72.39 - Autómatas, Teoría de Lenguajes y Compiladores

2º Cuatrimestre del 2021

Trabajo Práctico Especial

~ PSEUDOC ~

“Desarrollo completo de un lenguaje y su compilador”

Integrantes del grupo
Chao, Florencia - 60054
García Montagner, Santiago - 60352
Cerdeira, Tomás - 60051
Catolino, Lucas - 61817

Fecha de entrega: 28/11/2021

Índice

Objetivo e idea principal del lenguaje

Como objetivo principal de este trabajo especial se propuso llevar a cabo el desarrollo de un lenguaje de programación teniendo en cuenta la totalidad de su gramática; además del desarrollo de su respectivo compilador. El mismo, no solo tiene que ser completamente funcional y simple de usar, sino que también ofrecer algún tipo de diferencial.

Teniendo en cuenta que son muchas las personas que están interesadas en comenzar a entender y desarrollar su conocimiento en el rubro de la programación, pero se ven abrumadas al intentarlo; decidimos implementar un lenguaje de programación simple y ágil, incluyendo todas las funcionalidades básicas de un lenguaje de alto alcance: definición de variables con sus respectivos tipos, bloques condicionales, ciclos condicionales, uso de la salida estándar, entre otros.

Para hacer aún más nutritiva su experiencia, el lenguaje desarrollado no solo es completamente útil y capaz de llevar a cabo el programa deseado, sino que también genera un archivo de salida en C, uno de los lenguajes de programación más potentes de esta rama. El mismo, reproduce el programa de nuestro lenguaje en C, incluyendo comentarios explicativos de su gramática.

Consideraciones realizadas

Al estar desarrollando un lenguaje de programación y su compilador, a pesar de traducir el código al lenguaje C para generar su respectivo ejecutable, se debe prestar mucha atención a los posibles errores de la gramática definida.

Para el usuario final la traducción debe ser completamente transparente, no debe estar al tanto de que ocurre. Por esta razón, se implementaron mecanismos que, mediante el uso de la función yyerror definida por YACC, se encargan de abortar la compilación en caso de ocurrir un error de sintaxis enviando el mensaje apropiado.

Desarrollo e implementación

Los 2 pilares del desarrollo llevado a cabo son el uso de las herramientas Lex y YACC. Usandolas de manera conjunta permiten implementar, sin mucha complejidad, un analizador lexicográfico (Lex) y sintáctico (YACC) capaz de analizar y validar que cadenas de texto pertenezcan a una gramática definida.

Además de formular una gramática bien definida, cumpliendo con el objetivo planteado, la misma debe respetar algunas convenciones. La principal de ellas, es la definición y uso de variables de la forma correcta:

- No se puede definir 2 veces una misma variable
- El valor de la variable debe respetar el tipo de la misma
- El nombre de la variable debe cumplir con los criterios establecidos en la gramática
- No se puede utilizar una variable sin que haya sido inicializada

Para implementar las variables en nuestro lenguaje teniendo en cuenta lo mencionado, se utilizó una lista simplemente encadenada cuyo objetivo es guardar las variables definidas con sus respectivos tipos.

Al inicializar una variable, se agrega un nodo a la lista con su información. Cada vez que se define o utiliza una variable en el código, se revisa si la misma forma parte de la lista y en base a eso se decide si su definición/uso es posible o si se debe lanzar un error en compilacion.

Descripción de la gramática

Definición de la gramática

G = < Vn, Vt, S, P >

Vn: Símbolos NO terminales

Vn = { S, INICIO_PROGRAMA, PROGRAMA, FIN_PROGRAMA, DEFINICION, definicion_cons, lista_sentencias, lista_sentencias_bloques, fin_sentencia, sentencia, sentencia_bloques, operacion_sobre_variable_igual, variable_int, const_var, multiple_operadores, operacion_sobre_variable, operador, operador_igual, nueva_variable, tipo_int, tipo_char, concat, leer, imprimir, ASIGNACION, ASIGNACION_CONS, F_INT, F_CHAR, nombre_int, nombre_char, nombre_const, condicional, estructura_if, estructura_while, estructura_else, definicion_if, definicion_while, definicion_else, inicio_condicional, fin_condicional, condicion, or, and, comparador, condicion_logica }

Vt: Símbolos terminales

Vt = { INICIO, FIN, NUM, LETRAS,ASIGN_VAR, ASIGNACION_IGUAL, DISTINTO, IGUAL, MAYOR, MAYOR_IGUAL, MENOR, MENOR_IGUAL, FIN_LINEA, FIN_CONDICIONAL, ELSE_VAR, INICIO_CONDICIONAL, IF_VAR, WHILE_VAR, AND, OR, MULTIPLICACION,SUMA, RESTA, DIVISION, MODULO, MAS_IGUAL, MENOS_IGUAL, MULTIPLICACION_IGUAL, DIVISION_IGUAL, MODULO_IGUAL, IMPRIMIR_VAR, LEER_VAR, IMPRIMIR_VAR_LINEA, DEF,CONCAT_VAR ,PARENTESIS_ABRE ,PARENTESIS_CIERRA, COMA }

S: Símbolo inicial de la gramática

P: Conjunto de producciones

Las producciones se encuentran declaradas en el anexo del informe.

