Tecnológico de Monterrey

Ing. Elda Quiroga

Dr. Hector Ceballos

Diseño de Compiladores

L2C-bot

Documentacion

Entrega Final

Arturo Rojas Ortiz

A01039185

Diego Jiménez Torres

A01139513

Monterrey, Nuevo León a 27 de noviembre del 2019

Indice

Proyecto

Proposito

Utilizar este proyecto como un reto a nuestros conocimientos de programación con la visión de poder aprender el proceso de desarrollar un compilador básico, desde la definición de su gramática, hasta la generación de código. Agregándole la capacidad de que nuestro lenguaje pueda funcionar exitosamente con el microcontrolador de Arduino, y usarlo como a nosotros nos sea más conveniente.

Objetivo

Elaborar un lenguaje básico didáctico específico, que funcione como una introducción a la programación para niños de secundaria, donde a través de comandos sencillos puedan controlar los movimientos e interactuar con sensores de un carrito pre-ensamblado en Arduino. Con la finalidad de que pueda ser utilizado como una herramienta para la enseñanza, además de que sea divertido para operar.

Alcance

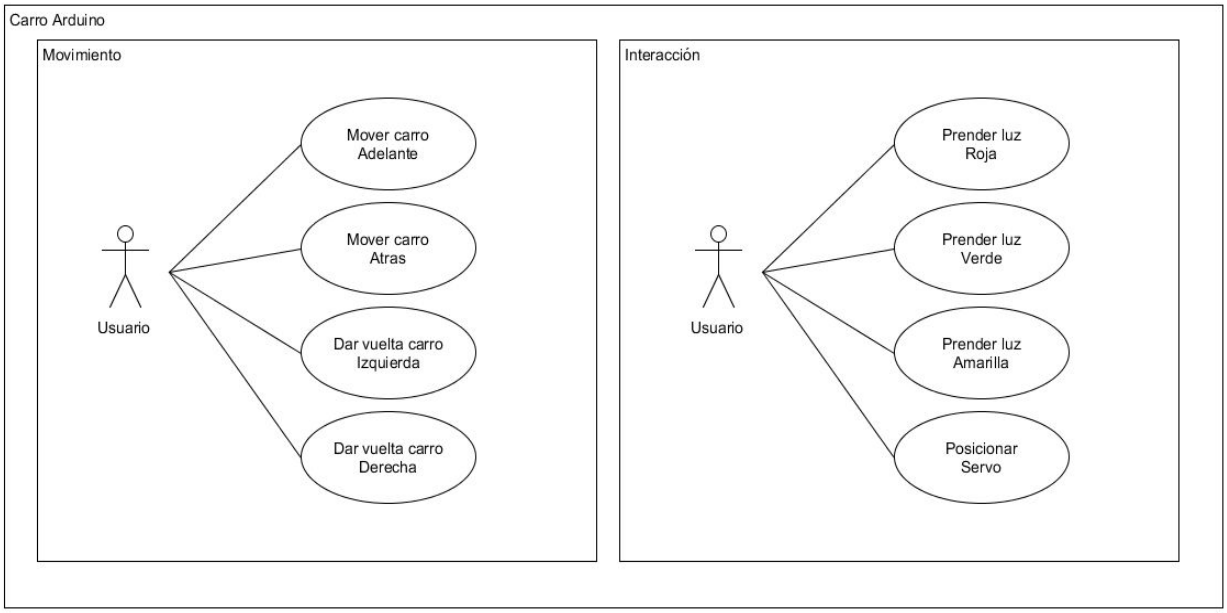
Fomentar un ámbito de aprendizaje con el uso de tecnologías de información elaborando un lenguaje imperativo didáctico basado en Arduino que sea: simple, fácil de enseñar y programar enfocado a niños de secundaria (11 a 15 años). Con la posibilidad elaborar el esqueleto básico funcional el cual pudiera en un futuro serle agregado aún más funcionalidades.

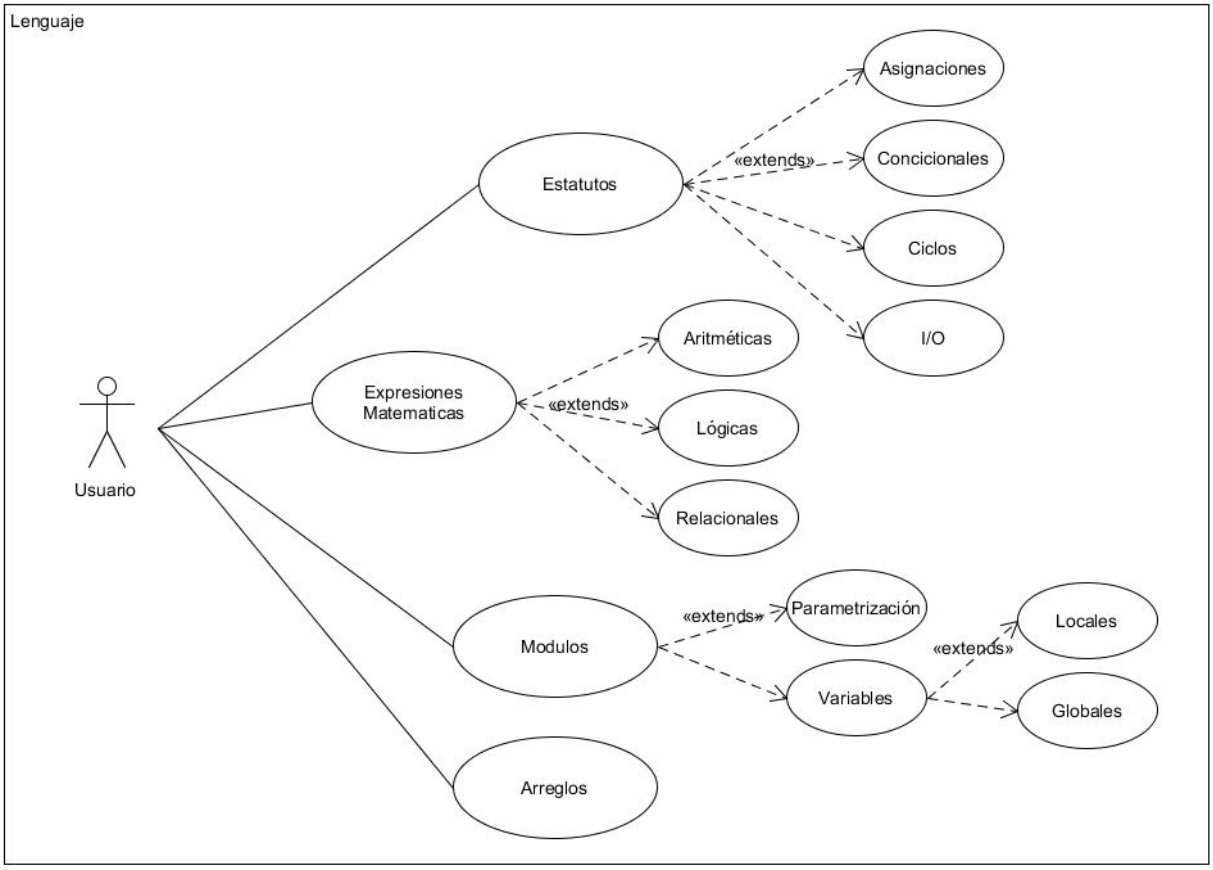
Análisis de Requerimientos

Este proyecto consta de elaborar un lenguaje de programación que contenga los elementos esenciales de cualquier otro lenguaje. Incluyendo pero no limitado a el uso de estatutos como lo son las asignaciones de variable, estatutos condicionales, ciclos y elementos de lectura y escritura. También, dentro del proyecto se incluirán expresiones matemáticas simples; entre ellos la aritmética, lógica matemática y matemática relacional.

