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 analizador sintáctico 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

Definición Dirigida por Sintaxis

REGLAS DE PRODUCCIÓN	REGLAS SEMÁNTICAS
programa→declaraciones funciones	PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0
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 = PilaTT.top().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 ! PilaTS.top().buscar(id) Entonces PilaTS.top().insetar(id, lista_var.tipo, dir, "var", NULO) dir = dir + PilaTT.top().getTam(lista_var.tipo) Sino error("El id no está declarado") FinSi
lista_var → **, id**	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insetar(id, lista_var.tipo, dir, "var", NULO) dir = dir + PilaTT.top().getTam(lista_var.tipo) Sino error("El id no está declarado") FinSi
funciones → **func** tipo **id (** argumentos **)** bloque funciones	ListaRetorno = NULO PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos) PilaDir.push(dir) dir = 0 Si ! PilaTS.top().buscar(id) Entonces Si equivalentesLista(ListaRetorno, tipo.tipo) Entonces PilaTS.top().insetar(id, tipo.tipo, –, 'func', argumentos.lista) genCod(label(id)) bloque.sig = nuevaEtq() genCod(label(bloque.sig)) Sino error("Los tipos o el número de argumentos no es correcto") FinSi Sino error("El id no está declarado") FinSi PilaTS.pop() PilaTT.pop() dir = PilaDir.pop()
funciones → ε	
argumentos→ lista_args	argumentos.lista = lista_args.lista
argumentos→ ε	argumentos.lista = NULO
lista_args→ lista_args$_1$, tipo **id**	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insetar(id, tipo.tipo, dir, "param", NULO) dir = dir + PilaTT.top().getTam(lista_var.tipo) Sino error("El id no está declarado") FinSi

	lista_args.lista = lista_args$_1$.lista lista_args.lista.agregar(tipo.tipo)
lista_args→tipo **id**	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insetar(id, tipo.tipo, dir, "param", NULO) dir = dir + PilaTT.top().getTam(lista_var.tipo) Sino error("El id no está declarado") FinSi lista_args.lista = nuevaListaArgs() lista_args.lista.agregar(tipo.tipo)
bloque → { declaraciones instrucciones }	instrucciones.sig = bloque.sig genCod(label(instrucciones.sig))
instrucciones → instrucciones$_1$ sentencia	instrucciones$_1$.sig = nuevaEtq() setencia.sig = instrucciones.sig genCod(label(instrucciones$_1$.sig))
instrucciones → sentencia	setencia.sig = instrucciones.sig
sentencia → localizacion **=** bool **;**	Si equivalentes(localizacion.tipo, bool.tipo) Entonces d$_1$ = reducir(bool.dir, bool.tipo, localizacion.tipo) genCod(localizacion.dir '=' d1) Sino error("Tipos incompatibles") FinSi
sentencia → **if(** bool **)** sentencia$_1$	bool.vddr = nuevaEtq() bool.fls = sentencia.sig sentencia$_1$.sig = sentencia.sig genCod(label(bool.vddr))
sentencia → **if(** bool **)** sentencia$_1$ **else** sentencia$_2$	bool.vddr = nuevaEtq() bool.fls = nuevaEtq() sentencia$_1$.sig = sentencia.sig sentencia$_2$.sig = sentencia.sig genCod(label(bool.vddr)) genCod('goto' sentencia.sig) genCod(label(bool.fls))
sentencia → **while(** bool **)** sentencia$_1$	sentencia$_1$.sig = nuevaEtq() bool.vddr = nuevaEtq() bool.fls = sentencia.sig genCod(label(sentencia$_1$.sig)) genCod(label(bool.vddr)) genCod('goto' sentencia$_1$.sig)
sentencia → **do** sentencia$_1$ **while(** bool **)**	bool.vddr = nuevaEtq() bool.fls = sentencia.sig sentencia$_1$.sig = nuevaEtq() genCod(label(bool.vrdd)) genCod(label(sentencia$_1$.sig))
sentencia → **break ;**	genCod(goto sentencia.sig)
sentencia → bloque	bloque.sig = sentencia.sig

sentencia → **return** exp **;**	ListaRetorno.agregar(exp.tipo)
sentencia → **return;**	ListaRetorno.agregar(void)
sentencia → **switch(** bool **)** { casos }	casos.sig = sentencia.sig casos.id = bool.dir
bool → bool || comb	
bool →comb	
comb → comb **&&** igualdad	
comb → igualdad	
igualdad → igualdad **==** rel	
igualdad →igualdad **!=** rel	
igualdad →rel	
rel → exp < exp	
rel →exp <= exp	
rel →exp >= exp	
rel →exp > exp rel → exp	
exp → exp + term	
exp →exp − term	
exp →term	
term → term ∗ unario	
term →term / unario	
term →term **%** unario	
term →unario	
unario → **!**unario	
unario →− unario	
unario →factor	
factor → **(**bool**)**	
factor →localizacion	
factor →**numero**	
factor →**cadena**	

factor →**true**	
factor →**false**	
factor → **id(**parametros**)**	
parametros → lista_param	parametros.lista = lista_parametros.lista
parametros → ε	parametros.lista = NULO
lista_param → lista_param$_1$ **,** bool	lista_param.lista = lista_param$_1$.lista lista_param.lista.agregar(bool.tipo)
lista_param →bool	lista_param.lista = nuevaListaArgs() lista_param.lista.agregar(bool.tipo)
localizacion→ localizacion **(** bool **)** id	

- equivalentesLista, es una función que recibe una lista con el tipo de retorno de cada una de las sentencias return, y el tipo de retorno de la función. Compara cada uno de los tipos de retorno con el tipo de la función y si todos son equivalentes al tipo de la función retorna verdadero en caso contrario falso.
- nuevaListaArgs(), crea una nueva lista donde se puedan almacenar los tipos de los argumentos