



# PROYECTO

## Compiladores e Interpretes

Instituto Tecnológico de Costa Rica. – II semestre

Estudiantes: André Solis Barrantes 2013389035

Daniel Alvarado Chou 201235858

Karen Lépez Chacón 2013005341

Sede: Cartago

Fecha de entrega: 30 de noviembre, 2016

### Abstract

This document will inform you of the tools used to create a compiler for the Arduino language. Where the grammar will be translated into Spanish and will present slight changes, these will be presented later. Also, this content the grammar BNF used, as well as examples of the functionality of the new version of grammar of the language.

It will detail the tools that were used for the execution of the project, and will show important definitions that the reader should handle for a correct understanding of the work

## Contenido

Introducción.....	2
Objetivo.....	3
Objetivo General.....	3
Objetivos Específicos.....	3
Justificación.....	4
Definiciones.....	5
Desarrollo.....	6
Análisis de Resultados.....	13
Prueba 1.....	13
Prueba 2.....	14
Prueba 3.....	15
Prueba 4.....	16
Conclusiones.....	17
Apéndice.....	18
Apéndice 1: Gramática BNF.....	18
Apéndice 2: programas de prueba.....	23
Bibliografía.....	32

## Introducción.

En este proyecto se quiere crear un compilador en donde se tiene como una base la gramática en el que se quiere trabajar, que, en este caso, corresponde a la gramática de Arduino. La gramática del lenguaje de programación Arduino será traducida al español, en donde no solo se le va a realizar el parser, scanner, sino también va a ser capaz de manejar errores léxicos, sintácticos y semánticos.

Se va a presentar el diseño del lenguaje traducido al español, el alfabeto y delimitadores que se utilizaron, así como la explicación de las causas de los errores léxicos, sintácticos y semánticos que se pueden presentar en el lenguaje, y también la manera en el que el programa se recupera del mismo.

## Objetivo.

### Objetivo General.

- Crear un compilador con la gramática traducida al español para el Lenguaje de programación: Arduino.

### Objetivos Específicos.

- Realizar un buen diseño de gramática para el lenguaje Arduino.
- Manejar herramientas como JFlex y Cup para el desarrollo de un parser y scanner.
- Implementar los componentes necesarios que necesita un compilador para ejecutarse correctamente.

### Justificación.

El lenguaje de programación seleccionado para realizarle la traducción a su gramática es Arduino, y se elige debido a que estamos familiarizados con la gramática que se utiliza este lenguaje y la sencillez de su implementación. Además, el acceso a sus reglas gramáticas iba a ser más sencilla de conseguir debido a que es código libre.

Para realizar el analizador léxico y el semántico se eligieron las herramientas JFlex y Cup respectivamente. A continuación, se mostrará un cuadro con un pequeño resumen de las razones por las cuales se eligieron estas herramientas.

JFlex	Cup
Permite generar analizadores de manera rápida.	Declaración de símbolos no terminales y terminales.
La sintaxis es fácil de comprender y de emplear.	Declaraciones de precedencia.
Independiente de la plataforma.	Se puede definir el símbolo inicial de la gramática.
Integración con Cup.	Definición de las reglas de producción.

*Tabla1. Características de JFlex y Cup.*

## Definiciones.

- Analizador léxico(Scanner): Opera bajo la petición del analizador sintáctico devolviendo un componente léxico conforme el analizador va avanzando atreves de la gramática dada.
- Analizador semántico(Parser): Es el encargado de comprobar que el árbol sintáctico detectado cumpla con las restricciones de tipo y otras limitaciones semánticas para después pasar a la siguiente fase: generación de código.
- Arduino: es una plataforma de hardware, en la que se basa en un microcontrolador que facilita el uso de la electrónica, además también contiene un software de código libre en la que está basada en el lenguaje de programación Processing.
- BFN (Backus-Naur-Form): es una notación formal para definir la sintaxis de un lenguaje.
- Compilador: analiza el programa y lo traduce a lenguaje máquina.
- Cup: Es un generador de analizadores sintáctico que pude ser integrado con java.
- Gramática: reglas y principios sintácticos que se utiliza en un lenguaje.
- Interprete: Analiza el programa fuente y lo ejecuta de manera directa.
- Java: es un lenguaje de programación orientado a objetos.
- JFlex: Es un generador de analizadores léxicos que pude ser integrado con java.
- Lenguaje de programación: Estructura con ciertas reglas sintácticas y semánticas, con las cuales se pueden impartir instrucciones que son traducidas a un lenguaje que la máquina pueda comprender.
- Programación Orientada a Objetos: es un paradigma de programación que define los programas en términos de clases de objetos.

- Tokens: símbolos terminales de una gramática.

## Desarrollo

El diseño de la gramática lenguaje Arduino que se va a utilizar será bastante similar a la gramática original del mismo lenguaje, con la excepción que las funciones LOOP() y SETUP(), serán funciones en las cuales el usuario no podrá sobrescribir, y estarán junto con las palabras reservadas. Otro cambio notorio en la “nueva” gramática, es que esta estará traducida al idioma español. A continuación, se mostrarán dos tablas, en donde la primera tabla (tabla 2) se detallará características de la sintaxis y en la tabla 3 se mostrará las palabras reservadas que tendrá el lenguaje, así como la descripción de su funcionamiento.

