

Compilador 'Baky'



Daniel Treviño Quiroga
A01283132



José Ernesto García Torales
A01365919



Diseño de Compiladores
21/11/2022

Índice

I : Descripción y Documentación Técnica del Proyecto	4
DESCRIPCIÓN DEL PROYECTO	5
Propósito y alcance del proyecto	6
Propósito	6
Alcance	6
Análisis de requerimientos y descripción de los principales test cases	6
Análisis de requerimientos	6
Test cases	8
Descripción del proceso de desarrollo	10
Bitácora general	10
Lista de commits	11
Reflexión	13
DESCRIPCIÓN DEL LENGUAJE	15
Nombre del lenguaje	16
Descripción de las principales características	16
Listado de errores que pueden ocurrir	16
DESCRIPCIÓN DEL COMPILADOR	19
Descripción de herramientas utilizadas	20
Descripción análisis léxico	20
Patrones de construcción	20
Tokens	20
Descripción análisis sintáctico	22
Gramática Formal	22
Descripción de generación de código intermedio y análisis semántico	26
Códigos de operación	26
Direcciones virtuales	27
Diagramas de sintaxis	28
Descripción de las acciones semánticas y de generación de código	33
Tabla de consideraciones semánticas	37
Descripción de la administración de memoria en compilación	39
DESCRIPCIÓN DE LA MÁQUINA VIRTUAL	42
Descripción de la administración de memoria en ejecución	43
Especificación gráfica y justificación	43
Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución).	44
PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE	45
Pruebas	46
Factorial Cíclico	46
Factorial Recursivo	47

Fibonacci Cíclico	49
Fibonacci Recursivo	50
Bubble Sort	52
Find	55
Multiplicación de Matrices	58
Compilación y Ejecución en el IDE	65
DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO	67
Repositorio	68
Descripción de módulos	68
Extractos de código	69
II : Manual de Usuario	72
QUICK REFERENCE MANUAL	73
Ambiente de ejecución	74
Ejecutar Baky	76
IDE de Baky	77
Código en Baky	78
Ejemplo de Código y Videos Demo	82

I : Descripción y Documentación Técnica del Proyecto

DESCRIPCIÓN DEL PROYECTO

Propósito y alcance del proyecto

Propósito

El propósito del presente proyecto es crear un lenguaje de programación nuevo y su respectivo compilador con la finalidad de poner en práctica los conocimientos adquiridos durante nuestra carrera como Ingenieros en Tecnologías Computacionales, así como también reforzar todos los conceptos vistos en la clase de Diseño de Compiladores.

De igual forma, se tiene planeado crear un entorno de desarrollo integrado (IDE) para dispositivos móviles en el que se pueda crear, editar, compilar y correr programas escritos en este nuevo lenguaje de programación.

Alcance

La visión de este nuevo lenguaje es que sea básico y sencillo de utilizar, con la intención de poder aprender a programar desde un dispositivo móvil. Se busca que en el IDE se pueda programar y que con un solo botón el programa se compile y ejecute de manera nativa, dando los resultados en la misma aplicación.

Es importante mencionar que el lenguaje contará con todos los elementos básicos de cualquier lenguaje de programación, como lo son operaciones aritméticas, estatutos de interacción (input/output), ciclos, decisiones, funciones, arreglos y matrices, dejando la opción a futuro de poder complementar y/o mejorar el presente proyecto.

Análisis de requerimientos y descripción de los principales test cases

Análisis de requerimientos

Requerimientos funcionales

Requerimiento	Descripción
RF1: Declaración de un programa	El lenguaje debe permitirte declarar un programa con sus respectivas funciones y estatutos.
RF2: Declaración y uso de funciones	El lenguaje debe soportar la declaración de funciones parametrizadas y su uso dentro de los estatutos.
RF3: Declaración y uso de variables globales	El lenguaje debe soportar la declaración y uso de variables globales.
RF4: Declaración y uso de variables locales	El lenguaje debe soportar la declaración y uso de variables locales.

RF5: Declaración y uso de constantes	El lenguaje debe soportar la declaración y uso de constantes (string,boolean,char,int,double).
RF6: Expresiones aritméticas con variables/constantes	El lenguaje debe soportar el uso de expresiones aritméticas.
RF7: Expresiones lógicas con variables/constantes	El lenguaje debe soportar el uso de expresiones lógicas.
RF8: Expresiones relacionales con variables/constantes	El lenguaje debe soportar el uso de expresiones relacionales.
RF9: Entradas/Salidas	El lenguaje debe soportar el uso de estatutos de lectura y escritura.
RF10: Ciclos/Condicionales	El lenguaje debe soportar el uso de estatutos cíclicos y condicionales.
RF11: Arreglos/Matrices	El lenguaje debe soportar el uso de elementos estructurados como arreglos y matrices.
RF12: Recursividad	El lenguaje debe soportar recursividad en sus funciones.
RF13: Asignación	El lenguaje debe soportar el uso de estatutos de asignación.
RF14: Errores léxicos	El compilador debe detectar el uso de tokens no definidos en el lenguaje.
RF15: Errores sintácticos	El compilador debe detectar errores sintácticos del programa.
RF16: Errores semánticos	El compilador debe detectar errores semánticos del programa.
RF17: Errores en ejecución	La máquina virtual debe detectar errores de ejecución.
RF18: Cuádruplos	El compilador debe generar los cuádruplos correspondientes del programa.
RF19: Ejecución de cuádruplos	La máquina virtual debe ejecutar todos los cuádruplos generados.
RF20: Memoria virtual	La memoria virtual contiene los scopes locales y globales de forma dinámica.
RF21: Entorno de desarrollo integrado (IDE)	El IDE debe soportar la creación, compilación y ejecución de un nuevo programa.

Requerimientos no funcionales

Requerimiento	Descripción
RNF1: Compilador óptimo	El compilador debe de optimizar la memoria y el tiempo de ejecución.
RNF2: IDE intuitivo	El IDE debe ser muy intuitivo para el usuario.
RNF3: Lenguaje sencillo	El lenguaje debe ser sencillo de entender y escribir.
RNF4: Uso de lexer/parser	El compilador debe de usar una herramienta para el análisis léxico/sintáctico.

Test cases

Todos los test cases listados abajo se codificaron en Baky y se encuentran en la carpeta test_cases dentro del repositorio del proyecto.

No.	Nombre	Descripción
1	Error_Grammar.bky	Prueba detectar un token que no soporta Baky.
2	Error_Syntax.bky	Prueba detectar un error de sintaxis en el programa.
3	Grammar_Syntax.bky	Prueba un programa correcto para el análisis léxico/sintáctico.
4	Error_DupGlobalId.bky	Prueba detectar una variable global duplicada.
5	Error_DupLocalId.bky	Prueba detectar una variable local duplicada.
6	SameId.bky	Prueba aceptar un mismo id para una variable global y una local.
7	Error_DupFunc.bky	Prueba detectar un id de una función duplicada.
8	Error_VoidReturn.bky	Prueba que una función void no tenga un return.
9	Error_NoReturn.bky	Prueba que una función con tipo tenga un return.
10	Error_BadReturn.bky	Prueba que el return sea del tipo esperado.
11	Error_ParamsType.bky	Prueba que el tipo de parámetros sea el correcto.
12	Error_NumParams.bky	Prueba que el número de parámetros sea el correcto.

13	Error_VoidFunction.bky	Prueba que una función void no se pueda utilizar en una expresión.
14	CorrectScope.bky	Prueba que si existe una variable global y una local con el mismo id, las operaciones se ejecuten sobre la local.
15	Error_ArrayDec.bky	Prueba que no se pueda declarar un arreglo con dimensiones negativas.
16	Error_ArrayIndex.bky	Prueba que el id de un arreglo tenga el index al utilizarlo.
17	Error_MatrixIndex.bky	Prueba que el id de una matriz tenga los dos indexes al utilizarlo.
18	Error_NoIndex.bky	Prueba que una variable simple no acepte tener indexes.
19	Error_UndecVar.bky	Prueba que una variable siempre exista antes de usarla.
20	Error_UndecFunc.bky	Prueba que una función siempre exista antes de usarla.
21	Error_TooManyVar.bky	Prueba el error que ocurre al intentar utilizar muchas variables.
22	Error_WhileEval.bky	Prueba que la evaluación del while sea booleana.
23	Error_IfEval.bky	Prueba que la evaluación del if sea booleana.
24	Error_ForStart.bky	Prueba que el inicio del for sea una variable int/double declarada.
25	Error_ForEnd.bky	Prueba que el fin del for sea una expresión int/double.
26	WriteEndl.bky	Prueba que al añadir endl en el write se cree un salto de línea.
27	ExpArith.bky	Calcula una expresión aritmética y la imprime.
28	Arrays.bky	Realiza operaciones sobre un array e imprime el resultado.
29	Error_Read.bky	Prueba que el read sea del tipo esperado.
30	Error_OutOfBounds.bky	Prueba que detecte el error si se intenta acceder a casillas inválidas de un array.
31	Error_ExecuteReturn.bky	Prueba que en una función con tipo se ejecute un return.

Descripción del proceso de desarrollo

Bitácora general

No de Avance	Fecha	Desarrollo
0	-	Desarrollamos la propuesta de Baky con el lenguaje de programación Python, utilizando la librería de Lex & Yacc.
1	3 de Octubre	Investigamos acerca de cómo podríamos correr nuestro compilador en python de forma nativa, buscamos muchas opciones pero decidimos que lo mejor sería cambiar nuestro lenguaje de programación.
2	12 de Octubre	Decidimos que la mejor manera de hacer un compilador que pudiera correr de manera nativa en IOS o Android sería haciéndolo con JavaScript. Sabiendo esto investigamos las mejores librerías para hacer un compilador y nos decidimos por hacerlo con Jison.
3	17 de Octubre	Programamos nuestra gramática y nuestras reglas de sintaxis.
4	4 de Noviembre	Agregamos las validaciones semánticas al igual que la generación de cuádruplos para expresiones aritméticas, estatutos, condicionales y ciclos.
5	7 de Noviembre	Hicimos la generación de cuádruplos para funciones.
6	14 de Noviembre	Creamos la memoria virtual y la máquina virtual, corriendo todos los cuádruplos que teníamos al momento. Empezamos a programar también la aplicación para el IDE en React Native.
7	16 de Noviembre	Agregamos arreglos y matrices a nuestros cuádruplos, memoria y máquina virtual. Logramos terminar la aplicación del IDE y poder ejecutar el código nativamente en un dispositivo móvil.
8	20 de Noviembre	Corregimos algunos errores mínimos del compilador y mejoramos el write, de igual forma empezamos la documentación del proyecto.

Lista de commits

10/17/2022	Ernesto García	init
10/17/2022	Ernesto García	Propuesta Inicial
10/17/2022	Daniel Trevino	grammar
10/17/2022	Ernesto García	Cambio Params
10/17/2022	Ernesto García	Avance 3 readme
10/29/2022	Ernesto García	Grammar correction
10/30/2022	Ernesto García	Grammar Syntax "for" correction
11/1/2022	Ernesto García	Semantics 1.0
11/1/2022	Ernesto García	AND correction, quads stacks
11/1/2022	Ernesto García	Add semantics to quadruple
11/2/2022	Ernesto García	Linear quads generation
11/2/2022	Daniel Trevino	if & while
11/2/2022	Daniel Trevino	for
11/2/2022	Ernesto García	For changes quads
11/4/2022	Ernesto García	Avance 4 Readme
11/4/2022	Ernesto García	Update README.md
11/4/2022	Ernesto García	Update README.md
11/7/2022	Daniel Trevino	add function quads
11/7/2022	Ernesto García	Avance5Readme
11/12/2022	Ernesto García	== and = priority and add = to semantic cube
11/12/2022	Ernesto García	Memory allocation in semantics
11/12/2022	Ernesto García	Add virtual memory
11/12/2022	Daniel Trevino	base app ide
11/12/2022	Daniel Trevino	basic vm

11/12/2022	Daniel Trevino	constantes no repetidas
11/13/2022	Daniel Trevino	funcs
11/14/2022	Ernesto García	functions bug
11/14/2022	Daniel Trevino	funcs recursivas
11/14/2022	Daniel Trevino	Merge pull request #1 from Ernesto1608/functions
11/16/2022	Ernesto García	Corrections grammar, return, functions void
11/16/2022	Ernesto García	Correction functions jumps
11/16/2022	Ernesto García	Arrays and Matrix
11/16/2022	Ernesto García	return error function
11/16/2022	Ernesto García	Added uminus
11/16/2022	Daniel Trevino	ide jalando
11/16/2022	Daniel Trevino	gitignore
11/17/2022	Daniel Trevino	input output ide
11/17/2022	Daniel Trevino	Merge branch 'functions'
11/17/2022	Daniel Trevino	detalles ide y update grammar
11/17/2022	Daniel Trevino	linea processArray
11/17/2022	Ernesto García	read array
11/17/2022	Ernesto García	Readme Avance 6
11/19/2022	Ernesto García	Add prompt to read files on console
11/20/2022	Ernesto García	Tests and bugs corrections
11/20/2022	Ernesto García	Write on same line
11/20/2022	Ernesto García	Write on same line IDE
11/20/2022	Ernesto García	tests
11/22/2022	Daniel Trevino	agregar comments

Reflexión

Daniel:

La verdad este proyecto me pareció muy interesante y retador, todo lo que teníamos que hacer lo vimos en clase, pero a mi me gusto basarme en eso para encontrar mi propia manera de hacer las cosas. Había trabajado anteriormente en un compilador, pero nunca me había metido tan a fondo y creo que ahora entiendo mucho mejor cómo funcionan las cosas. Tuvimos que resolver muchos problemas en equipo por lo cual fue de gran ayuda tener dos perspectivas de donde basarnos para cuando uno de los dos estaba atorado poder pedir ayuda. Crear el IDE para poder correr nuestro compilador fue su propio reto del cual aprendí mucho, como no siempre buscar una solución que ya exista como en librerías sino que siempre puedes adaptar una o crear una nueva.



