

ProgramAR

72.39 - Autómatas, Teoría de Lenguajes y Compiladores Segundo Cuatrimestre 2021

Integrantes:

- Julián Francisco **Arce**, 60509
 - Gastón **De Shant**, 60755
 - Paula Andrea **Domingues**, 60148
 - Gian Luca **Pecile**, 59235
-

Índice

Idea subyacente y objetivo del lenguaje	2
Consideraciones realizadas	2
Desarrollo	2
Gramática	3
Tipos de Datos	4
Constantes	4
Delimitadores	4
Operadores aritméticos	4
Operadores relacionales	4
Operadores lógicos	5
Operadores de asignación	5
Operadores booleanos	5
Bloque Condicional	5
Bloque Do-While	5
Entrada estándar	5
Salida estándar	6
Separadores de contexto	6
Dificultades encontradas	6
Posibles extensiones	6
Bibliografía y referencias	7

Idea subyacente y objetivo del lenguaje

La idea subyacente es crear un lenguaje que sea en español y, a su vez, tenga como objetivo ser lo más didáctico posible. El enfoque está en que sea sencillo de usar por chicos de escuelas primarias/secundarias al igual que gente sin conocimiento previo sobre programación de cualquier edad, donde no sea necesario lidiar con el proceso de aprender un lenguaje en específico, sino aprender a resolver problemas de programación y lógica. Para ello, proponemos el lenguaje "ProgramAR".

En primer instancia esa fue la idea detrás del lenguaje y al consultar con la cátedra nos encontramos que otro grupo había elegido una idea similar y decidimos proponer una estructura para "forzar" ciertas buenas prácticas en cuanto a estilo de código, manteniendo el fin didáctico y el proponer que los programas escritos en nuestro lenguaje posean buenas prácticas; en el sentido de siempre definir primero variables, luego trabajar y ejercitar esas variables, por último, mostrar adecuadamente los resultados. Dicha estructura se asemeja a la que posee un test unitario, donde primero se setean las precondiciones, luego se ejercita el método a testear y luego se validan resultados; así se puede generar una organización del código bien diferenciada y seccionada para que facilite la lectura del mismo.

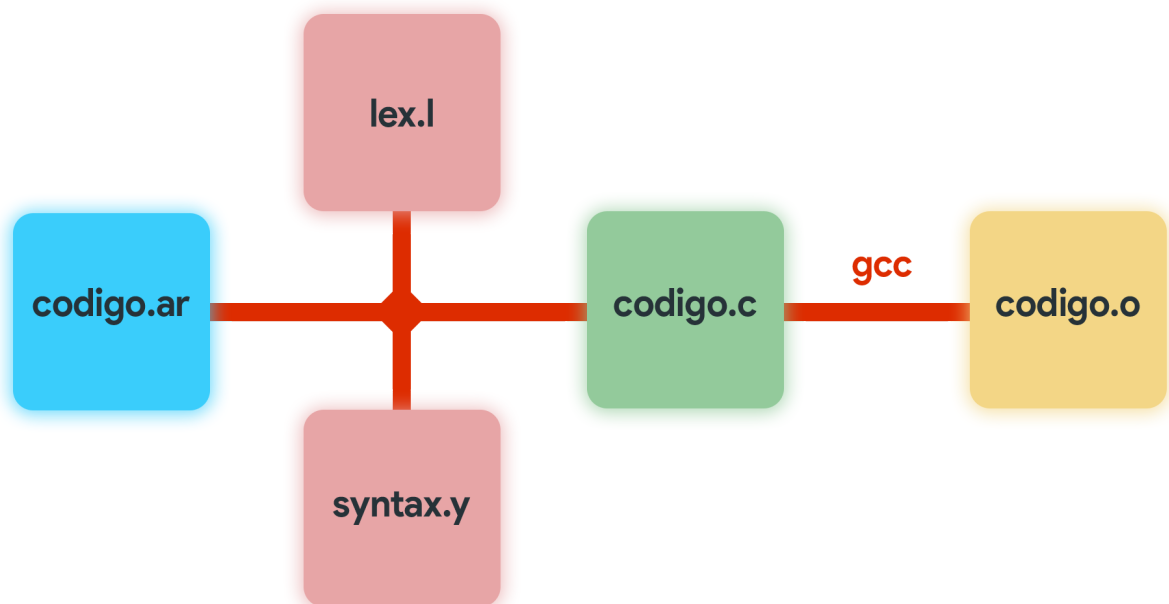
Consideraciones realizadas

El lenguaje ProgramAR cumple con lo pedido por el enunciado, además no hubo consideraciones realizadas fuera de las halladas en la consigna.

