) { statement }

If (condición) { statement } else if (condicion) { statement }

Tomar en cuenta que un if y else pueden tener anidados muchos ifs en conjunto con else y que la condición a evaluar debe ser de tipo Booleana.

if ::= IF PAR_ABRE **expresion** PAR_CIERRA **statement elsE**

elsE ::= ELSE **statement**

| ELSE **if**

|

Producción de Switch:

Esta producción acepta una entrada como la del switch convencionalmente vista en el lenguaje de programación JAVA, en donde a partir de una condición a evaluar se compara con los distintos casos que existen y en caso de que ninguna opción cumpla se ejecutará la opción por defecto si es que existe:

Switch (opción) { lista_casos default }

switch ::= SWITCH PAR_ABRE **expresion** PAR_CIERRA LL_ABRE **lista_case default** LL_CIERRA

| SWITCH PAR_ABRE **expresion** PAR_CIERRA LL_ABRE **default** LL_CIERRA

lista_case ::= **lista_case case**

| **case**

case ::= CASE **expresion** DOSPUNTOS **statement_case**

statement_case ::= **instrucciones**

|

default ::= DEFAULT DOS_PUNTOS **statement_case**

|

Producción de While y Do While:

Esta producción al igual que la del switch sigue la sintaxis de un bucle while, similar a la de los lenguajes más comunes:

While (condición) { statement }

Do { statement } while (condición)

Tomar en cuenta que la condición a evaluar debe ser de tipo Booleana.

while ::= WHILE PAR_ABRE **expresion** PAR_CIERRA **statement**

do_while ::= DO **statement** WHILE PAR_ABRE **expresion** PAR_CIERRA

Producción de For:

Esta producción se encarga de recibir una entrada como las del bucle for convencional, en donde se tienen 3 parámetros:

For (declaracion/asignacion ; condición ; actualización) { statement }

for ::= FOR PAR_ABRE **declaracion** PUNTO_Y_COMA **expresion** PUNTO_Y_COMA **incremento_decremento** PAR_CIERRA **statement**

| FOR PAR_ABRE **declaracion** PUNTO_Y_COMA **expresion** PUNTO_Y_COMA **asignacion** PAR_CIERRA **statement**

| FOR PAR_ABRE **asignacion** PUNTO_Y_COMA **expresion** PUNTO_Y_COMA **incremento_decremento** PAR_CIERRA **statement**

| FOR PAR_ABRE **asignacion** PUNTO_Y_COMA **expresion** PUNTO_Y_COMA **asignacion** PAR_CIERRA **Statement**

Producción Statement (Cuerpo):

Esta producción es una de las más importantes, ya que es la encargada de estableces todo el cuerpo de las instrucciones evaluadas dentro de las estructuras más conocidas (if, else, while, switch, for, etc.). Se establece de la siguiente forma:

{ instrucciones }

{ }

statement ::= LL_ABRE **instrucciones** LL_CIERRA

| LL_ABRE LL_CIERRA

Producción de Funciones:

Esta producción es la encargada del reconocimiento de funciones, ya sea que tengan un tipo concreto(void, int, double, char, boolean, string) o que no tengan uno establecido(en este caso se le asigna void por defecto).

Id () : tipo { statement }

Id () { statement }

Id (parametros) : tipo { statement }

Id (parametros) { statement }

Tomar en cuenta que los parámetros al crear la función tienen que ser del mismo tipo al ser llamada esta función y se ingresan en el orden que los pide la función.

function ::= IDENTIFICADOR PAR_ABRE PAR_CIERRA DOS_PUNTOS **tipo_dato statement**

| IDENTIFICADOR PAR_ABRE **parametros** PAR_CIERRA DOS_PUNTOS **tipo_dato statement**

| IDENTIFICADOR PAR_ABRE PAR_CIERRA **statement**

| IDENTIFICADOR PAR_ABRE **parametros** PAR_CIERRA **statement**

tipo_dato ::= INT

| DOUBLE

| CHAR

| BOOLEAN

| STRING

| VOID

Producción de Parámetros:

En la creación de las funciones puede que estas tengan parámetros dentro de los paréntesis. Esta producción es la que se encarga de reconocer todos los parámetros que reciben las funciones y pueden ser de la siguiente forma:

Tipo id

Tipo [ ] id

Tipo id [ ]