Sintaxis Utilizada	
<b>Sintaxis Básica</b>	
• Delimitadores	{ }
• Comentarios	//,/**/
• Asignación	=
• Operadores aritméticos	+, -, %, /, *, ^
• Operadores de comparación.	<, <=, >, >=
• Operadores Booleanos	==, !=, not, and, or
• Operadores de bits	&,  , ^, ~, <<, >>
• Operadores Compuestos	
- Incremento y decremento	++, --
- Asignación y operación	+=, -=, *=, /=, %=, &=,  =
<b>Estructuras de control</b>	Si, mientras, sino, sino si
<b>Variables</b>	Tipo de dato(int,string,etc) nombreVariable “=” sentencia

<b>Constantes</b>	ALTO, BAJO, ENTRADA, SALIDA, FALSO, VERDAD
<b>Tipos de datos</b>	Boolean, char, unsigned_char, byte, int, unsigned_int, void, Word, long, unsigned_long, short, float, double, string, String, array
<b>Entrada/Salida digital</b>	PinMode(), DigitalWrite(), DigitalRead()
<b>Tiempo</b>	Esperar, microsegundos
<b>Matemáticas</b>	Min, max, abs, constrain, map
<b>Números Aleatorios</b>	Random
<b>Bits y Bytes</b>	ByteBajo, byteAlto, leerBit, escribirBit, bitSet, limpiarSet, bit
<b>Alfabeto</b>	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z A, B, C, D, E, F, G, H, I, J, K, L, O, P, Q, R, S, T, U, V, W, X, Y, Z
<b>Números</b>	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Tabla2. Sintaxis que se usará para el Lenguaje Arduino.

<b>Palabras Reservadas</b>	<b>Descripción o Significado</b>
<b>ABS</b>	Nombre de una función para calcular el valor absoluto de un número.
<b>ALEATORIO</b>	Encargado de generar números aleatorios de acuerdo a un rango determinado.
<b>ALTO</b>	Se utilizará para que el voltaje del pin sea mayor de 3 o 2 voltios, dependiendo de la entrada, salida y el pinMode que se le indique.
<b>ARREGLO</b>	Indica un arreglo de un tipo de objeto especificado.
<b>BAJO</b>	Se utilizará para que el voltaje del pin sea menor de 3 o 2 voltios, dependiendo de la entrada, salida y el pinMode que se le indique.
<b>BOLEANO</b>	Tipo de dato booleano que puede ser VERDAD o FALSO



<b>BYTE</b>	Representa los valores de un bit específico.
<b>BYTE_ALTO</b>	Función que extra el bit de orden superior (el que se encuentra más a la izquierda) de una variable
<b>BYTE_BAJO</b>	Función que extra el bit de orden inferior (el que se encuentra más a la derecha) de una variable
<b>BYTE_SET</b>	Declara un bit con un valor numérico.
<b>CHAR</b>	Tipo de dato para representar caracteres.
<b>CONST</b>	Nombre de una función que restringe un número para estar dentro de un rango.
<b>DEOTROMODO</b>	Nombre de una estructura de control que procede a ejecutarse cuando la condición del SI no se cumple, pero el DEOTROMODO se ejecutará solo si se cumple la condición establecida en el mismo.
<b>DIGITALREAD</b>	Nombre de una función que lee el pin digital que se ha especificado.
<b>DIGITALWRITE</b>	Nombre de una función que le asigna a un pin un valor de voltaje ALTO o BAJO
<b>ENTERO</b>	Tipo de dato para representar números enteros.
<b>ENTERO_CORTO</b>	Tipo de dato para representar números enteros cortos.
<b>ENTERO_LARGO</b>	Tipo de dato para representar números enteros largos.
<b>ENTRADA</b>	Es un modo en el que se puede configurar un pin, en donde estos tendrán un estado de alta impedancia.
<b>ENTRADA_PULLOP</b>	Este modo de pin es usando cuando estos tienen resistencias internas, es decir, resistencias que se conectan a la alimentación interna.
<b>ES_ALPHA</b>	Función que se encarga de indicar si el carácter está entre las letras del alfabeto definidas en el lenguaje.
<b>ES_ASCII</b>	Función que se encarga de indicar si el carácter es ASCII.

