Universidad Nacional Autónoma de México
Semestre 2021-1
Compiladores
Gramática
Profesor: Adrián Ulises Mercado Martínez

Elaborar una front-end para la gramática descrita en la sección de gramática bajo las siguientes especificaciones

1. Elaborar el analizador léxico en flex: debe reconocer los tokens y retornar un entero por cada token, además debe considerar una variable global al programa que permita almacenar el lexema de los tokens que lo requieran(Token actual).
2. Debe aceptar colocar comentarios del tipo $< *$ comentario $* >$ para varias líneas y para una sola línea $--$comentario de una sola línea
3. Elaborar un parser recursivo para la gramática que determine si el archivo de código fuente pertenece o no al lenguaje generado por la gramática
4. Elaborar el analizador semántico que realice:
   (a) Buscar si un identificador ya fue declarado al momento de declaraciones de variables y funciones
   (b) Cada vez que se use un identificador en una instrucción donde no se declaran variables, buscar que exista el identificador para poder usarlo.
   (c) Validar los tipos de operandos en las operaciones aritméticas y booleanas
   (d) Validar el uso de indices en los arreglos
   (e) Validar el tipo de retorno de la función contra las instrucciones de retorno de la función.
   (f) Validar el número de argumentos y tipo en las llamadas a funciones
5. Agregar las acciones semánticas para la generación de código intermedio, entre las que se encuentran la generación de etiquetas y de variables temporales.
6. El programa debe leer un programa fuente especificado desde línea de comandos y mostrar lo siguiente:
   (a) Mostrar la tabla de símbolos (Para cada función)
   (b) Mostrar la tabla de tipos(Para cada función)
   (c) Escribir en un archivo con el nombre del programa de entrada y extensión ci, el código intermedio generado para ese programa
   (d) En caso de ocurrir errores indicar el tipo de error (léxico, sintáctico o semántico), la línea donde ocurrió el error, y el caracter o token que lo genera
7. Documentos a entregar
   (a) Diseño de las expresiones regulares
   (b) Proceso para quitar la recursividad y los factores de la gramática
   (c) Diagramas de sintaxis de la gramática
   (d) La definición dirigida por sintaxis
   (e) El esquema de traducción obtenido,

Gramática

PRODUCCIÓN
programa → declaraciones funciones
declaraciones → tipo lista_var **;** declaraciones | ε
tipo → basico compuesto
basico → **int** | **float** | **char** | **double**| **void**
compuesto → **(numero)** compuesto | ε
lista_var → lista_var **, id** | **id**
funciones → **func** tipo **id (** argumentos **)** bloque funciones | ε
argumentos→ lista_args | ε
lista_args→ lista_args, tipo **id** | tipo **id**
bloque → { declaraciones instrucciones }
instrucciones → instrucciones sentencia | sentencia
sentencia → localizacion **=** bool| **if(** bool **)** sentencia
| **if(** bool **)** sentencia **else** sentencia | **while(** bool **)** sentencia
| **do** sentencia **while(** bool **)** | **break ;** | bloque | **return** exp ;| **return;**
| **switch(** bool **)** { casos }
casos → caso casos| ε | predeterminado
caso → **case numero:** instrucciones
predeterminado→ **default:** instrucciones
bool → bool || comb | comb
comb → comb **&&** igualdad | igualdad
igualdad → igualdad **==** rel | igualdad **!=** rel | rel
rel → exp < exp | exp <= exp | exp >= exp |
| exp > exp | exp
exp → exp + term | exp − term | term
term → term ∗ unario | term / unario | term **%** unario | unario
unario → **!**unario | − unario | factor
factor → **(**bool**)** | localizacion | **numero** | **cadena** | **true** | **false** | **id(**parametros**)**
parametros → lista_param | ε
lista_param → lista_param **,** bool | bool
localizacion→ localizacion **(** bool **)** | id

REGLAS DE PRODUCCIÓN	REGLAS SEMÁNTICAS
declaraciones → tipo lista_var **;** declaraciones	lista_var.tipo = tipo.tipo
declaraciones → ε	
tipo → basico compuesto	compuesto.base = basico.base tipo.tipo = compuesto.tipo
basico → **int**	base.tipo = int
basico →**float**	base.tipo = float
basico → **char**	base.tipo = char
basico → **double**	base.tipo = double
basico → **void**	base.tipo = void
compuesto → **(numero)** compuesto_1	compuesto.tipo = TT.insertar("array", num.val, compuesto_1.tipo) compuesto_1.base = compuesto.base
compuesto → ε	compuesto.tipo = compuesto.base

lista_var → lista_var **, id**	lista_var$_1$.tipo = lista_var.tipo
	Si !TS.buscar(id) Entonces
	  TS.insetar(id)
	  dir = dir + TT.getTam(lista_var.tipo)
	Sino
	  error("El id no está declarado")
	FinSi

lista_var → **, id**	Si !TS.buscar(id) Entonces
	  TS.insetar(id)
	  dir = dir + TT.getTam(lista_var.tipo)
	Sino
	  error("El id no está declarado")
	FinSi