INFORME TRABAJO PRÁCTICO ESPECIAL

Nivel y Área:

72.39 – Autómatas, Teoría de Lenguajes y Compiladores

2º Cuatrimestre 2018

Comisión: S - **Carrera:** Informática

Fecha: 30 de noviembre de 2018

Alumnos Expositores:

AQUILI, Alejo Ezequiel (57432)

BASSANI, Santiago (57435)

SANGUINETI ARENA, Francisco Javier (57565)

Informe: Trabajo Práctico Especial - ArgenC

Índice

Idea subyacente y objetivo del lenguaje

El lenguaje de programación ***argenC*** fue concebido como posible suplemento para el sistema de educación secundaria Argentina. En el medio del debate actual, de si enseñar programación o no como parte de la curricula básica en los colegios secundarios, este lenguaje de programación simple y estructurado nace para evitar que la frontera idiomática sea una limitación para los alumnos que den sus primeros pasos en el mundo de la programación. Muchos colegios secundarios que no son bilingües enfrentarían la perdida de interés de sus alumnos a la hora de programar porque les resultaría poco familiar e intuitivo utilizar palabras reservadas en ingles para realizar su código (en muchos casos se genera "una complicación adicional" el no saber que están haciendo en realidad y se tiende a memorizar sintaxis que no es amigable con los alumnos).
El objetivo del lenguaje ***argenC*** es promocionar y alentar a los alumnos a programar en un lenguaje lo mas similar al lenguaje cotidiano en su lengua nativa y que ellos aprecien el mágico arte de programar sin fantasmas idiomáticos que lo rodeen ni perdidas de interés o atención, pudiéndose concentrar en lo importante que es el mero hecho de escribir un código estructurado capaz de computar sus algoritmos.

Si bien tenemos antecedentes de lenguajes de programación utilizados en el contexto educativo, como puede ser el anecdótico LOGO, la idea de ***argenC*** es desprender la contextualización o adaptación del lenguaje y que sea un leguaje íntegramente imperativo al igual que el lenguaje de programación C. Si un alumno sabe el idioma ingles, o en un futuro aprende el mismo, puede programar en C o cualquier otro lenguajes a fin dado que ya sabe programar en ***argenC*** y sabiendo ingles solo tiene que amigarse con la sintaxis correspondiente y consideramos que esto ultimo es mucho mas provechoso que realizar: "Avanza tortuga, Retrocede Tortuga" o cualquier otra abstracción de un lenguaje de programación teniendo en cuenta que se trata de un nivel de educación secundario y no primario.

Consideraciones realizadas

El lenguaje ***argenC*** incluye tipos de datos de números enteros o decimales o cadenas de caracteres, sumado a eso, se flexibiliza el esquema fuertemente tipado del lenguaje de programación C siendo además dinámico el tipado lo cual genera un enfoque mas abstracto y simplista para los alumnos, que no tendrán que preocuparse por los tipos de datos y sus variables.

Se incorpora no solo sentencias condicionales, sino que también además del "while" como bloque de repetición incorpora un ciclo "for" para las repeticiones. Se agrego la estructura de datos de listas lo cuales se implementan mediante arreglos e incorpora el famoso "for each" al estilo JAVA.

El lenguaje ***argenC*** genera código C como salida al compilar, que luego es compilado con GCC o Clang para obtener el archivo binario ejecutable ELF.

Para eliminar la nocion de la biblioteca standard de C, nuestro lenguaje incorpora funciones built-in para manipulación de cadenas de caracteres, leer de entrada estándar STDIN, o imprimir en salida estándar STDOUT. Tambiém se incorporaron, la generación de numeros aleatorios.

Por otra parte, se decidio que incluya comentarios fieles al estilo C de comentarios que también son compatibles con otros lenguajes como JAVA. Pero esto pareciera contradecirce con la idea de dejar de lado los tecnisismos de estos lenguajes y por ello la forma principal de realizar comentarios es como lo haria una persona normalmente utilizando las sentencias "comentario: contenido del comentario." O "nota: contenido del comentario."

Descripción del desarrollo

Una vez definida la gramática de ***argenC*** se procedió a armar un escáner con **LEX** para ser utilizado como tokenizador. Este se probo de forma simple, si bien no es test de unidad, era un testeo dinámico con un programa de prueba que permite comprobar en tiempo de ejecución que se este tokenizando como se deseaba.