Otros elementos que similarmente serán incluidos en nuestro lenguaje será el manejo de módulos parametrizados utilizando variables tanto globales como locales. Adicionado con el uso de elementos estructurados en arreglos de una dimensión. Todo esto en conjunto formará la base de nuestro lenguaje de programación simple orientado para la enseñanza.

Casos de Uso


Carro Arduino
Movimiento
Mover carro Adelante
Mover carro Atras
Dar vuelta carro Izquierda
Dar vuelta carro Derecha
Usuario
Interacción
Prender luz Roja
Prender luz Verde
Prender luz Amarilla
Posicionar Servo
Usuario


Lenguaje
Usuario
Estatutos
Asignaciones
«extends»
Concicionales
Ciclos
I/O
Expresiones Matematicas
Aritméticas
«extends»
Lógicas
Relacionales
Modulos
«extends»
Parametrización
Variables
«extends»
Locales
Globales
Arreglos

Principales Test Cases

Primeramente las pruebas estarán divididas acorde se fueron trabajando, separadas por el tipo de prueba entre elementos del vehículo con arduino y los principales elementos del lenguaje de programación imperativo.

Arduino

ID	**CP001**
Objetivo	Probar cada pin del Arduino funcione correctamente.
Precondición	● Se encuentra compilado Standard Firmata en Arduino. ● El puerto (socket) de Arduino se encuentra especificado (*eg. COM3)* ● Tener un led conectado en el Protoboard
Entradas	Definir cada pin digital como Entrada, Salida o PWM.
Resultados esperados	Prender luz led con cada uno de los pines conectado.
Resultados reales	Led se prendió con los pines 3,4,5,6,7,8,9,10,11,12,13.
Resultado de prueba	Aprobada

ID	**CP002**
Objetivo	Probar cada uno de los motores se encuentre funcionando correctamente
Precondición	● Puente H se encuentra alimentado con al menos 5v. ● Cada motor se encuentra conectado a los puertos Out 1 y Out 2, Out3 y Out 4, respectivamente.
Entradas	Alimentar In1 y In2, In3 y In4 para cada motor, intercambiando las polaridades.
Resultados esperados	1. Motor A y Motor B se mueven al recibir una alimentación a sus respectivas entradas. 2. Motor A y Motor B se mueven en rotación contraria al intercambiar sus respectivas entradas.

Resultados reales	Ambos motores se movieron hacia adelante y hacia atrás.
Resultado de prueba	Aprobada

ID	**CP003**
Objetivo	Probar cada led funciona correctamente
Precondición	● Se encuentra conectados los leds en la protoboard. ● Cada led tiene su propia resistencia de al menos 220(Ohms) ● Asegurarse de la polaridad del led se encuentre en la posición correcta. ● Se tiene fuente de alimentación de al menos 5v.
Entradas	Alimentar cada led uno por uno.
Resultados esperados	Al conectar cada led, este se prenderá acorde.
Resultados reales	Todos los leds se prendieron uno por uno.
Resultado de prueba	Aprobada

ID	**CP004**
Objetivo	Probar que el servomotor funciona correctamente
Precondición	● Se encuentra conectado en el protoboard. ● El pin naranja del servo se encuentra en un pin digital PWM en el Arduino. ● Asegurarse de que el servo se encuentre alimentado correctamente.
Entradas	Probar grados de movimiento 0°, 30°, 60°, 90°, 120°, 150° y 180° del servo.
Resultados esperados	1. Al recibir alimentación, el servo se coloca en 0°. 2. Se espera un segundo 3. Se incremente el grado en 30° hasta llegar a 180°
Resultados reales	El servo se fue moviendo en lapsos de un segundo 30°.

Resultado de prueba	Aprobada

Lenguaje de Programación

ID	**CP005**
Objetivo	Probar que en la declaración de variables se detecte correctamente su tipo, su scope y se le asigne una dirección de acuerdo a la estructura de nuestra memoria virtual.
Precondición	● No existen errores lexicos, sintácticos y semánticos.
Entradas	Se crean una variable entera, una flotante, una char, y una bool
Resultados esperados	La variable se encuentra definida en la tabla de variables. Se genera el quad de manera correcta.
Resultados reales	Las variables se crearon correctamente en la tabla de variables con su ubicación de memoria correspondiente- int: 11001 | float: 12001 | bool: 13001 | char: 14001 Se generaron los quads con cada una de las variables.
Resultado de prueba	Aprovada

ID	**CP006**
Objetivo	Probar ciclos y decisiones. Los cuádruplos de saltos se están generando correctamente y las decisiones esperadas se realizan correctamente.
Precondición	● No existen errores lexicos, sintácticos y semánticos. ● Hay estatutos dentro del ciclo/decisiones
Entradas	Ciclo que da 3 vueltas, con una variable dentro del ciclo y una variable booleana fuera del ciclo, y existen estatutos de decisión dentro del ciclo que utilizan a la variable.
Resultados esperados	La generación de quads de salto son generados correctamente. La variable actualiza su valor 3 veces. En la tercera vuelta el estatuto de decisión es accesado y modifica la variable booleana.

Resultados reales	La variable dentro del ciclo imprime el valor correcto. La variable booleana fuera del ciclo presenta un valor distinto al original. Se generaron los saltos correctos en los quads de salto.
Resultado de prueba	Aprovada

ID	**CP007**
Objetivo	Probar recursión de funciones.
Precondición	● No existen errores lexicos, sintácticos y semánticos. ● Se tiene una llamada de función dentro de la misma función.
Entradas	Cinco llamadas recursivas de una función, una variable booleana y una variable de retorno.
Resultados esperados	Después de 5 iteraciones de llamadas recursivas se modifica el valor de la variable booleana, utilizando la variable de retorno para controlar las llamadas recursivas.
Resultados reales	Se modificó la variable booleana.
Resultado de prueba	Aprovada

ID	**CP008**
Objetivo	Probar que al declarar una variable dimensionada se detecte correctamente su tamaño y se pueda acceder a sus índices de manera estática y cíclica
Precondición	● No existen errores lexicos, sintácticos y semánticos. ● El acceso a la variable dimensionada se encuentra dentro del rango de la variable.
Entradas	Una variable dimensionada de 5 espacios, un ciclo que realiza 5 vueltas.
Resultados esperados	Las casillas de la variable dimensionada son modificadas una a una dentro del ciclo, asignandoles un valor diferente a null.
Resultados	Las casillas dentro de la variable dimensionada son diferentes a las originales

reales	(null)
Resultado de prueba	Aprovada

Proceso del Proyecto

Durante el desarrollo de este proyecto, se estuvo trabajando siguiendo una metodología ágil incremental. Semanalmente, se le extendía a la versión base una parte funcional con el contenido del siguiente entregable. Asimismo, se fue elaborando el armado del vehículo con Arduino con las diferentes partes necesarias para su funcionamiento de manera simultánea.

El desarrollo del lenguaje tomó lugar de la siguiente manera::

- Lexico/Sintaxis
- Semántica de Variables y expresiones
- Generacion de codigo expresiones aritméticas, estatutos secuenciales y condicionales, y funciones
- Manejo de memoria
- Maquina Virtual
- Recursión de funciones
- Manejo de arreglos

El ensamblado del vehículo tomó lugar de la siguiente manera:

- Montaje de microcontrolador y protoboard al chasis
- Montaje de puente H y motores al chasis
- Montaje de baterías en la parte inferior del chasis
- Montaje de Servo al chasis
- Elaboración de la pala recolectora para el servo
- Conexión de todos los componentes del vehiculo.

Reflexión de Arturo Rojas:

Durante este semestre llevamos a cabo la tarea de desarrollar un lenguaje de programación desde su gramática hasta su máquina virtual. Llevo 5 años cursando la carrera de ITC y una de las preguntas que constantemente me hacía era la de cómo podía ser posible que una persona como **Guido** (Python) fuera capaz de desarrollar algo tan complejo como un lenguaje de programación por sí solo.

