Proyecto Final: Compiladores CS3402

v0.3

Editado por: Prof. Rommel Quintanilla
Desarrollado por: Prof. José Fiestas

UTEC

Instrucciones para desarrollar el proyecto:

- Grupos de máximo 2 integrantes.
- El entregable estará compuesto por el código fuente en C/C++ más un informe final. En caso se opte por utilizar algún generador de lexers y/o parsers, se deberá incluir los fuentes de lex/yacc (flex/bison) según corresponda.
- El informe final debe contener los capítulos de: **Introducción**, **Antecedentes** (trabajos relacionados), **Fundamento teórico** (con la base teórica del problema), **Métodos y Desarrollo**, **Resultados y Conclusiones**.
- De acuerdo a la rúbrica, la calificación total será sobre 20, con los siguientes pesos por criterio: 0.6*Desarrollo de Software + 0.2*Presentación escrita + 0.1*Presentación oral + 0.1*Mejora en desarrollo.
- **Fecha de Entrega:** TBD.
- Presentaciones Orales: TBD.

Lenguaje CEs (C en Español)

Implemente el lenguaje CEs de acuerdo a las indicaciones a continuación y genere código tal que reciba de entrada dos enteros, luego calcule su máximo común divisor y calcule e imprima el resultado.

1. Lexicografía en CEs

1. Las palabras reservadas son: **entero retorno sin_tipo mientras si sino main** (deben estar escritas en minúscula).

2. Símbolos especiales: + − ∗ / < <= > >= == ! = = ; , ( ) [ ] { }

3. **ID** y **NUM** son tokens definidos como:

   ID = letra letra*
   NUM = (+-)? digito digito*
   letra =$a|...|z|A|...|Z$
   digito = 0|...|9

4. Los espacios en blanco se componen de blancos, retornos de línea y tabulaciones. El espacio en blanco es ignorado, excepto cuando deba separar ID, NUM y palabras reservadas.

5. Los comentarios son como en el lenguaje C /∗...∗/. Los comentarios se pueden colocar en cualquier lugar donde pueda aparecer un espacio en blanco (es decir, los comentarios no pueden ser colocados dentro de los tokens) y pueden incluir más de una línea. Los comentarios no pueden estar anidados.

2. Gramática en CEs

La gramática está definida por las siguientes reglas:

1. **programa** → **lista_declaracion**

2. **lista_declaracion** → **lista_declaracion declaracion** | **declaracion**

3. **declaracion** → **var_declaracion** | **fun_declaracion**

Un **programa** se compone de una lista (o secuencia) de declaraciones (**lista_declaracion**), las cuales pueden ser declaraciones de variable o función, en cualquier orden.

Debe haber al menos una **declaración**. Algunas restricciones semánticas son dadas a continuación (algo distintas a C). Todas las variables y funciones deben ser declaradas antes de utilizarlas (esto evita las referencias de retroajuste). La última declaración en un programa debe ser una declaración de función con el nombre **main**. CEs no hace ninguna distinción entre declaraciones y definiciones (como en el lenguaje C).

4. **var_declaracion** → **entero ID ;** | **entero ID [NUM];**

5. **tipo** → **entero** | **sin_tipo**

Una declaración de variable declara una variable simple de tipo entero o una variable de arreglo cuyo tipo base es entero, y cuyos índices abarcan desde 0 hasta NUM−1. Observe que en CEs los únicos tipos básicos son entero y vacío (**sin_tipo**). En una declaración de variable sólo se puede utilizar el especificador de tipo **entero**. Luego, **sin_tipo** es para declaraciones de función. Advierta también que sólo se puede declarar una variable por cada declaración.