Ernesto:

Personalmente considero que este proyecto ha sido el más “pesado” y retador que he realizado a lo largo de mi carrera, debido a que requiere de muchos aprendizajes previos y de la materia como tal desde el día uno en el que empiezas a planificar tu lenguaje, esto mismo hace que necesites llevar un análisis muy profundo de los temas vistos en clase pues a la hora de aplicarlos para el proyecto necesitas tener un buen entendimiento de lo que estás haciendo.

Uno de los mayores aprendizajes que me llevo del proyecto es el de la organización en equipo, al ser un proyecto tan teórico y largo era muy importante el tener una buena comunicación con mi compañero para el correcto avance del compilador y evitar la duplicidad de código al igual que mantener una buena calidad dentro del proyecto para mitigar la refactorización en cada uno de los avances que se iban entregando. Considero que fue de suma importancia el tener dos visiones diferentes a la hora de enfrentarnos con un nuevo reto, pues esto nos permitió poder abordar los problemas y solucionarlos de una manera más rápida para que el proyecto fluyera adecuadamente.

En definitiva es un proyecto que me ha ayudado mucho a crecer como Ingeniero de Software y que me gustaría poder mejorar en el futuro.



DESCRIPCIÓN DEL LENGUAJE

Nombre del lenguaje

El nombre del lenguaje es **Baky**, se decidió poner este nombre en representación a las dos mascotas de los creadores del lenguaje, por una parte un husky siberiano llamado **Baltto** y una cruce de golden retriever llamado **Cuky**. De igual manera podemos observar que la imagen del lenguaje es una mezcla entre un husky y un golden.



Descripción de las principales características

Baky es un lenguaje de programación procedural. Cuenta con una función principal que debe estar declarada en todo programa. Para identificar la función principal, donde empezará la ejecución del programa, se debe de declarar con el nombre "Baky", de igual forma esta palabra reservada nos indica el inicio de nuestro programa escrito en Baky. El lenguaje tiene las características típicas de cualquier lenguaje de programación como variables (globales o locales) simples y estructuras no atómicas como arreglos y matrices, es importante mencionar que Baky soporta cinco diferentes tipos (int, double, char, string, boolean). Entre estas características también se encuentra el uso de aritmética simple, condicional y lógica, funciones con parámetros, ciclos, condicionales, entradas y salidas.

En adición a todo esto, Baky tiene como meta el poder ser un lenguaje sencillo de programar y que todos puedan aprender desde cualquier lugar, por lo que es un lenguaje optimizado para correr nativamente en un dispositivo móvil y cuenta con su propio IDE híbrido (ios/android) que permite la creación, edición, compilación e interacción con programas escritos en Baky.

Listado de errores que pueden ocurrir

Mensaje de error	Descripción
Unsupported symbols on line LINE	El compilador encuentra un símbolo que no está en la gramática
Out of memory for SCOPE of type TYPE	Se acabo la memoria de un tipo en un scope (global, local, constante)
Array index must be int on line LINE	El índice de un arreglo no es un número entero

Matrix index must be int on line LINE	El índice de una matriz no es un entero
Cannot write void value on line LINE	Se quiere imprimir un valor VOID a consola
Conditions must have type boolean on line LINE	Una condición (if/while) con un valor no booleano en su evaluación
For must have int or double on line LINE	Un ciclo for no tiene asignada una variable entera o flotante. También aparece cuando su expresión de término no es entera o flotante.
Number of parameters doesn't match function definition for FUNCTION in line LINE	El número de parámetros de una llamada a una función no coincide con la declaración
Wrong parameter type on line LINE	Se manda un tipo de parámetro incorrecto
Missing return on function SCOPE	Una función que se espera que regrese un valor no regresa nada
Unexpected return on line LINE	Una función VOID regresa un valor
Wrong return type on line LINE	Una función regresa un tipo incorrecto
Invalid operation TYPE OPERATOR TYPE on line LINE	Una operación que no es válida de acuerdo con nuestro cubo semántico.
Duplicated function name ID on line LINE	Declaración de funciones con el mismo nombre
Duplicated variable name ID on line LINE on scope SCOPE	Declaración de variables con el mismo nombre dentro del mismo scope
Undeclared function ID on line LINE	Se intenta llamar una función que no existe
Undeclared variable ID on line LINE	Se intenta usar una variable que no existe
Missing indexes, variable ID on line LINE	Faltan los índices para un arreglo o matriz
Not an array, variable ID on line LINE	Se intenta usar como arreglo una variable que no es arreglo
Not a matrix, variable ID on line LINE	Se intenta usar como matriz una variable que no es matriz
Unable to cast VALUE to type TYPE	Se intenta ingresar como input un valor que no corresponde al tipo
Value VALUE out of bounds for array	Se intenta acceder a una posición de un

	arreglo o matriz que está fuera del rango
Expecting valid return on function FUNCTION	Una función que espera regresar un valor no regresa en tiempo de ejecución.

DESCRIPCIÓN DEL COMPILADOR

Descripción de herramientas utilizadas

Utilizamos el lenguaje de programación JavaScript para la codificación de nuestro compilador e IDE. Elegimos este lenguaje para poder correr Baky nativamente en un dispositivo móvil de manera sencilla y que fuera compatible tanto con Android como con IOS con la ayuda de React Native.

Para las librerías utilizadas nos apoyamos de npm (node package manager), herramienta que nos permitió el uso de “datastructures-js”, librería que cuenta con la implementación de algunas estructuras de datos en javascript que utilizamos en el proyecto. De igual manera la librería que utilizamos para ayudarnos a construir el compilador fue “Jison” que sirve para hacer parsers bottom-up y es una adaptación de Flex & Bison para javascript.

En cuanto a los equipos de cómputo utilizados para el desarrollo de Baky se usaron máquinas con Windows 10 y macOS Monterey, al apoyarnos de Node.js y npm, el desarrollo de Baky se puede continuar en cualquier sistema operativo que soporte estas dos herramientas.

Descripción análisis léxico

Patrones de construcción

Elemento	Expresión Regular
Comentarios (sin código)	<code>\\(.*)</code>
BOOLEAN_VALUE	<code>(true false)</code>
DOUBLE_VALUE	<code>[0-9]+\.[0-9]+</code>
INT_VALUE	<code>[0-9]+</code>
STRING_VALUE	<code>\"[^\"]*\"</code>
CHAR_VALUE	<code>'\.'</code>
ID	<code>[a-zA-z]\w*</code>

Tokens

Código	Token
BAKY	Baky

VAR	var
INT	int
DOUBLE	double
STRING	string
CHAR	char
BOOLEAN	boolean
FUNCTION	function
VOID	void
RETURN	return
READ	read
WRITE	write
IF	if
ELSE	else
WHILE	while
FROM	from
TO	to
DO	do
SEMICOLON	;
COMA	,
OPEN_SQUARE_BRACKET	[
CLOSE_SQUARE_BRACKET]
OPEN_CURLY_BRACKET	{
CLOSE_CURLY_BRACKET	}
OPEN_PARENTHESIS	(
CLOSE_PARENTHESIS)
EQ	==

EQUAL	=
OR	
AND	&&
TIMES	*
DIVIDED	/
NOT_EQUAL	!=
LESS_OR_EQ_THAN	<=
GREATER_OR_EQ_THAN	>=
LESS_THAN	<
GREATER_THAN	>
PLUS	+
MINUS	-

Descripción análisis sintáctico

Gramática Formal

baky →

BAKY ID SEMICOLON vars funcns main

vars →

ϵ |

vars_aux vars

vars_aux →

VAR type vars_aux2 SEMICOLON

vars_aux2 →

ID |

ID COMA vars_aux2 |

ID OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET |

ID OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET COMA

vars_aux2 |

ID OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET
 OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET |
 ID OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET
 OPEN_SQUARE_BRACKET INT_VALUE CLOSE_SQUARE_BRACKET COMA
 vars_aux2

funcs →

ϵ |
 function funcs

main →

VOID BAKY OPEN_PARENTHESIS CLOSE_PARENTHESIS vars block

type →

INT |
 DOUBLE |
 CHAR |
 STRING |
 BOOLEAN

function →

FUNCTION type ID OPEN_PARENTHESIS params CLOSE_PARENTHESIS vars block |
 FUNCTION VOID ID OPEN_PARENTHESIS params CLOSE_PARENTHESIS vars
 block

block →

OPEN_CURLY_BRACKET block_aux CLOSE_CURLY_BRACKET

block_aux →

ϵ |
 statute block_aux

params →

ϵ |
 params_aux

params_aux →

type ID |
 type ID COMA params_aux

statute →

call SEMICOLON |
 return |

```

    read |
    write |
    if |
    assign |
    while |
    for
call →
    ID OPEN_PARENTHESIS CLOSE_PARENTHESIS |
    ID OPEN_PARENTHESIS call_aux CLOSE_PARENTHESIS
call_aux →
    exp |
    exp COMA call_aux
return →
    RETURN exp SEMICOLON
read →
    READ OPEN_PARENTHESIS read_aux CLOSE_PARENTHESIS SEMICOLON
read_aux →
    var |
    var COMA read_aux
write →
    WRITE OPEN_PARENTHESIS write_aux CLOSE_PARENTHESIS SEMICOLON
write_aux →
    exp |
    exp COMA write_aux
assign →
    var EQUAL exp SEMICOLON
if →
    IF OPEN_PARENTHESIS exp CLOSE_PARENTHESIS block |
    IF OPEN_PARENTHESIS exp CLOSE_PARENTHESIS block ELSE block
while →
    WHILE OPEN_PARENTHESIS exp CLOSE_PARENTHESIS block
for →
    FROM var TO exp DO block
exp →

```

superexp |
 superexp OR exp

var →

ID |
 ID OPEN_SQUARE_BRACKET exp CLOSE_SQUARE_BRACKET |
 ID OPEN_SQUARE_BRACKET exp CLOSE_SQUARE_BRACKET
 OPEN_SQUARE_BRACKET exp CLOSE_SQUARE_BRACKET

superexp →

megaexp |
 megaexp AND superexp

megaexp →

hiperexp |
 hiperexp comp hiperexp

hiperexp →

term |
 term PLUS hiperexp |
 term MINUS hiperexp

comp →

LESS_THAN |
 GREATER_THAN |
 EQ |
 NOT_EQUAL |
 GREATER_OR_EQ_THAN |
 LESS_OR_EQ_THAN

term →

factor |
 factor TIMES term |
 factor DIVIDED term

factor →

OPEN_PARENTHESIS exp CLOSE_PARENTHESIS |
 var |
 call |
 INT_VALUE |
 MINUS INT_VALUE |

DOUBLE_VALUE |
 MINUS DOUBLE_VALUE |
 CHAR_VALUE |
 STRING_VALUE |
 BOOLEAN_VALUE

Descripción de generación de código intermedio y análisis semántico

Códigos de operación

Código	Descripción
returnError	Muestra un error en tiempo de ejecución debido a que una función con tipo jamás tuvo un return.
end	Realiza todos los writes pendientes al final de la ejecución del programa.
ver	Verifica que un valor dado en memoria esté en un rango esperado.
sumP	Suma dos valores dados en memoria y el resultado lo guarda como un apuntador.
write	Realiza un output en consola del valor dado en memoria.
read	Realiza el input desde consola y lo guarda en una dirección de memoria.
gotoF	Mueve el puntero de instrucciones a una posición determinada si el valor dado en memoria es falso, de lo contrario mueve secuencialmente el puntero.
goto	Mueve el puntero de instrucciones a una posición determinada.
init	Reserva la memoria necesaria para ejecutar una función y la inserta en nuestra pila de memoria.
golib	Mueve el puntero de instrucciones a una posición determinada y guarda la posición actual en una pila.
popScope	Elimina la memoria actual de la pila de memoria, elimina la posición de la parte superior de la pila de punteros y mueve el puntero de instrucciones a este valor obtenido.
	Evalúa la operación lógica “or” y guarda el resultado en memoria temporal.
&&	Evalúa la operación lógica “and” y guarda el resultado en memoria temporal.
=	Asigna un valor de memoria dado en otro valor de memoria.