Este proyecto me ayudó a dimensionar mejor el proceso de esto, ya que al dividir los entregables del proyecto por semana, cada domingo (o lunes) que se entregaba un avance podías ver la

funcionalidad que iba agarrando tu lenguaje. El producto final de nuestro proyecto claramente no se acerca mucho a la complejidad que ofrece Python, C++ entre otros pero aún así me sentí contento al ver como podíamos ejecutar código con sintaxis diseñada por nosotros.

Considero que este proyecto me ayudó también a trabajar de manera más consistente con mi lógica de codificación, ya que muchas de las trabas que salían durante el desarrollo era por inconsistencias de declaración de variables o manejo de stacks.

Reflexión de Diego Jiménez:

La verdad es que todo este proyecto sí fue un sube y baja de emociones. Al principio me encontraba un poco confundido pues los rumores que existen de la materia son bastante nombrados en la carrera. Pero una vez que nos juntamos y comenzamos a platicar, poco a poco me fui dando cuenta que no era algo inalcanzable como alguna vez imaginé. Había tenido la suerte de trabajar anteriormente con Arduino, y la verdad es que me siento muy agusto trabajando, y me llenó de alegría saber que como propuesta de este proyecto lo podía utilizar.

Una vez listos, fuimos administrando cómo es que lo haríamos una realidad, por que no estábamos tan convencidos al inicio. De ahí en adelante si fue un dolor de cabeza todo lo que se ha logrado, pero me llena de alegría ver a nuestra creacion tomar vida con una serie de instrucciones simples. Quizá y algun dia se pueda retomar este conocimiento y aplicarlo a otra área dentro de nuestra industria, por que la verdad me llena de emocion si ese será el caso.

Lenguaje

Nombre del Lenguaje

Optamos por nombrar a nuestro lenguaje **L2C-Bot**, con el significado de “Learn-to-Code: Bot” que lo hacen fácil de recordar.

Características principales

El L2C-Bot está basado en su utilidad para que sea sencillo de aprender, por lo que muchas de sus características están basadas en aquellas funcionalidades básicas que se encuentran en un lenguaje de desarrollo imperativo. Por estar pensado como una manera alternativa de utilizar el lenguaje de Arduino, todas las configuraciones del microcontrolador, no las realizará el usuario. Solamente se enfocará en darle la funcionalidad requerida por el mismo, además de las funcionalidades base.

Constará de una función main, donde estarán ubicadas principalmente las instrucciones, y es aquí donde el usuario trabajará la mayor parte del tiempo. También, las variables y funciones miembro deben de ser declaradas antes de utilizarse en main. Cada variable que se declare,

tiene que tener especificado su tipo sin obligar a ser inicializada; y en dado caso tendrá el valor de nulo. Sin embargo, si se es requerida una función reservada tendrá que obligatoriamente tener un valor de entrada.

Y como especificación clave, toda instrucción termina con un punto y coma (;).

Posibles errores

`>> ERROR: Memory overflow.`	Se han sobrepasado el espacio de memoria del módulo al asignar una nueva constante.
`>> ERROR: " + a + ope +  b + " is not a valid operation.`	Operación no aceptada por cubo semántico
`>> ERROR: dimension of array must be of type INT`	Se declara o llamo un arreglo con una expresión NO tipo int.
`>> ERROR: couldn't find called variable for len()`	Se llamo len con una variable no existente
`>> ERROR: index out of range`	Posición de arreglo fuera de su tamaño
`>> ERROR: Could not compile file`	Error léxico/sintáctico en el código a compilar.
`ERROR: Caracter ilegal`	Declaración errónea de ID
`>> ERROR: ID '" + x + "' already assigned to a parameter or variable`	ID duplicado
`>> ERROR : variable not declared or of not supported type.`	Variable en expresión no declarada o de tipo incorrecto.
`>> ERROR: Number of parameters don't match.`	Llamada a función con cantidad incorrecta de parámetros
`>> ERROR: Types of parameters don't match.`	Llamada a función con parámetros con tipos que no coinciden con la función
`ERROR: function " + calledFunc + " hasn't been declared`	Llamada a función de ID no declarado
`>> ERROR: delay() was called with a " + rtyp + " but only works with int values.`	Algunas funciones de arduino solo pueden recibir INT como parámetro.

Compilador

Desarrollo

Para el desarrollo del compilador se utilizaron dos laptops con el SO de Windows 10, y la distribución de linux Ubuntu 2.19. Como lenguaje de desarrollo se utilizó Python 3, haciendo uso de PLY, una implementación de Lex/Yacc diseñada para su uso en Python.

Analisis Lexico

Patrones de Construcción (ER)

id → [a-z] [a-zA-Z0-9]*
int → [0-9]+
float → [0-9]+ . [0-9]*
char → [a-z A-Z 0-9 ! ” # $ % & / ( ) ...]
string → [a-z A-Z 0-9 ! ” # $ % & / ( ) ...]+
bool → ‘true’ | ‘false’
arreglo de una dimensión que almacena uno de los tipos de datos listados

Tokens y Código asociado

ASSIGN	=
PLUS	+
MINUS	-
MULTI	*
DIVI	/
LPAREN	(
RPAREN	)
LBRACKET	[
RBRACKET	]
LCURLY	{

RCURLY	}
EQUALS	==
NOTEQUALS	<>
LESSTHAN	<
GREATERTHAN	>
SEMICOLON	;
COMMA	,
AND	&&
OR	||
NOT	!=
CTE_INT	[0-9]
CTE_CHAR	[a-zA-Z]
CTE_FLOAT	[0-9]+.[0-9]
CTE_BOOL	[true|false]

Análisis Sintáctico

Gramática Formal

```
program -> PROGRAM varsblock funcsblock main FIN SEMICOLON

main -> MAIN setmain LPAREN RPAREN LCURLY varsblock main1 block RCURLY

funcsblock -> funcs funcsblock | ε

funcs -> FUNCDEF type LPAREN paramsblock RPAREN LCURLY varsblock block RCURLY

varsblock -> vars varsblock | ε

vars -> VARDEF type ID array SEMICOLON

array -> LBRACKET express RBRACKET | ε

paramsblock -> params paramsblock | COMMA params paramsblock | ε
```

```
params -> type ID | ε

block -> statute SEMICOLON block | ε

statute -> cond
         | assign
         | call
         | cin
         | cout
         | delay
         | forward
         | backward
         | turnleft
         | turnright
         | servo
         | lights
         | distance
         | stop
         | while
         | return

cond -> IF LPAREN express RPAREN LCURLY block RCURLY else

else -> ELSE LCURLY block RCURLY else | ε

assign -> ID array ASSIGN express

call -> ID LPAREN paramcall RPAREN

paramcall -> express paramcall1 | ε

paramcall1 -> COMMA paramcall | ε

cin -> CIN cin1

cin1 -> cin2 | cin3

cin2 -> LPAREN ID cin3 RPAREN

cin3 -> LBRACKET CTE_INT RBRACKET

cout -> COUT LPAREN express RPAREN

delay -> DELAY LPAREN express RPAREN

forward -> FORWARD LPAREN express RPAREN

backward -> BACKWARD LPAREN express RPAREN

turnleft -> TURNLEFT LPAREN express RPAREN

turnright -> TURNRIGHT LPAREN express RPAREN
```

```
servo -> SERVO LPAREN express RPAREN

lights -> LIGHTS LPAREN express COMMA express RPAREN

distance -> DISTANCE LPAREN RPAREN

stop -> STOP LPAREN RPAREN

while -> WHILE LPAREN express RPAREN LCURLY block RCURLY

return -> RETURN LPAREN express RPAREN

len -> LEN LPAREN ID RPAREN

type -> INT
      | FLOAT
      | BOOL
      | CHAR
      | VOID

constant -> ID array
          | CTE_INT
          | CTE_FLOAT
          | CTE_CHAR
          | CTE_BOOL

array -> LBRACKET express RBRACKET | ε

express -> relational express1

express1 -> andor express | ε

andor -> AND | OR

relational -> exp relational1 | NOT

relational1 -> compare exp | empty

compare -> LESSTHAN
          | GREATERTHAN
          | EQUALS
          | NOTEQUALS

exp -> term exp1

exp1 -> plusminus exp | ε

plusminus -> PLUS | MINUS
term -> factor term1

term1 -> multidivi term | ε

multidivi -> MULTI | DIVI
```

```
factor -> LPAREN express RPAREN | constant | call | len
```

