Universidad Nacional Autónoma de México
Semestre 2025-1
Compiladores
Gramática
Profesora: Ariel Adara Mercado Martínez

Objetivo:

Elaborar un compilador para la gramática descrita en la sección de gramática que genere código objeto para MIPS.

Gramática

PRODUCCIÓN
programa → decl_proto decl_var decl_func
decl_proto → **proto** tipo **id (** argumentos **) ;** decl_proto | ε
decl_var → tipo lista_var **;** decl_var | ε
tipo → basico compuesto | **struct {** decl_var **}** | puntero
puntero → **ptr** basico
basico → **int** | **float** | **double** | **complex** | **rune** | **void** | **string** |
compuesto → [**literal_entera**] compuesto | ε
lista_var → lista_var **,** **id** | **id**
decl_func → **func** tipo **id (** argumentos **)** bloque decl_func | ε
argumentos→ lista_args | ε
lista_args→ lista_args, tipo **id** | tipo **id**
bloque → **{** declaraciones instrucciones **}**
instrucciones → instrucciones sentencia | sentencia
sentencia → parte_izquierda **=** exp **;** | **if(** exp **)** sentencia | **if(** exp **)** sentencia **else** sentencia | **while(** exp **)** sentencia | **do** sentencia **while(** exp **)** | **break ;** | bloque | **return** exp **;** | **return;** | **switch(** exp **) {** casos **}** | **print** exp **;** | **scan** parte_izquierda
casos → caso casos| ε | predeterminado
caso → **case** opcion **:** instrucciones
opcion → literal_enter | literal_runa
predeterminado→ **default:** instrucciones
parte_izquierda → **id** localizacion |**id**
exp → exp || exp | exp **&&** exp | exp **==** exp | exp **!=** exp | exp $<$ exp | exp $<=$ exp | exp $>=$ exp | exp $>$ exp | exp $+$ exp | exp $-$ exp | exp $*$ exp | exp / exp | exp **%** exp | exp // exp | **!** exp | $-$ exp | **(**exp**)** | **id** localizacion | **false** | **literal_cadena** | **true** | **literal_runa** | **literal_entera** | literal_flotante | literal_doble | literal_compleja | **id(**parametros**)**| **id**
parametros → lista_param | ε
lista_param → lista_param **,** exp | exp
localizacion→ arreglo | estructurado
arreglo → arreglo [exp] | [exp]
estructurado → estructurado **.** **id** | **.** **id**

Análisis Léxico

1. Los identificadores deben comenzar con letra o guión bajo, y podrán o no estar seguido de cero más dígitos, letras o guiones bajos. Las letras aceptadas con todas las letras válidas para el español, incluyendo letras con acentos y la u con diéresis.

2. Los números deben clasificarse en tres tipos: enteros, decimales y complejos. Los números decimales deben aceptar notación exponencial. Además, se debe contemplar la notación de lenguajes tipo C para diferenciar entre números de precisión simple y doble. En los números complejos, tanto la parte real como la

imaginaria pueden representarse con números flotantes de precisión simple o doble.

3. Una runa es un carácter, pero este tipo de dato incluye letras con acentos, diéresis y la letra "ñ". Las runas deben estar encerradas entre comillas simples. También deben considerarse en la expresión regular los caracteres de escape, así como símbolos como @, #, % y otros.
4. Una cadena es una secuencia de caracteres delimitada por comillas dobles.

Análisis Sintáctico

1. Se puede implementar utilizando análisis sintáctico descendente recursivo, análisis sintáctico descendente LL(1) utilizando Javacc o análisis sintáctico LR(1) utilizando Byacc
2. Se deben generar diagramas de sintaxis para la gramática. *https://rr.red-dove.com/ui*
3. En caso de que exista, se debe eliminar la ambigüedad en la gramática.
4. Se deben eliminar los factores izquierdos.
5. Si se utiliza un análisis sintáctico por descenso recursivo, se debe eliminar la recursividad izquierda.