Instituto Tecnológico y de Estudios Superiores de Monterrey

Diseño de Compiladores

TC3048

Documentación Final

Forever Alone

Profesores:

Dr. Héctor Ceballos
Ing. Elda G. Quiroga

Alumno:

Jesús Alfredo Mejía Gil A00816657

Grupo: 1

Monterrey, Nuevo León. Junio 3, 2020

Contenido

Descripción del Proyecto

Propósito

El propósito de este proyecto fue el diseñar y codificar un lenguaje de programación imperativo, tales como C, C++, Java, etc; para obtener el conocimiento acerca de toda la funcionalidad que hay detrás de un lenguaje y con estas enseñanzas poder adaptarse más fácilmente a lenguajes nuevos y escribir mejor código en distintos lenguajes.

Objetivos

El objetivo de este proyecto es crear un lenguaje de programación desde 0, empezando por la definición de la gramática y de la sintaxis hasta la implementación de arreglos unidimensionales y que con este lenguaje se pueda codificar problemas básicos y que compile con un tiempo de respuesta aceptable.

Alcance del proyecto

El lenguaje de programación diseñado incluye las características principales de un lenguaje de programación tales como:

- Declaración de variables y arreglos unidimensionales de diferente tipo
- Asignación a variables y arreglos unidimensionales
- Declaración de funciones de diferente tipo
- Llamadas a funciones
- Estatutos condicionales
- Estatutos de repetición
- Desplegar información en consola
- Escribir información por consola

Análisis de Requerimientos

Requerimientos Funcionales

- RF001: El lenguaje permitirá declarar variables de manera global y en las funciones especificando que tipo de variables son las que se crearán (int, float, char)
- RF002: El lenguaje permitirá declarar funciones donde se tendrá que especificar su tipo de retorno (int, float, char, void).
- RF003: El lenguaje permitirá hacer llamadas a funciones previamente declaradas.
- RF004: Las llamadas a funciones deberán tener el número exacto de parámetros o de lo contrario se mostrará un error.
- RF005: Las llamadas a funciones de tipo void no podrán ser asignadas y de lo contrario se mostrará un error.

- RF006: Las llamadas a funciones diferentes a void tendrán que ser asignadas o de lo contrario se mostrará un error.
- RF007: Se deberá utilizar una expresión entera para accesar los valores de un arreglo.
- RF008: Para accesar un arreglo se necesita un número menor o igual al tamaño del arreglo, de lo contrario se mostrará un mensaje de error correspondiente.
- RF009: El lenguaje permitirá mostrar información en la consola por medio del estatuto print.
- RF010: El lenguaje permitirá obtener información mediante la consola por medio del estatuto read.
- RF011: El lenguaje permitirá utilizar llevar a cabo condiciones por medio de los estatutos if y else.
- RF012: El lenguaje permitirá hacer ciclos por medio de los estatutos while y from-to.

Requerimientos No Funcionales

- RNF001: El lenguaje deberá ser fácil de utilizar para nuevos usuarios.
- RNF002: El lenguaje deberá tener un tiempo de compilación aceptable para los usuarios.
- RNF003: El lenguaje deberá mostrar mensajes de error adecuados para los casos donde se hayan cometidos fallas en el desarrollo de código.

Casos de uso generales

Caso de Uso: Hello World
ID: CU01
Actores: Usuario
Descripción: El usuario programará un Hello World por medio del lenguaje ForeverAlone.
Precondiciones: 1. Contar con los archivos del lenguaje ForeverAlone.
Flujo de eventos: 1. Crear archivo de texto. 2. En el archivo de texto escribir lo siguiente: *program helloworld;* *main(){* *print("Hello world");* *}* 3. Ejecutar archivo main.py con la ubicación de python3 y la ubicación del archivo de texto recién creado como parámetro.
Postcondiciones: 1. Se mostrará en pantalla el mensaje Hello world.