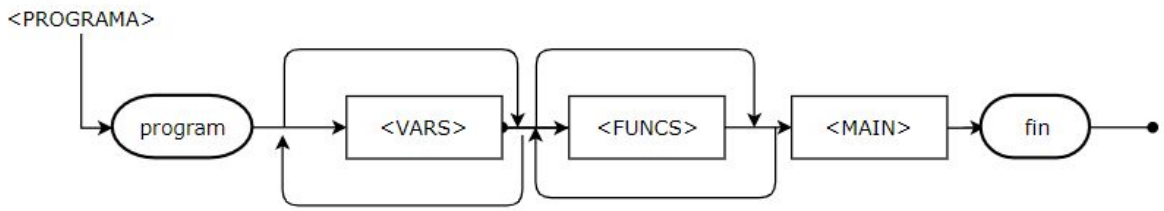
Generación de código intermedio y análisis semántico

Listado de instrucciones de cuádruplos

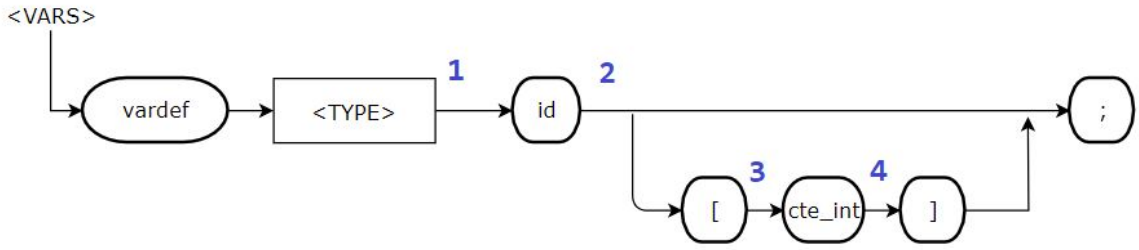
Instrucción	Código	Descripción
GOTO	0	Salto
GOTOF	1	Salto condicional
GOSUB	2	Salto de contexto
PARAM	3	Lectura de parámetro
ENDPROC	4	Fin de contexto
VER	5	Verificación de dimensión
K	6	Suma de K a tamaño
DIM	7	Posición + dir base
ERA	8	Nuevo espacio de memoria
+	9	Suma
-	10	Resta
*	11	Multiplicación
/	12	División
<	13	Menor que
>	14	Mayor que
==	15	Igual a
!=	27	No igual a
=	26	Asignación
PRINT	16	Impresión
LIGHTS	17	Arduino: LEDs

FORWARD	18	Arduino: Movimiento
BACKWARD	19	Arduino: Movimiento
TURNLEFT	20	Arduino: Movimiento
TURNRIGHT	21	Arduino: Movimiento
SERVO	22	Arduino: Posición de servo
DELAY	23	Arduino: Pausa
RETURN	24	Retorno de función
CIN	25	Lectura de teclado

Listado de diagramas con puntos neurálgicos

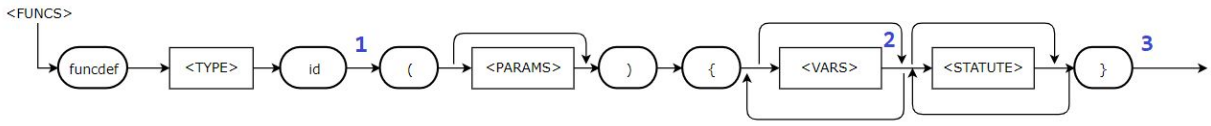


Estructura general del programa. Se declaran las variables locales, luego las funciones y al final la función principal MAIN.



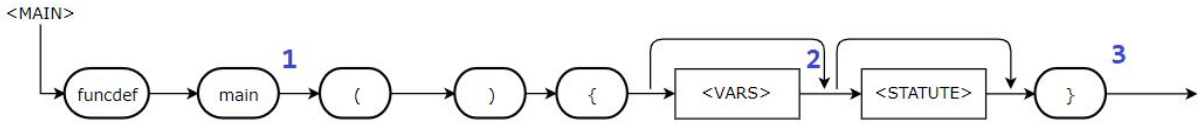
Definición de variables:

1. Se guarda el tipo leído, para usarse después en la declaración de variable.
2. Se busca el id. Si no existe, se declara con el tipo guardado, si existe regresa error.
3. Se detecta que es una variable dimensionada.
4. Se añade a la variable declarada su atributo ‘dim’ con el tamaño declarado.

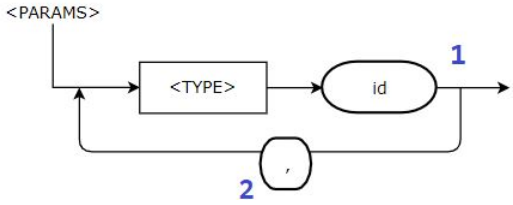


Definición de funciones:

1. Se declara el ID como el scope actual y se crea nueva entrada en tabla de func.
2. Si se terminó de leer variables, se asigna el ERA de la función y su cantidad de params.
3. Al terminar de leer la función, se hace limpieza de memoria (era, varstable, direcciones).

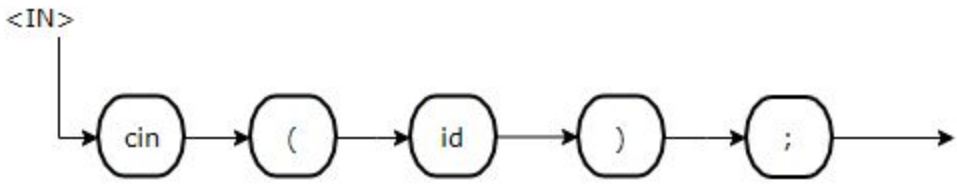


La declaración de main sigue la misma semántica que una función normal, con la diferencia de que su scope o ID está predeterminado a siempre llamarse ‘main’ y esta no contiene parámetros.



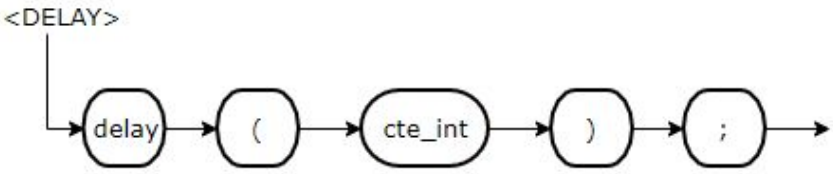
Parámetros:

1. Un parámetro se declara como variable en la tabla de variables, con su ID y tipo.
2. Se incrementa el contador de parámetros.

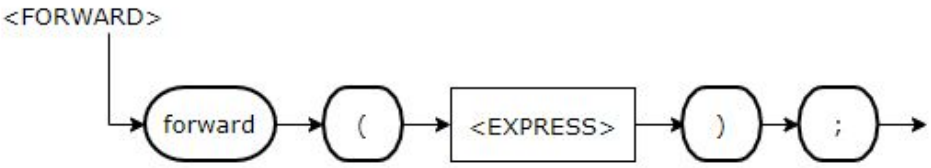


Estructura para tener input en el transcurso de la ejecución, se especifica la palabra reservada *cin* seguido del id de la variable que almacenará el valor proporcionado.