6. **fun_declaracion** → **tipo ID (params) sent_compuesta**

7. **params** → **lista_params** | **sin_tipo**

8. **lista_params** → **lista_params , param** | **param**

9. **param** → **tipo ID** | **tipo ID [ ]**

Una declaración de función consta de un especificador de tipo de retorno, un identificador y una lista de parámetros separados por comas dentro de paréntesis, seguida por una sentencia compuesta con el código para la función. Si el tipo de retorno de la función es **sin_tipo**, entonces la función no devuelve valor alguno (es decir, es un procedimiento). Los parámetros de una función pueden ser **sin_tipo** (es decir, sin parámetros) o una lista que representa los parámetros de la función. Los parámetros seguidos por corchetes son parámetros de arreglo cuyo tamaño puede variar. Los parámetros enteros simples son pasados por valor. Los parámetros de arreglo son pasados

por referencia (es decir, como punteros) y deben ser igualados mediante una variable de arreglo durante una llamada. Advierta que no hay parámetros de tipo "función". Los parámetros de una función tienen un ámbito igual a la sentencia compuesta de la declaración de función, y cada invocación de una función tiene un conjunto separado de parámetros. Las funciones pueden ser recursivas.

10. **sent_compuesta** → {**declaracion_local lista_sentencias**}

Una sentencia compuesta se compone de llaves que encierran un conjunto de declaraciones y sentencias. Una sentencia compuesta se realiza al ejecutar Ia secuencia de sentencias en el orden dado. Las declaraciones locales tienen un ámbito igual al de la lista de sentencias de la sentencia compuesta y reemplazan cualquier declaración global.

11. **declaracion_local** → **declaracion_local var_declaracion** | **vacio**

12. **lista_sentencias** → **lista_sentencias sentencia** | **vacio**

Advierta que tanto la lista de declaraciones como la lista de sentencias pueden estas vacías. (El no terminal **vacio** representa la cadena vacía, que se describe en ocasiones como ϵ).

13. **sentencia** → **sentencia_expresion** | **sentencia_seleccion** | **sentencia_iteracion** | **sentencia_retorno**

14. **sentencia_expresion** → **expresion ;** | **;**

Una sentencia de expresión tiene una expresión opcional seguida por un signo de punto y coma. Tales expresiones por lo regular son evaluadas por sus efectos colaterales. Por consiguiente, esta sentencia se utiliza para asignaciones y llamadas de función.

15. **sentencia_seleccion** → **si (expresion) sentencia** | **si (expresion) sentencia sino sentencia**

La sentencia **si** tiene la semántica habitual: la expresión es evaluada; un valor distinto de cero provoca la ejecución de la primera sentencia; un valor

de cero ocasiona la ejecución de la segunda sentencia, si es que existe. Esta regla produce la ambigüedad clásica del **sino** (else) ambiguo, la cual se resuelve de la manera estándar: la parte else siempre se analiza sintácticamente de manera inmediata como una subestructura del **si** actual (la regla de eliminación de ambiguedad "de anidación más cercana")

16. **sentencia_iteracion → mientras (expresion) {lista_sentencias}**

La sentencia **mientras** es la única sentencia de iteración en el lenguaje CEs. Se ejecuta al evaluar de manera repetida la expresión y al ejecutar entonces la sentencia si la expresión evalúa un valor distinto de cero, finalizando cuando la expresión se evalúa a 0.

17. **sentencia_retorno → retorno ; | retorno expresion ;**

Una sentencia de retorno puede o no devolver un valor. Las funciones no declaradas como **sin_tipo** deben devolver valores. Las funciones declaradas **sin tipo** no deben devolver valores. Un retorno provoca la transferencia del control de regreso al elemento que llama (o la terminación del programa si está dentro de **main**).

