PROGRAMACION PARA PEQUEÑOS

AUTÓMATAS, TEORÍA DE LENGUAJES Y COMPILADORES
GRUPO AMIGOS DE ALAN

Alumnos:

MARTIN GRABINA	57360
JUAN GRETHE	57370
MICAELA BANFI	57293

Instituto Tecnológico de Buenos Aires
ITBA

30-06-2019

Contenidos

1 Introducción

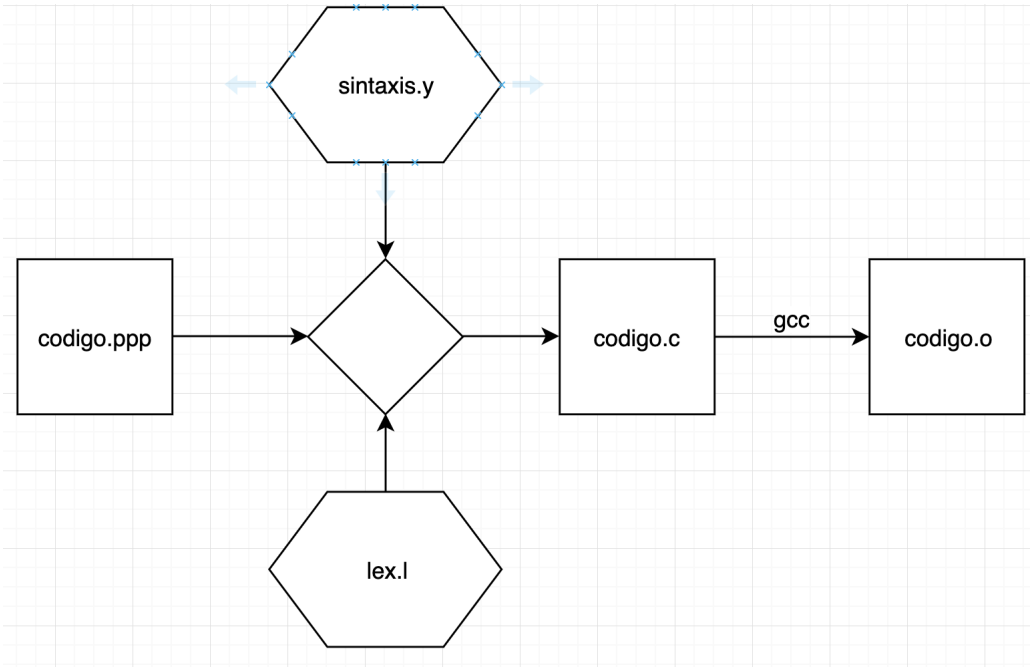
En el siguiente informe se describe el trabajo realizado para la construcción de un compilador, utilizando herramientas como Yacc y Lex. A si mismo se describirán las dificultades encontradas, decisiones en general y una descripción detallada de la gramática.

2 Idea subyacente y objetivo del lenguaje

Cada día mas niños comienzan a estudiar programación desde mas jóvenes, y a esta creciente demanda surgen día a día nuevas herramientas para aprender a programar desde chicos, pero generalmente estas herramientas están en inglés entonces quedan fuera del alcance de los niños que solamente hablan español. Decidimos colaborar con la causa y desarrollar un lenguaje, en español, lo mas simple y coloquial posible, que permita hacer las cosas básicas de programación imperativa para dar los primeros pasos en la materia.

3 Desarrollo del TP

Para el desarrollo se utilizaron Yacc y Lex, herramientas vistas en clase. Primero se definieron las palabras propias del lenguaje y se creó el archivo *lex.l*. Luego se creó el archivo *sintaxis.y*, donde se definen los diferentes terminales finales y no finales y generar el analizador sintáctico de nuestro lenguaje. Se decidió que a medida que se recorre el programa, se va guardando todo en un string, hasta llegar al símbolo final e imprimir todo. La salida es en lenguaje C, compilado con gcc, sin los warnings, ya que la idea es que nuestro compilador agarre y muestre los errores. Es importante notar que hay que tener la version 7 de gcc. Puede notarse que la extencion de los archivos es .ppp, refiriéndose a Programación Para Pequeños.