Caso de Uso: Tabla del 7
ID: CU02
Actores: Usuario
Descripción: El usuario escribirá un programa que muestre la tabla de multiplicar del 7 en consola por medio del lenguaje ForeverAlone.
Precondiciones: 1. Contar con los archivos del lenguaje ForeverAlone.
Flujo de eventos: 1. Crear archivo de texto. 2. En el archivo de texto escribir lo siguiente: *program foreveralone;* *var int i;* *main(){* *from i=1 to 11 do{* *print(i*7);* *}* *}* 3. Ejecutar archivo main.py con la ubicación de python3 y la ubicación del archivo de texto recién creado como parámetro.
Postcondiciones: 1. Se mostrará en pantalla el mensaje los números 7, 14, 21, 28, 35, 42, 49, 56, 63, 70

Caso de Uso: Aprobado o reprobado
ID: CU03
Actores: Usuario
Descripción: El usuario escribirá un programa que muestre si la calificación ingresada es aprobatorio o reprobatoria por medio del lenguaje ForeverAlone.
Precondiciones: 1. Contar con los archivos del lenguaje ForeverAlone.
Flujo de eventos: 1. Crear archivo de texto. 2. En el archivo de texto escribir lo siguiente: *program foreveralone;* *var int i;* *func void calificar(int x){* *if(x >= 70) then{* *print("Calificacion aprobatoria");* *}* *else{* *print("Calificacion reprobatoria");* *}* *}* *main(){* *read(i);* *calificar(i);*

3. Ejecutar archivo main.py con la ubicación de python3 y la ubicación del archivo de texto recién creado como parámetro.
4. Escribir un número entre el 0 y 100.

Postcondiciones:
2. Se mostrará en pantalla el mensaje correspondiente a la calificación escrita.

Casos de prueba

Caso de Prueba: Fibonacci Iterativo

ID: CP01

Actores: Usuario

Descripción:
Programa escrito de forma iterativa que mostrará la serie fibonacci hasta el número dado por el usuario.

Precondiciones:
1. Contar con los archivos del lenguaje ForeverAlone.

Flujo de eventos:
1. Crear archivo de texto.
2. En el archivo de texto escribir lo siguiente:

```
program Fibonacci;
var int numero, fibo1, fibo2, i;

main(){
    numero = 0;
    while(numero <= 1) do{
        print("Introduce un numero mayor que 1");
        read(numero);
    }
    print("Los", numero, "primeros numeros de la seria de Fibonacci son:");
    fibo1 = 0;
    fibo2 = 1;
    print(fibo1);
    from i=2 to numero+1 do{
        print(fibo2);
        fibo2 = fibo1 + fibo2;
        fibo1 = fibo2 - fibo1;
    }
}
```

3. Ejecutar archivo main.py con la ubicación de python3 y la ubicación del archivo de texto recién creado como parámetro.
4. Escribir un número.

Postcondiciones:
1. Se mostrará en pantalla la serie de Fibonacci con una cantidad de números igual a la cantidad ingresada.

Caso de Prueba: Fibonacci Recursivo
ID: CP02
Actores: Usuario
Descripción: Programa escrito de forma recursiva que mostrará la serie fibonacci hasta el número dado por el usuario.
Precondiciones: 1. Contar con los archivos del lenguaje ForeverAlone.
Flujo de eventos: 1. Crear archivo de texto. 2. En el archivo de texto escribir lo siguiente: *program foreveralone;* *var* *int numero, i, Arreglo[120];* *float prueba;* *func int fibonacci(int n){* *if((n == 0) || (n==1)) then {* *return(1);* *}* *else{* *return(fibonacci(n-1) + fibonacci(n-2));* *}* *}* *main(){* *while(numero <= 1) do{* *print("Introduce un numero mayor que 1");* *read(numero);* *}* *from i = 0 to numero + 1 do{* *print(fibonacci(i));* *}* *}* 3. Ejecutar archivo main.py con la ubicación de python3 y la ubicación del archivo de texto recién creado como parámetro. 4. Escribir un número.
Postcondiciones: 2. Se mostrará en pantalla la serie de Fibonacci con una cantidad de números igual a la cantidad ingresada.

Proceso general seguido

El proceso de desarrollo fue por medio de una metodología iterativa ágil en la cual se siguió el orden dictaminado por los profesores de la clase en fechas distintas a las estipuladas debido a conflictos con las fechas. Cada cambio considerable fue pusheado a un repositorio en Github por lo que se puede ver la lista de commits realizados en este proyecto. En los casos que hubiera bugs y fallos, se resolvieron antes de proseguir con el desarrollo de código para no impactar negativamente a los avances ya elaborados y a la nueva funcionalidad que se iba a agregar.