<b>ES_CONTROL</b>	Función que se encarga de indicar si el carácter es un carácter de control.
<b>ES_DIGITO</b>	Función que se encarga de indicar si el carácter es un dígito.
<b>ES_ESPACIO</b>	Función que se encarga de indicar si el carácter es un carácter de espacio.
<b>ES_ESPACIO_BLANCO</b>	Función que se encarga de indicar si el carácter es un espacio en blanco.
<b>ES_GRAFICO</b>	Función que se encarga de indicar si el carácter es imprimible.
<b>ES_HEXADECIMAL</b>	Función que se encarga de indicar si el carácter esta en hexadecimal.
<b>ES_IMPRIMIBLE</b>	Función que se encarga de indicar si el carácter es imprimible.
<b>ES_MAYUSCULA</b>	Función que se encarga de indicar si el carácter esta en mayúscula.
<b>ES_MINUSCULA</b>	Función que se encarga de indicar si el carácter esta en minúscula.
<b>ES_NUMERO_ALPHA</b>	Función que se encarga de indicar si el carácter es alpha numérico.
<b>ES_SIGNO_PUNTUACION</b>	Función que se encarga de indicar si el carácter es un signo de puntuación.
<b>ESCRIBIR_BIT</b>	Función encargada de escribir/declarar un bit.
<b>ESPERAR</b>	Encargado de definir un tiempo de espera para ejecutar la siguiente instrucción.
<b>FALSO</b>	Constante que representa que lo que se está definiendo es falso.

<b>HACER</b>	Nombre de una estructura de control que funciona similar a la estructura MIENTRAS, con la excepción de que siempre se va a ejecutar al menos una vez.
<b>LED_CONECTADO</b>	Indica si el led se encuentra conectado o no.
<b>LEER_BIT</b>	Función encargada de leer un bit.
<b>LIMPIAR_BIT</b>	Encargado de limpiar el valor que se le ha dado previamente a un bit
<b>LOOP</b>	Esta función debe escribirse después del SETUP(), y será el encargado de generar ciclos consecutivamente permitiendo que el programa cambie y responda.
<b>MAP</b>	Nombre de una función que se va a encargar de reasignar el rango de un número a otro.
<b>MAX</b>	Nombre de una función para calcular el máximo entre dos números.
<b>MICROSEGUNDOS</b>	Será el encargado de devolver el número de microsegundos desde que el arduino entro en ejecución.
<b>MIENTRAS</b>	Nombre de una estructura de control que se va a encargar de realizar ciclos mientras que la condición que se declare se cumpla
<b>MIN</b>	Nombre de una función para calcular el mínimo entre dos números.
<b>PIN_MODE</b>	Nombre de una función que se encarga de configurar el comportamiento de un pin, en este caso: entrada o salida.
<b>POR</b>	Nombre de una estructura de control que se va a encargar de realizar ciclos mientras que la condición que se declare se cumpla

<b>SALIDA</b>	Es un modo en el que se puede configurar un pin, en donde estos tendrán un estado de baja impedancia.
<b>SETUP(PREPARAR)</b>	Función que se utiliza para empezar un “sketch”, en este espacio se inician variables, modos de los pines, las librerías que se van a utilizar.
<b>SI</b>	Nombre de una estructura de control en donde utiliza operaciones de comparación y el comportamiento que vaya a tener dependerá si el resultado de la comparación es VERDAD o FALSO.
<b>SINO</b>	Nombre de una estructura de control que procede a ejecutarse cuando la condición del SI no se cumple. Es decir, Si la condición establecida no se cumple, ejecuta lo escrito dentro del SINO.
<b>STRING</b>	Tipo de dato para representar una cadena caracteres.
<b>VERDADERO</b>	Constante que representa que lo que se está definiendo es verdadero.