==	Evalúa si dos valores de memoria son iguales y guarda el resultado en memoria temporal.
!=	Evalúa si dos valores de memoria no son iguales y guarda el resultado en memoria temporal.
>	Evalúa si un valor es mayor que otro y guarda el resultado en memoria temporal.
<	Evalúa si un valor es menor que otro y guarda el resultado en memoria temporal.
>=	Evalúa si un valor es mayor o igual que otro y guarda el resultado en memoria temporal.
<=	Evalúa si un valor es menor o igual que otro y guarda el resultado en memoria temporal.
+	Realiza la suma de dos valores y guarda el resultado en memoria temporal.
-	Realiza la resta de dos valores y guarda el resultado en memoria temporal.
*	Realiza la multiplicación de dos valores y guarda el resultado en memoria temporal.
/	Realiza la división de dos valores y guarda el resultado en memoria temporal.

Direcciones virtuales

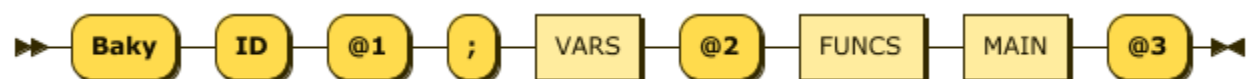
Inicio	Fin	Scope	Tipo
0	999	Global	Int
1000	1999	Global	Double
2000	2999	Global	String
3000	3999	Global	Char
4000	4999	Global	Boolean
5000	5999	Local	Int
6000	6999	Local	Double
7000	7999	Local	String
8000	8999	Local	Char
9000	9999	Local	Boolean

10000	10999	Local	Int Temporal
11000	11999	Local	Double Temporal
12000	12999	Local	String Temporal
13000	13999	Local	Char Temporal
14000	14999	Local	Boolean Temporal
15000	15999	Local	Pointer
16000	16999	Constante	Int
17000	17999	Constante	Double
18000	18999	Constante	String
19000	19999	Constante	Char
20000	20999	Constante	Boolean

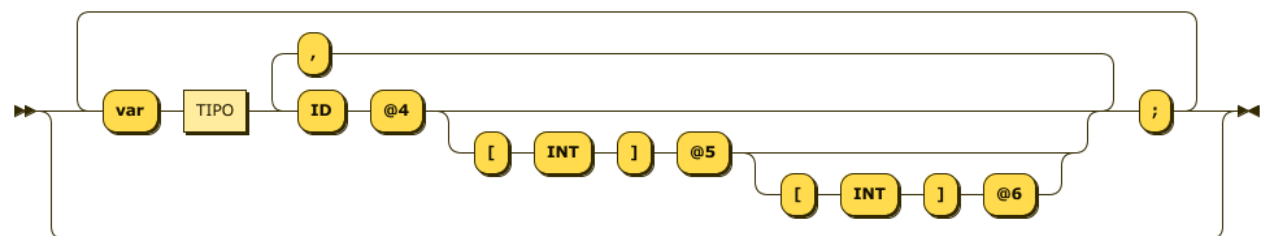
Diagramas de sintaxis

En los siguientes diagramas los puntos neurálgicos vienen con el siguiente formato '@n' siendo n el número consecutivo que corresponde a cada punto y que más adelante se describe con más detalle.

