“Cpanish”

Teoría de Lenguajes, Automatas y Compiladores

Trabajo Práctico Especial

Autores:

- Ezequiel Martín Keimel Álvarez (Legajo 58325)
- Nicolás Enrique Barrera (Legajo 57694)

Índice

Idea subyacente y objetivo del lenguaje

Cpanish fue diseñado apuntando a los hispanohablantes sin experiencia alguna en programación. Se pretende generar una gramática razonablemente compatible con el lenguaje *C*, a fin no refugiar a los programadores novicios del funcionamiento real de la computadora, que el equipo considera es un conocimiento fundamental, y que debe inculcarse desde el principio, pero ofreciéndoles simultáneamente una sintaxis inmediatamente familiar, amigable, y sobre todo, entendible.

En resumidas cuentas, se pretende ofrecer una puerta de entrada sencilla a la programación, que a diferencia de la programación por bloques u otras herramientas didácticas, abstraiga lo menos posible de la lógica de la computadora.

Consideraciones realizadas

Se utilizaron los conocimientos adquiridos a lo largo de la materia en el presente cuatrimestre, en particular los expuestos durante las clases relativas a *Yacc* y *Lex*. Incontables sitios de Internet sirvieron también como material de consulta.

Se decidió compilar a *C* y posteriormente a lenguaje máquina por medio de *GCC* (con una interfaz alrededor del mismo incluído en el propio compilador, que engloba el proceso completo y genera un archivo ejecutable). Esto se debe, ante todo, a que la capacidad de los integrantes del equipo de producir código correcto y eficiente en *C* se entiende inmensamente superior a la de escribir código en lenguaje ensamblador que garantice la eficiencia y calidad que se pretende. Adicionalmente, y en vista de que que existen compiladores de *C* para casi todas las arquitecturas y sistemas operativos, el uso del mismo como lenguaje intermedio permite proyectar la posibilidad de ampliar el soporte de *Cpanish* a cualquiera de estas plataformas. Se impuso, en ese sentido, la restricción de no apoyarse en características de un sistema operativo en general. El código del compilador pretende ser totalmente portable. En esta primera versión, sin embargo, no se garantiza esta condición.

Descripción del desarrollo

Se comenzó definiendo la gramática en papel, y construyendo un primer programa para el que se graficó su árbol y correspondiente grafo de dependencias a fin de facilitar la definición de atributos y reglas de producción necesarias para el desarrollo. Posteriormente, se construyó de manera gradual el compilador usando *YACC* y *LEX*, definiendo primeramente tipos de dato básicos como entero y cadena de texto, así como una función built-in *Mostrar* para llevar a cabo las primeras pruebas. Gradualmente se agregaron los bloques condicionales e iterativos, y finalmente las funciones.

Se contempló el contexto en la definición de variables (scope), pudiendo éstas existir dentro de bloques, funciones, o a nivel global. El grupo pretende revisar, sin embargo, la implementación del contexto, por considerar que se trata de un prototipo muy burdo y que debe mejorarse.

Finalmente, se agregaron funciones adicionales que permiten interpretar los parámetros que recibe el compilador, así como volcar el código intermedio a un archivo en *C* que posteriormente se borra (a menos que el usuario explícitamente decida conservarlo), después de enviarse a *GCC* para su traducción a lenguaje ensamblador.

Ejemplos y características adicionales

Además de encontrarse en español, otra característica del lenguaje que pretenden facilitar el uso del mismo a usuarios principiantes se encuentra en el manejo de las cadenas de texto. Las mismas no solo pueden concatenarse utilizando el operador “+” (o incluso el operador “*” junto con un entero), sino que además las comparaciones se realizan de acuerdo a su contenido y no al puntero, dado que usualmente la primera es la deseada.

Por otro lado, una de las ventajas del compilador es la capacidad de resolver operaciones entre enteros y cadenas de texto constantes. Es decir, si se realizan operaciones entre estos tipos de datos, las mismas se verán reflejadas en el código C que se obtiene previo a la compilación final.