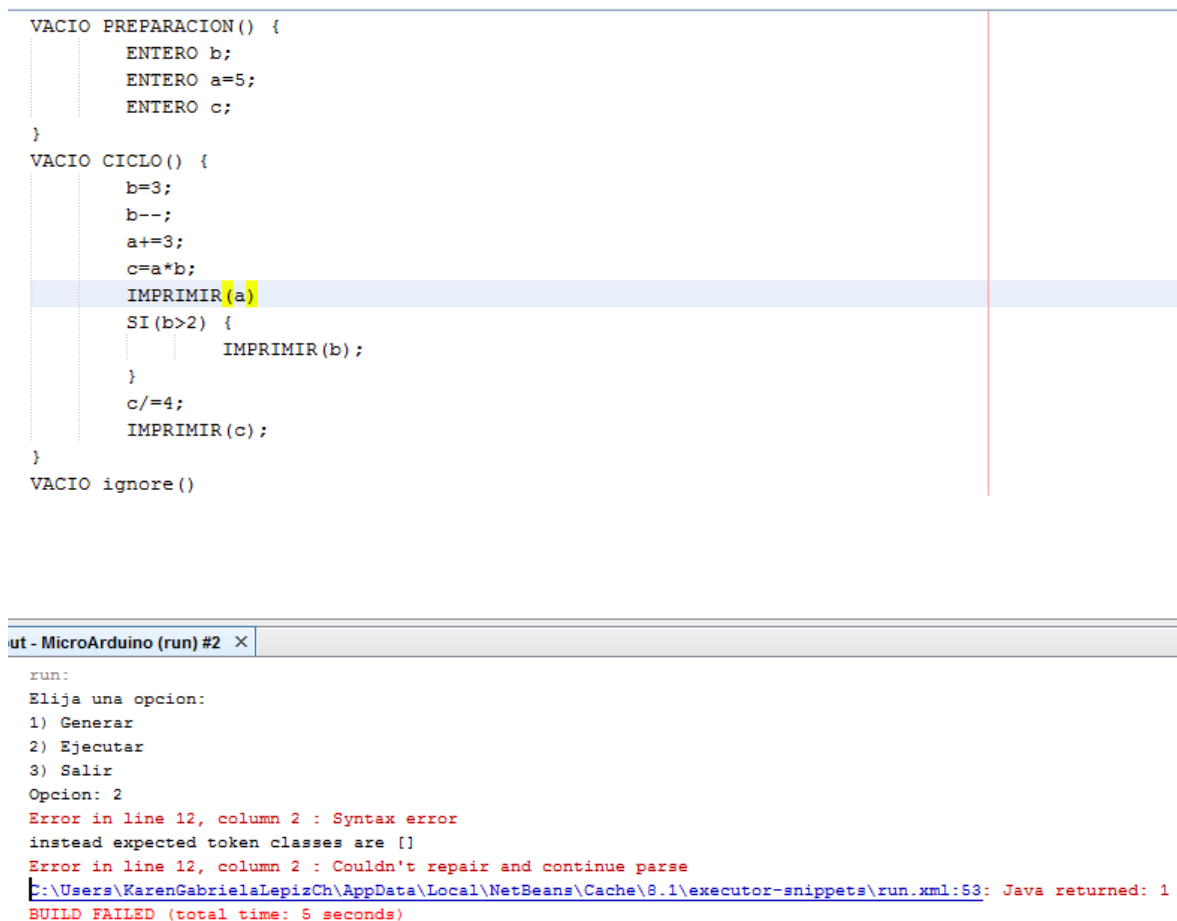
*Tabla3. Palabras reservadas con su respectiva descripción que tendrá el Lenguaje Arduino.*

Después de realizar el analizador léxico y el semántico, se ejecutan poniendo en prueba la gramática y la lógica que se ha especificado en el mismo. Para este proyecto, lo que se puede realizar es:

- Operaciones aritméticas sencillas.
- Operaciones aritméticas usando variables. Por ejemplo:  $a = 1$  y  $b = 1$ ,  $a + b = 2$ .
- Operaciones con asignaciones compuestas.
- Operaciones de incremento y decremento ( $++$ ,  $--$ ).
- Operaciones de comparación.
- Imprimir resultados.
- Menú para dar facilidad de uso del programa.

Es común, que el usuario al estar escribiendo código cometa errores, ya sea ortográficos o semánticos, por lo que el lenguaje debe estar diseñado para soportar estos tipos de errores. En este caso, los errores sintácticos serán reconocidos, y se le indicara al usuario que tuvo un error de ese tipo, sin embargo, no será aprueba de errores, es decir, solo le avisará al programa del error, pero no hará nada al respecto para que el programa no termine de manera inesperada.

Por ejemplo:



The image shows a code editor window with the following code:

```
VACIO PREPARACION() {  
    ENTERO b;  
    ENTERO a=5;  
    ENTERO c;  
}  
VACIO CICLO() {  
    b=3;  
    b--;  
    a+=3;  
    c=a*b;  
    IMPRIMIR(a)  
    SI (b>2) {  
        IMPRIMIR(b);  
    }  
    c/=4;  
    IMPRIMIR(c);  
}  
VACIO ignore()
```

The line `IMPRIMIR(a)` is highlighted in yellow. Below the code editor is a terminal window titled "ut - MicroArduino (run) #2". The terminal output shows the following:

```
run:  
Elija una opcion:  
1) Generar  
2) Ejecutar  
3) Salir  
Opcion: 2  
Error in line 12, column 2 : Syntax error  
instead expected token classes are []  
Error in line 12, column 2 : Couldn't repair and continue parse  
E:\Users\KarenGabrielaLepizCh\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 5 seconds)
```

*Imagen1. Muestra de un ejemplo de error sintáctico y la manera en la que se comporta el programa.*

En la imagen anterior, se puede mostrar como hay un error de sintaxis en la línea 12, el error se produce porque le hace falta un “;” al final de la línea, y como se mencionó antes,

se le indicará al usuario que se tuvo un error de sintaxis en la línea respectiva, pero no se manejarán los errores de manera correcta.