Desarrollo

Para el desarrollo del lenguaje ProgramAR se hizo uso de Yacc y Lex, herramientas vistas en clase y provistas por la cátedra con más información sobre la documentación y referencias utilizadas en dicha [sección](#).

En primer lugar se definieron las palabras propias del lenguaje y se creó el archivo *lex.l*. Luego se creó el archivo de sintaxis llamado *syntax.y*, donde se encuentran definidos los diferentes terminales finales y no finales, al igual que generar el analizador sintáctico para nuestro lenguaje. Para almacenar dichas variables usadas al igual que sus nombres, se hace uso de una implementación de una estructura de datos, en específico una lista encadenada, dónde se guarda el nombre de cada variable y su tipo (entero o texto) además del puntero al siguiente. La salida es en lenguaje C, compilado con gcc. Se puede notar que la extensión de los archivos propios del lenguaje ProgramAR poseen la extensión *.ar* debido al énfasis en el lenguaje en español que usa jerga proveniente Argentina. Para ilustrar mejor la compilación se presenta el siguiente diagrama de flujo:



Todo el desarrollo se realizó mediante el [repositorio de github](#) donde se encuentra este informe.

Gramática

La gramática detrás del lenguaje programar incluye lo siguiente:

Tipos de Datos

- texto
 - Representa un string que se usa en lenguajes como Java.
- letra
 - Representa el tipo de dato char en lenguaje C.
- numero
 - Representa un entero.

Constantes

Precede al tipo de dato con no cambia.

Delimitadores

- ;
 - Actúa como indicador de fin de línea.
- ()
- { }
- " "

Operadores aritméticos

- +
- -
- *
- /
- modulo

Operadores relacionales

- vale menos que
- vale mas que

- es igual o vale menos que
- es igual o vale mas que
- es igual a
- es distinto de

Operadores lógicos

- y
- o
- opuesto de

Operadores de asignación

- vale

Operadores booleanos

- verdadero
- falso

Bloque Condicional

- si se cumple(condición) { // código }
- } si no { // código }

Bloque Do-While

- hacer { // código } mientras(condición);

Entrada estándar

- leer(variable);

Salida estándar

- imprimir(variable);

Separadores de contexto

- al inicio { //código }
- rutina { // código }
- al finalizar { // código }

Dificultades encontradas

La principal dificultad encontrada fue la falta de conocimiento al respecto de tanto Lex como Yacc. Se recurrió a las clases dadas por la cátedra al igual que la bibliografía y los manuales disponibles online que se encuentran en la sección de [bibliografía y referencias](#).


A medida del avance del desarrollo del lenguaje se vieron conflictos con *shift/reduce* y *reduce/reduce*, los mismos pudieron ser resueltos con las herramientas vistas en clase y lo encontrado en la sección de referencias.

Posibles extensiones

Una posible extensión para generar al lenguaje es la inclusión de objetos, en esta iteración se consideró que sería afrontar mucho contenido para una primer versión del lenguaje ya que el paradigma orientado a objetos no suele ser introductorio a un lenguaje de programación y ya se abarca contenido de programación imperativa al igual que la estructura subyacente detrás de todo testeo unitario en la iteración actual del lenguaje programAR.

Recibir argumentos al momento de ejecutar el programa similar a cómo funciona en C.

Bibliografía y referencias

- Clases y bibliografía aportada por la cátedra:
 - [Intro a los Compiladores \(Presentación sobre Compiladores\)](#)
 - [Lex \(Presentación sobre Lex\)](#)
 - [Yacc \(Presentación sobre Yacc\)](#)
 - [Ejemplos Yacc/Lex](#)
 - [Arquitectura de Compiladores](#)
 - [faturita/YetAnotherCompilerClass: Lex and Yacc samples and tools repository for Languages and Compiler class.](#)
 - Lex & Yacc - Doug Brown, John R. Levine, and Tony Mason.
- [Ubuntu Manpage: flex - generador de analizadores léxicos rápidos.](#)
- [Condiciones de arranque para separadores de contexto.](#)
- [Introducción a yacc.](#)
-  Part 02: Tutorial on lex/yacc. .
- [Implementación de Linked List.](#)