Sintaxis

- Declaración de constantes

Las constantes representan variables numericas que **no pueden ser modificadas luego en el código**. Como en el lenguaje C, las mismas se deben definir antes del inicio del

programa usando la palabra clave *definicion:* . Además, el nombre de cada constante debe comenzar con *const_* :

e.j.

```
definicion:
    const_dias_de_la_semana = 7
    const_horas_del_dia = 24
```

- Bloque principal del programa (a.k.a. main)

Comenzando la línea con la palabra clave *inicio:* se define el bloque principal del código. Todo el programa debe estar contenido dentro de él; la definición de las variables, bloques condicionales, bloques iterativos y todo el resto de la lógica deseada. El mismo debe ser "cerrado" con la palabra clave *fin*.

e.j.

```
inicio:
    num var_a = 1;
    num var_b = 2;
    num var_suma_a_b = 0;
    var_suma_a_b = var_a + var_b;
    imprimirln(var_suma_a_b);
fin
```

- Declaración y asignación de variables

Para definir una nueva variable, se debe comenzar la línea estableciendo el tipo de la misma:

- *num :* entero (tipo *int* en C)
- *letras :* cadena de caracteres (tipo *char* * en C)

Seguido del tipo, el nombre de la variable comenzando con *var_* y la asignación de su valor.
e.j.

```
num var_a = 1;
letras var_a_en_texto = "uno";
```

Para redefinir el valor de una variable definida anteriormente, en vez de usar el =, se

debe usar :=
e.j.

```
var_a := 2;
var_a_en_texto := "dos"
```

Además de poder asignarle un valor, a las variables se les puede asignar, luego de su declaración, una operación. La misma puede ser entre números, variables de tipo num y números, y entre variables de tipo num:

e.j.

```
num var_horas = 2;
num var_minutos = 40;
num var_tiempo_total = 0;
var_minutos_total = var_horas * 60 + var_minutos;
```

- Operaciones

La operaciones soportadas por el lenguaje son:

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Módulo (%)

Además, existen operaciones que se pueden hacer directamente sobre las variables:

- Suma y asignación al valor de la variable (var_a += 1)
- Resta y asignación al valor de la variable (var_a -= 1)
- Multiplicación y asignación al valor de la variable (var_a *= 2)
- División y asignación al valor de la variable (var_a /= 2)
- Módulo y asignación al valor de la variable (var_a %= 2)

- Bloque condicional

Muchas veces se desean llevar a cabo operaciones, asignaciones a variables o impresiones a salida estándar si una condición se cumple y otra u otras cuando no. Parecido al *if else* de C, un bloque condicional se define de la siguiente manera:
e.j.

```
si (var_segundos > 0) inicio sentencia
    var_minutos = var_segundos / 60;
    var_segundos_restantes = var_segundos % 60;
terminar
sino
    var_minutos = 0;
    var_segundos_restantes = 0;
terminar
```

- Bloque iterativo

Parecido al objetivo del bloque condicional, muchas veces se desea contar con bloques de código que se ejecutan repetitivamente mientras que, o hasta que, se cumpla cierta condición. Parecido al *while* de C, un bloque iterativo se define de la siguiente manera:

e.j.

```
mientras (var_segundos > 0) inicio sentencia
    imprimir("Cantidad de segundos restantes: ");
    imprimirln(var_segundos);
    var_segundos -= 1;
terminar
    imprimirln("Te quedaste sin tiempo :(");
```

- Comparadores y Operadores lógicos

Los comparadores soportados por el lenguaje son:
- Igual (==)
- Distinto (!=)
- Menor (<)
- Mayor (>)
- Menor o igual (<=)
- Mayor o igual (>=)

Los operadores lógicos soportados por el lenguaje son:
- AND (y)
- OR (o)

- Imprimir y leer de salida estándar

Nunca está demás contar con funciones que permiten imprimir texto y/o valores del código directamente a la salida estándar para que el usuario lo vea. Sumado a eso, muchas veces se desea recibir un input del usuario para llevar a cabo tal o cual operación. Para esto, el lenguaje de programación cuenta con las siguientes 3 funciones:

- Imprimir

```
imprimir("Texto que va a salir por STDOUT seguido por el valor de una variable: ");
imprimir(var_variable);
imprimir("\n");
```

- Imprimir línea (imprime el texto pasado como parámetro y luego un \n)

```
imprimirln("Imprimir linea");
```

- Leer (lee de entrada estándar una palabra introducida por el usuario guardandola en la variable pasada como parámetro)

```
leer(var_texto);
leer(var_numero);
```

OBS: el tipo de la entrada esperada depende del tipo de la variable pasada como parámetro en la función leer. De enviarle un número a una variable de tipo texto, se guarda correctamente; pero de enviarle un texto a una variable de tipo texto, la variable queda con un valor de 0.

- Comentarios

Existen 2 tipos de comentarios en el lenguaje:

- Uni-línea

```
//Esto es un comentario unilinea
```

- Multi-línea

```
/*
Esto es un comentario multilinea
*/
```

Dificultades encontradas

Como el lenguaje implementado es de propósito general, encapsulando mucho comportamiento similar al de C, definir las producciones de la gramática llevó bastante tiempo y trabajo. A su vez, al tener que familiarizarse con las herramientas Lex y YACC, no solo se invirtió tiempo en la definición en sí, sino que también en su correcta implementación en el entorno de trabajo definido para este TP.

Por otra parte, el manejo de errores también presentó dificultades ya que en un principio, no se consideró el hecho de que el error también sea arrojado en c. Como consecuencia se tuvo que modificar la lógica implementada en primer lugar para poder lograr el correcto funcionamiento de los mismos.

Limitaciones

Una limitación del lenguaje se encuentra en que las operaciones aritméticas y lógicas solo soportan operaciones con caracteres de tipo int, sean así variables o no. Otra limitación, es el hecho de que no se puede tabular el archivo de salida.

Futuras extensiones

Las futuras extensiones diseñadas para el programa son:

- La implementación del tipo de dato arreglo
- La implementación de la estructura case
- Explicación más precisa de los errores
- Mayor cantidad de operaciones en variables de tipo int como reverse y replace

Ejemplos

- Ejemplo 1

```
definicion:
   const_segundos_en_min = 60

inicio:
num var_segundos =0;

imprimirln("Ingrese una cantidad de segundos y te dire cuantos minutos son.");
leer(var_segundos);

mientras (var_segundos < 0) inicio sentencia
imprimirln("Ingrese una cantidad de segundos mayor que 0 y te dire cuantos minutos
son.");
leer(var_segundos);
terminar

num var_minutos = 0;

var_minutos = var_segundos / const_segundos_en_min;

imprimir("La cantidad de minutos son: ");
imprimirln(var_minutos);

fin
```

El programa anterior recibe desde entrada estándar una cantidad de segundos e imprime cuantos minutos serían.

- Ejemplo 2

```
// Ejemplo provisto en la consigna

inicio:
   num var_x = 6;
   num var_y = 1;

   mientras (var_x > 0) inicio sentencia
      var_y = var_y * var_x;
      var_x -=1;
   terminar

imprimir("El factorial de 6 es: ");
imprimirln(var_y);
imprimir("El valor de x+1 es: ");
var_x +=1;
imprimirln(var_x);
```

```
fin
```

El programa anterior calcula el factorial del número 6.

- Ejemplo 3

```
inicio:

num var_entrada = 0;
imprimirln("Ingrese un año y te dire si es bisiesto.");
leer(var_entrada);

mientras (var_entrada < 0) inicio sentencia
imprimirln("Ingrese un año mayor que 0 ");
leer(var_entrada);
terminar


num var_mod4 = 0;
var_mod4 = var_entrada % 4;

num var_mod100 = 0;
var_mod100 = var_entrada % 100;

num var_mod400 = 0;
var_mod400 = var_entrada % 400;

si (var_mod4 == 0 y var_mod100 != 0 o var_mod400 == 0) inicio sentencia
   imprimirln("Es un año bisiesto!");
terminar
sino
   imprimirln("No es un año bisiesto!");
terminar

fin
```

El programa anterior recibe por entrada estándar un año y devuelve si el mismo es bisiesto o no.