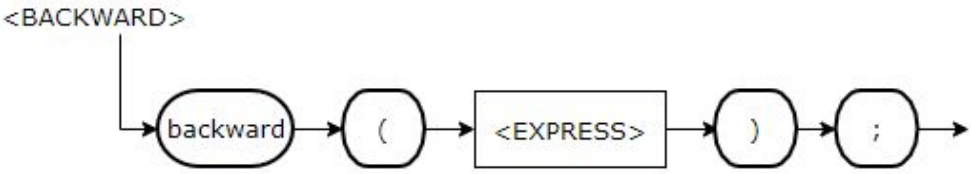
Estructura para desplegar información de cualquier tipo de dato, se proporciona la expresión y esta se muestra en la terminal. Adicionalmente se prende la luz led azul, cada vez que se muestra algo, haciendo que sea más notorio.



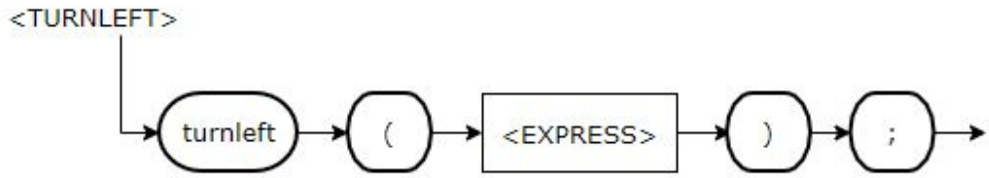
Estructura de espera, que recibe un número entero y detiene la momentáneamente la ejecución, en el número determinado de segundos.



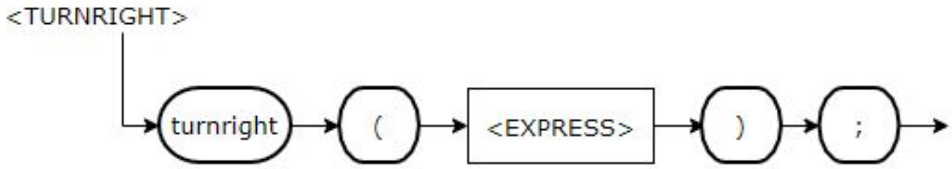
Utilizado para el movimiento del carro, se recibe una expresión entera y proporciona una rotación a los motores del vehículo con un pulso de 5v por una duración especificada en segundos, con dirección hacia adelante.



Utilizado para el movimiento del carro, se recibe una expresión entera y proporciona una rotación a los motores del vehículo con un pulso de 5v por una duración especificada en segundos, con dirección hacia atrás.

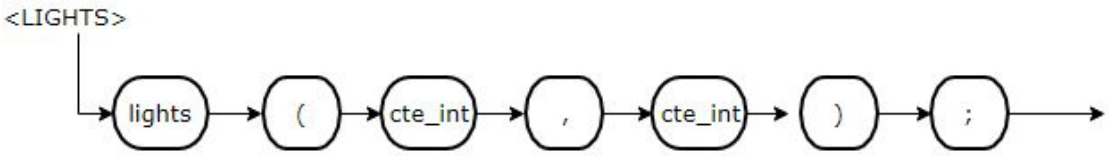


Utilizado para el movimiento del carro, se recibe una expresión entera y proporciona una rotación a los motores del vehículo en sentidos opuestos, con un pulso de 5v por una duración especificada en segundos, con dirección hacia la izquierda.



Utilizado para el movimiento del carro, se recibe una expresión entera y proporciona una rotación a los motores del vehículo en sentidos opuestos, con un pulso de 5v por una duración especificada en segundos, con dirección hacia la derecha.

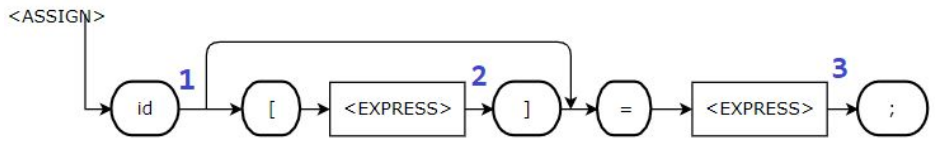
Utilizado para el movimiento del servomotor, recibe una expresión entera y le proporciona un grado de rotación, entre 0 y 180.



Utilizado para modificar las propiedades de las diferentes luces led del vehículo.

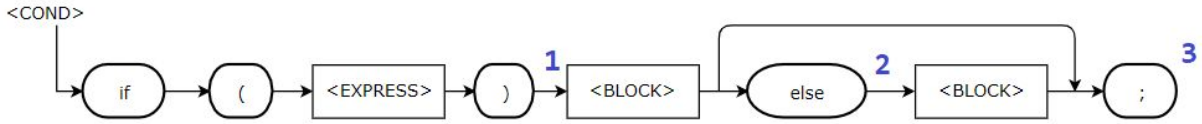
La primer constante va de un valor entre 1, 2 y 3, donde se elige (1 : para led rojo, 2 : para led amarillo, 3 : para led verde)
La segunda constante también se elige una constante entre 1, 2 y 3, donde se elige (1 :

prendido, 2 : apagado, 3 : parpadeando)



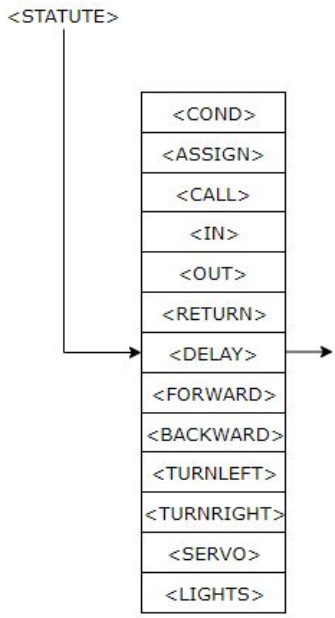
Asignación

1. Se detecta el ID y se busca en las tablas de variables.
2. Se resuelve la expresión, y se generan los cuádruplos de acceso a variable dimensionada.
3. Se obtiene el resultado de la expresión con un pop y se genera el cuádruplo de asignación del id y el valor detectado.

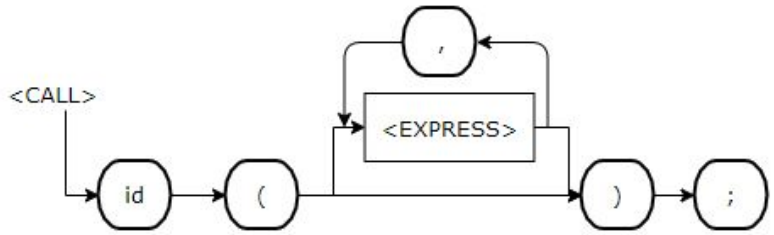


Condición:

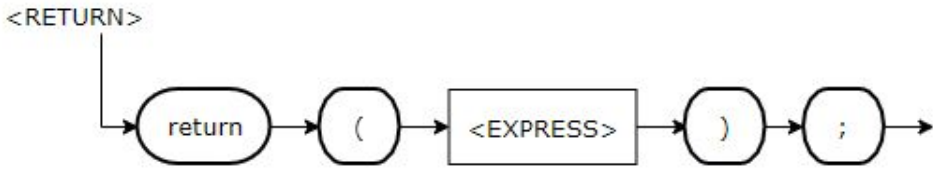
1. Se genera el GOTOF vacío.
2. Se genera el GOTO vacío.
3. Se actualiza ell GOTOF y el GOTO con el # de cuádruplo en el que este termina.



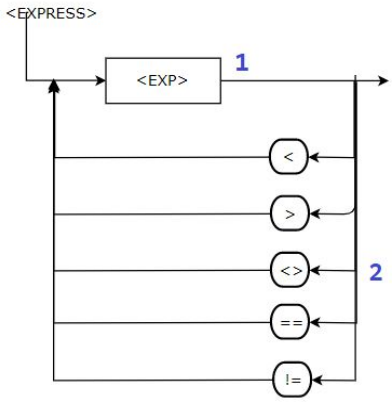
Statute nos permite identificar cuál de las diferentes funciones reservadas es la que se está tratando de acceder.