## Análisis de Resultados.

### Prueba 1.

```
VACIO PREPARACION() {  
    ENTERO b;  
    ENTERO a=5;  
    ENTERO c;  
}  
VACIO CICLO() {  
    b=3;  
  
    a+=3;  
    c=a*b;  
    IMPRIMIR(a);  
    IMPRIMIR(c);  
    SI (b>2) {  
        IMPRIMIR(b);  
    }  
    b--;  
    c/=4;  
    IMPRIMIR(b);  
    IMPRIMIR(c);  
}
```

*Imagen2. Prueba de funcionalidad.*

**Análisis de Resultados:** El código de prueba que se está ejecutando no posee errores de ningún tipo, por lo que se comporta de la manera que le corresponde, mostrando el siguiente resultado:

```
Elija una opcion:  
1) Generar  
2) Ejecutar  
3) Salir  
Opcion: 2  
8  
24  
3  
2  
6
```

*Imagen 3. Resultado de la ejecución de la prueba 1.*

Se puede apreciar como realiza la suma correcta a la variable a, que fue inicializada con 5, después se le suma 3, con un resultado final de 8. El segundo valor que se aprecia, es el producto de la operación de  $8 \times 3 = 24$ , sigue la impresión del valor que le corresponde a b, el penúltimo resultado que se aprecia es del decremento de b, es decir, --b. Y por último, el valor de 6 corresponde al resultado de la división de c entre 4.

#### Prueba 2.

```
VACIO PREPARACION() {  
    ENTERO b;  
    ENTERO a=5;  
    ENTERO c;  
}  
VACIO CICLO() {  
    b=3;  
    SI (b>2) {  
        IMPRIMIR (b) ;  
    }  
}
```

*Imagen4. Prueba de funcionalidad.*

**Análisis de Resultados:** El código de prueba que se está ejecutando no posee errores de ningún tipo, por lo que se comporta de la manera que le corresponde, mostrando el siguiente resultado:

```
Elija una opcion:  
1) Generar  
2) Ejecutar  
3) Salir  
Opcion: 2  
3
```

*Imagen5. Resultados de la ejecución de la prueba2.*

Se puede apreciar como la validación de  $b > 2$  se realiza correctamente, por eso mismo se imprime el valor que le corresponde a esa variable.

### Prueba 3.

```
VACIO PREPARACION() {  
    ENTERO b;  
}  
VACIO CICLO() {  
    b=3;  
    IMPRIMIR(b) ;  
    b=3+2;  
    IMPRIMIR(b) ;  
    b=3*2;  
    IMPRIMIR(b) ;  
    b=6/2;  
    IMPRIMIR(b) ;  
    b+=1;  
    IMPRIMIR(b) ;  
}
```

*Imagen6. Prueba de funcionalidad.*

**Análisis de Resultados:** El código de prueba que se está ejecutando no posee errores de ningún tipo, por lo que se comporta de la manera que le corresponde, mostrando el siguiente resultado:



Elija una opcion:

- 1) Generar
- 2) Ejecutar
- 3) Salir

Opcion: 2

3

5

6

3

4

Imagen7. Resultados de la ejecución de la prueba 3.

- El resultado de 3, corresponde al primer valor que se le asigna a b.
- El resultado de 5, corresponde al resultado de la segunda operación que se implementa que es  $3+2$ .
- El resultado de 6, corresponde al resultado de la tercera operación que se implementa que es  $3*2$ .
- El resultado de 3, corresponde al resultado de la cuarta operación que se implementa que es  $3/2$ .
- El resultado de 4, corresponde al resultado de la quinta operación que se implementa que es  $b+=1$ , en donde el valor de b corresponde a 3, que fue el resultado de la última operación que se realizó.

#### Prueba 4.

```
1 VACIO PREPARACION() {  
2     ENTERO b;  
3 }  
4 VACIO CICLO() {  
5     b=3;  
6  
7     SI (b<2)  
8         IMPRIMIR (b) ;  
9     }  
10  
11 }  
12 VACIO ignore()
```

*Imagen8. Prueba de error.*

**Análisis de Resultados:** El código de prueba que se está ejecutando posee errores de ningún tipo, por lo que se comporta de la siguiente manera:

```
Elija una opcion:  
1) Generar  
2) Ejecutar  
3) Salir  
Opcion: 2  
Error in line 8, column 3 : Syntax error  
instead expected token classes are []
```

*Imagen9. Resultados de la ejecución de la prueba 4.*

Como se aprecia, el programa indicará en que línea se está cometiendo el error sintáctico, en este caso ocurre porque en la línea 8, hace falta un “{”. El programa indicara el error que se tiene, pero no la razón por la que ocurre, tampoco se manejarán los errores.

## Conclusiones.

El desarrollo de un compilador para un lenguaje es una tarea bastante tediosa, ya que todo depende del buen diseño de la gramática. Si la gramática no está bien construida o diseñada, ya sea por errores e ambigüedad o por escritura, el resto del proceso puede resultar defectuoso o incluso se tenga que volver a crear. Se tiene que tener bien claro la manera en la que se quiere que el lenguaje funcione y como interpretar las sentencias que se escriban en el mismo.

La buena comprensión sobre el funcionamiento de un compilador, así como la tarea que realiza cada parte, así como comprender la manera en la que funciona el lenguaje en el que se va a trabajar, ya sea de manera sintáctica y de manera semántica. Entender el lenguaje facilitará ligeramente la construcción de la gramática que se desea usar para el compilador.