Como se ha mencionado, ***argenC*** es compilado a lenguaje C que luego se obtiene su código Assembler o directamente su código ejecutable formato ELF mediante un compilador de C. Para llevar a cabo la generación de código C las instrucciones o código ***argenC*** son llevadas a funciones auxiliares o wrappers en C que embeben el comportamiento deseado en el lenguaje de salida.

Se construye un árbol de parsing, más específicamente un árbol de sintaxis abstracta, que es recorrido en profundidad para generar el código de salida.
El árbol se construye usando como materia prima las producciones de nuestra gramática.

El árbol se construye con un enfoque "bottom-up" característico del análisis sintáctico ascendente para la dada gramática **LALR(1)** mediante el uso de **YACC**.

Las variables se guardan en un mapa que las asocia con su nombre lo cual se traduce en que al generar el código C asociado se sigue el estilo de declaración de variables al principio del procedimiento, tal como se lo realizaba en el estándar C89.

Por último, durante la realización de los ejemplos se fue modificando la gramática para hacerla aun mas amigable y generando la sensación de uno esta escribiendo seudocódigo por la cotidiano y amigable de la misma.

Descripción de la gramática

Definición de la gramática:

Grammar

```
PROGRAM' → PROGRAM.
PROGRAM →  BLOCK
         | BLOCK PROGRAM.
BLOCK →    ASIG
         | ARRASIGN
         | PRINT
         | WHILE
         | FOREACH
         | FOR
         | IF
         | FUNCCALL
         | EXIT.
EXIT →     exit.
ASIG →     id es E.
ARRASIGN → ARRINDEX es E.
E →        string
         | entero
         | decimal
         | ARRAY
         | id
         | ARRINDEX
         | E plus E
         | E minus E
         | E mul E
         | E p E
         | E v E
         | E div E
         | minus E
         | parentesisA E parentesisC. | aleatorio
FUNCCALL → concatenar E a id
         | poner id en mayuscula
         | poner id en minuscula
         | incrementar id
         | decrementar id
         | leer numero a id
         | leer texto a id.
ARRAY →    corcheteA EXPLIST corcheteC
         | corcheteA corcheteC.
```

```
ARRINDEX → id corcheteA E corcheteC
         | ARRINDEX corcheteA E corcheteC.
EXPLIST →  E
         | E coma EXPLIST.
PRINT →    imprimir E.
WHILE →    mientras separador CONDICION separador hacer
           PROGRAM fin
         | hasta separador CONDICION separador hacer PROGRAM
           fin.
FOREACH →  porcada id en id PROGRAM fin.
FOR →      desde ASIG hasta separador CONDICION separador
           hacer PROGRAM fin
         | desde hasta separador CONDICION separador hacer
           PROGRAM fin.
IF →       si separador CONDICION separador hacer PROGRAM fin
         | si separador CONDICION separador hacer PROGRAM
           ELSEIF fin.
ELSEIF →   sinosi separador CONDICION separador PROGRAM
         | sinosi separador CONDICION separador PROGRAM
           ELSEIF
         | sino PROGRAM.
CONDICION
→          E BOOL E
         | CONDICION or CONDICION
         | CONDICION and CONDICION
         | negar CONDICION.
BOOL →     menorigual
         | menor
         | mayorigual
         | mayor
         | igual
         | desigual.
```

Tabla de primeros y siguientes:

No Terminal	Primeros	Siguientes
PROGRAM'	imprimir mientras hasta porcada desde siconcatenar poner incre mentar decrementarleer exit id	∅
BLOCK	imprimir mientras hasta porcada desde siconcatenar poner incre mentar decrementarleer exit id	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
EXIT	exit	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
ARRASIGN	id	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
FUNCCALL	concatenar poner incrementar de crementarleer	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
ARRAY	corcheteA	fin sinosi sino separador or and p lus minus mul p v div parentesis C a corcheteC coma menorigual menormayorigual mayor igual de sigual imprimir mientras hasta p orcada desde si concatenar poner incrementardecrementar leer exi t id
ARRINDEX	id	fin sinosi sino separador or and p lus minus mul p v div parentesis C a corcheteC coma menorigual menormayorigual mayor igual de sigual es corcheteA imprimir mi entras hasta porcada desde si con catenar ponerincrementar decre mentar leer exit id
EXPLIST	string entero decimal id minus p arentesisAcorcheteA	corcheteC
PRINT	imprimir	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
WHILE	mientras hasta	fin sinosi sino imprimir mientras hasta porcada desde si concatena