Bitácora:

Fecha	Avances
Miércoles 13 de mayo del 2020	• Creación del Scaner y Parser
Domingo 17 de mayo del 2020	• Arreglo de shift/reduce
Sábado 23 de mayo del 2020	• Tablas de variables. • Directorio de Funciones.
Domingo 24 de mayo del 2020	• Cubo Semántico. • Cuádruplo.
Domingo 31 de mayo del 2020	• Condicional.
Martes 2 de junio del 2020	• Tabla de direcciones. • Direcciones para variables globales, locales, temporales y constantes. • Ciclo From.
Miércoles 3 de junio del 2020	• Arreglos. • Llamada de funciones. • Reparación de bugs en la generación de cuádruplos.
Lunes 8 de junio del 2020	• Modificación de errores en las llamadas de funciones.
Martes 9 de junio del 2020	• Clase Memoria. • Máquina Virtual. • Arreglo de bugs en uso de arreglos y operador goto. • Modificación de las variables globales para aceptar que no se declaren.

Reflexión:

Por medio de este proyecto pude aprender mucho acerca de que es lo que conlleva un lenguaje de programación, todas las funciones y elementos que necesitan tener para poder funcionar apropiadamente que, aunque usualmente no los veas son de suma importancia, y pude desarrollar las diferentes rubricas para desarrollar mi propio lenguaje imperativo, que es cierto que es un lenguaje con varias limitantes debido al corto tiempo que se tiene en el semestre pero es funcional y es factible añadir nueva operatividad gracias al conocimiento adquirido en el semestre que los profesores me brindaron en cada clase.

Descripción del Lenguaje

Nombre del lenguaje

El lenguaje se llama “ForeverAlone” debido a que mi compañero y yo tuvimos que separarnos ya que la velocidad del internet en su tierra natal es muy pobre y esto nos hubiera causado muchas problemáticas al momento de tener que ponernos en contacto por video llamadas, aunque al final tristemente pienso que termino siendo más trabajo, pero debido a la situación del Covid no fue posible hacer algo al respecto.

Descripción genérica

El lenguaje ForeverAlone es un lenguaje imperativo que te permite hacer la mayoría de las cosas que un lenguaje imperativo común puede, pero tiene ciertas limitantes debido al alcance del proyecto. ForeverAlone te permite declarar variables globales que pueden ser accesadas desde cualquier función y que mantienen su valor, al declarar variables se necesita especificar el tipo, ya sea int, float o char. Igualmente se pueden declarar funciones que necesitan tener un valor de retorno valido (void, int, float, char) y en caso de ser diferente a void un estatuto de return que regresará el valor indicado por el programador. Antes de describir el cuerpo de la función se pueden declarar variables locales que solo podrán ser accesadas por la función en la que se declaró y en caso de tener el mismo nombre de una variable global se dará privilegio a la variable local. Cada programa necesitará de una función main en la cual no se pueden declarar variables por lo que es necesario crearlas de manera global si se necesita utilizar alguna variable en main. ForeverAlone también cuenta con los estatutos condicionales if-else, los ciclos while y from-to, funciones de read, print y declaración de arreglos unidimensionales.

Lista de errores

- Error de sintaxis generado por PLY.
- Stack overflow en la declaración de variables.
- Asignación de funciones de tipo void.
- Llamada a funciones de tipo diferente a void que no son asignadas.
- Llamada a una función no existente.
- Llamada a una función que es declarada posteriormente.
- Llamada a una función con el número incorrecto de parámetros.
- Llamada a una función con parámetros de un tipo distinto al declarado.
- Falta del estatuto return en funciones de tipo diferente a void.
- Creación de una función con nombre global.
- Creación de una función con nombre main.
- Declaración de una variable con el mismo nombre a una declarada previamente.
- Creación de una variable global con el nombre de alguna función declarada.
- Operaciones con variables no declaradas.
- Type mismatch con operaciones y tipos no compatibles.