Algunas de las observaciones presentes durante la elaboración del proyecto se encuentran las siguientes:

- Las herramientas que se elijan utilizar puede afectar de manera positiva o negativa en el proyecto, en este caso se utilizaron JFlex para el scanner y Cup

para el parser, las razones por las cuales se eligieron se encuentran en la sección de Justificación de este mismo documento.

- Encontrar información sobre cómo realizar los analizadores y entender la manera que funciona, para poder crear nuestra propia lógica fue difícil, afectando la dificultad de la creación del compilador.
- El lenguaje Arduino en particular, se tuvo que crear una propia gramática en BNF, puesto que no estaba publicada de manera pública en ningún repositorio (git) o en la página oficial del lenguaje.

## Apéndice.

### Apéndice 1: Gramática BNF.

```
terminal SETUP, LOOP, CALL, INSTANTIATE, VAR, END, FUNCTION, RETURN, NULL, ARRAY, WAIT, PRINT, TRUE, FALSE, HIGH, LOW,
IN, OUT, FOR, DO, WHILE, IF, ELSE, ELSEIF, BOOLEAN, STRING, CHAR, FLOAT, INT, LONG, BYTE, VOID, ISALPHANUMERIC,
ISALPHA, ISASCII, ISWHITEPACE, ISCONTROL, ISDIGIT, ISGRAFIC, ISLOWER, ISPRINT, ISPUNCTUATION, ISSPACE, ISUPPER,
ISHX, ABS, RANDOM, READBIT, WRITEBIT, SETBYTE, CLEANBIT, DIGITALWRITE, DIGITALREAD, CONST, MAP, MAX, MIN, MICRO, MILLI, PINMODE;
terminal PLUS, MINUS, MULT, DIV, SORT, POW, MOD, EQ, PLUSEQ, MINUSEQ, MULTEQ, DIVEQ, INC, DEC, EQEQ, BIG, BIGEQ, LESS, LESSEQ, NOTEQ,
LPAREN, RPAREN, LBRACE, RBRACE, COMMA, SEMI, NOT, AND, OR;
terminal String TAG;
terminal Integer NUM;

non terminal Object Programa, Cuerpo, Declaracion, Asignacion, Funciones, Function, Token_tipo, Token_asignacion, Token_asigna_arit,
Token_ment, Token_comparacion, Token_aritmetico, Token_constantes_VoF, Token_constantes_AoB, Token_constantes_EoS,
Sentencias, Sentencia, Imprimir_var, Cuando;
non terminal Integer Expr, Factor, Termino;
non terminal Boolean Comparacion;

precedence left PLUS, MINUS;
precedence left MULT, DIV, MOD;
precedence left SORT, POW;
precedence left LPAREN, RPAREN, LBRACE, RBRACE;
precedence left SEMI, COMMA;
precedence right ELSE;

start with Programa;

//Estructura base del programa
Programa ::= VOID SETUP LPAREN RPAREN Cuerpo VOID LOOP LPAREN RPAREN Cuerpo Funciones
;
Cuerpo ::= LBRACE Sentencias RBRACE
;
```

```

//Sentencias de codigo
Sentencias ::= Sentencia Sentencias
            | Sentencia
            ;

Sentencia  ::= Declaracion
            | Asignacion
            | Comparacion
            | Imprimir_var
            | Cuando //if
            | Expr:e SEMI
            ;

//Variables
Declaracion ::= Token_tipo TAG:id Token_asignacion Expr:val SEMI //Declara y asigna
            { :
              table.put( id, val );
            : }
            | Token_tipo TAG:id SEMI //Solo declara
            { :
              table.put( id, 0 );
            : }
            ;

Asignacion  ::= TAG:id Token_asignacion Expr:val SEMI //Asigna
            { :
              Integer value = ( Integer ) table.get( id );
              if ( value == null ) {
                parser.report_error( "Undeclared Identifier " + id,
                                     new Symbol( sym.TAG ) );
                value = new Integer( 0 );
              } else {
                table.put( id, val );
              }
            : }

            | TAG:id Token_asigna_arit:tk Expr:e SEMI //Asigna y aritmetica juntos
            { :
              Integer value = ( Integer ) table.get( id );
              if ( value == null ) {
                parser.report_error( "Undeclared Identifier " + id,
                                     new Symbol( sym.TAG ) );
                value = new Integer( 0 );
              } else {
                if(tk.equals("+")) { table.put( id, value+e.intValue() ); }
                if(tk.equals("-")) { table.put( id, value-e.intValue() ); }
                if(tk.equals("*")) { table.put( id, value*e.intValue() ); }
                if(tk.equals("/")) { table.put( id, value/e.intValue() ); }
              }
            : }

            | TAG:id Token_ment:tk SEMI //Incrementa y decrementa por 1
            { :
              Integer value = ( Integer ) table.get( id );
              if ( value == null ) {
                parser.report_error( "Undeclared Identifier " + id,
                                     new Symbol( sym.TAG ) );
                value = new Integer( 0 );
              } else {
                if(tk.equals("++")) { table.put( id, value+1 ); }
                if(tk.equals("--")) { table.put( id, value-1 ); }
              }
            : }
            ;