Tipo [ ] [ ] id

Tipo id [ ] [ ]

parametros ::= **parametros** COMA **tipos** IDENTIFICADOR

| **tipos** IDENTIFICADOR

| **parametros** COMA **tipos** COR_ABRE COR_CIERRA IDENTIFICADOR

| **tipos** COR_ABRE COR_CIERRA IDENTIFICADOR

| **parametros** COMA **tipos** IDENTIFICADOR COR_ABRE COR_CIERRA

| **tipos** IDENTIFICADOR COR_ABRE COR_CIERRA

| **parametros** COMA **tipos** COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR

| **tipos** COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR

| **parametros** COMA **tipos** IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA

| **tipos** IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA

tipos ::= INT

| DOUBLE

| CHAR

| BOOLEAN

| STRING

| VECTOR

Producción de Llamada Función:

Esta producción se encarga de reconocer las llamadas que se realicen a funciones dentro del programa, se hacen de la siguiente manera:

Id ();

Id(parámetros) ;

Tener en cuenta que si una función es llamada y la cantidad de parámetros es incorrecta o si el tipo no es del mismo tipo que recibe la función al ser declarada esto ocasionará un error.

llamada_funcion ::= IDENTIFICADOR PAR_ABRE PAR_CIERRA

| IDENTIFICADOR PAR_ABRE **ListaExpr** PAR_CIERRA

| RUN IDENTIFICADOR PAR_ABRE PAR_CIERRA

| RUN IDENTIFICADOR PAR_ABRE **ListaExpr** PAR_CIERRA

Producción de Return:

Esta producción es la que se encarga de reconocer la instrucción return dentro del lenguaje.

Return ;

Return valor;

return ::= RETURN

| RETURN **expresion**

Producción de Incremento y Decremento:

Esta producción se encarga de reconocer una entrada como la siguiente;

Id ++;

Id --;

incremento_decremento ::= IDENTIFICADOR MAS_MAS

| IDENTIFICADOR MENOS_MENOS

Producción de Operador Ternario:

Esta producción se encarga de reconocer una entrada como la siguiente:

(condición) ? valor : valor ;

ternario ::= **expresion** QUESTION **expresion** DOS_PUNTOS **expresion**

Producción de Casteos:

Esta producción se encarga de reconocer una entrada como la siguiente:

(tipo) valor;

casteos ::= PAR_ABRE DOUBLE PAR_CIERRA **expresion**

| PAR_ABRE CHAR PAR_CIERRA **expresion**

| PAR_ABRE INT PAR_CIERRA **expresion**

| PAR_ABRE BOOLEAN PAR_CIERRA **expresion**

| PAR_ABRE STRING PAR_CIERRA **expresion**

Producción de Vectores(1 y 2 dimensiones):

Esta producción genera otras dos producciones, la cuales sirven para reconocer entradas como las siguientes:

Tipo id [ ] = new tipo [#];

Tipo [ ] id = new tipo[#];

Tipo id [ ] = [n1, n2, n3, …];

Tipo [ ] id = [n1, n2, n3, …];

Tipo id [ ] [ ] = new tipo [#] [#];

Tipo [ ] [ ] id = new tipo [#] [#];

Tipo id [ ] [ ] id = [[n1, n2, n3, …], [n1, n2, n3, …], …];

Tipo [ ] [ ] id = [[n1, n2, n3, …], [n1, n2, n3, …], …];

Donde # es el tamaño de las filas o columnas y este debe de ser de tipo entero(int).

declaracion_vectores ::= **vector_una_dimension**

| **vector_dos_dimensiones**

vector_una_dimension ::= INT IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL NEW INT COR_ABRE **expresion** COR_CIERRA

| INT IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| DOUBLE IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL NEW DOUBLE COR_ABRE **expresion** COR_CIERRA

| DOUBLE IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL NEW CHAR COR_ABRE **expresion** COR_CIERRA

| CHAR IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL **to_char_array**

| BOOLEAN IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL NEW BOOLEAN COR_ABRE **expresion** COR_CIERRA

| BOOLEAN IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| STRING IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL NEW STRING COR_ABRE **expresion** COR_CIERRA

| STRING IDENTIFICADOR COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| INT COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW INT COR_ABRE **expresion** COR_CIERRA