- Asignación a una variable con un tipo diferente al de la variable declarada.
- Indexación de arreglo con un número mayor al tamaño.
- Indexación a una variable que no es arreglo.
- Nombrar una variable con alguna de las palabras reservadas del lenguaje.
- Strings con el carácter ~

Descripción del Compilador

Equipo de Cómputo, Lenguaje y Librerías

El equipo que se utilizó para el desarrollo del lenguaje de programación ForeverAlone fue una laptop Lenovo Legion Y520 con el sistema operativo de Windows 10 que cuenta con las siguientes características:

- Procesador Intel Core i5-7300HQ (4 núcleos, 6MB caché, 2.5GHz - 3.5GHz)
- Memoria RAM 8GB DDR4 2400MHz, 2 slots, máximo 16GB
- Almacenamiento 1TB 7MM 5400RPM
- Controlador gráfico Nvidia GeForce® GTX 1050 4G GDDR5

El lenguaje para el desarrollo del código del proyecto fue Python 3, específicamente la versión 3.7.3 que es con la que contaba el equipo previamente mencionado. También fue necesario el uso de la librería de PLY para el desarrollo de ForeverAlone, la cual adapta las herramientas de Lex y Yacc para ser utilizadas en Python y permitir el análisis de léxico y sintaxis, además de librerías estándar de Python tales como sys y fileinput.

Análisis de Léxico

Los elementos principales que requirieron del uso de patrones de construcciones y expresiones regulares complejas fueron los siguientes:

Token	Expresión Regular
C_FLOAT	\d+\.\d+
C_INT	\d+
C_STRING	"([^"\n]|(\\"))*"
C_CHAR	\'[a-zA-z_0-9]\'
NAME	[a-zA-Z_][a-zA-Z_0-9]*

Igualmente, su utilizo palabras reservadas idénticas al nombre de los tokens para los siguientes tokens:

- INT
- FLOAT
- CHAR
- VOID
- VAR
- PROGRAM
- FUNC
- RETURN
- READ
- PRINT
- IF
- ELSE
- 'WHILE
- FROM
- TO
- DO
- THEN
- MAIN

La siguiente lista representa los otros tokens utilizados y su código correspondiente:

Tokens	Código	Tokens	Código	Tokens	Código
PLUS	r'\+'	**MINUS**	r'\-'	**DIVIDE**	r'\/'
MULTIPLY	r'*'	**EQUALS**	r'\='	**COMMA**	r'\,'
SEMICOLON	r'\;'	**LPAREN**	r'\('	**RPAREN**	r'\)'
LBRACKET	r'\{'	**RBRACKET**	r'\}'	**AND**	r'\&\&'
OR	r'||'	**GREATER_THAN**	r'\>'	**LESS_THAN**	r'\<'
LSBRACKET	r'\['	**RSBRACKET**	r'\]'	**GREATER_OR_EQUAL**	r'\>\='
LESS_OR_EQUAL	r'\<\='	**IS_EQUAL**	r'\=\='	**NOT_EQUAL**	r'\!\='

Análisis de Sintaxis

La sintaxis utilizada se realizó en base a las siguientes definiciones:

- **PROGRAMA**: PROGRAM NAME SEMICOLON VARS F PRINCIPAL
- **VARS**: VAR V1 | empty
- **F**: FUNCION F | empty
- **V1:** TIPO VARIABLEDer V2 SEMICOLON V3
- **V2**: COMMA VARIABLEDer V2
- **V3**: V1 | empty
- **FUNCION**: FUNC TIPO_FUNCION NAME LPAREN PARAMS RPAREN VARS CUERPO
- **PARAMS**: TIPO NAME P1
- **P1**: P2 | empty
- **P2**: COMMA PARAMS P1
- **CUERPO**: LBRACKET E RBRACKET
- **E**: E1 | empty
- **E1**: ESTATUTO | ESTATUTO E1
- **TIPO**: INT | FLOAT | CHAR
- **TIPO_FUNCION**: INT | FLOAT | CHAR | VOID
- **PRINCIPAL**: MAIN LPAREN RPAREN CUERPO
- **VARIABLE**: NAME E2
- **E2**: LSBRACKET EXP RSBRACKET | empty
- **VARIABLEDer**: NAME E2Der
- **E2Der**: LSBRACKET C_INT RSBRACKET | empty
- **ESTATUTO**: ASIGNACION | LLAMADA | RETORNO | LECTURA | ESCRITURA | CONDICION | CICLO_W | CICLO_F
- **ASIGNACION**: VARIABLE EQUALS EXP SEMICOLON
- **LLAMADA**: NAME LPAREN E4 RPAREN SEMICOLON
- **E4**: EXP | EXP COMMA E4 | empty
- **LLAMADAF**: NAME LPAREN E4 RPAREN
- **RETORNO**: RETURN LPAREN EXP RPAREN SEMICOLON
- **LECTURA** : READ LPAREN V4 RPAREN SEMICOLON
- **V4** : VARIABLE COMMA V4 | VARIABLE
- **ESCRITURA**: PRINT LPAREN E3 RPAREN SEMICOLON
- **E3**: S_EXP | S_EXP COMMA E3
- **CONDICION**: IF LPAREN H_EXP RPAREN THEN CUERPO ELSE1
- **ELSE1**: ELSE CUERPO | empty
- **CICLO_W**: WHILE LPAREN H_EXP RPAREN DO CUERPO
- **CICLO_F**: FROM VARIABLE EQUALS EXP TO EXP DO CUERPO
- **H_EXP**: T_EXP | T_EXP OR H_EXP