		r poner incrementar decrementar leer exit id
FOREACH	porcada	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
FOR	desde	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
ASIG	id	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
IF	si	fin sinosi sino imprimir mientras hasta porcada desde si concatena r poner incrementar decrementar leer exit id
ELSEIF	sinosi sino	fin
PROGRAM	imprimir mientras hasta porcada desde siconcatenar poner incre mentar decrementarleer exit id	fin sinosi sino
E	string entero decimal id minus p arentesisAcorcheteA	fin sinosi sino separador or and p lus minus mul p v div parentesis C a corcheteC coma menorigual menormayorigual mayor igual de sigual imprimir mientras hasta p orcada desde si concatenar poner incrementardecrementar leer exi t id
CONDICION	negar string entero decimal id m inusparentesisA corcheteA	separador or and
BOOL	menorigual menor mayorigual m ayor igualdesigual	string entero decimal id minus pa rentesisA corcheteA

Grafo/Autómata LALR(1):

Para ver la tabla del autómata LALR(1) se puede ingresar la gramática en el siguiente enlace:

http://smlweb.cpsc.ucalgary.ca/start.html

NOTA: no se adjunto en el informe la tabla dado que el automata tiene 147 estados y resultaba muy extensa la tabla.

Ejemplos realizados:

Se realizaron varios ejemplos de algoritmos codificados en lenguaje argenC, la lista completa de estos ejemplos se encuentra en el archivo README.md.
Mostraremos uno de ellos en el presente informe a modo de ejemplificar más alla de la definición formal de la grmática del lenguaje.

Programa primos.arg:

```
hacer:
    imprimir "Ingrese un numero hasta el cual desee
calcular sus primos anteriores"
    leer numero en n
fin
mientras que n sea menor o igual a 1

vectorBooleano es []

desde i = 0 hasta i menor a n, hacer:
    vectorBooleano es vectorBooleano + 1
    vectorBooleano[i] = 1
    incrementar i
fin

desde i = 2 hasta que i sea menor que n, hacer:
    si es que vectorBooleano[i] igual a 1, entonces:
        comentario: checkeo solo hasta la raiz de n.

        desde j = i*i hasta que j sea menor que n, hacer
            vectorBooleano[j] = 0
            j es j+i
        fin
    fin
    incrementar i
fin

nota: los primos encontrados son los indices del arreglo
      booleano que esten en 1.

imprimir "Estos son los primos hasta " + n

desde i = 2 hasta i menor a n, hacer:
    si es que vectorBooleano[i] igual a 1, entonces
        imprimir i
    fin
    incrementar i
fin
```

Dificultades encontradas

Como principal dificultad nos enfrentamos el manejo de tipado dinamico y el relajamiento de los tipos para contraponernos a lenguajes fuertemente tipados como JAVA o C. Esto ultimo nos obligo a realizar explotación de metodos o funciones built-in y nos complico el proceso de implementación, trayendonos problemas con las implementaciones tanto de arreglos o listas y del mencionado ciclo de repetición estilo For each.

Futuras extensiones

Posibles extensiones o expansiones para el lenguaje de programación estructurado argenC son:

- Permitir insertar código C inline (análogo a como C permite Assembler inline), para que aquellos programadores que posean conocimiento del lenguaje C y deseen inyectar fragmentos de código C en su programa.

- Poder extender el lenguaje en tempo de ejecución, agregando funciones personalizadas al set de funciones built-in de ***argenC*** o incorporar sets enteros personalizados de comportamiento como una especie de módulos de funciones personalizadas.

- Incorporar otra nueva estructura de datos tipo Tablas de Hash.

- Poder incluir código ***argenC*** de otros archivos o librerías externas (como los “#include “file” o “#include <lib>” del lenguaje de programación C).

- Tener la capacidad de definir funciones e invocarlas desde otras al igual que el el lenguaje de programación C.

Referencias

- Jhon R. Levine, Tony Mason & Doug Brown. (1990). Unix Programming Tools - lex & yacc. Estados Unidos: O'REILLY.

- Presentaciones de Lex y Yacc provistas por la cátedra de Autómatas, Teoría de Lenguajes y compiladores.

- Alfred V. Aho, Ravi Sethi. (1990). Compiladores, Principios, Técnicas Y Herramientas. Estados unidos: Pearson Educación.