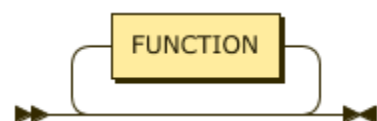
BAKY →



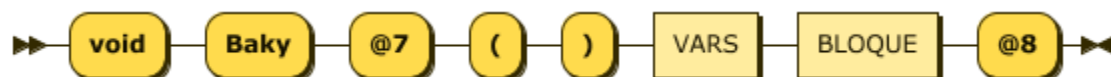
VARS →



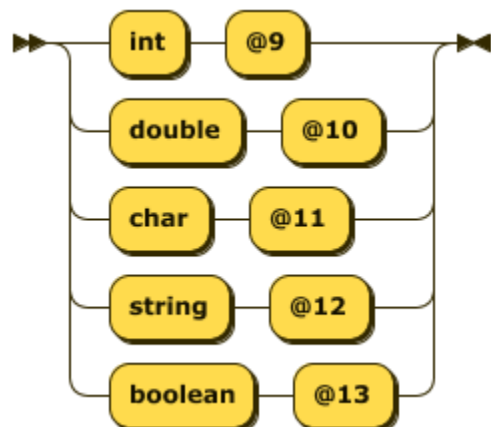
FUNCS →



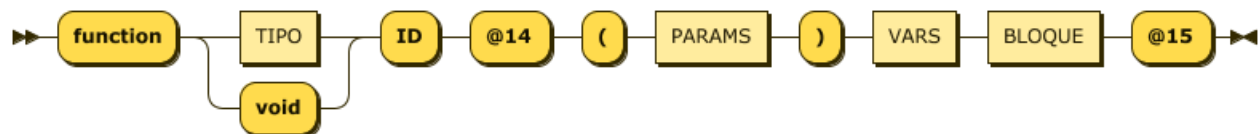
MAIN →



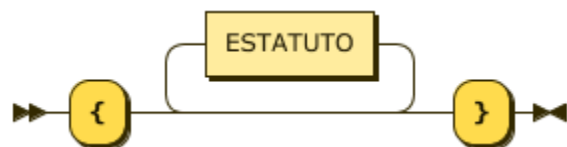
TIPO →



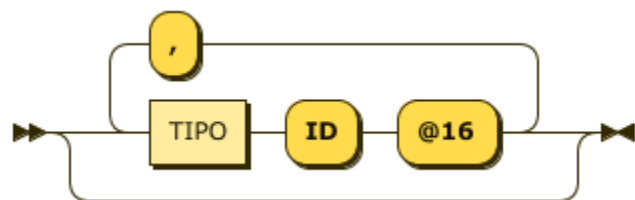
FUNCTION →



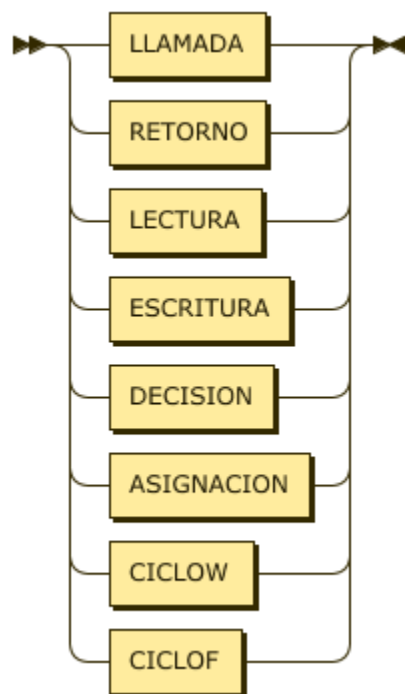
BLOQUE →



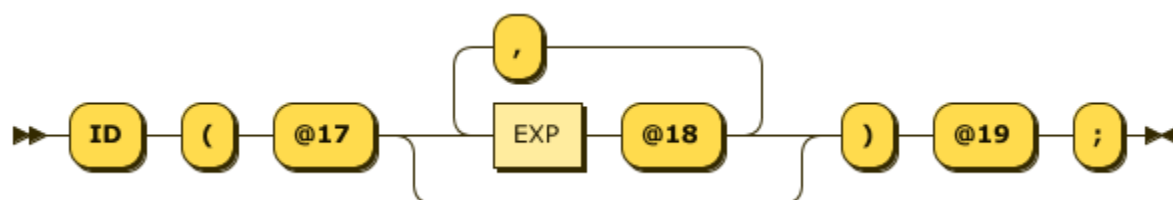
PARAMS →



ESTATUTO →



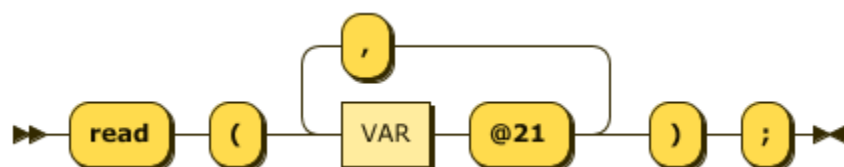
LLAMADA →



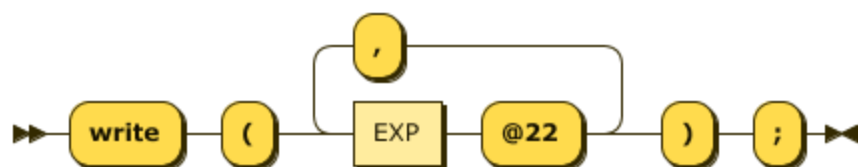
RETORNO →



LECTURA →



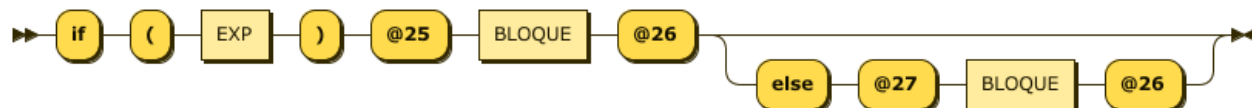
ESCRITURA →



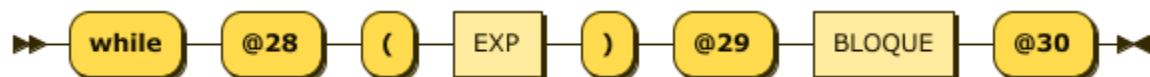
ASIGNACIÓN →



DECISIÓN →



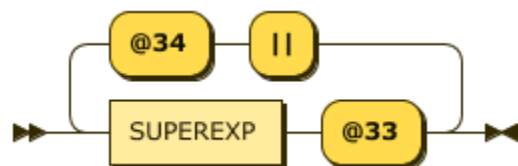
CICLOW →



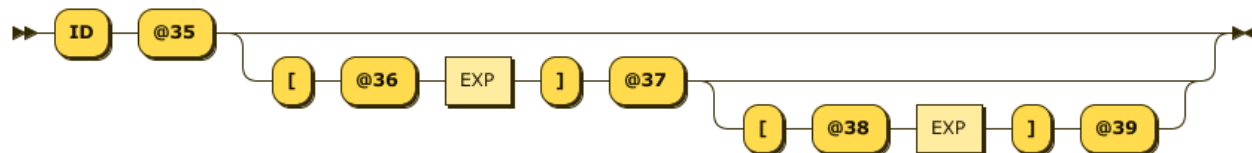
CICLOF →



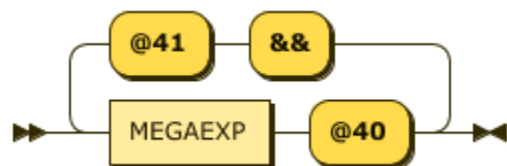
EXP →



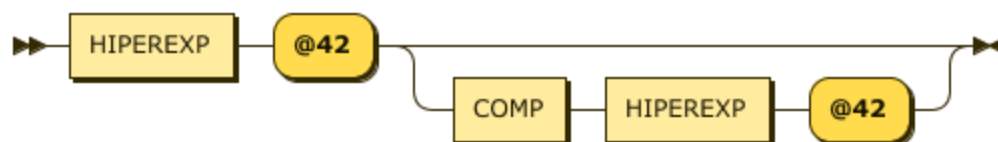
VAR →



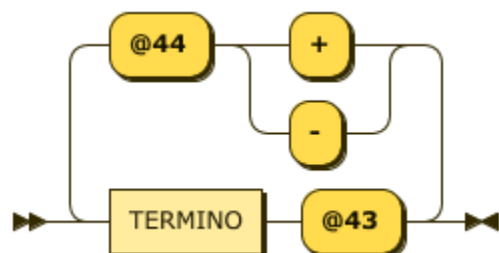
SUPEREXP →



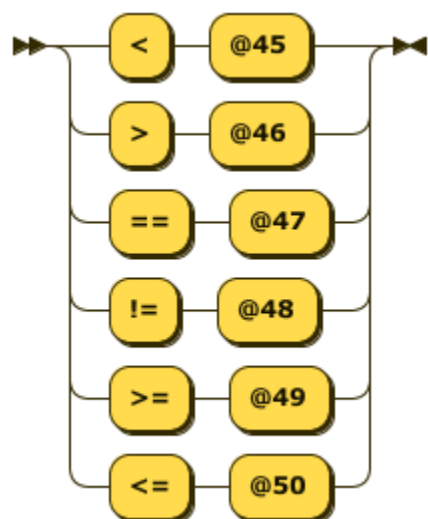
MEGAEXP →



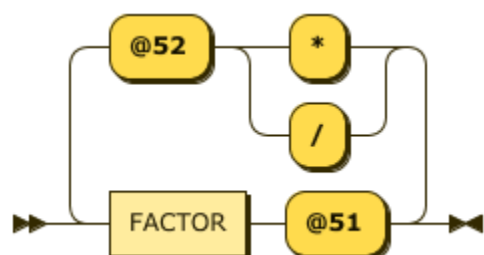
HIPEREXP →



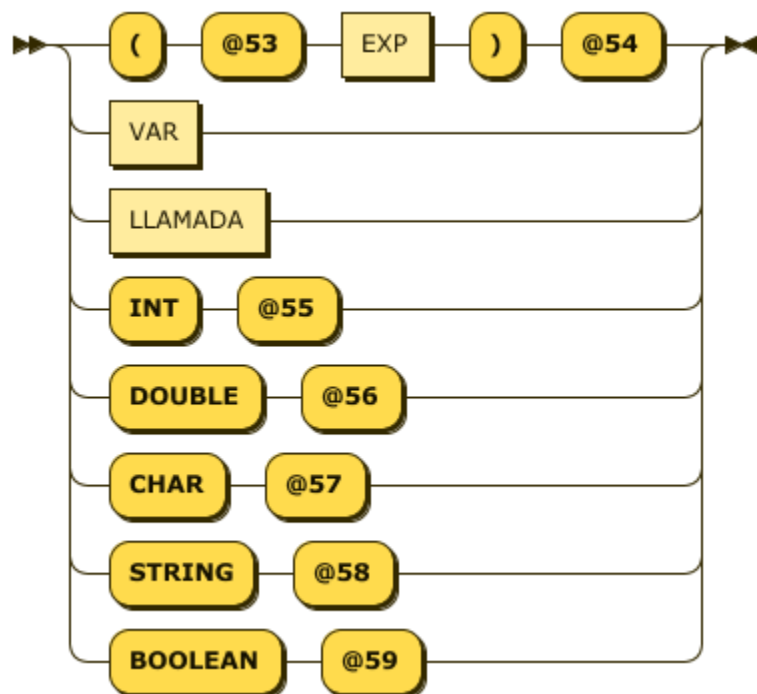
COMP →



TERMINO →



FACTOR →



Descripción de las acciones semánticas y de generación de código

No	Descripción
1	Se crea el directorio de funciones y el scope global.
2	Se reserva el espacio necesario en la memoria virtual para el scope global.
3	Se crea el cuádruplo "end".
4	Si no está duplicado el id se inserta la variable y el tipo en la tabla de variables y se aumenta el tipo de recurso para la función actual, también se le asigna una dirección.
5	Se inserta la dimensión en la tabla de variables actual, se aumenta los recursos y las direcciones de acuerdo al tipo.
6	Se inserta la dimensión en la tabla de variables actual, se aumenta los recursos y las direcciones de acuerdo al tipo.

7	Si no está duplicado el id de la función, se crea un nuevo registro en el directorio de funciones con su tabla de variables vacía, si tiene un tipo se crea una variable global para el valor de retorno.
8	Elimina la tabla de variables del scope actual, si es una función con tipo válida que tenga por lo menos un return declarado y crea el cuádruplo de "popScope".
9	Asigna el tipo actual a INT
10	Asigna el tipo actual a DOUBLE
11	Asigna el tipo actual a CHAR
12	Asigna el tipo actual a STRING
13	Asigna el tipo actual a BOOLEAN
14	Si no está duplicado el id de la función, se crea un nuevo registro en el directorio de funciones con su tabla de variables vacía, si tiene un tipo se crea una variable global para el valor de retorno.
15	Elimina la tabla de variables del scope actual, si es una función con tipo válida que tenga por lo menos un return declarado y crea el cuádruplo de "popScope" y "returnError".
16	Si no está duplicado el id se inserta la variable y el tipo en la tabla de variables y de parámetros, se aumenta el tipo de recurso para la función actual, también se le asigna una dirección.
17	Verifica que la función esté declarada y hace push de un fondo falso en el stack de operadores.
18	Verifica que el parámetro sea del tipo esperado.
19	Hace pop del fondo falso en el stack de operadores, válida que el número de parámetros sea el esperado, crea variables globales para pasar los parámetros, crea el cuádruplo de "init", "gosub" y la asignación del valor de retorno a un temporal.
20	Valida que no sea un return a un void y que el tipo de return sea el correcto, crea el cuádruplo "popScope".
21	Crea el cuádruplo de "read".
22	Valida que no sea void y crea el cuádruplo "write".
23	Inserta el operador "=" a la pila de operadores.
24	Hace pop de la pila de operadores y operandos, verifica que sea una operación válida y crea el cuádruplo del operador.
25	Valida que sea un boolean, crea el cuádruplo "gotoF" e inserta en la pila de saltos el contador actual.

26	Hace pop de la pila de saltos y rellena el cuádruplo del salto con el contador actual.
27	Crea el cuádruplo "goto", hace pop de la pila de saltos y rellena el cuádruplo del salto con el contador actual, inserta en la pila de saltos el contador actual.
28	Inserta en la pila de saltos el contador actual.
29	Valida que sea un boolean, crea el cuádruplo "gotoF" e inserta en la pila de saltos el contador actual.
30	Hace dos pop de la pila de saltos, crea el cuádruplo "goto" con el segundo pop y rellena el cuádruplo del primer pop con el contador actual.
31	Válida que la variable y la expresión sean INT o DOUBLE, inserta en la pila de saltos el contador actual, crea el cuádruplo de comparación, inserta nuevamente el contador a los saltos, crea el "gotoF".
32	Aumenta la variable en uno, hace dos pop de la pila de saltos, crea el "goto" con el segundo pop y rellena el cuádruplo del primer pop con el contador actual.
33	Mientras el top de la pila de operadores sea realiza la operación haciendo dos pop a la pila de operandos, verifica que sea una operación válida y lo guarda en un temporal.
34	Inserta el operador " " a la pila de operadores.
35	Valida que la variable este declarada e inserta su address a la pila de operandos y su tipo a la pila de tipos.
36	Hace push de un fondo falso en el stack de operadores.
37	Hace pop del fondo falso en el stack de operadores, valida que la expresión sea un INT, crea el cuádruplo "ver" con los límites de la dimensión, crea "sumP" con el valor de la expresión, la dirección base y lo guarda en un pointer.
38	Hace push de un fondo falso en el stack de operadores.
39	Hace pop del fondo falso en el stack de operadores, valida que ambas expresiones sean un INT, crea el cuádruplo "ver" con los límites de las dos dimensiones, hace el $s1*d2+s2$, crea "sumP" con el valor de la expresión, la dirección base y lo guarda en un pointer.
40	Mientras el top de la pila de operadores sea && realiza la operación haciendo dos pop a la pila de operandos, verifica que sea una operación válida y lo guarda en un temporal.
41	Inserta el operador "&&" a la pila de operadores.
42	Mientras el top de la pila de operadores sea un token relacional realiza la operación haciendo dos pop a la pila de operandos, verifica que sea una operación válida y lo guarda en un temporal.
43	Mientras el top de la pila de operadores sea +/- realiza la operación haciendo dos pop a la pila de operandos, verifica que sea una operación válida y lo guarda en un temporal.

44	Inserta el operador “+/-” a la pila de operadores.
45	Inserta el operador “<” a la pila de operadores.
46	Inserta el operador “>” a la pila de operadores.
47	Inserta el operador “==” a la pila de operadores.
48	Inserta el operador “!=” a la pila de operadores.
49	Inserta el operador “>=” a la pila de operadores.
50	Inserta el operador “<=” a la pila de operadores.
51	Mientras el top de la pila de operadores sea * o / realiza la operación haciendo dos pop a la pila de operandos, verifica que sea una operación válida y lo guarda en un temporal.
52	Inserta el operador “* o /” a la pila de operadores.
53	Hace push de un fondo falso en el stack de operadores.
54	Hace pop del fondo falso en el stack de operadores.
55	Verifica que la constante INT no esté ya declarada, si no está declarada la crea en la tabla de constantes y le asigna una dirección, da de alta el valor en la memoria virtual y hace push de su dirección y tipo en las respectivas pilas.
56	Verifica que la constante DOUBLE no esté ya declarada, si no está declarada la crea en la tabla de constantes y le asigna una dirección, da de alta el valor en la memoria virtual y hace push de su dirección y tipo en las respectivas pilas.
57	Verifica que la constante CHAR no esté ya declarada, si no está declarada la crea en la tabla de constantes y le asigna una dirección, da de alta el valor en la memoria virtual y hace push de su dirección y tipo en las respectivas pilas.
58	Verifica que la constante STRING no esté ya declarada, si no está declarada la crea en la tabla de constantes y le asigna una dirección, da de alta el valor en la memoria virtual y hace push de su dirección y tipo en las respectivas pilas.
59	Verifica que la constante BOOLEAN no esté ya declarada, si no está declarada la crea en la tabla de constantes y le asigna una dirección, da de alta el valor en la memoria virtual y hace push de su dirección y tipo en las respectivas pilas.

Tabla de consideraciones semánticas

La siguiente tabla muestra todas las operaciones válidas entre dos tipos de operandos con su respectivo tipo de resultado, es importante mencionar que cualquier operación entre dos tipos que no esté descrita en esta tabla genera un error semántico.