- Ejemplo 4

```
inicio:

num var_entrada = 0;
imprimirln("Ingrese un numero y le calculare su factorial");
leer(var_entrada);
num var_aux = 0;
mientras (var_entrada < 0) inicio sentencia
   imprimirln("Ingrese un numero mayor que 0 ");
   leer(var_entrada);
terminar
var_aux := var_entrada;

num var_salida = 1;

mientras (var_entrada > 0) inicio sentencia
   var_salida = var_salida * var_entrada;
   var_entrada -=1;
terminar

imprimir("El factorial de ");
imprimir(var_aux);
imprimir(" es ");
imprimirln(var_salida);

fin
```

El programa anterior recibe por entrada estándar un número y devuelve su factorial.

- Ejemplo 5

```
inicio:

letras var_leido1 = "";
letras var_leido2 = "";
num var_cant = 0;
num var_indice = 1;


imprimirln("Ingrese la cantidad de palabras a concatenar ");
leer(var_cant);

letras var_salida = "Tus palabras concatenadas son: ";

mientras(var_cant>0) inicio sentencia
   imprimir("Ingrese la palabra numero ");
   imprimir(var_indice);
   imprimirln(" para concatenar. ");
```

```
    leer(var_leido1);
    concatenar(var_salida,var_leido1);
    var_indice +=1;
    var_cant -=1;

terminar

imprimirln(var_salida);

fin
```

El programa anterior concatena la cantidad de cadenas de caracteres recibidas por la entrada estándar.

Conclusión

Como grupo creemos poder afirmar que luego de varios días de trabajo, hemos logrado cumplir con el objetivo de este proyecto, creando un lenguaje de programación simple y ágil.

Sin mucha complejidad, el mismo permite implementar programas básicos, capaces de resolver problemas “comunes” con los que se enfrenta una persona que recientemente comenzó con sus prácticas y estudios sobre la programación. No solo sirve como herramienta para darle una primera aproximación a la creación de un programa, sino que también, la traducción del mismo a C con sus respectivos comentarios sobre lo llevado a cabo, permite abrirle el panorama indagando en la programación imperativa con uno de los lenguajes más potentes como lo es C.

Por último, nos quedamos con lo útiles (y relativamente simples) que son las herramientas Lex y YACC para llevar a cabo un proyecto como este. Contando con una documentación clara y una enorme comunidad, hacer un proyecto como el llevado a cabo pasa de ser un trabajo arduo, a uno posible de completarse en un mediano-corto plazo.

Demás está decir que sin los conocimientos aprendidos durante la cursada de la materia, esta conclusión y el desarrollo del trabajo serían completamente distintos. Aplicando la teoría estudiada y practicada, logramos comprender con mayor profundidad el funcionamiento del mismo en su totalidad además de los conflictos y complicaciones con las que nos topamos.

Referencias

- Manual YACC - LINK
- Manual LEXX - LINK
- Implementación de una linked-list - LINK

Anexo

Gramática en BNF:

- Identificadores para variables y números

INTEGER [-]?[0-9]+
NOMBRE var_[a-z|A-Z|0-9|_]+
TEXTO ["][^\n]*["]
NOMBRE_CONST const_[a-z|A-Z|0-9|_]+

- Producciones

El símbolo distinguido de la gramática es S

S::= INICIO_PROGRAMA PROGRAMA FIN_PROGRAMA
| DEF DEFINICION INICIO_PROGRAMA PROGRAMA FIN_PROGRAMA

INICIO_PROGRAMA::= INICIO

FIN_PROGRAMA::= FIN

PROGRAMA::= lista_sentencias

DEFINICION::= definicion_cons
| definicion_cons definicion_cons

definicion_cons::= nombre_const ASIGNACION_CONS F_INT

lista_sentencias::= sentencia fin_sentencia
| sentencia fin_sentencia lista_sentencias
| condicional lista_sentencias
| condicional
| ERROR_COMENTARIO

lista_sentencias_bloques::= sentencia_bloques fin_sentencia
| sentencia_bloques fin_sentencia lista_sentencias_bloques
| condicional lista_sentencias_bloques
| condicional

fin_sentencia::= FIN_LINEA
| ERROR_COMENTARIO

sentencia::= nueva_variable
| operacion_sobre_variable
| operacion_sobre_variable_igual
| imprimir
| concat
| leer

sentencia_bloques::= operacion_sobre_variable
| operacion_sobre_variable_igual
| imprimir
| concat
| leer

operacion_sobre_variable_igual::= variable_int operador_igual F_INT
| variable_int operador_igual variable_int
| variable_int operador_igual const_var

variable_int::= NOMBRE

const_var::= NOMBRE_CONST

multiple_operadores::= operador variable_int
| operador variable_int multiple_operadores
| operador F_INT
| operador F_INT multiple_operadores
| operador const_var
| operador const_var multiple_operadores