4 Descripción de la gramática

La gramática permite declarar variables del tipo **numero** y **texto** (int o string en otros lenguajes). Permite utilizar bloques **si - o si - y si no** (if - elseif - else), **hacer - mientras** (do - while) y **para - desde - hasta** (for). Estos bloques pueden anidarse como se desee. También permite realizar operaciones aritméticas, imprimir en pantalla y leer desde el teclado. Se decidió no utilizar ningún signo de puntuación para la finalización de una linea. A continuación se detalla su uso. En la carpeta del repositorio, se encontraran ejemplos concretos. Las tabulaciones en los siguientes ejemplos no son necesarias, solo se aplican para organizar visualmente el código.

4.1 Símbolos

La gramática cuenta con los siguientes símbolos no terminales:

- PROGRAM : símbolo inicial de la gramática, constituido por MAIN (final e indica inicio) y END (final e indica el fin). En el medio esta el CODE (código)
- CODE : constituido por STATEMENT o STATEMENT CODE

- STATEMENT : contiene principalmente todas las funcionalidades del lenguaje, print/imprimir, write/escribir, end line/fin de linea, declaraciones, asignaciones, definiciones y bloques if - else, do - while y for.
- ELIF : contiene los else if del bloque if - else, con sus posibles variaciones, con una o mas condiciones.
- CONDITIONAL : contiene condiciones, las cuales están constituidas por expresiones y operandos aritméticos.
- EXPRESSION : las expresiones puede ser un termino o la suma, resta, multiplicación y división de expresiones.
- TERM : pueden ser del tipo numero, texto o ID.
- DECLARATION : permite la declaración de variables del tipo numero o texto.
- ASSIGNMENT : permite asignarle a una variable su valor.
- DEFINITION : permite declarar y asignarle valor a una variable en la misma linea

4.2 Operaciones aritméticas

A continuación se detallan las expresiones escritas para cada operación aritmética soportada.
La suma, resta, división y multiplicación se dejo con sus respectivos símbolos
+, -, /, *

- == es igual a
- != es distinto a
- >= es mayor o igual a
- <= es menor o igual a
- > es mayor a
- < es menor a
- || o
- && y

El programa permite realizar estas operaciones entre variables del mismo tipo. Si no lo son, va a mostrar un error. Esto se detalla en la sección de Manejo de Errores.

4.3 Iniciar programa

Para declarar donde comienza y donde termina el programa, la primer linea del código debe ser **inicio** y finalizar el código con **fin**.

```
inicio

código

fin
```

4.4 Declaración de variables

Las variables deben ser declaradas al inicio del programa. En caso de declararlas posterior a un bloque tirara un error de compilación.

- Para declarar una variable

```
inicio
numero minumero
text mitexto
fin
```

- Para asignarle un valor o texto a una variable

```
inicio
minumero es 3
mitexto es "hola"
fin
```

  Cabe resaltar que el valor de las variables del tipo texto van entre " "

- Para realizar la declaración y la asignación en la misma linea

```
inicio
numero  minumero es 3
texto  mitexto es "hola"
fin
```

4.5 Bloque If - elseif - else

Para utilizar este bloque se deben usar las palabras **si - o si - y si no**. Hay que utilizar **fin si** para delimitar el fin del bloque. Si se desea utilizar una sola condición, no se utilizan parentesis. Si se desean verificar una o más condiciones, los parentesis son necesarios. Como en cualquier otro lenguaje el **o si** y **si no** son optativos.

```
inicio
si  condición
    código
o si  condición
    código
y si no
    código
fin si
fin
```

Tambien se pueden utilizar de manera anidada con una o mas condiciones.

```
inicio
si (condición1 o condición2)
    código
y si no
    si condición
       código
    y si no
       código
    fin si
fin si
fin
```

4.6 Bloque Do - while

Para su uso se emplean las palabras **hacer - mientras**.

```
inicio
hacer
    código
mientras condición
fin
```

4.7 Bloque for

Para este bloque hay que usar las palabras **para - desde - hasta - de a**. Muy importante cerrar el ciclo con **fin ciclo**. El **de a**, es decir el incremento, es opcional. Si no se especifica, se toma como default 1.

```
inicio
para nombrevariable desde numeroinicial hasta numerofinal de a incremento
    código
fin ciclo
fin
```

4.8 Imprimir en pantalla

Para realizar una impresión en pantalla, se utiliza la palabra **imprimir** seguida de lo que se desee imprimir. Para un texto específico se lo debe poner entre " ". También se pueden imprimir suma, resta, etc. de variables.

```
inicio
imprimir   mivariable
imprimir    "\n"
imprimir   5
fin
```