Tipo 1	Operador	Tipo 2	Resultado
INT	+	INT	INT
INT	-	INT	INT
INT	*	INT	INT
INT	/	INT	DOUBLE
INT	==	INT	BOOLEAN
INT	=	INT	INT
INT	!=	INT	BOOLEAN
INT	>	INT	BOOLEAN
INT	>=	INT	BOOLEAN
INT	<	INT	BOOLEAN
INT	<=	INT	BOOLEAN
INT	+	DOUBLE	DOUBLE
INT	-	DOUBLE	DOUBLE
INT	*	DOUBLE	DOUBLE
INT	/	DOUBLE	DOUBLE
INT	==	DOUBLE	BOOLEAN
INT	=	DOUBLE	INT
INT	!=	DOUBLE	BOOLEAN
INT	>	DOUBLE	BOOLEAN
INT	>=	DOUBLE	BOOLEAN
INT	<	DOUBLE	BOOLEAN
INT	<=	DOUBLE	BOOLEAN
DOUBLE	+	DOUBLE	DOUBLE
DOUBLE	-	DOUBLE	DOUBLE
DOUBLE	*	DOUBLE	DOUBLE
DOUBLE	/	DOUBLE	DOUBLE

DOUBLE	==	DOUBLE	BOOLEAN
DOUBLE	=	DOUBLE	DOUBLE
DOUBLE	!=	DOUBLE	BOOLEAN
DOUBLE	>	DOUBLE	BOOLEAN
DOUBLE	>=	DOUBLE	BOOLEAN
DOUBLE	<	DOUBLE	BOOLEAN
DOUBLE	<=	DOUBLE	BOOLEAN
DOUBLE	+	INT	DOUBLE
DOUBLE	-	INT	DOUBLE
DOUBLE	*	INT	DOUBLE
DOUBLE	/	INT	DOUBLE
DOUBLE	==	INT	BOOLEAN
DOUBLE	=	INT	DOUBLE
DOUBLE	!=	INT	BOOLEAN
DOUBLE	>	INT	BOOLEAN
DOUBLE	>=	INT	BOOLEAN
DOUBLE	<	INT	BOOLEAN
DOUBLE	<=	INT	BOOLEAN
STRING	=	STRING	STRING
STRING	==	STRING	BOOLEAN
STRING	!=	STRING	BOOLEAN
CHAR	=	CHAR	CHAR
CHAR	==	CHAR	BOOLEAN
CHAR	!=	CHAR	BOOLEAN
BOOLEAN	=	BOOLEAN	BOOLEAN
BOOLEAN	==	BOOLEAN	BOOLEAN
BOOLEAN	!=	BOOLEAN	BOOLEAN

BOOLEAN		BOOLEAN	BOOLEAN
BOOLEAN	&&	BOOLEAN	BOOLEAN

Descripción de la administración de memoria en compilación

Para la compilación, utilizamos diferentes estructuras para guardar los datos necesarios para la generación de cuádruplos y su validación semántica. Como lo son:

- **Arreglo de cuádruplos:** utilizamos un arreglo de arreglos de cuatro posiciones para cada cuádruplo, para esto solo necesitábamos que la estructura pudiera estar en el orden en el que se generan.
- **Stack de operandos:** el stack de operandos se utiliza para poder obtener el último operando que fue identificado en el código.
- **Stack de tipos:** este stack se utiliza en conjunto con el de operandos para poder saber el tipo de los operandos.
- **Stack de operadores:** se utilizó un stack para poder llevar cuenta de los operadores y poder sacar el último cuando se tenga que procesar.
- **Stack de saltos:** se utiliza un stack porque los saltos se van resolviendo del último al primero.
- **String globalName:** un string para guardar el nombre del programa (acceso a variables globales).
- **Stack scopes:** inicialmente teníamos la idea de tener scopes anidados por lo que utilizamos un stack para saber la jerarquía de scopes, pero finalmente no utilizamos scopes anidados pero se quedó el stack para poder implementarlo en un futuro.
- **String currentType:** guardamos en un string el último tipo procesado por el parser.
- **Hash table dir funciones:** se genera una entrada con el nombre de la función como llave, se guarda el tipo de la función, el scope pasado (en este caso global para todos, pero lo dejamos implementado para scopes anidados), una tabla de variables, una tabla de parámetros, el inicio de la función (línea de cuádruplos), un arreglo para los recursos utilizados y una bandera booleana de si se encontró un valor de retorno o no. Se utilizó esta estructura porque tiene que poderse hacer una búsqueda eficiente.
 - **Tabla de variables:** la tabla de variables es una hash table, para una búsqueda eficiente, se guarda el id de la variable como la llave y dentro del objeto se guarda el tipo de la variable, su dirección de memoria y sus dimensiones en caso de ser un arreglo.
 - **Tabla de parámetros:** un arreglo de objetos para cada parámetro, con el id del parámetro, el tipo y su dirección, se utilizó un arreglo para poder accederlos de manera secuencial.
 - **Arreglo de recursos:** El arreglo de recursos es un arreglo de números enteros para poder saber la cantidad de recursos necesarios para cada tipo de datos, dependiendo de la posición en el arreglo es como lo identificamos, similar a como manejamos la memoria en ejecución que se detalla más adelante.

- **Hash table constsTable:** Hash table que mapea un valor constante a una dirección de memoria, se utiliza así para poder buscar de manera eficiente y no tener constantes repetidas que gasten memoria de más.
- **Cubo semántico:** Nuestro cubo semántico es un arreglo de tres dimensiones, en donde las primeras dos dimensiones son el tipo de operandos y la tercera dimensión es el operador dando como resultado si es una operación válida o si produce un error semántico. Por ejemplo si queremos consultar si la operación INT*CHAR es válida, tenemos que consultar al arreglo indexado como Cubo["INT"]["CHAR"]["*"].
- **Memory:** Este objeto representa el instanciamiento de la memoria de ejecución que se detalla más adelante desde compilación, debido a que nuestro proyecto tenía como finalidad correr en un IDE en donde con un botón se compila y ejecuta al mismo tiempo decidimos tomar ventaja de esto y cruzar esa línea compilación/ejecución para que dentro de la compilación se puedan inicializar las constantes en su respectiva dirección y se reserve el espacio necesario para las variables globales dentro de la memoria de ejecución, esto con la finalidad de podernos ahorrar algunos cuádruplos que no consideramos que valieran la pena.
- **String currentFunctionCall:** un string que guarda el nombre de la última función que se ha llamado con la finalidad de poder acceder a sus recursos y parámetros en las validaciones semánticas para la generación de los cuádruplos de las funciones.
- **Int paramsCounter:** entero que se inicializa en 0 en cada llamada y va aumentando secuencialmente conforme encuentre un nuevo parámetro con la finalidad de poder validar semánticamente el tipo de los parámetros y el número final esperado de parámetros.
- **Int expectedParams:** entero que guarda los parámetros esperados de la última función que se ha llamado con la finalidad de validar al final de la llamada si el paramsCounter es igual al expectedParams.

DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

Descripción de la administración de memoria en ejecución

Especificación gráfica y justificación

Arreglo para variables globales	[[ints], [doubles], [strings], [chars], [bools]]
Stack para scope actual	STACK([[ints], [doubles], [strings], [chars], [bools], [temp_ints], [temp_doubles], [temp_strings], [temp_chars], [temp_bools], [temp_pointers]])
Arreglo para constantes	[[ints], [doubles], [strings], [chars], [bools]]

Utilizamos un arreglo para la memoria, en la primera posición se encuentra un arreglo de arreglos para cada tipo de las variables globales. En la siguiente posición se encuentra un stack de arreglos de arreglos para cada tipo de las variables de ese scope específico, incluyendo temporales. Finalmente tenemos un arreglo de arreglos para cada tipo de las constantes. Es importante mencionar que estos arreglos se crean de manera dinámica dependiendo de los recursos exactos que se necesiten para cada scope correspondiente.

Asociación hecha entre las direcciones virtuales (compilación) y las reales (ejecución).

Los índices que utilizamos para la memoria son los siguientes:

```
global: {
    INT: 0,
    DOUBLE: 1000,
    STRING: 2000,
    CHAR: 3000,
    BOOLEAN: 4000,
},
local: {
    INT: 5000,
    DOUBLE: 6000,
    STRING: 7000,
    CHAR: 8000,
    BOOLEAN: 9000,
    TINT: 10000,
    TDOUBLE: 11000,
    TSTRING: 12000,
    TCHAR: 13000,
    TBOOLEAN: 14000,
    TPOINTER: 15000,
},
cons: {
    INT: 16000,
    DOUBLE: 17000,
    STRING: 18000,
    CHAR: 19000,
    BOOLEAN: 20000,
}
```

Para pasar de las direcciones virtuales a las reales primero encontramos si es global, local o constante, para ver en qué posición de nuestra memoria hacer el cambio. Después de encontrar eso vemos con la dirección el tipo de variable que es para identificar en qué arreglo (de los tipos) se encuentra esa dirección. Finalmente sacamos el módulo de 1000 de la dirección para poder encontrar la posición en el arreglo de tipos.

Por ejemplo si tuviéramos la dirección 11004, encontramos que es local, por lo que el primer índice es 1, después sabemos que es un double temporal, por lo que está en el arreglo de tipos en la posición 6 y finalmente el módulo de 1000 es 4 por lo que se encuentra en memoria[1][6][4].

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

Pruebas

Factorial Cíclico

Código fuente

```
Baky FactorialC;

function int factorial(int n)
var int i,f;
{
    f = 1;
    i = 1;
    from i to n+1 do {
        f = f*i;
    }
    return f;
}

void Baky()
var int n;
{
    read(n);
    n = factorial(n);
    write(n);
}
```

Código intermedio

```
0 : ["=",5002,16000,null]
1 : ["=",5001,16000,null]
2 : ["+",5000,16000,10000]
3 : ["<",5001,10000,14000]
4 : ["gotoF",14000,null,9]
5 : ["*",5002,5001,10001]
6 : ["=",5002,10001,null]
7 : ["+",5001,16000,5001]
8 : ["goto",null,null,3]
```

```

9 : ["=",0,5002,null]
10 : ["popScope",null,null,null]
11 : ["returnError","factorial",null,null]
12 : ["popScope",null,null,null]
13 : ["read",5000,null,null]
14 : ["=",1,5000,null]
15 : ["init","factorial",null,null]
16 : ["=",5000,1,null]
17 : ["gosub",null,null,0]
18 : ["=",10000,0,null]
19 : ["=",5000,10000,null]
20 : ["write",5000,null,null]
21 : ["end",null,null,null]

```

Resultado de ejecución

```

Successful compilation of program FactorialC
(input) 6
(BAKY) 720

```

Factorial Recursivo

Código fuente

```

Baky FactorialR;

function int factorial(int n){
    if(n <= 0) {return 1;}
    return n * factorial(n-1);
}

void Baky()
var int n;
{
    read(n);
    n = factorial(n);
    write(n);
}

```

```
}

```

Código intermedio

```
0 : ["<=",5000,16000,14000]
1 : ["gotoF",14000,null,4]
2 : ["=",0,16001,null]
3 : ["popScope",null,null,null]
4 : ["-",5000,16001,10000]
5 : ["=",1,10000,null]
6 : ["init","factorial",null,null]
7 : ["=",5000,1,null]
8 : ["gosub",null,null,0]
9 : ["=",10001,0,null]
10 : ["*",5000,10001,10002]
11 : ["=",0,10002,null]
12 : ["popScope",null,null,null]
13 : ["returnError","factorial",null,null]
14 : ["popScope",null,null,null]
15 : ["read",5000,null,null]
16 : ["=",1,5000,null]
17 : ["init","factorial",null,null]
18 : ["=",5000,1,null]
19 : ["gosub",null,null,0]
20 : ["=",10000,0,null]
21 : ["=",5000,10000,null]
22 : ["write",5000,null,null]
23 : ["end",null,null,null]
```

Resultado de ejecución

```
Successful compilation of program FactorialR
(input) 20
(BAKY) 2432902008176640000
```

Fibonacci Cíclico

Código fuente

```
Baky FibonacciC;

function int fibonacci(int n)
var int i,j, temp;
{
    i = 1;
    j = 1;
    while(n>2){
        temp = i;
        i = j;
        j = temp + j;
        n = n-1;
    }
    return j;
}

void Baky()
var int n;
{
    read(n);
    n = fibonacci(n);
    write(n);
}

// Serie de fib: 1 1 2 3 5 8 13 21 34 55 89 144 233
```

Código intermedio

```
0 : ["=",5001,16000,null]
1 : ["=",5002,16000,null]
2 : [ ">",5000,16001,14000]
3 : ["gotoF",14000,null,11]
4 : ["=",5003,5001,null]
5 : ["=",5001,5002,null]
```

```

6 : ["+",5003,5002,10000]
7 : ["=",5002,10000,null]
8 : ["-",5000,16000,10001]
9 : ["=",5000,10001,null]
10 : ["goto",null,null,2]
11 : ["=",0,5002,null]
12 : ["popScope",null,null,null]
13 : ["returnError","fibonacci",null,null]
14 : ["popScope",null,null,null]
15 : ["read",5000,null,null]
16 : ["=",1,5000,null]
17 : ["init","fibonacci",null,null]
18 : ["=",5000,1,null]
19 : ["gosub",null,null,0]
20 : ["=",10000,0,null]
21 : ["=",5000,10000,null]
22 : ["write",5000,null,null]
23 : ["end",null,null,null]

```

Resultado de ejecución

```

Successful compilation of program FibonacciC
(input) 20
(BAKY) 6765

```

Fibonacci Recursivo

Código fuente

```

Baky FibonacciR;

function int fibonacci(int n){
    if(n<=2) {return 1;}
    return fibonacci(n-2)+fibonacci(n-1);
}

void Baky()
var int n;