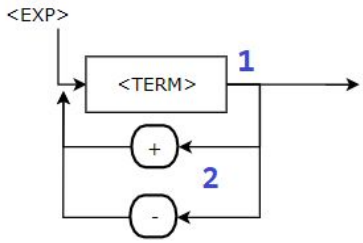
Call es la definición de la estructura de cada una de las funciones, donde id es el nombre de la función, y dentro de los paréntesis es donde se van aglutinando los parámetros necesarios dependiendo de dicha función.



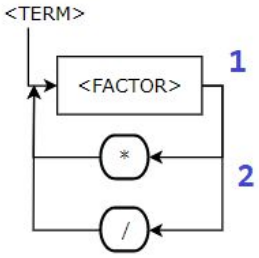
Return nos proporciona el valor de retorno de una función, con el valor de la expresión dentro del paréntesis.



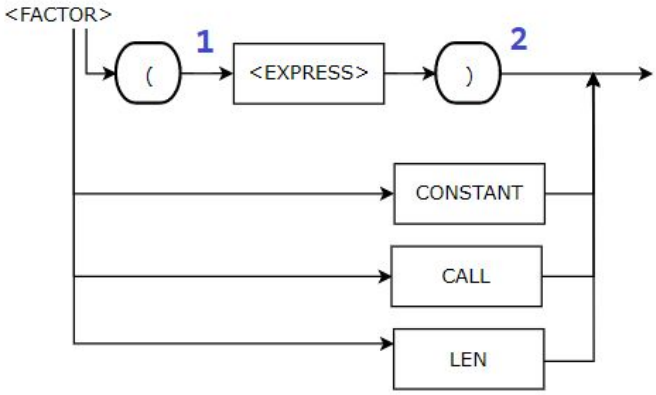
1. Si el tope de la pila contiene un relacional, se genera su cuádruplo y se expulsa.
2. Se mete a la pila de operandos el relacional detectado.



1. Si el tope de operandos contiene una suma/resta, se genera su cuádruplo y se expulsa.
2. Se agrega a la pila de operandos el suma/resta detectado.



1. Si el tope de operandos contiene una multiplicación/división, se genera su cuádruplo y se expulsa.
2. Se agrega a la pila de operandos el multiplicación/división detectado.



1. Se agrega un piso falso a la pila de operandos
2. Se remueve el piso falso de la pila de operandos

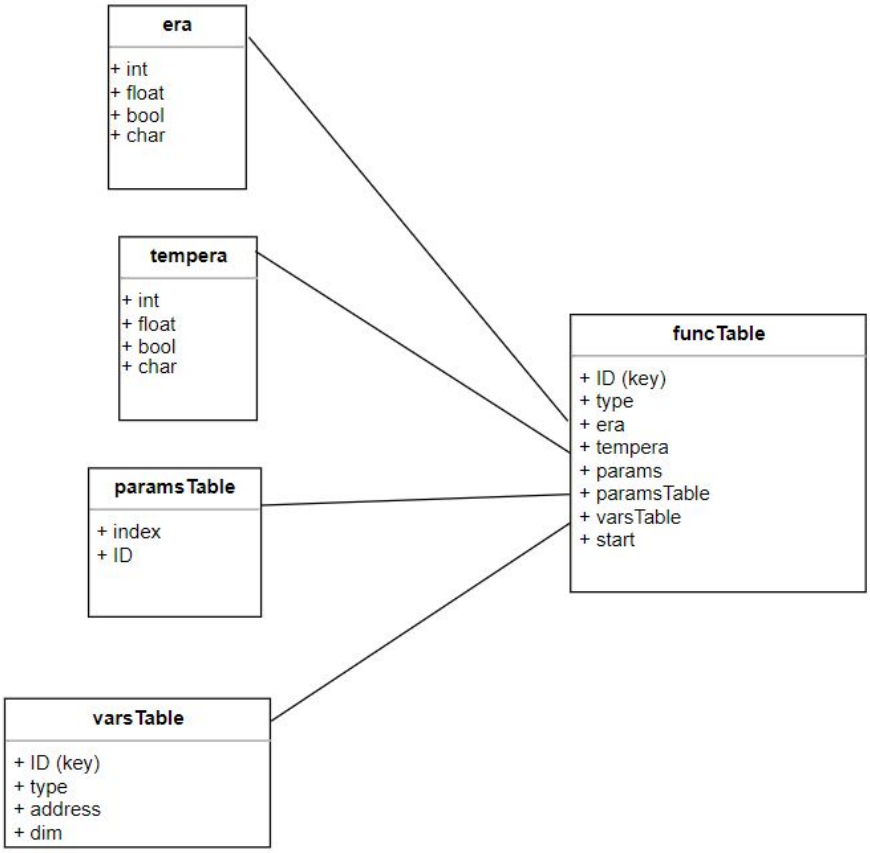
Diccionario de consideraciones semánticas

```
semCube = {'int' : { 'int' : {  '+' : 'int',
                                '-' : 'int',
                                '/' : 'float',
                                '*' : 'int',
                                '<' : 'bool',
                                '>' : 'bool',
                                '==': 'bool',
                                '!=': 'bool',
                                '=' : 'int'
                             },
                   'float' : { '+' : 'float',
                               '-' : 'float',
                               '/' : 'float',
                               '*' : 'float',
                               '<' : 'bool',
                               '>' : 'bool',
                               '==': 'bool',
                               '!=': 'bool',
                               '=' : 'int'
                             }
                   },
           'float' :{ 'int' : {'+' : 'float',
                               '-' : 'float',
                               '/' : 'float',
                               '*' : 'float',
                               '<' : 'bool',
                               '>' : 'bool',
                               '==': 'bool',
                               '!=': 'bool',
                               '=' : 'float'
                               },
                      'float':{'+': 'float',
                               '-': 'float',
                               '/': 'float',
                               '*': 'float',
                               '<': 'bool',
                               '>': 'bool',
                               '==': 'bool',
                               '!=': 'bool',
                               '=': 'float'
                               }
                   },
           'bool' : {'bool' : {'&&': 'bool',
                               '||': 'bool',
                               '=' : 'bool',
                               '==': 'bool',
                               '!=': 'bool'
```

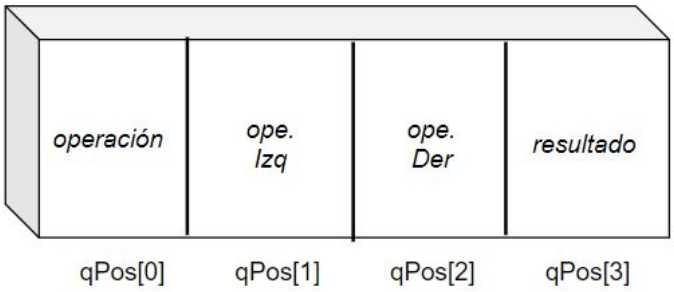
```
                            }
                },
        'char' : {'char' : { '==' : 'bool',
                             '!=' : 'bool',
                             '='  : 'char'
                            }
                }
    }
```

Administración de memoria en compilación

Tabla de funciones y Tabla de Variables



Cuádruplos



La memoria se divide en módulos de globales, locales, temporales y constantes y cada uno de estos módulos se divide en submódulos de los 4 tipos de constantes que permite el lenguaje.

Estos submódulos tienen la capacidad de contener 1,000 elementos cada uno.

```
virMem = {'global': {'int'   : [],
                     'float' : [],
                     'bool'  : [],
                     'char'  : []
                     },
          'local'  : {'int'   : [],
                     'float' : [],
                     'bool'  : [],
                     'char'  : []
                     },
          'temp'   : {'int'   : [],
                     'float' : [],
                     'bool'  : [],
                     'char'  : []
                     },
          'const'  : {'int'   : [],
                     'float' : [],
                     'bool'  : [],
                     'char'  : []
                     }
          }
```

Cada función almacena un diccionario de ERA y TEMPERA donde se guarda el espacio de memoria que ocupa la función en ejecución. Estos diccionarios se van incrementando por cada variable declarada en era o cada temporal utilizado en tempera.