- **T_EXP**: G_EXP | G_EXP AND T_EXP
- **G_EXP**: EXP B
- **B**: GREATER_OR_EQUAL G_EXP | LESS_OR_EQUAL G_EXP | GREATER_THAN G_EXP | LESS_THAN G_EXP | IS_EQUAL G_EXP | NOT_EQUAL G_EXP | empty
- **S_EXP**: C_STRING | EXP
- **EXP**: TERMINO T
- **T**: PLUS EXP | MINUS EXP | empty
- **TERMINO**: FACTOR F2
- **F2**: DIVIDE TERMINO | MULTIPLY TERMINO | empty
- **FACTOR**: LPAREN H_EXP RPAREN | C_INT | C_FLOAT | C_CHAR | VARIABLE | LLAMADAF
- **empty**:

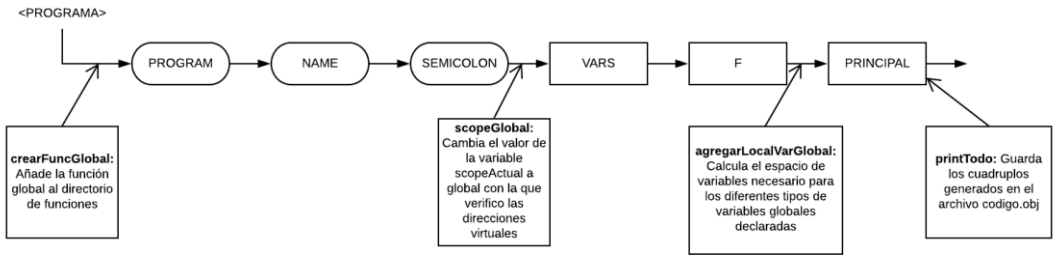
Generación de Código Intermedio y Análisis Semántico

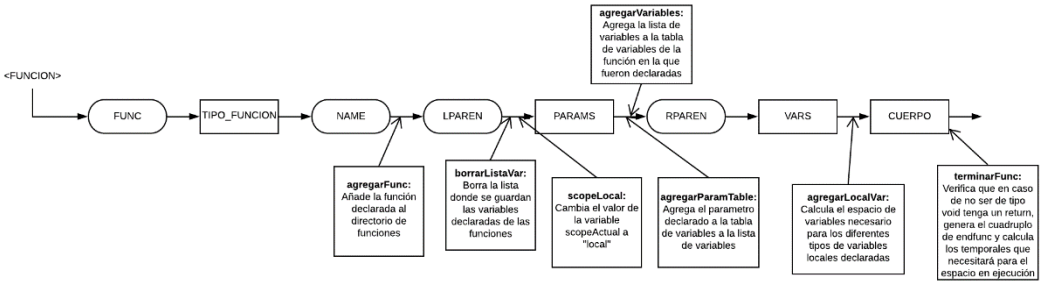
Código de Operación y Direcciones Virtuales