```



```

{
    read(n);
    n = fibonacci(n);
    write(n);
}

// Serie de fib: 1 1 2 3 5 8 13 21 34 55 89 144 233

```

Código intermedio

```

0 : ["<=",5000,16000,14000]
1 : ["gotoF",14000,null,4]
2 : ["=",0,16001,null]
3 : ["popScope",null,null,null]
4 : ["-",5000,16000,10000]
5 : ["=",1,10000,null]
6 : ["init","fibonacci",null,null]
7 : ["=",5000,1,null]
8 : ["gosub",null,null,0]
9 : ["=",10001,0,null]
10 : ["-",5000,16001,10002]
11 : ["=",1,10002,null]
12 : ["init","fibonacci",null,null]
13 : ["=",5000,1,null]
14 : ["gosub",null,null,0]
15 : ["=",10003,0,null]
16 : ["+",10001,10003,10004]
17 : ["=",0,10004,null]
18 : ["popScope",null,null,null]
19 : ["returnError","fibonacci",null,null]
20 : ["popScope",null,null,null]
21 : ["read",5000,null,null]
22 : ["=",1,5000,null]
23 : ["init","fibonacci",null,null]
24 : ["=",5000,1,null]
25 : ["gosub",null,null,0]
26 : ["=",10000,0,null]
27 : ["=",5000,10000,null]
28 : ["write",5000,null,null]

```

```
29 : ["end",null,null,null]
```

Resultado de ejecución

```
Successful compilation of program FibonacciR
(input) 30
(BAKY) 832040
```

Bubble Sort

Código fuente

```
Baky BubbleSort;
var int arr[10];

function void readArray()
var int i;
{
    from i to 10 do{
        read(arr[i]);
    }
}

function void writeArray()
var int i;
{
    write("El arreglo ordenado: endl[ ");
    from i to 10 do{
        write(arr[i], " , ");
    }
    write("]");
}

function void bubbleSort()
var int i,j,temp;
var boolean s;
{
    s = true;
```

```

while(i<9 && s){
    s = false;
    j = 0;
    while(j<9-i){
        if (arr[j] > arr[j + 1]){
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
            s = true;
        }
        j = j+1;
    }
    i = i+1;
}

void Baky()
var int n;
{
    write("Llena el arreglo de 10 posiciones");
    readArray();
    bubbleSort();
    writeArray();
}

```

Código intermedio

```

0 : ["<",5000,16000,14000]
1 : ["gotoF",14000,null,7]
2 : ["ver",5000,0,"10"]
3 : ["sumP",5000,16001,15000]
4 : ["read",15000,null,null]
5 : ["+",5000,16002,5000]
6 : ["goto",null,null,0]
7 : ["popScope",null,null,null]
8 : ["write",18000,null,null]
9 : ["<",5000,16000,14000]
10 : ["gotoF",14000,null,17]
11 : ["ver",5000,0,"10"]

```

```
12 : ["sumP",5000,16001,15000]
13 : ["write",15000,null,null]
14 : ["write",18001,null,null]
15 : ["+",5000,16002,5000]
16 : ["goto",null,null,9]
17 : ["write",18002,null,null]
18 : ["popScope",null,null,null]
19 : ["=",9000,20000,null]
20 : ["<",5000,16003,14000]
21 : ["&&",14000,9000,14001]
22 : ["gotoF",14001,null,55]
23 : ["=",9000,20001,null]
24 : ["=",5001,16001,null]
25 : ["-",16003,5000,10000]
26 : ["<",5001,10000,14002]
27 : ["gotoF",14002,null,52]
28 : ["ver",5001,0,"10"]
29 : ["sumP",5001,16001,15000]
30 : ["+",5001,16002,10001]
31 : ["ver",10001,0,"10"]
32 : ["sumP",10001,16001,15001]
33 : [">",15000,15001,14003]
34 : ["gotoF",14003,null,49]
35 : ["ver",5001,0,"10"]
36 : ["sumP",5001,16001,15002]
37 : ["=",5002,15002,null]
38 : ["ver",5001,0,"10"]
39 : ["sumP",5001,16001,15003]
40 : ["+",5001,16002,10002]
41 : ["ver",10002,0,"10"]
42 : ["sumP",10002,16001,15004]
43 : ["=",15003,15004,null]
44 : ["+",5001,16002,10003]
45 : ["ver",10003,0,"10"]
46 : ["sumP",10003,16001,15005]
47 : ["=",15005,5002,null]
48 : ["=",9000,20000,null]
49 : ["+",5001,16002,10004]
50 : ["=",5001,10004,null]
```

```

51 : ["goto",null,null,25]
52 : ["+",5000,16002,10005]
53 : ["=",5000,10005,null]
54 : ["goto",null,null,20]
55 : ["popScope",null,null,null]
56 : ["write",18003,null,null]
57 : ["init","readArray",null,null]
58 : ["gosub",null,null,0]
59 : ["init","bubbleSort",null,null]
60 : ["gosub",null,null,19]
61 : ["init","writeArray",null,null]
62 : ["gosub",null,null,8]
63 : ["end",null,null,null]

```

Resultado de ejecución

```

Successful compilation of program BubbleSort
(BAKY) Llena el arreglo de 10 posiciones
(input) 56
(input) 34
(input) 2
(input) 3
(input) 546
(input) 34
(input) 2
(input) 1
(input) 34
(input) 78
(BAKY) El arreglo ordenado:
(BAKY) [ 1 , 2 , 2 , 3 , 34 , 34 , 34 , 56 , 78 , 546 , ]

```

Find

Código fuente

```

Baky Find:
var int arr[10];

function void readArray()

```

```

var int i;
{
    from i to 10 do{
        read(arr[i]);
    }
}

function int findArray(int n)
var int i;
{
    from i to 10 do {
        if(arr[i] == n){
            return i;
        }
    }
    return -1;
}

void Baky()
var int n;
{
    write("Llena el arreglo de 10 posiciones");
    readArray();
    write("Ingresa el número a buscar:");
    read(n);
    n = findArray(n);
    write("El index del número es:", n);
}

```

Código intermedio

```

0 : ["<",5000,16000,14000]
1 : ["gotoF",14000,null,7]
2 : ["ver",5000,0,"10"]
3 : ["sumP",5000,16001,15000]
4 : ["read",15000,null,null]
5 : ["+",5000,16002,5000]
6 : ["goto",null,null,0]
7 : ["popScope",null,null,null]

```

```
8 : ["<",5001,16000,14000]
9 : ["gotoF",14000,null,18]
10 : ["ver",5001,0,"10"]
11 : ["sumP",5001,16001,15000]
12 : ["==",15000,5000,14001]
13 : ["gotoF",14001,null,16]
14 : ["=",10,5001,null]
15 : ["popScope",null,null,null]
16 : ["+",5001,16002,5001]
17 : ["goto",null,null,8]
18 : ["=",10,16003,null]
19 : ["popScope",null,null,null]
20 : ["returnError","findArray",null,null]
21 : ["popScope",null,null,null]
22 : ["write",18000,null,null]
23 : ["init","readArray",null,null]
24 : ["gosub",null,null,0]
25 : ["write",18001,null,null]
26 : ["read",5000,null,null]
27 : ["=",11,5000,null]
28 : ["init","findArray",null,null]
29 : ["=",5000,11,null]
30 : ["gosub",null,null,8]
31 : ["=",10000,10,null]
32 : ["=",5000,10000,null]
33 : ["write",18002,null,null]
34 : ["write",5000,null,null]
35 : ["end",null,null,null]
```

Resultado de ejecución

```

Successful compilation of program Find
(BAKY) Llena el arreglo de 10 posiciones
(input) 5
(input) 67
(input) 34
(input) 23
(input) 12
(input) 45
(input) 34
(input) 23
(input) 12
(input) 89
(BAKY) Ingresa el número a buscar:
(input) 23
(BAKY) El index del número es:3

```

Multiplicación de Matrices

Código fuente

```

Baky MulMatrix;
var int a[18][18], b[18][18], c[18][18], n, k, m;

function void generateM()
var int i,j;
{
    from i to n do{
        from j to k do {
            a[i][j] = i*j*n;
        }
        j=0;
    }
    i=0;
    from i to k do{
        from j to m do {
            b[i][j] = i*j*k;
        }
        j=0;
    }
}
}

```



```

function void printM()
var int i,j;
{
    write("A: endl");
    from i to n do{
        write("[ ");
        from j to k do {
            write(a[i][j], " , ");
        }
        write("] endl");
        j=0;
    }
    i=0;
    write("B: endl");
    from i to k do{
        write("[ ");
        from j to m do {
            write(b[i][j], " , ");
        }
        write("] endl");
        j=0;
    }
    i=0;
    write("C: endl");
    from i to n do{
        write("[ ");
        from j to m do {
            write(c[i][j], " , ");
        }
        write("] endl");
        j=0;
    }
}

function void multiply()
var int i,j,l,num;
{
    from i to n do {

```

```

        j = 0;
        from j to m do {
            l = 0;
            num = 0;
            from l to k do {
                num = num + a[i][l] * b[l][j];
            }
            c[i][j] = num;
        }
    }
}

void Baky(){
    write("Las dimensiones de las matrices serán N*K y K*M endl");
    write("Ingresa N");
    read(n);
    write("Ingresa K");
    read(k);
    write("Ingresa M");
    read(m);
    generateM();
    multiply();
    printM();
}

```

Código intermedio

```

0 : ["<",5000,972,14000]
1 : ["gotoF",14000,null,17]
2 : ["<",5001,973,14001]
3 : ["gotoF",14001,null,14]
4 : ["ver",5000,0,"18"]
5 : ["ver",5001,0,"18"]
6 : ["*",5000,16000,10000]
7 : ["+",10000,5001,10000]
8 : ["sumP",10000,16001,15000]
9 : ["*",5000,5001,10001]
10 : ["*",10001,972,10002]

```

```
11 : ["=",15000,10002,null]
12 : ["+",5001,16002,5001]
13 : ["goto",null,null,2]
14 : ["=",5001,16001,null]
15 : ["+",5000,16002,5000]
16 : ["goto",null,null,0]
17 : ["=",5000,16001,null]
18 : ["<",5000,973,14002]
19 : ["gotoF",14002,null,35]
20 : ["<",5001,974,14003]
21 : ["gotoF",14003,null,32]
22 : ["ver",5000,0,"18"]
23 : ["ver",5001,0,"18"]
24 : ["*",5000,16000,10003]
25 : ["+",10003,5001,10003]
26 : ["sumP",10003,16003,15001]
27 : ["*",5000,5001,10004]
28 : ["*",10004,973,10005]
29 : ["=",15001,10005,null]
30 : ["+",5001,16002,5001]
31 : ["goto",null,null,20]
32 : ["=",5001,16001,null]
33 : ["+",5000,16002,5000]
34 : ["goto",null,null,18]
35 : ["popScope",null,null,null]
36 : ["write",18000,null,null]
37 : ["<",5000,972,14000]
38 : ["gotoF",14000,null,55]
39 : ["write",18001,null,null]
40 : ["<",5001,973,14001]
41 : ["gotoF",14001,null,51]
42 : ["ver",5000,0,"18"]
43 : ["ver",5001,0,"18"]
44 : ["*",5000,16000,10000]
45 : ["+",10000,5001,10000]
46 : ["sumP",10000,16001,15000]
47 : ["write",15000,null,null]
48 : ["write",18002,null,null]
49 : ["+",5001,16002,5001]
```

```
50 : ["goto",null,null,40]
51 : ["write",18003,null,null]
52 : ["=",5001,16001,null]
53 : ["+",5000,16002,5000]
54 : ["goto",null,null,37]
55 : ["=",5000,16001,null]
56 : ["write",18004,null,null]
57 : ["<",5000,973,14002]
58 : ["gotoF",14002,null,75]
59 : ["write",18001,null,null]
60 : ["<",5001,974,14003]
61 : ["gotoF",14003,null,71]
62 : ["ver",5000,0,"18"]
63 : ["ver",5001,0,"18"]
64 : ["*",5000,16000,10001]
65 : ["+",10001,5001,10001]
66 : ["sumP",10001,16003,15001]
67 : ["write",15001,null,null]
68 : ["write",18002,null,null]
69 : ["+",5001,16002,5001]
70 : ["goto",null,null,60]
71 : ["write",18003,null,null]
72 : ["=",5001,16001,null]
73 : ["+",5000,16002,5000]
74 : ["goto",null,null,57]
75 : ["=",5000,16001,null]
76 : ["write",18005,null,null]
77 : ["<",5000,972,14004]
78 : ["gotoF",14004,null,95]
79 : ["write",18001,null,null]
80 : ["<",5001,974,14005]
81 : ["gotoF",14005,null,91]
82 : ["ver",5000,0,"18"]
83 : ["ver",5001,0,"18"]
84 : ["*",5000,16000,10002]
85 : ["+",10002,5001,10002]
86 : ["sumP",10002,16004,15002]
87 : ["write",15002,null,null]
88 : ["write",18002,null,null]
```

```

89 : ["+",5001,16002,5001]
90 : ["goto",null,null,80]
91 : ["write",18003,null,null]
92 : ["=",5001,16001,null]
93 : ["+",5000,16002,5000]
94 : ["goto",null,null,77]
95 : ["popScope",null,null,null]
96 : ["<",5000,972,14000]
97 : ["gotoF",14000,null,130]
98 : ["=",5001,16001,null]
99 : ["<",5001,974,14001]
100 : ["gotoF",14001,null,128]
101 : ["=",5002,16001,null]
102 : ["=",5003,16001,null]
103 : ["<",5002,973,14002]
104 : ["gotoF",14002,null,120]
105 : ["ver",5000,0,"18"]
106 : ["ver",5002,0,"18"]
107 : ["*",5000,16000,10000]
108 : ["+",10000,5002,10000]
109 : ["sumP",10000,16001,15000]
110 : ["ver",5002,0,"18"]
111 : ["ver",5001,0,"18"]
112 : ["*",5002,16000,10001]
113 : ["+",10001,5001,10001]
114 : ["sumP",10001,16003,15001]
115 : ["*",15000,15001,10002]
116 : ["+",5003,10002,10003]
117 : ["=",5003,10003,null]
118 : ["+",5002,16002,5002]
119 : ["goto",null,null,103]
120 : ["ver",5000,0,"18"]
121 : ["ver",5001,0,"18"]
122 : ["*",5000,16000,10004]
123 : ["+",10004,5001,10004]
124 : ["sumP",10004,16004,15002]
125 : ["=",15002,5003,null]
126 : ["+",5001,16002,5001]
127 : ["goto",null,null,99]