| INT COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| DOUBLE COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW DOUBLE COR_ABRE **expresion** COR_CIERRA

| DOUBLE COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW CHAR COR_ABRE **expresion** COR_CIERRA

| CHAR COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL **to_char_array**

| BOOLEAN COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW BOOLEAN COR_ABRE **expresion** COR_CIERRA

| BOOLEAN COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| STRING COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW STRING COR_ABRE **expresion** COR_CIERRA

| STRING COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

vector_dos_dimensiones ::= INT IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL NEW INT COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| INT IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| DOUBLE IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL NEW DOUBLE COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| DOUBLE IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL NEW CHAR COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| CHAR IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| BOOLEAN IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL NEW BOOLEAN COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| BOOLEAN IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| STRING IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL NEW STRING COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| STRING IDENTIFICADOR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| INT COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW INT COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| INT COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| DOUBLE COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW DOUBLE COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| DOUBLE COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| CHAR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW CHAR COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| CHAR COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| BOOLEAN COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW BOOLEAN COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| BOOLEAN COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

| STRING COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL NEW STRING COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

| STRING COR_ABRE COR_CIERRA COR_ABRE COR_CIERRA IDENTIFICADOR IGUAL COR_ABRE **ListaExpr** COR_CIERRA

Producción de Acceso a vectores(1 y 2 dimensiones):

En esta producción se reconocen las entradas que hacen referencia a la posición de un arreglo en cierta posición específica:

Id [#];

Id [#] [#];

Tener en cuenta que # debe de ser de tipo entero(int) de lo contrario se considera un error.

acceso_vectores ::= IDENTIFICADOR COR_ABRE **expresion** COR_CIERRA

| IDENTIFICADOR COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA

Producción de Modificación de vectores(1 y 2 dimensiones):

modificacion_vectores ::= IDENTIFICADOR COR_ABRE **expresion** COR_CIERRA IGUAL **expresion**

| IDENTIFICADOR COR_ABRE **expresion** COR_CIERRA COR_ABRE **expresion** COR_CIERRA IGUAL **expresion**

Producciones de funciones nativas:

Estas producciones siguen una sintaxis bastante parecida, la cual consta de los siguientes tokens:

Tipo_funcion_nativa (expresion);

En donde Tipo_funcion_nativa puede ser:

- typeOf
- toString
- toCharArray
- length
- toLower
- toUpper
- Round

type_of ::= TYPEOF PAR_ABRE **expresion** PAR_CIERRA

to_string ::= TOSTRING PAR_ABRE **expresion** PAR_CIERRA

to_char_array ::= TOCHARARRAY PAR_ABRE **expresion** PAR_CIERRA

length_ ::= LENGTH PAR_ABRE **expresion** PAR_CIERRA

to_lower ::= TOLOWER PAR_ABRE **expresion** PAR_CIERRA

to_upper ::= TOUPPER PAR_ABRE **expresion** PAR_CIERRA

round ::= ROUND PAR_ABRE **expresion** PAR_CIERRA

Producción de Expresión:

En esta producción se reconocen todas las entradas que van a ser asignadas a diferentes funciones o variables:

expresion ::= MENOS **expresion**

| **expresion** MAS **expresion**

| **expresion** MENOS **expresion**

| **expresion** POR **expresion**

| **expresion** DIVIDIR **expresion**

| **expresion** POTENCIA **expresion**

| **expresion** MODULO **expresion**

| **expresion** IGUAL_IGUAL **expresion**

| **expresion** DIFERENTE **expresion**

| **expresion** MAYOR_IGUAL **expresion**

| **expresion** MENOR_IGUAL **expresion**

| **expresion** MAYOR **expresion**

| **expresion** MENOR **expresion**

| PAR_ABRE **expresion** PAR_CIERRA

| **expresion** AND **expresion**

| **expresion** OR **expresion**

| NOT **expresion**

| CADENA

| ENTERO

| DECIMAL

| TRUE

| FALSE

| CARACTER

| IDENTIFICADOR

| **ternario**

| **incremento_decremento**

| COR_ABRE **ListaExpr** COR_CIERRA

| **acceso_vectores**

| **type_of**

| **to_string**

| **to_char_array**

| **length_**

| **to_lower**

| **to_upper**

| **round**

| **llamada_funcion**