Todos los módulos inician vacíos y en ejecución se les irán agregando espacios de acuerdo al ERA de cada función llamada.

```
#Función para limpiar todos los índices al
empezar a leer una nueva función.
def memClear():
    global dirMem
    global iParams
    global era
    global tempera
    global funcTable

    iParams = 0

    era =  {'int' : 0,
    dirMem['local']['int'] = 21001
    dirMem['local']['float'] = 22001
    dirMem['local']['bool'] = 23001
    dirMem['local']['char'] = 24001
    dirMem['temp']['int'] = 31001
    dirMem['temp']['float'] = 32001
    dirMem['temp']['bool'] = 33001
    dirMem['temp']['char'] = 34001

    virMem['local']['int'] = []
    virMem['local']['float'] = []
    virMem['local']['bool'] = []
```

```
        'float' : 0,
        'bool' : 0,
        'char' : 0}

    tempera =  {'int' : 0,
        'float' : 0,
        'bool' : 0,
        'char' : 0}
```

```
    virMem['local']['char'] = []
    virMem['temp']['int'] = []
    virMem['temp']['float'] = []
    virMem['temp']['bool'] = []
    virMem['temp']['char'] = []
```

Al finalizar la lectura de una función, memClear() reinicia los valores de direcciones durante compilación para poder empezar con los índices desde 0 con la nueva función que se lea.

```
#Generación de un nuevo cúadruplo. Agrega el cuádruplo al stack e incrementa el
contador.
def newQuad(ope, a, b, res):
    global iQuads
    quads.append([ope, a, b, res])
    iQuads = iQuads + 1

#PN: Cuádruplo de GOTOIF para condicional
def p_gotoif(p):
    'gotoif : empty'
    global iQuads
    if ptypes[-1] == 'bool' :
        x = pconsts.pop()
        ptypes.pop()
        pjumps.append(iQuads)
        newQuad(op['GOTOF'], x, '', '')
```

Máquina Virtual

Desarrollo

Para el desarrollo de la máquina virtual se utilizaron dos laptops con el SO de Windows 10, y la distribución de linux Ubuntu 2.19. Como lenguaje de desarrollo se utilizó Python 3 haciendo uso de pyFirmata, la cual es una interpretación en python del Protocolo Firmata. Lo cual permite comunicarse con el software en una computadora host. Esto permite escribir firmware personalizado sin tener que crear nuestro propio protocolo y objetos para el entorno de programación Arduino.

En otras palabras, pyFirmata nos permite utilizar comandos hechos en python para que sean interpretados por el microcontrolador de Arduino directamente.

Arquitectura - Administración de Memoria

virMem	Global	Int	->	11,001
		Float	->	12,001
		Bool	->	13,000
		Char	->	14,000
	Local	Int	->	21,001
		Float	->	22,001
		Bool	->	23,001
		Char	->	24,001
	Temporal	Int	->	31,001
		Float	->	32,001
		Bool	->	33,001
		Char	->	34,001
	Constant	Int	->	41,001
		Float	->	42,001
		Bool	->	43,001
		Char	->	44,001

```
#Estructura de declaración de variable
def p_vars(p):
    'vars : VARDEF type ID dimvar SEMICOLON'
    global dirMem
    global virMem
    global era
    x = p[3]
    erasize = 1
    #Se empieza a clasificar la variable
detectada.
    if scope == 'global' :
        address = dirMem['global'][tempType]
        dirMem['global'][tempType] = address
+ 1
```

```
        checkOverflow('global', tempType)
        virMem['global'][tempType].append(x)
    else :
        address = dirMem['local'][tempType]
        dirMem['local'][tempType] = address +
1
        checkOverflow('local', tempType)
   ...
```

Un ejemplo de asignación de memoria durante compilación. Durante declaracion de variables se accede al índice de memoria local o global y dependiendo del tipo de variable se consigue su nueva dirección, después se añade +1 al índice para cuando llegue una variable nueva.

Si la variable es global, se le hace append a la memoria virtual ya que las globales permanecen constantes durante ejecución.

```
#Escritura de memoria
#Recibe como parámetro un valor y una
dirección.
#Almacena en la dirección el valor recibido.
def memWrite(val, dir):
    if type(dir) == str:
        value =int(dir.replace('*', ''))
        dir = memRead(value)
    scopeFloor = 0
    dir = str(dir)
    pos = int(dir[2] + dir[3] + dir[4])-1
    if dir[0] == '1':
        scope = 'global'
    elif dir[0] == '2':
        scope = 'local'
    elif dir[0] =='3':
        scope = 'temp'
    elif dir[0] == '4':
        scope = 'const'

    if dir[1] == '1':
        typ = 'int'
        if scope == 'local' :
            scopeFloor = era[typ]
        virMem[scope][typ][pos + scopeFloor]
= int(val)
    elif dir[1] == '2':
        typ = 'float'
        if scope == 'local' :
            scopeFloor = era[typ]
        virMem[scope][typ][pos + scopeFloor]
= float(val)
    elif dir[1] == '3':
        typ = 'bool'
        if scope == 'local' :
            scopeFloor = era[typ]
        virMem[scope][typ][pos + scopeFloor]
= bool(val)
    elif dir[1] == '4':
        typ = 'char'
        if scope == 'local' :
            scopeFloor = era[typ]
        virMem[scope][typ][pos + scopeFloor]
= str(val)

    return val
```

```
  def memRead(dir):
    if type(dir) == str:
        value =int(dir.replace('*', ''))
        dir = memRead(value)

    scopeFloor = 0
    dir = str(dir)
    pos = int(dir[2] + dir[3] + dir[4])-1

    if dir[0] == '1':
        scope = 'global'
    elif dir[0] == '2':
        scope = 'local'
    elif dir[0] =='3':
        scope = 'temp'
    elif dir[0] == '4':
        scope = 'const'

    if dir[1] == '1':
        typ = 'int'
        if scope == 'local' :
            scopeFloor = era[typ]
        val = virMem[scope][typ][pos +
scopeFloor]
        return int(val)
    elif dir[1] == '2':
        typ = 'float'
        if scope == 'local' :
            scopeFloor = era[typ]
        val = virMem[scope][typ][pos +
scopeFloor]
        return float(val)
    elif dir[1] == '3':
        typ = 'bool'
        if scope == 'local' :
            scopeFloor = era[typ]
        val = virMem[scope][typ][pos +
scopeFloor]
        return bool(val)
    elif dir[1] == '4':
        typ = 'char'
        if scope == 'local' :
            scopeFloor = era[typ]
```

	`val = virMem[scope][typ][pos + scopeFloor]`
	`return str(val)`
	`else :`
	`sys.exit("ERROR: Tried to access unassigned memory space")`

memwrite(val, dir) recibe como parámetro un valor y una dirección. Primero convierte la dirección en string para poder decodificarla.

Por ejemplo en una dirección 4 2 0 0 5 :

El 4 nos indica que es una variable constante

El 2 nos indica que es de tipo flotante

Y el 0 0 5 nos indica su posición en el módulo.

Entonces ya se puede asignar el valor dentro de la casilla correcta.

scopeFloor funciona para cuando se trabaja durante un cambio de contexto, y se aplica un piso que esconde las variables que se estaban utilizando durante la función anterior. Entonces a la dirección que se espera acceders, se le agrega el scopeFloor para representar el cambio de contexto.

memRead(dir) funciona de manera muy similar con la diferencia de que en vez de asignar un valor en la casilla identificada, extrae lo que esté almacenado ahí y lo manda al return.