```

```

Imprimir_var ::= PRINT LPAREN TAG:id RPAREN SEMI
{
    Integer value = ( Integer ) table.get( id );
    if ( value == null ) {
        parser.report_error( "Undeclared Identifier " + id,
            new Symbol( sym.TAG ) );
        value = new Integer( 0 );
    } else {
        System.out.println(value);
    }
}
;

//Comparacion
Comparacion ::= Expr:e1 Token_comparacion:tk Expr:e2
{
    if(tk.equals("==")) { if(e1.intValue() == e2.intValue()){ RESULT = true; } else { RESULT = false; } }
    if(tk.equals("!=")) { if(e1.intValue() != e2.intValue()){ RESULT = true; } else { RESULT = false; } }
    if(tk.equals("<")) { if(e1.intValue() < e2.intValue()){ RESULT = true; } else { RESULT = false; } }
    if(tk.equals("<=")) { if(e1.intValue() <= e2.intValue()){ RESULT = true; } else { RESULT = false; } }
    if(tk.equals(">")) { if(e1.intValue() > e2.intValue()){ RESULT = true; } else { RESULT = false; } }
    if(tk.equals(">=")) { if(e1.intValue() >= e2.intValue()){ RESULT = true; } else { RESULT = false; } }
}
;

//Estructuras de control
Cuando ::= IF LPAREN Comparacion:c RPAREN LBRACE Sentencias:s RBRACE
{
    if(c){ RESULT = s; }
}
;

//Expresion:
Expr ::= Expr:e PLUS Factor:f
{
    RESULT = new Integer(e.intValue() + f.intValue());
}
| Expr:e MINUS Factor:f
{
    RESULT = new Integer(e.intValue() - f.intValue());
}
| Factor:f
{
    RESULT = f;
}
;

Factor ::= Factor:f MULT Termino:t
{
    RESULT = new Integer(f.intValue() * t.intValue());
}
| Factor:f DIV Termino:t
{
    RESULT = new Integer(f.intValue() / t.intValue());
}
| Termino:t
{
    RESULT = t;
}
;

```

```

Termino ::= LPAREN Expr:e RPAREN
        {
            RESULT = e;
        }
        | NUM:n
        {
            RESULT = n;
        }
        | TAG:id
        {
            Integer value = ( Integer ) table.get( id );
            if ( value == null ) {
                parser.report_error( "Undeclared Identifier " + id,
                new Symbol( sym.TAG ) );
                value = new Integer( 0 );
            } else {
                RESULT = value;
            }
        }
        :}
;

//Funciones: TODO
Funciones ::= Function Funciones
           | Function
           ;
Function ::= Token_tipo TAG LPAREN RPAREN
           ;

```

```

//No Terminales para agrupar Tokens
Token_tipo ::= CHAR:s { : RESULT = s; : }
           | INT:s { : RESULT = s; : }
           | LONG:s { : RESULT = s; : }
           | FLOAT:s { : RESULT = s; : }
           | BYTE:s { : RESULT = s; : }
           | BOOLEAN:s { : RESULT = s; : }
           | STRING:s { : RESULT = s; : }
           | VOID:s { : RESULT = s; : }
           ;

Token_asignacion ::= EQ:s { : RESULT = s; : }
                 ;

Token_asigna_arit ::= PLUSEQ:s { : RESULT = s; : }
                  | MINUSEQ:s { : RESULT = s; : }
                  | MULTEQ:s { : RESULT = s; : }
                  | DIVEQ:s { : RESULT = s; : }
                  ;

Token_ment ::= INC:s { : RESULT = s; : }
            | DEC:s { : RESULT = s; : }
            ;

Token_comparacion ::= EQEQ:s { : RESULT = s; : }
                  | NOTEQ:s { : RESULT = s; : }
                  | BIG:s { : RESULT = s; : }
                  | BIGEQ:s { : RESULT = s; : }
                  | LESS:s { : RESULT = s; : }
                  | LESSEQ:s { : RESULT = s; : }
                  ;

Token_aritmetico ::= PLUS:s { : RESULT = s; : }
                 | MINUS:s { : RESULT = s; : }
                 | MULT:s { : RESULT = s; : }
                 | DIV:s { : RESULT = s; : }
                 | MOD:s { : RESULT = s; : }
                 | POW:s { : RESULT = s; : }
                 | SQRT:s { : RESULT = s; : }
                 ;

Token_constantes_VoF ::= TRUE:s { : RESULT = s; : }
                     | FALSE:s { : RESULT = s; : }
                     ;

Token_constantes_AoB ::= HIGH:s { : RESULT = s; : }
                     | LOW:s { : RESULT = s; : }
                     ;

Token_constantes_EoS ::= IN:s { : RESULT = s; : }
                     | OUT:s { : RESULT = s; : }
                     ;

```

## Apéndice 2: programas de prueba.

----- Pruebas con ejecución correctamente -----

-----

----- Prueba 1.