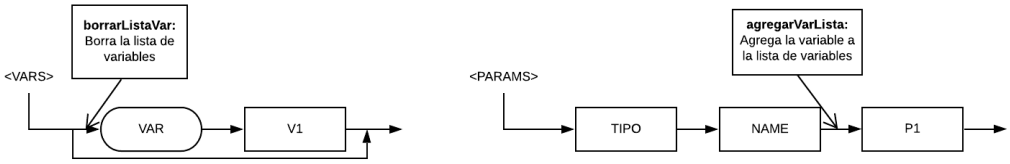
Las direcciones virtuales que se utilizaron para los diferentes alcances de las variables fueron las siguientes:

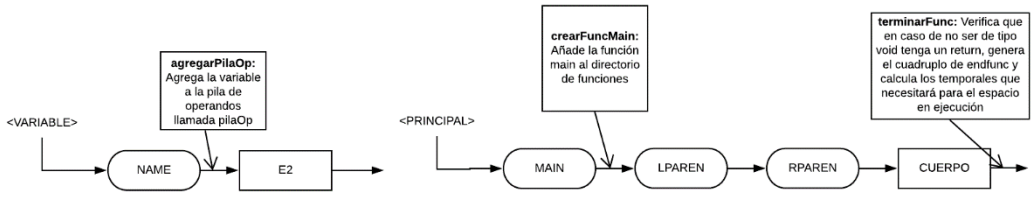
Alcance	Tipo	Rango de Direcciones Virtuales
Global	Int	1000 – 3999
	Float	4000 – 6999
	Char	7000 – 9999
Local	Int	10000 – 12999
	Float	13000 – 15999
	Char	16000 - 18999
Temporal	Int	19000 – 21999
	Float	22000 – 24999
	Char	25000 – 27999
	Bool	28000 – 30999
Constante	Int	31000 – 33999
	Float	34000 – 36999
	Char	37000 – 39999
	String	40000 - 42999