Pruebas

Factorial

Codigo

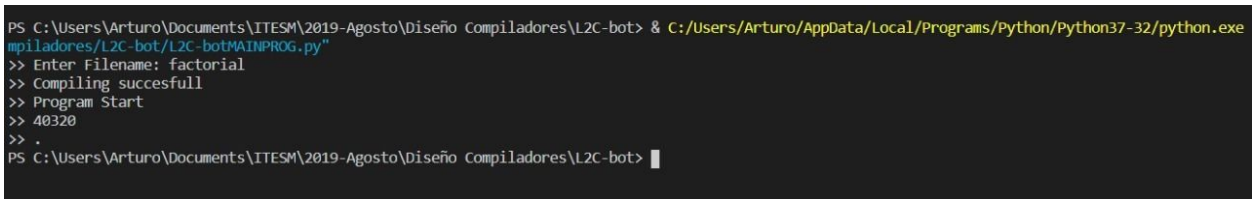
```
program

vardef int i;

funcdef int uno(int a){
	if (a == 1 ) {
		return(a);
	}else{
		return(uno(a-1)*a);
	};
}

main(){
	vardef int z;
	z = 8;
	cout(uno(z));
	}
fin;
```

Quads

```
[0, '', '', 15]
[15, 21001, 41001, 33001]
[1, 33001, '', 6]
[24, 21001, '', 11002]
[4, '', '', '']
[0, '', '', 14]
[8, 'uno', '', [1, 0, 0, 0]]
[10, 21001, 41001, 31001]
[3, 31001, '', 21001]
[2, 'uno', '', 1]
[26, 11002, '', 31002]
[11, 31002, 21001, 31003]
[24, 31003, '', 11002]
[4, '', '', '']
[4, '', '', '']
[26, 41002, '', 21001]
[8, 'uno', '', [1, 0, 0, 0]]
[3, 21001, '', 21001]
[2, 'uno', '', 1]
[26, 11002, '', 31001]
[16, 31001, '', '']
```

```
{'int': ['i', 'uno'], 'float': [], 'bool': [], 'char': []}
{'int': ['1', '8'], 'float': [], 'bool': [], 'char': []}
```

Resultado



Factorial Ciclico

Codigo

```
program

main(){
	vardef int i;
	vardef int fact;
	fact = 1;
	i = 7;
	while (i > 0){
		fact = fact * i;
		i = i - 1;
	};
	cout(fact);
}

fin;
```

Quads

```
[0, '', '', 1]
[26, 41001, '', 21002]
[26, 41002, '', 21001]
[14, 21001, 41003, 33001]
[1, 33001, '', 10]
[11, 21002, 21001, 31001]
[26, 31001, '', 21002]
[10, 21001, 41001, 31002]
[26, 31002, '', 21001]
[0, '', '', 3]
[16, 21002, '', '']
{'int': [], 'float': [], 'bool': [], 'char': []}
```

```
{'int': ['1', '7', '0'], 'float': [], 'bool': [], 'char': []}
```

Resultado

```
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot> & C:/Users/Arturo/AppData/Local/Programs/Python/Python37-32/python.exe
mpiladores/L2C-bot/L2C-botMAINPROG.py"
>> Enter Filename: factorialciclico
>> Compiling succesfull
>> Program Start
>> 5040
>> .
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot>
```

Fibonacci

Codigo

```
program

vardef int i;

funcdef int fibo(int n){
	vardef int a;
	vardef int b;
	vardef int r;
	if (n < 2) {
		return(n);
	}else{
		a = fibo(n-1);
		b = fibo(n-2);
		r = a+b;
		return(r);
	};
}

main(){
	cout(fibo(10));
	}
fin;
```

Quads

```
[0, '', '', 23]
[13, 21001, 41001, 33001]
[1, 33001, '', 6]
[24, 21001, '', 11002]
[4, '', '', '']
[0, '', '', 22]
[8, 'fibo', '', [4, 0, 0, 0]]
[10, 21001, 41002, 31001]
[3, 31001, '', 21001]
[2, 'fibo', '', 1]
[26, 11002, '', 31002]
[26, 31002, '', 21002]
[8, 'fibo', '', [4, 0, 0, 0]]
[10, 21001, 41001, 31003]
[3, 31003, '', 21001]
[2, 'fibo', '', 1]
[26, 11002, '', 31004]
[26, 31004, '', 21003]
[9, 21002, 21003, 31005]
[26, 31005, '', 21004]
[24, 21004, '', 11002]
[4, '', '', '']
[4, '', '', '']
[8, 'fibo', '', [4, 0, 0, 0]]
[3, 41003, '', 21001]
[2, 'fibo', '', 1]
[26, 11002, '', 31001]
[16, 31001, '', '']
{'int': ['i', 'fibo'], 'float': [], 'bool': [], 'char': []}
{'int': ['2', '1', '10'], 'float': [], 'bool': [], 'char': []}
```

Resultado

```
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot> & C:/Users/Arturo/AppData/Local/Programs/Python/Python37-32/python.exe
mpiladores/L2C-bot/L2C-botMAINPROG.py"
>> Enter Filename: factorialciclico
>> Compiling succesfull
>> Program Start
>> 5040
>> .
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot>
```

Fibonacci Ciclico

Codigo

```
program

funcdef int fibo(int n){
	vardef int a;
	vardef int b;
	vardef int c;
	vardef int i;

	if( n < 3){
		return(1);
	};

	a = 1;
	b = 1;
	i = 2;
	while(i < n){
		c = a + b;
		a = b;
		b = c;
		i = i + 1;
	};
	return(c);
}

main(){
	cout(fibo(10));
	}
fin;
```

Quads

```
[0, '', '', 20]
[13, 21001, 41001, 33001]
[1, 33001, '', 5]
[24, 41002, '', 11001]
[4, '', '', '']
[26, 41002, '', 21002]
[26, 41002, '', 21003]
[26, 41003, '', 21005]
[13, 21005, 21001, 33002]
[1, 33002, '', 17]
[9, 21002, 21003, 31001]
[26, 31001, '', 21004]
[26, 21003, '', 21002]
```

```
[26, 21004, '', 21003]
[9, 21005, 41002, 31002]
[26, 31002, '', 21005]
[0, '', '', 8]
[24, 21004, '', 11001]
[4, '', '', '']
[4, '', '', '']
[8, 'fibo', '', [5, 0, 0, 0]]
[3, 41004, '', 21001]
[2, 'fibo', '', 1]
[26, 11001, '', 31001]
[16, 31001, '', '']
{'int': ['fibo'], 'float': [], 'bool': [], 'char': []}
{'int': ['3', '1', '2', '10'], 'float': [], 'bool': [], 'char': []}
```

Resultado

```
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot> & C:/Users/Arturo/AppData/Local/Programs/Python/Python37-32/python.exe
mpiladores/L2C-bot/L2C-botMAINPROG.py"
>> Enter Filename: fibonaccicicliclico
>> Compiling succesfull
>> Program Start
>> 55
>> .
PS C:\Users\Arturo\Documents\ITESM\2019-Agosto\Diseño Compiladores\L2C-bot>
```

Troubleshoot

Puerto (Socket)

De los errores más comunes, se encuentran la asignación del puerto USB de la conexión al microcontrolador de Arduino. Es sumamente importante colocarle a la máquina virtual el valor del socket correcto, de lo contrario el programa no ejecuta. Donde tiene que especificarse el puerto correcto, dependiendo del sistema operativo. En el caso de Windows se utiliza comúnmente el puerto **‘COM3’** y en el caso de linux en su distro de Ubuntu utiliza comúnmente el puerto **‘/dev/ttyACM0’.**
Para revisar el puerto que el sistema operativo le asignó al microcontrolador es necesario abrir el Arduino IDE y revisar dentro de la sección de Tools > Port, el puerto que aparece a seleccionar.

Llamada de función con 2 o más funciones como argumento de parámetro.

Cuando se requiere utilizar una función que recibe como parámetro, el resultado de 2 o más funciones, el valor de retorno es inesperado. Los quads que se generan a partir de ese tipo de función, se generan erróneamente. De manera que es difícil encontrar en qué sección es en donde está ocurriendo el problema. Ya sea por que se esta yendo por algún camino distinto, o que el orden en el cual se generan pueden variar, por tanto se optó por reorganizar su estructura, y solo limitar a poder utilizar solamente una función como parámetro de otra.