```

```

128 : ["+",5000,16002,5000]
129 : ["goto",null,null,96]
130 : ["popScope",null,null,null]
131 : ["write",18006,null,null]
132 : ["write",18007,null,null]
133 : ["read",972,null,null]
134 : ["write",18008,null,null]
135 : ["read",973,null,null]
136 : ["write",18009,null,null]
137 : ["read",974,null,null]
138 : ["init","generateM",null,null]
139 : ["gosub",null,null,0]
140 : ["init","multiply",null,null]
141 : ["gosub",null,null,96]
142 : ["init","printM",null,null]
143 : ["gosub",null,null,36]
144 : ["end",null,null,null]

```

Resultado de ejecución

```

Successful compilation of program MulMatrix
(BAKY) Las dimensiones de las matrices serán N*K y K*M
(BAKY) Ingresa N
(input) 3
(BAKY) Ingresa K
(input) 2
(BAKY) Ingresa M
(input) 4
(BAKY) A:
(BAKY) [ 0 , 0 , ]
(BAKY) [ 0 , 3 , ]
(BAKY) [ 0 , 6 , ]
(BAKY) B:
(BAKY) [ 0 , 0 , 0 , 0 , ]
(BAKY) [ 0 , 2 , 4 , 6 , ]
(BAKY) C:
(BAKY) [ 0 , 0 , 0 , 0 , ]
(BAKY) [ 0 , 6 , 12 , 18 , ]
(BAKY) [ 0 , 12 , 24 , 36 , ]
(BAKY)

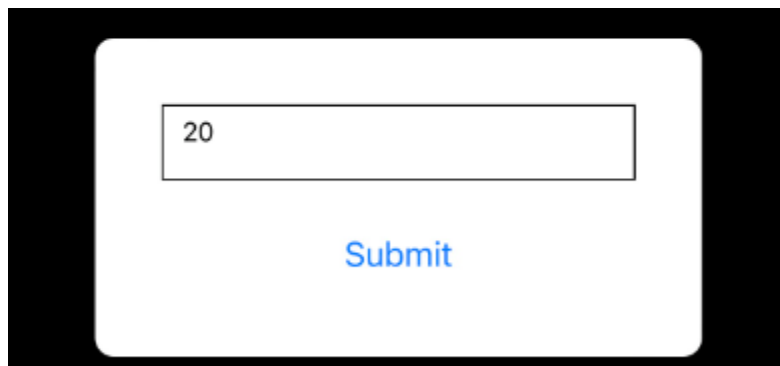
```

Compilación y Ejecución en el IDE

Código fuente en IDE

```
CODE          CONSOLE
1 Baky FibonacciR;
2
3 function int fibonacci(int n){
4     if(n<=2) {return 1;}
5     return fibonacci(n-2)+fibonacci(n-1);
6 }
7
8 void Baky()
9 var int n;
10 {
11     read(n);
12     n = fibonacci(n);
13     write(n);
14 }
15
16 // Serie de fib: 1 1 2 3 5 8 13 21 34 55 89 144 233
17
```

Inputs en el IDE



A screenshot of the IDE's input interface. It features a white rounded rectangle on a black background. Inside this rectangle is a text input field containing the number "20". Below the input field is a blue button with the text "Submit" in a lighter blue font.

Resultado de ejecución en el IDE

```
CODE          CONSOLE
(BAKY) 6765
```

DOCUMENTACIÓN DEL CÓDIGO DEL PROYECTO

Repositorio

<https://github.com/Ernesto1608/Baky>

Descripción de módulos

En Baky contamos con dos ambientes, uno en la terminal de una computadora y en la aplicación del IDE nativo a un dispositivo móvil. Como estos ambientes funcionan de manera distinta, tenemos esos ambientes divididos.

En el directorio base tenemos las siguientes carpetas:

- ide: donde se encuentra todo el código de la aplicación del IDE al igual que una copia de la carpeta src adaptada al ambiente nativo
- src: todos los archivos necesarios para la compilación y ejecución de un programa
- test_cases: el código en .bky de nuestros casos de prueba
- test_functional: el código en .bky de las pruebas de funcionalidad

Los archivos principales son:

- grammar.json: Archivo con la gramática y la sintaxis
- index.js: Este es el archivo principal, donde se manda el archivo (en el caso de la terminal) o el código (en el caso nativo) para tomarlo como input y hacer la compilación y ejecución del mismo.
- memory.js: Archivo encargado de manejar la memoria del programa. Se utiliza en compilación para obtener las direcciones de memoria y en ejecución para acceder a la memoria.
- quadruple.js: El trabajo de este archivo es manejar los cuádruplos del programa.
- semantic-constants.js: Define las constantes de operadores y tipo, al igual que el cubo semántico.
- semantics.js: En este archivo se maneja la memoria de compilación para la generación de cuádruplos y recursos para memoria virtual. Además se hacen las validaciones semánticas.
- vm.js: Archivo con la máquina virtual que procesa los cuádruplos, los cuales recibe como input, junto con todas propiedades de la clase Quadruple, como el objeto de semántica.

Extractos de código

getValueFromAddress, getValueFromPointer

En estas dos funciones se puede ver como obtenemos el valor de una dirección de memoria virtual de la memoria real.

En “getValueFromAddress” primero tiene una condición para ver si la dirección es de un pointer, en ese caso primero obtiene el valor del pointer y continua con ese nuevo valor (que sabemos que es una dirección). Después obtenemos el scope y el tipo con otras funciones auxiliares simples que solamente regresan un valor dependiendo del rango de la dirección. Finalmente con el scope, el tipo y el módulo de 1000 podemos obtener el valor. Es importante notar que cuando el scope es local (1) tenemos que hacer un .peek() para obtener el último arreglo en el stack de memoria local.

```
getValueFromAddress(address) {
    if(address >= 15000 && address < 16000)
        address = this.getValueFromPointer(address);
    const scopeMem = this.getScopeFromAddress(address);
    const typeMem = this.getTypeFromAddress(address);
    return scopeMem == 1 ?
        this.virtualMemory[scopeMem].peek()[typeMem][address % 1000] :
        this.virtualMemory[scopeMem][typeMem][address % 1000];
}

getValueFromPointer(address) {
    const scopeMem = this.getScopeFromAddress(address);
    const typeMem = this.getTypeFromAddress(address);
    return scopeMem == 1 ?
        this.virtualMemory[scopeMem].peek()[typeMem][address % 1000] :
        this.virtualMemory[scopeMem][typeMem][address % 1000];
}
```

processOperator

En esta función se genera el cuádruplo para una operación. Primero se obtienen los valores y tipos de los dos operandos de la pila de operandos y tipos. Después se obtiene el resultado de la operación y manda error si no es posible la operación. Se genera una nueva variable temporal con este tipo y se agrega uno a los recursos de este tipo. Finalmente se genera el cuádruplo con el operador, los operandos y la dirección a donde se va a asignar el valor y se agrega esa dirección con su tipo a sus pilas correspondientes.

```

processOperator(operator, line) {
  const [rightO, leftO] = [this.operands.pop(), this.operands.pop()];
  const [rightT, leftT] = [this.types.pop(), this.types.pop()];
  const type =
    this.semantics.semantiConstants.CUBE.validOperation(leftT, rightT,
      operator, line);
  const address = this.semantics.memory.assignMemory("local", type,
    true, 1);
  const typeMem = this.semantics.memory.getTypeFromAddress(address);
  this.semantics.functionsTable[this.semantics.scopeStack.peak()]
    .resources[typeMem]++;
  this.quadruples.push([operator, leftO, rightO, address]);
  this.operands.push(address);
  this.types.push(type);
  this.operators.pop();
}

```

getInput (IDE)

Esta es la función que se manda a llamar cuando se procesa el cuádruplo de lectura en el IDE. Se abre el modal del input y se regresa una promesa, se guarda la función para resolver la promesa en una variable global. Cuando se presiona el botón de “Submit” en el modal, se resuelven la promesa con el valor del input, se cierra el modal y se borra el valor del input.

```

function getInput() {
  setModalVisible(true);
  return new Promise((res) => {
    resolver = res;
  });
}

<Button
  title='Submit'
  onPress={() => {
    resolver(input);
    setModalVisible(false);
    setInput("");
  }}/>

```

II : Manual de Usuario

QUICK REFERENCE MANUAL

Ambiente de ejecución

Baky tiene dos maneras diferentes de utilizarse, puede ser en una computadora directamente desde la terminal nativa o desde un IDE adaptado para ejecutarse en un dispositivo móvil. Desafortunadamente para el segundo caso Baky aún no sale para ninguna tienda de apps por lo que actualmente se debe correr el IDE desde una computadora y conectarse mediante una red local.

En cualquier caso para poder utilizar Baky necesitamos tener instalados Node.js y NPM, existen versiones para macOS y Windows al igual que diferentes formas de poder instalarlos, para consultar una de ellas les dejamos el siguiente enlace.

<https://radixweb.com/blog/installing-npm-and-nodejs-on-windows-and-mac>

Una vez instalado Node.js y NPM necesitamos clonar el repositorio de Baky, para esto tenemos que tener instalado previamente git, de nuevo les dejamos un enlace con información al respecto.

<https://github.com/git-guides/install-git>

Con todo lo listado previamente es hora de instalar Baky en tu computadora, se necesita correr el siguiente comando:

```
git clone https://github.com/Ernesto1608/Baky.git
```

Una vez creada la carpeta de Baky en tu computadora se necesitan instalar todas las dependencias necesarias para su funcionamiento, para lograr esto dentro de la carpeta principal /Baky se corre el siguiente comando:

```
npm i
```

Al término de la instalación debes tener la carpeta /node_modules dentro de la carpeta principal /Baky.

Es importante mencionar que estas dependencias sirven únicamente para correr Baky en la terminal nativa si queremos utilizar el IDE se necesitan ejecutar los siguientes comandos:

```
//Acceder a la carpeta del IDE, en la carpeta principal /Baky ejecutar
```

```
cd ide
```

```
//Una vez situado en la carpeta /Baky/ide ejecutar
```

```
npm i
```

Nuevamente al término de la instalación se debe tener la carpeta /node_modules pero esta vez dentro de /Baky/ide.