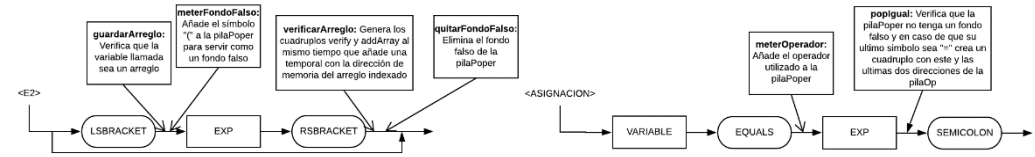
Diagramas de Sintaxis con Acciones

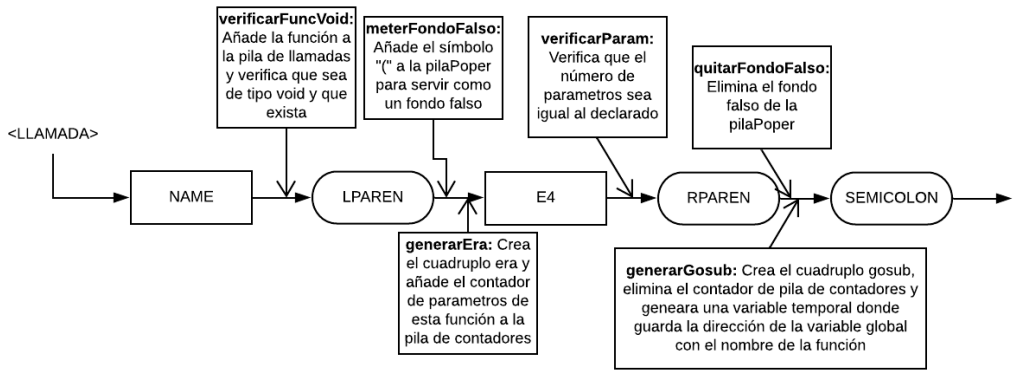


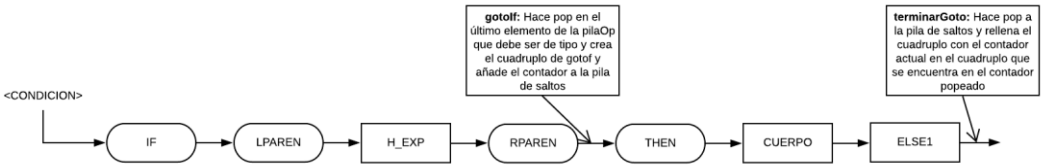

<FUNCION>
FUNC
TIPO_FUNCION
NAME
LPAREN
PARAMS
RPAREN
VARS
CUERPO
agregarVariables: Agrega la lista de variables a la tabla de variables de la función en la que fueron declaradas
agregarFunc: Añade la función declarada al directorio de funciones
borrarListaVar: Borra la lista donde se guardan las variables declaradas de las funciones
scopeLocal: Cambia el valor de la variable scopeActual a "local"
agregarParamTable: Agrega el parametro declarado a la tabla de variables a la lista de variables
agregarLocalVar: Calcula el espacio de variables necesario para los diferentes tipos de variables locales declaradas
terminarFunc: Verifica que en caso de no ser de tipo void tenga un return, genera el cuadruplo de endfunc y calcula los temporales que necesitará para el espacio en ejecución

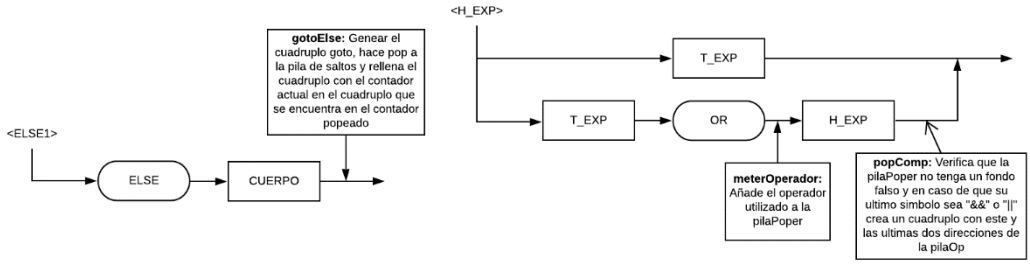

borrarListaVar: Borra la lista de variables
<VARS>
VAR
V1
agregarVarLista: Agrega la variable a la lista de variables
<PARAMS>
TIPO
NAME
P1

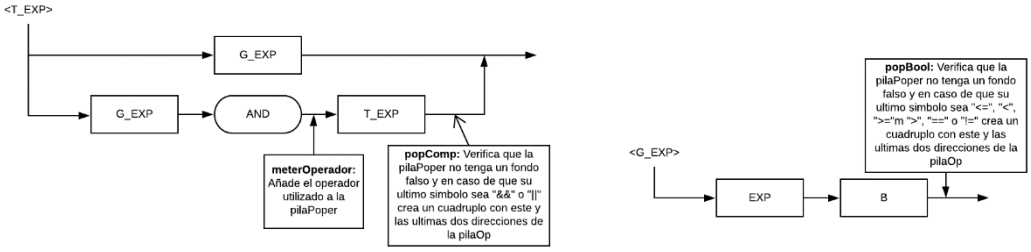

agregarPilaOp: Agrega la variable a la pila de operandos llamada pilaOp
<VARIABLE>
NAME
E2
crearFuncMain: Añade la función main al directorio de funciones
terminarFunc: Verifica que en caso de no ser de tipo void tenga un return, genera el cuadruplo de endfunc y calcula los temporales que necesitará para el espacio en ejecución
<PRINCIPAL>
MAIN
LPAREN
RPAREN
CUERPO