operacion_sobre_variable::= variable_int ASIGNACION variable_int multiple_operadores
| variable_int ASIGNACION F_INT multiple_operadores
| variable_int ASIGNACION const_var multiple_operadores

operador::= MULTIPLICACION
| SUMA
| RESTA
| DIVISION
| MODULO

operador_igual::= MAS_IGUAL
| MENOS_IGUAL
| MULTIPLICACION_IGUAL
| DIVISION_IGUAL
| MODULO_IGUAL

| ASIGNACION_IGUAL

nueva_variable::= tipo_int nombre_int ASIGNACION F_INT
| tipo_int nombre_int ASIGNACION NOMBRE_CONST
| tipo_char nombre_char ASIGNACION F_CHAR

tipo_int::= NUM

tipo_char::= LETRAS

concat::= CONCAT_VAR PARENTESIS_ABRE NOMBRE COMA NOMBRE PARENTESIS_CIERRA
| CONCAT_VAR PARENTESIS_ABRE NOMBRE COMA TEXTO PARENTESIS_CIERRA

leer::= LEER_VAR PARENTESIS_ABRE NOMBRE PARENTESIS_CIERRA

imprimir::= IMPRIMIR_VAR TEXTO
| IMPRIMIR_VAR_LINEA TEXTO
| IMPRIMIR_VAR PARENTESIS_ABRE TEXTO PARENTESIS_CIERRA
| IMPRIMIR_VAR_LINEA PARENTESIS_ABRE TEXTO PARENTESIS_CIERRA
| IMPRIMIR_VAR NOMBRE
| IMPRIMIR_VAR PARENTESIS_ABRE NOMBRE PARENTESIS_CIERRA
| IMPRIMIR_VAR_LINEA NOMBRE
| IMPRIMIR_VAR_LINEA PARENTESIS_ABRE NOMBRE PARENTESIS_CIERRA
| error

ASIGNACION::= ASIGN_VAR

ASIGNACION_CONS::= ASIGN_VAR

F_INT::= INTEGER

F_CHAR::= TEXTO

nombre_int::= NOMBRE

nombre_char::= NOMBRE

nombre_const::= NOMBRE_CONST

condicional::= estructura_if
| estructura_if estructura_else

| estructura_while

estructura_if::= definicion_if condicion inicio_condicional lista_sentencias_bloques fin_condicional
| definicion_if PARENTESIS_ABRE condicion PARENTESIS_CIERRA inicio_condicional lista_sentencias_bloques fin_condicional

estructura_while::= definicion_while condicion inicio_condicional lista_sentencias_bloques fin_condicional
| definicion_while PARENTESIS_ABRE condicion PARENTESIS_CIERRA inicio_condicional lista_sentencias_bloques fin_condicional

estructura_else::= definicion_else lista_sentencias fin_condicional

definicion_if::= IF_VAR

definicion_while::= WHILE_VAR

definicion_else::= ELSE_VAR

inicio_condicional::= INICIO_CONDICIONAL

fin_condicional::= FIN_CONDICIONAL

condicion::= F_INT
| condicion_logica or condicion
| condicion_logica and condicion
| condicion_logica

or::= OR

and::= AND

comparador::= IGUAL
| MENOR
| MAYOR
| MENOR_IGUAL
| MAYOR_IGUAL
| DISTINTO

condicion_logica::= variable_int comparador variable_int
| F_INT comparador F_INT
| variable_int comparador F_INT
| F_INT comparador variable_int

Lista de tokens:

INICIO "inicio:"
FIN "fin"
NUM "num "
LETRAS "letras "
ASIGN_VAR "="
ASIGNACION_IGUAL ":="
DISTINTO "!="
IGUAL "=="
MAYOR ">"
MAYOR_IGUAL ">="
MENOR "<"
MENOR_IGUAL "<="
FIN_LINEA ";"
FIN_CONDICIONAL "terminar"
ELSE_VAR "sino"
INICIO_CONDICIONAL "inicio sentencia"
IF_VAR "si"
WHILE_VAR "mientras"
AND "y"
OR "o"
MULTIPLICACION "*"
SUMA "+"
RESTA "-"
DIVISION "/"
MODULO "%"
MAS_IGUAL "+="
MENOS_IGUAL "-="
MULTIPLICACION_IGUAL "*="
DIVISION_IGUAL "/="
MODULO_IGUAL "%="
IMPRIMIR_VAR "imprimir"
LEER_VAR "leer"
IMPRIMIR_VAR_LINEA "imprimirln"
DEF "definicion:"
CONCAT_VAR "concatenar"
PARENTESIS_ABRE "("
PARENTESIS_CIERRA ")"
COMA ","