Ahora viene un punto muy importante para la correcta ejecución del IDE, debido a temas de compatibilidad con React Native y las librerías utilizadas, se necesitan comentar algunas líneas de código dentro de los /Baky/node_modules las cuales son las siguientes:

Archivo - node_modules\jison\lib\jison.js

Líneas sin comentar:

```
1283     var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
1284     return exports.parser.parse(source);
```

Líneas comentadas:

```
1283     // var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
1284     // return exports.parser.parse(source);
```

Archivo - node_modules\lebnf-parser\transform-parser.js

Líneas sin comentar:

```
623     var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8"); 211 (gzipped: 165)
624     return exports.parser.parse(source);
```

Líneas comentadas:

```
623     // var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
624     // return exports.parser.parse(source);
```

Archivo - node_modules\lebnf-parser\parser.js

Líneas sin comentar:

```
796     var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
797     return exports.parser.parse(source);
```

Líneas comentadas:

```
796     // var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
797     // return exports.parser.parse(source);
```

Archivo - node_modules\lex-parser\lex-parser.js

Líneas sin comentar:

```
845     var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
846     return exports.parser.parse(source);
```

Líneas comentadas:

```
845     // var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
846     // return exports.parser.parse(source);
```

Ejecutar Baky

La ejecución de Baky es realmente sencilla, primero veamos cómo poder ejecutarlo en la terminal nativa del ordenador.

Situado en la carpeta principal de /Baky ejecutar el siguiente comando:

```
npm start
```

Una vez ejecutado, la terminal va a pedir por la ruta hacia el archivo del código fuente escrito en Baky, simplemente queda escribir esta ruta, presionar enter y listo Baky se ejecutará de forma automática.

Ejemplo:

```
PS C:\Users\ergarcia\Desktop\Baky> npm start

> Baky@1.0.0 start
> node src/index.js

prompt: filename: test_functional/FibonacciR.bky
Successful compilation of program FibonacciR
(input) 12
(BAKY) 144
PS C:\Users\ergarcia\Desktop\Baky> █
```

En este ejemplo yo tenía un archivo llamado FibonacciR.bky dentro de una carpeta llamada /test_functional, por lo que la ruta hacia el archivo es /test_functional/FibonacciR.bky.

Ahora veamos cómo ejecutar el IDE de Baky en tu dispositivo móvil. Para lograr esto nos ayudamos de una librería llamada Expo, es por esto que es necesario contar con la App Expo Go instalada en tu celular, esta la puedes encontrar tanto en android como en ios y luce así:



Una vez instalada la app dentro de la carpeta /Baky/ide ejecutamos el siguiente comando:

```
npm start
```

Se desplegará un código QR en la terminal como el siguiente:



Solo queda escanear el código QR desde tu dispositivo móvil y la app de Expo Go se abrirá sola y te desplegará el IDE, es importante mencionar que necesitas contar con una conexión a una red local y que tanto tu computadora como tu dispositivo móvil deben de estar conectados a la misma red.

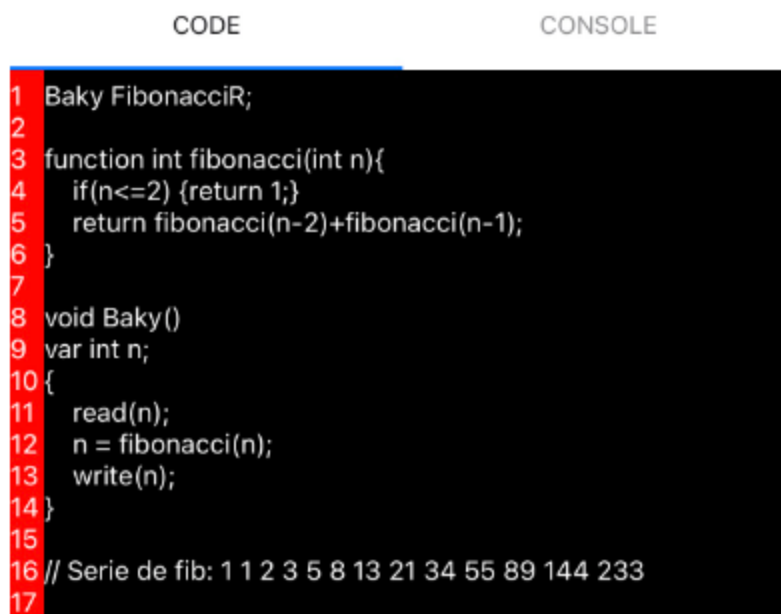
IDE de Baky

El IDE de Baky es muy simple e intuitivo, luce de la siguiente forma:



Tenemos dos pestañas en la parte superior, una para escribir código y otra para mostrar el resultado en la consola y pedir los inputs si es que se tienen, de igual manera en la parte inferior de la pestaña de código tenemos el botón de correr que es el que nos permite compilar y ejecutar nuestro código en Baky, por otra parte en la pestaña de consola tenemos el botón de clear, que nos permite limpiar los logs de nuestra consola.

Así luce el código en nuestro IDE

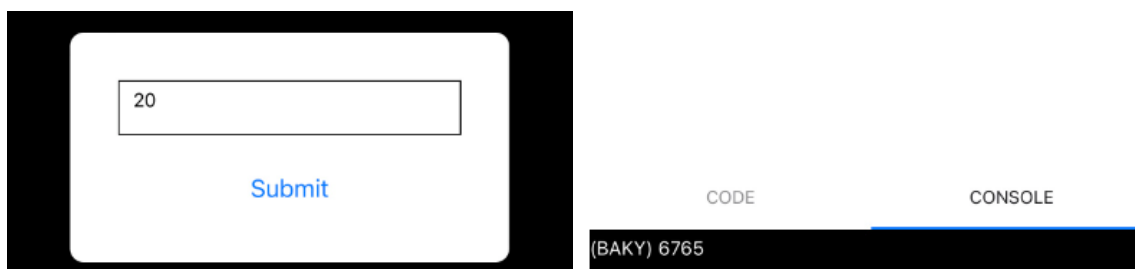


```

1 Baky FibonacciR;
2
3 function int fibonacci(int n){
4     if(n<=2) {return 1;}
5     return fibonacci(n-2)+fibonacci(n-1);
6 }
7
8 void Baky()
9 var int n;
10 {
11     read(n);
12     n = fibonacci(n);
13     write(n);
14 }
15
16 // Serie de fib: 1 1 2 3 5 8 13 21 34 55 89 144 233
17

```

Y así se piden los inputs y se muestran los logs en la consola



Submit

CODE
CONSOLE

(BAKY) 6765

Código en Baky

Baky cuenta con una sintaxis muy sencilla de comprender, aquí explicamos a grandes rasgos cómo programar en Baky pero si quieres aprender más te invitamos a leer la documentación del Baky que se encuentra en el repositorio de Git.

Todo programa en Baky empieza con el siguiente código:

```
Baky nombreDelPrograma;
```

En donde tu puedes nombrar como quieras a tu programa, seguido de esto tenemos una sección opcional de declaración de variables globales siendo su sintaxis y tipos soportados los siguientes:

```
var int i,j;
var double s[10],p;
var char c[2][2], w;
var string s2;
var boolean b;
```

Cómo podemos observar los tipos soportados son int, double, char, string y boolean, podemos declarar variables simples, arreglos y matrices de cada una, es importante mencionar que los indexes de los array y matrices empiezan en 0 a n-1.

Seguido de las variables globales viene una sección igual opcional de funciones parametrizadas cuya sintaxis es la siguiente:

```
//El valor de retorno de la función puede ser void,int,double,char,string,boolean
// Los parámetros son opcionales

function void nombreDeLaFuncion(int i, double s, char w, string m, boolean b)
//La sección de variables locales es opcional
var int q;
var char p;
{
    //Aquí se colocan los estatutos del lenguaje
}
```

Cómo podemos observar cada función tiene su ID propio, tipo de retorno, parámetros opcionales, variables locales opcionales y un bloque de estatutos, es importante mencionar que puedes definir cuantas funciones quieras dentro de esta sección.

Por último en nuestro programa tenemos la declaración de nuestra función principal, que es la que se manda a llamar al inicio de la ejecución, su sintaxis es la siguiente:

```
void Baky()
//Variables locales opcionales
int p;
{
    //Estatutos
}
```

Cómo se observa su nombre es Baky, es de tipo void y no tiene parámetros iniciales.

Pasemos a los estatutos que soporta Baky, para empezar tenemos la llamada a una función que tiene la siguiente estructura:

```
//Los parámetros a mandar deben coincidir con los esperados por la función
functionID(idVariable, idVariable, idArray[2]);
```

Como se observa llamar a una función es muy fácil y dentro de los parámetros podemos mandar variables, constantes o valores dentro de un array o matriz.

El siguiente estatuto es el de el return:

```
return exp;
```

El return se utiliza para regresar un valor esperado en una función con tipo diferente a void, podemos regresar el valor resultante de una expresión.

El siguiente estatuto es el read:

```
read(s,r[1]);
```

Este estatuto nos sirve para recibir valores desde la consola y guardarlos en una variable simple o array.

El siguiente estatuto es el write:

```
write(exp1, exp2);
```

Este estatuto nos sirve para escribir valores en la consola, podemos desplegar cualquier valor resultante de una expresión.

El siguiente estatuto es el if:

```
//El else es opcional
if(booleanExp){
    //Estatutos
} else {
    //Estatutos
}
```

Este estatuto hace una comparación inicial y de acuerdo con el resultado ejecuta el primer o el segundo bloque de estatutos, siendo el segundo bloque totalmente opcional y en dado caso solo ejecutaría el primer bloque si la condición se cumple.

El siguiente estatuto es el while:

```
while(booleanExp){
```

```
//Estatutos
}
```

Este estatuto hace una comparación inicial y ejecuta el bloque de estatutos de manera cíclica mientras la comparación resulte verdadera.

El siguiente estatuto es el for:

```
from idVariable to expIntODouble do {
    //Estatutos
}
```

Este estatuto requiere de una variable de tipo int o double de comienzo y una expresión que resulte en un int o double de fin, ejecuta de manera iterativa el bloque de estatutos mientras que el valor de idVariable sea menor que el de expIntODouble incrementando en uno el valor de idVariable al final de cada iteración.

El siguiente estatuto es el de asignación:

```
idVariable = exp;
```

Este estatuto asigna el valor resultante de la expresión al idVariable.

A lo largo del código hemos utilizado el término de expresión así que vamos a explicarlo más a detalle. Una expresión es una secuencia de operandos y operadores que al evaluarse de acuerdo a una jerarquía establecida dan un resultado de tipo variable de acuerdo a la expresión.

En el caso de Baky tenemos operadores aritméticos, relacionales y lógicos, siendo su definición y jerarquía la siguiente:

Operador	Descripción	Jerarquía
()	Paréntesis que indican prioridad en las operaciones.	1
/ *	División y Multiplicación	2
- +	Suma y Resta	3
>, <, >=, <=, !=, ==	Mayor, Menor, Mayor o Igual, Menor o Igual, Diferente, Igual	4
&&	And	5
	Or	6
=	Asignación	7

Por otro lado los operandos en Baky pueden ser cualquiera de los siguientes:

- Variable simple o no atómica
- Llamada a una función con tipo de retorno
- Constante entera
- Constante double
- Constante char
- Constante string
- Constante boolean

Esto permite que en Baky se puedan evaluar expresiones como las siguientes:

```
1*2.5+23--5/idVariable <= idArray[2]*idMatrix[1][0]

"salmon" == "ramon" || false && true && functionBoolean(int i)
```

Ejemplo de Código y Videos Demo

A continuación dejamos el ejemplo de un código que calcula el enésimo número de la serie de fibonacci de manera recursiva.

```
Baky FibonacciR;

function int fibonacci(int n){
    if(n<=2) {return 1;}
    return fibonacci(n-2)+fibonacci(n-1);
}

void Baky()
var int n;
{
    read(n);
    n = fibonacci(n);
    write(n);
}

// Serie de fib: 1 1 2 3 5 8 13 21 34 55 89 144 233
```

Demo Baky en Terminal: <https://youtu.be/Ci5OvhqXdUo>

Demo IDE de Baky: <https://youtu.be/iTlddhkHQC4>