4.9 Leer de teclado

Nuestro lenguaje permite poder leer desde el teclado, así tener una mayor interacción con el usuario. Para su uso se utiliza la palabra reservada **escribir en**. Esto se permite únicamente para variables del tipo texto, es decir, no se puede utilizar estas variables como números.

```
inicio
texto a
imprimir "ingrese su nombre"
escribir en a
imprimir "Su nombre es " + a
fin
```

4.10 Manejo de operaciones aritméticas

Como se mencionó anteriormente, se pueden realizar operaciones aritméticas entre variables del mismo tipo.

```
inicio
texto t es "holacomoestas"
texto m es "chau"
si t es mayor a m
    imprimir "Texto1 es mayor a texto2"
fin si
fin
```

```
inicio
numero t es 30
numero m es 3
si t es mayor a m
    imprimir "t es mayor a m"
fin si
fin
```

5 Manejo de errores

Para esta sección se crearon diferentes funciones, cada una chequea lo que su nombre indica. Se lleva una cuenta de las lineas del código, gracias a esto se muestra en que linea ocurre el error. Se muestran ejemplos a continuación, con el código y su correspondiente salida.

- Se chequean errores de asignación (no asignarle un número a algo declarado como texto, por ejemplo)

```
inicio
numero minumero
minumero es "hola"
fin
```

Se obtiene la salida

```
Los tipos no coinciden en la linea 3
```

- Si se quieren restar, multiplicar, dividir variables del tipo texto, se muestra el siguiente error

```
inicio
texto a es "hola"
numero b es 5
numero rta es a-b
fin
```

```
Operación Invalida - se requieren operandos del tipo numero.
```

- Si se definen variables dentro de algun bloque, ocurre un error. Éstas deben ser definidas al inicio, o fuera de los bloques if - else y do - while.

```
inicio
texto a es "hola"
numero b es 5
si b es igual a 0
    numero z es 4
fin si
fin
```

```
Las variables deben ser definidas al inicio. Definición errónea en la linea 5.
```

- Si se redefine una variable, se observa el siguiente error

```
inicio
numero b es 0
texto a es "hola"
numero b es 10
fin
```

```
Redefinió de variable b en la linea 4.
```

- Si se quieren utilizar operaciones aritméticas entre variables del tipo numero y texto, se producirá un error

```
inicio
numero t es 30
texto a es "hola"
si t es menor a a
        imprimir "error"
fin si
fin
```

```
Los tipos no coinciden en la linea 4.
```

6 Dificultades encontradas

La primer y mayor dificultad fue que ningún integrante del grupo tenia conocimientos previos sobre Lex y Yacc, por eso se tuvo que realizar investigación en ese tema, utilizando también el material dado en clase. Sobre el principio, a través de la prueba y error, se fue avanzando lentamente sobre el lenguaje. Cuando se fue probando el código a través de ejemplos, al no tener un manejo de errores muy especifico, si había algún error ortográfico, como imprmir y no imprimir, era difícil de detectar. También fue difícil detectar los conflictos de shift/reduce, pero se pudieron eliminar todos. Para esto, fue de mucha ayuda contar con las herramientas de debugging, flags que permiten imprimir mas en detalle todo lo que se va haciendo en 2do plano para entender por donde y por que fallaban las cosas. Una vez avanzado el lenguaje, no fue difícil agregar funcionalidades.

7 Futuras extensiones

La primer extensión seria crear mas tipos de datos para las variables, así poder tener *double, float, long, char[], true, false*, etc, y no solamente *numero* y *texto*.
Se debería a su vez mejorar en que linea ocurre el error, ya que a veces no la indica con exactitud.
También se podría pensar en la declaración de funciones propias para poder tener código fuera del inicio - fin. Esto permitiría menos repetición de código y mas similitud a un lenguaje de programación mas grande.
Seria ideal ademas, incluir algun tipo de declaracion de objetos, ya que su utilizacion permite manejar estructuras de datos mas grandes de manera sencilla y seria de gran utilidad para el objetivo de este lenguaje.
Por último, extender el manejo de errores seria provechoso, para mas claridad a la hora de encontrar errores en el código lo que haría que sea aún mas fácil desarrollar.

8 Referencias.

- `https://www.linux.co.cr/lg/issue93/ramankutty.html`
- `https://github.com/faturita/YetAnotherCompilerClass`
- `http://dinosaur.compilertools.net/yacc/`
- `https://www.geeksforgeeks.org/variable-length-argument-c/`
- `https://codereview.stackexchange.com/questions/-142638/a-variadic-c-function-for-concatenating-multiple-strings`