18. **expresion → var=expresion | expresion_simple**

19. **var → ID | ID [expresion]**

Una expresión es una referencia de variable seguida por un símbolo de asignación (signo de igualdad) y una expresión, o solamente una expresión simple. La asignación tiene la semántica de almacenamiento habitual: se encuentra la localidad de la variable representada por **var**. Luego se evalúa la subexpresión a la derecha de la asignación, y se almacena el valor de la subexpresión en la localidad dada. Este valor también es devuelto como el valor de la expresión completa. Una **var** es una variable (entera) simple o bien una variable de arreglo subindizada. Un subíndice negativo provoca que el programa se detenga (a diferencia de C). Sin embargo, no se verifican los límites superiores de los subíndices. Las variables representan una restricción en CEs respecto a C. En C el objetivo de una asignación debe ser un **valor l**, y los valores l son direcciones que pueden ser obtenidas mediante muchas operaciones. En CEs los únicos valores l son aquellos dados por la sintaxis de **var**, y así esta categoría es verificada sintácticamente, en vez de hacerlo durante la verificación de tipos como en C. Por consiguiente, en CEs está prohibida la aritmética de punteros.

20. **expresion_simple** → **expresion_aditiva relop expresion_aditiva** | **expresión_aditiva**

21. **relop** → $<$ | $<=$ | $>$ | $>=$ | $==$ | $!=$

Una expresión simple se compone de operadores relacionales que no se asocian (es decir, una expresión sin paréntesis puede tener solamente un operador relacional). El valor de una expresión simple es el valor de su expresión aditiva si no contiene operadores relacionales, o bien, 1 si el operador relacional se evalúa como verdadero, o 0 si se evalúa como falso.

22. **expresion_aditiva** → **expresion_aditiva addop term** | **term**

23. **addop** → $+$ $-$

24. **term** → **term mulop factor** | **factor**

25. **mulop** → $*$ | $/$

Los términos y expresiones aditivas representan la asociatividad y precedencia típicas de los operadores aritméticos. El símbolo / representa la división entera.

26. **factor** → **(expresion)** | **var** | **call** | **NUM**

Un factor es una expresión encerrada entre paréntesis, una variable. que evalúa el valor de su variable; una llamada de una función, que evalúa el valor devuelto de la función; o un **NUM**, cuyo valor es calculado por el analizador léxico. Una variable de arreglo debe estar subindizada, excepto en el caso de una expresión compuesta por una **ID** simple y empleada en una llamada de función con un parámetro de arreglo.

27. **call** → **ID (args)**

28. **args** → **lista_arg** | **vacio**

29. **lista_arg** → **lista_arg, expresion** | **expresion**

Una llamada de función consta de un **ID** (el nombre de la función), seguido por sus argumentos encerrados entre paréntesis. Los argumentos pueden estar vacíos o estar compuestos por una lista de expresiones separadas mediante comas, que representan los valores que se asignarán a los parámetros durante una llamada. Las funciones deben ser declaradas antes de llamarlas, y el número de parámetros en una declaración debe ser igual al número de argumentos en una llamada. Un parámetro de arreglo en una declaración de función debe coincidir con una expresión compuesta de un identificador simple que representa una variable de arreglo.

Finalmente. las reglas anteriores no proporcionan sentencia de entrada o salida. Debemos incluir tales funciones en la definición de CEs, puesto que a diferencia del lenguaje C, CEs no tiene facilidades de ligado o compilación por separado. Por lo tanto, consideraremos dos funciones como predefinidas en el ambiente global, para ser consideradas como si tuvieran las declaraciones indicadas:

entero entrada(sin_tipo) {...}
sin_tipo salida(entero x) {...}

La función **input** no tiene parámetros y devuelve un valor entero desde el dispositivo de entrada estándar. La función **output** toma un parámetro entero, cuyo valor es transferido a la salida estándar, junto con un retorno de línea.

Así, el proyecto, consistirá de las siguientes partes:

- Programar el **scanner** correspondiente.
- Programar el **parser** correspondiente.
- Implementar reglas de identificación y recuperación de errores (léxicos, sintácticos, semánticos), reportando amigablemente los mismos.
- Implementar el módulo de generación de código intermedio y ejecutar sobre él algunas rutinas de optimización.
- El código final generado será interpretado por un simulador llamado TM[1] (Tiny Machine).

[1] http://www.cs.sjsu.edu/ louden/cmptext/