Universidad Nacional Autónoma de México
Facultad de Ciencias
Compiladores
2019-2
Proyecto Final
Profesor: Adrián Ulises Mercado Martínez

1 Objetivo:

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

2 Gramática

1. programa → declaraciones funciones
2. declaraciones → tipo lista **;** | ε
3. tipo → **int** | **float** | **double** | **char** | **void** | **struct {** declaraciones **}**
4. lista → lista **,** **id** arreglo | **id** arreglo
5. arreglo → [**numero**] arreglo | ε
6. funciones → **func** tipo **id(** argumentos **){** declaraciones sentencias **}** funciones | ε
7. argumentos → lista_argumentos | ε
8. lista_argumentos → lista_argumentos **,** tipo **id** parte_arreglo | tipo **id** parte_arreglo
9. parte_arreglo → [ ] parte_arreglo | ε
10. sentencia → sentencias sentencias | **if(** condición) sentencia | **if**(condición) sentencias **else** sentencia | **while(** condición **)** sentencia | **do** sentencia **while(** condición **);** | **for(** sentencia **;** condición**;** sentencia **)** sentencia | parte_izquierda **=** expresión **;** | **return** expresión**;** | **return;**| **{** sentencia **}** | **switch(** expresión **)**{ casos predeterminado} | **break;** | **print** expresión;
11. casos → **case:** **numero** sentencia predeterminado | ε
12. predeterminado → **default:** sentencia | ε
13. parte_izquierda → **id** | var_arreglo | **id.id**
14. var_arreglo → **id**[expresión] | var_arreglo [expresión]
15. expresión → expresión **+** expresión | expresión **-** expresión | expresión ***** expresión | expresión **/** expresión | expresión **%** expresión | var_arreglo | **cadena** | **numero** | **caracter** | **id(** parámetros **)**
16. parámetros → ε | lista_param
17. lista_param → lista_param **,** expresión | expresión
18. condición → condición **||** condición | condición **&&** condición | **!** condición | **(**condición**)** | expresión relacional expresión | **true** | **false**
19. relacional → < | > | >= | <= | ! = | ==

3 Análisis léxico

Objetivo:

Elaborar el analizador léxico para la gramática de la sección 2, utilizando lex.

Investigar:

1. Uso de estados léxicos en lex
2. Leer la entrada desde un archivo usando yyin

Descripción:

1. int, float, double, char y void: son palabras reservadas para declarar los tipos de las variables, que se deben sustituir por otras en algún otro idioma.
2. struct es una palabra reservada para definir variables del tipo registro o estructura, se debe sustituir por una palabra en otro idioma.
3. numero: debe poder reconocer tres tipos de números (enteros, flotantes y de doble precisión) relacionados con los tipos
4. id: son los identificadores para el lenguaje de programación.
5. if, else, while, do, return, switch, break, return, print, case, default, true y false: se deben sustituir por palabras reservadas den otro idioma que no sea el inglés, mismo idioma usado para las palabras reservadas de 1 y 2.
6. cadena: son secuencias de caracteres.
7. caracter: debe reconocer solo un caracteres.
8. los operadores: ||, && y !: deben ser sustituidos por algún otro símbolo o palabras.

3.1 Consideraciones

3.1.1 Constantes de caracteres

Este lenguaje admite dos tipos de constantes de cadenas: cadenas y de caracteres.Una cadena es cualquier secuencia de caracteres tipo ASCII. El símbolo \ sirve para usar caracteres especiales.

Los caracteres son cualquier símbolo ASCII.

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 el lenguaje se deben definir en cualquier otro idioma que no sea el inglés.

3.1.5 Identificadores

Los identificadores son nombres de las variables, funciones y estructuras.

3.2 Documentos a entregar:

Se debe entregar un documento que contenga la siguiente parte

1. Descripción del problema a resolver (no del programa)
2. Análisis del problema.
   (a) Expresiones regulares de los componentes léxicos.
   (b) Consideraciones y restricciones.
3. Diseño e implementación
   (a) De ser posible incluir el autómata que reconocerá al lenguaje.
   (b) La organización del código fuente.
4. Resultados y conclusiones individuales.

El código fuente:

1. Cada función debe contener una breve descripción de que es lo que hace y quién la programó.
2. Se debe respetar la identación de dependencia de instrucciones.
3. Se debe usar

4 Análisis Sintáctico

Objetivo:

Elaborar un programa que realice el análisis léxico y sintáctico de la gramática en la sección 2.

Investigar:

Para resolver la ambigüedad del if-else, es que se debe asignar una precedencia a else como si fuera el operador de mayor precedencia.(%prec).

4.1 Documentos a entregar:

Se debe entregar un documento que contenga la siguiente parte

1. Descripción del problema a resolver (no del programa)
2. Análisis del problema.
   (a) Eliminar la ambigüedad de la gramática.
   (b) Consideraciones y restricciones.
3. Diseño e implementación
   (a) Representarla en notación EBNF.
   (b) Diagramas de sintaxis del lenguaje
   (c) La organización del código fuente.
4. Resultados y conclusiones individuales.

El código fuente:

1. Cada función debe contener una breve descripción de que es lo que hace y quién la programó.
2. Se debe respetar la identación de dependencia de instrucciones.
3. Se debe usar

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 correspondientes.
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, pero si pueden usar la instrucción return.
7. Si los parámetros de una función son arreglos se debe validar que la variable que se recibe es un arreglo y tiene las mismas dimensiones pedidas por el parámetro de la función.
8. No pueden existir variables ni parámetros del tipo void.
9. En las expresiones se deben validar los tipos de los operandos, es decir; verificar que los tipos sean iguales o equivalentes.
10. 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 es la función main.
11. Todo programa en C−− debe tener una función main.
12. En caso de que no exista una función main el compilador debe enviar un mensaje de error.
13. Los índices en los arreglos sólo pueden ser números enteros.
14. Los índices en los arreglos se deben validar para que no sean números negativos.
15. Los índices de los arreglos no deben exceder el tamaño del que fueron declarados.
16. 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.
17. El valor numérico en los casos de un switch solo pude ser un entero.
18. Para las estructuras se debe validar que al hacer referencia a un campo de la estructura este exista.

6 Generación de Código Intermedio

Se debe generar código de tres direcciones. 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.

8 Documentos a entregar

1. El manual de usuario.
2. El diseño y especificación del analizador léxico.
3. La definición dirigida por sintaxis.
4. El esquema de traducción.
5. El código fuente del programa documentado en cada función (quién la programó, qué hace, la fecha en que fue programada).
6. En caso de optar por lenguaje C++, se debe usar programación orientada a objetos y entregar un diagrama de clases.