Por último, el lenguaje cuenta con dos funciones built-in para imprimir texto en la pantalla y para leer información del teclado. Las mismas se llaman “Mostrar” y “Leer un caracter del teclado y guardarlo en”. Esta última tiene un nombre extenso y verborrágico ya que se pretende que todas las definiciones del lenguaje sean claras y no requieran el uso de un instructivo para comprenderlas y utilizarlas.

Ejemplos concretos de este uso se pueden encontrar en la carpeta Examples del proyecto. Cada uno está comentado para dar una breve descripción de su contenido.

Cabe destacar que en el archivo README.md se encuentran especificadas las instrucciones de compilación del compilador y del uso de este último.

Descripción de la gramática

Como se especificó con anterioridad, se pretende que los programas escritos en *Cpanish* sean un subconjunto del lenguaje español. Para ello:

1. Se parte de un símbolo distinguido PROGRAMA
2. Del mismo derivan los PROTOTIPOS, que se definen al principio, así como el alcance global de variables NUEVO_ALCANCE (representa un ALCANCE que en este caso no tienen padre), seguido de la función de entrada PRINCIPAL, y posteriormente una sucesión opcional de FUNCIONES.
3. Los PROTOTIPOS y variables globales se definen con prefijos que se ilustran en los ejemplos, y pretenden facilitar la comprensión a primera vista del lenguaje.
4. Las FUNCIONES degeneran en no terminales FUNCION, cada una de las cuales devuelve un TIPO de datos, tiene un nombre, y recibe una lista de argumentos con nombre y TIPO propios. Las reglas de nomenclatura para funciones y variables son las mismas: conjunto de números, letras, y guiones precedido siempre de una letra.
5. Una FUNCION contiene un conjunto de LINEAS.
6. Un no terminal LINEA degenera en un BLOQUE o en una LINEA y más LINEAS.
7. Una LÍNEA produce una INSTRUCCION.

8. Una INSTRUCCION puede ser una DECLARACION o REASIGNACION de una variable, así como una EXPRESION (operaciones entre variables), una INCREMENTACION o DECREMENTACION de una variable numérica, o una FUNCION_BUILTIN (en este momento solo *Mostrar*).
9. Las variables, ya sea al definirlas o redefiniarlas, pueden adquirir valor el valor de una variable, de una expresión, del retorno de una función, o de un dato del tipo del que fueron definidas o al que pueda ser promovida.
10. Un TIPO puede ser entero y cadena de texto.
11. Las operaciones que se definen son las de aritmética básica para números, y la concatenación de dos cadenas de texto o de una cadena y un entero, que nos parece que es algo que falta en *C* y que facilita la lectura del código evitando hacer uso de funciones que requieran conocimiento de punteros.

Dificultades encontradas

La principal dificultad fue el desconocimiento de las herramientas Yacc y Lex, que requirieron de extensiva investigación previo y durante el desarrollo del trabajo.

Se encontró además la dificultad de validar la correcta definición de un prototipo para cada función que se entiende fundamental tanto desde lo sintáctico como lo funcional en lo que hace al código *C* generado. Es una falencia del trabajo que se pretende resolver en futuras versiones. Un último conflicto reducción desplazamiento nos fue esquivo, por lo que se pretende también solucionarlo en el futuro. Se entiende, en todo caso, que no trae problemas a la hora de procesar la gramática.

Futuras extensiones

- Se pretenden implementar todos los tipos built in de C, incluídos los vectores. Para estos últimos, la definición de una semántica amigable se prevé dificultosa. Los tipos básicos no deberían representar problemas, más allá del requerido adaptamiento de funcionalidades como *Mostrar* y la concatenación con cadenas.
- Mejor manejo de errores.
- Optimización del código en general.
- Reducción de dependencias que se agregan al código C generado al mínimo posible.
- Disponibilidad en distintas plataformas y arquitecturas.
- Transparencia en el proceso de compilación a C y manejo de posibles errores u advertencias relacionados.

Referencias

- http://westes.github.io/flex/manual/Start-Conditions.html (Comentarios multilínea)
- StackOverflow, Wikipedia, man en cantidades industriales.