Universidad Nacional Autónoma de México
Facultad de Ingeniería
Compiladores
2018-1
Proyecto Final
Profesor: Adrián Ulises Mercado Martínez

1 Objetivo:

Construir un compilador para el lenguaje definido por la gramática de la sección 2, Gramática, utilizando lex y yacc, que genere código objeto para MIPS.

2 Gramática

1. P → D F
2. D → T L | ε
3. T → **int** | **float** | **double** | **char** | **void** | **struct** { D }
4. L → L_1 , **id** C | **id** C
5. C → [**numero**] C_1 | ε
6. F → **func** T **id(** A **)**{ D S } F_1 | ε
7. A → G | ε
8. G → G_1, T **id** I | T **id** I
9. I → [ ] I_1 | ε
10. S → S_1 S_2 | **if**(B) S_1 | **if**(B) S_1 **else** S_2 | **while(** B **)** S_1 | **do** S_1 **while(** B **);** | **for(** S_1**;** B**;** S_2**)** S_3 | U = E ; | **return** E**;** | **return;**| { S } | **switch(** E **)**{J K} | **break;** | **print** E;
11. J → **case:** **numero** S J_1 | ε
12. K → **default:** S | ε
13. U → **id** | M | $\mathbf{id}_1\mathbf{.id}_2$
14. M → **id**[E] | M_1 [E]
15. E → E_1 + E_2 | E_1 - E_2 | E_1 * E_2 | E_1 / E_2 | E_1 % E_2 | U | **cadena** | **numero** | **caracter** | **id(** H **)**
16. H → H_1 , E | E
17. B → B_1 || B_2 | B_1 && B_2 | ! B_1 | (B_1) | E_1 R E_2 | **true** | **false**
18. R → < | > | >= | <= | ! = | ==

Símbolo	Nombre	Símbolo	Nombre
P	programa	D	declaración
T	tipo	B	expresión lógica
C	tipo arreglo	L	lista identificadores
F	definición de funciones	A	lista de definición parámetros
G	lista de parámetros	S	sentencias
U	identificadores	E	expresión
I	parámetro tipo arreglo	J	casos
K	caso predeterminado	M	arreglos
R	operadores relacionales	H	lista de paso de parámetros

3 Análisis léxico

El análisis léxico debe reconocer todos los símbolos terminales de la gramática, se debe recordar que de algunos símbolos terminales se deben definir expresiones regulares distintas del símbolo que los está representando en la gramática.

3.1 Consideraciones

3.1.1 Constantes

El lenguaje C−− admite dos tipos de constantes: de cadena y de carácter.Una cadena es cualquier secuencia de caracteres tipo ASCII entre comillas. El símbolo \ sirve para usar caracteres especiales. Los caracteres son cualquier símbolo ASCII delimitado por comillas simples.

3.1.2 Comentarios

Este lenguaje de programación acepta comentarios que comienzan con /* y terminan con */, se deben implementar utilizando estados en lex, un error léxico debe marcarse si el comentario no se cierra.

3.1.3 Espacios en blanco

Todos los espacios en blanco deben ser reconocidos para poder ser ignorados al igual que los comentarios.

3.1.4 Palabras reservadas

Las palabras reservadas para C−− son: int, double, float, char, void, struct, if, else, while, do, for, switch, return, break, case, default, true, false, func y print.

3.1.5 Identificadores

Los identificadores se forma igual que en el lenguaje C.

4 Análisis Sintáctico

En el análisis sintáctico se deben tomar varias decisiones, la gramática en algunas partes presenta ambigüedad, sin embargo; esta se puede ignorar utilizando en bison la precedencia de operadores y dejar que el programa la resuelva de manera automática o eliminar la ambigüedad antes de transcribir la gramática en bison.

Otra consideración que se debe tener en cuenta para resolver la ambigüedad del if-else, es que se debe asignar una precedencia a else como si fuera el operador de mayor precedencia.(Investigar %prec).

5 Análisis Semántico

Las consideraciones semánticas que se deben tomar en cuenta son:

1. El análisis semántico gestiona las tablas de tipos y de símbolos.
2. Los identificadores al ser declarados deben entrar en la tabla de símbolos.En caso de ser arreglos o estructuras se debe tener un apuntador a la tabla de tipos.
3. Los identificadores para poder ser usados en alguna expresión antes debieron ser declarados. Es decir, se deben encontrar en la tabla de símbolos local en caso contrario buscar en la tabla global.
4. En las llamadas a funciones se debe revisar, el número y tipo de parámetros que esta recibiendo.
5. En la declaración de una función se debe verificar el tipo de la función con el tipo del valor de retornado.
6. Las funciones del tipo void no pueden devolver valores.
7. No pueden existir variables ni los parámetros del tipo void.
8. En las expresiones se deben validar los tipos de los operandos, es decir; verificar que los tipos sean iguales o equivalentes.

9. En un programa en C−− como no existen los prototipos, todas las funciones de deben definir antes de ser usadas. La última función en declararse en la función main.
10. Todo programa en C−− debe tener una función main.
11. Los índices en los arreglos sólo pueden ser números enteros.
12. Los índices en los arreglos se deben validar para que no sean números negativos:

    Si se define un arreglo int a[3], este no podrá ser indexado con valores negativos ni con valores mayores a (3-1)
13. Cuando en los arreglos no se usa directamente un número sino una expresión se debe validar que el tipo de expresión sea obligadamente del tipo entero.
14. El valor numérico en los casos de un switch solo pude ser un entero.

6 Generación de Código Intermedio

Se puede generar código de tres direcciones o código de pila. Este código intermedio debe ser almacenado en alguna estructura de datos(usar cuádruplas o tripletas) para su posterior traducción a código objeto, sin embargo también se debe generar un archivo de salida con el código intermedio.

7 Generación de Código Objeto

La salida final del compilador debe ser un archivo que contenga código en ensamblador para MIPS el cuál será ejecutado en algún emulador de MIPS.