-----

```
VACIO PREPARACION() {
```

```
    ENTERO b;
```

```
    ENTERO a=5;
```

```
    ENTERO c;
```

```
}
```

```
VACIO CICLO() {
```

```
    b=3;
```

```
    a+=3;
```

```
    c=a*b;
```

```
    IMPRIMIR(a);
```

```
    IMPRIMIR(c);
```

```
    SI(b>2) {
```

```
        IMPRIMIR(b);
```

```
    }
```

```
    b--;
```

```
    c/=4;
```



```
IMPRIMIR(b);
```

```
IMPRIMIR(c);
```

```
}
```

-----

----- Prueba 2.

-----

```
VACIO PREPARACION() {
```

```
    ENTERO b;
```

```
}
```

```
VACIO CICLO() {
```

```
    b=3;
```

```
    SI(b>2) {
```

```
        IMPRIMIR(b);
```

```
    }
```

```
}
```

-----  
----- Prueba 3.  
-----

```
VACIO PREPARACION() {
```

```
    ENTERO b;
```

```
}
```

```
VACIO CICLO() {
```

```
    b=3;
```

```
    IMPRIMIR(b);
```

```
    b=3+2;
```

```
    IMPRIMIR(b);
```

```
    b=3*2;
```

```
    IMPRIMIR(b);
```

```
    b=6/2;
```

```
    IMPRIMIR(b);
```

```
    b+=1;
```

```
    IMPRIMIR(b);
```

```
}
```

----- Prueba 4.

VACIO PREPARACION() {

    ENTERO b;

}

VACIO CICLO() {

    b=3;

    IMPRIMIR(b);

    b-=2;

    IMPRIMIR(b);

    b\*=2;

    IMPRIMIR(b);

    b/=2;

    IMPRIMIR(b);

    b+=1;

    IMPRIMIR(b);

}

-----  
----- Prueba 5.  
-----

VACIO PREPARACION() {

    ENTERO b;

}

VACIO CICLO() {

    b=3;

    SI(b!=2) {

        IMPRIMIR(b);

    }

}

----- Pruebas con errores -----

-----

----- Prueba 6.

-----

VACIO PREPARACION() {

    ENTERO b;

}

VACIO CICLO() {

    b=3;

    SI(b<2)

        IMPRIMIR(b);

    }

}

-----  
----- Prueba 7.  
-----

VACIO PREPARACION() {

    ENTERO b

}

VACIO CICLO() {

    b=3;

    SI(b<2){

        IMPRIMIR(b);

    }

}

----- Prueba 8.

VACIO PREPARACION() {

    INT b

}

VACIO CICLO() {

    b=3;

    SI(b<2){

        IMPRIMIR(b);

    }

}

-----  
----- Prueba 9.  
-----

VACIO PREPARACION() {

    ENTERO b

}

VACIO CICLO() {

    b 3;

    SI(b<2){

        IMPRIMIR(b);

    }

}



----- Prueba 10.

VACIO PREPARACION( {

ENTERO b

}

VACIO CICLO() {

b=3;

SI(b<2){

IMPRIMIR(b);

}

}

## Bibliografía

*Capítulo 5. Análisis Semántico.* (s.f.). Recuperado el 23 de Noviembre de 2016, de <http://arantxa.ii.uam.es/~alfonsec/docs/compila5.htm>

- Castro, R. A. (Octubre de 2008). *Integración de JFlex y Cup*. Recuperado el 23 de Noviembre de 2016, de <http://www.rafaelvega.info/wp-content/uploads/Articulo.pdf>
- Compiladores e Interpretes*. (s.f.). Recuperado el 23 de Noviembre de 2016, de Fundamentos de Programación:  
<https://funprogramacion.wikispaces.com/Compiladores+e+Int%C3%A9rpretes>
- Definición: Arduino*. (5 de octubre de 2012). Recuperado el 23 de Noviembre de 2016, de Arduino:  
<http://jamangandi2012.blogspot.com/2012/10/que-es-arduino-te-lo-mostramos-en-un.html>
- EcuRed. (s.f.). *Lenguajes de Programación*. Recuperado el 23 de Noviembre de 2016, de EcuRed:  
[https://www.ecured.cu/Lenguaje\\_de\\_Programaci%C3%B3n](https://www.ecured.cu/Lenguaje_de_Programaci%C3%B3n)
- Rodríguez, A. (s.f.). *¿Qué es Java? Concepto de programación orientada a objetos vs programación estructurada*. Recuperado el 23 de Noviembre de 2016, de aprende a programar:  
[http://www.aprenderaprogramar.com/index.php?option=com\\_content&view=article&id=368:i-que-es-java-concepto-de-programacion-orientada-a-objetos-vs-programacion-estructurada-cu00603b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188](http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=368:i-que-es-java-concepto-de-programacion-orientada-a-objetos-vs-programacion-estructurada-cu00603b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188)
- Vigilante. (10 de Diciembre de 2007). *Analizador léxico, sintáctico y semántico con JFlex y CUP*. Recuperado el 23 de Noviembre de 2016, de CRYSQL: <http://crysol.org/es/node/819>