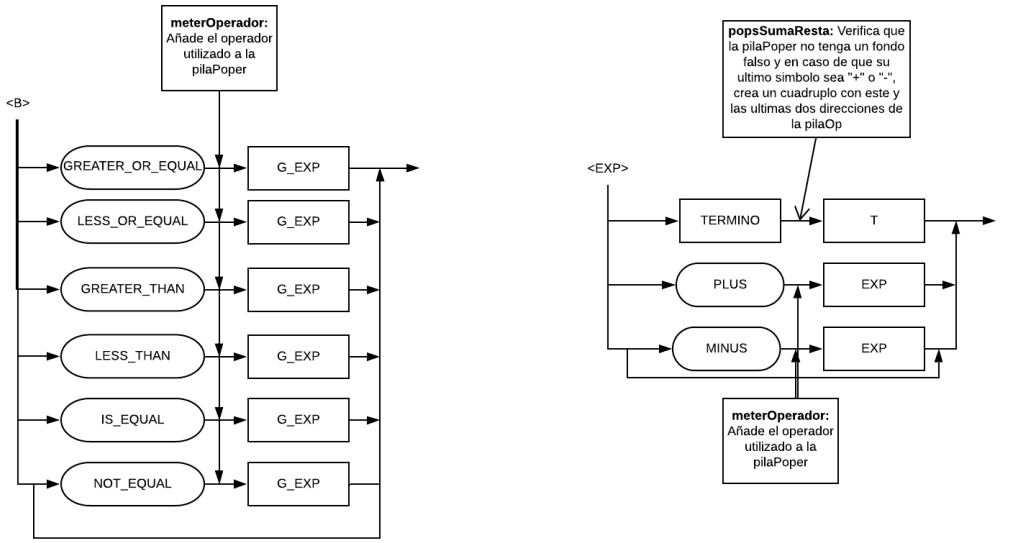

guardarArreglo: Verifica que la variable llamada sea un arreglo
meterFondoFalso: Añade el símbolo "(" a la pilaPoper para servir como un fondo falso
verificarArreglo: Genera los cuadruplos verify y addArray al mismo tiempo que añade una temporal con la dirección de memoria del arreglo indexado
quitarFondoFalso: Elimina el fondo falso de la pilaPoper
<E2>
LSBRACKET
EXP
RSBRACKET
meterOperador: Añade el operador utilizado a la pilaPoper
popIgual: Verifica que la pilaPoper no tenga un fondo falso y en caso de que su ultimo simbolo sea "=" crea un cuadruplo con este y las ultimas dos direcciones de la pilaOp
<ASIGNACION>
VARIABLE
EQUALS
EXP
SEMICOLON


verificarFuncVoid: Añade la función a la pila de llamadas y verifica que sea de tipo void y que exista
meterFondoFalso: Añade el símbolo "(" a la pilaPoper para servir como un fondo falso
verificarParam: Verifica que el número de parametros sea igual al declarado
quitarFondoFalso: Elimina el fondo falso de la pilaPoper
<LLAMADA>
NAME
LPAREN
E4
RPAREN
SEMICOLON
generarEra: Crea el cuadruplo era y añade el contador de parametros de esta función a la pila de contadores
generarGosub: Crea el cuadruplo gosub, elimina el contador de pila de contadores y geneara una variable temporal donde guarda la dirección de la variable global con el nombre de la función


<CONDICION>
IF
LPAREN
H_EXP
RPAREN
THEN
CUERPO
ELSE1
gotoIf: Hace pop en el último elemento de la pilaOp que debe ser de tipo y crea el cuadruplo de gotof y añade el contador a la pila de saltos
terminarGoto: Hace pop a la pila de saltos y rellena el cuadruplo con el contador actual en el cuadruplo que se encuentra en el contador popeado


<ELSE1>
ELSE
CUERPO
gotoElse: Genear el cuadruplo goto, hace pop a la pila de saltos y rellena el cuadruplo con el contador actual en el cuadruplo que se encuentra en el contador popeado
<H_EXP>
T_EXP
T_EXP
OR
H_EXP
meterOperador: Añade el operador utilizado a la pilaPoper
popComp: Verifica que la pilaPoper no tenga un fondo falso y en caso de que su ultimo simbolo sea "&&" o "||" crea un cuadruplo con este y las ultimas dos direcciones de la pilaOp


<T_EXP>
G_EXP
G_EXP
AND
T_EXP
meterOperador: Añade el operador utilizado a la pilaPoper
popComp: Verifica que la pilaPoper no tenga un fondo falso y en caso de que su ultimo simbolo sea "&&" o "||" crea un cuadruplo con este y las ultimas dos direcciones de la pilaOp
<G_EXP>
EXP
B
popBool: Verifica que la pilaPoper no tenga un fondo falso y en caso de que su ultimo simbolo sea "<=", "<", ">=m ">", "==" o "!=" crea un cuadruplo con este y las ultimas dos direcciones de la pilaOp


<B>
meterOperador: Añade el operador utilizado a la pilaPoper
GREATER_OR_EQUAL
LESS_OR_EQUAL
GREATER_THAN
LESS_THAN
IS_EQUAL
NOT_EQUAL
G_EXP
G_EXP
G_EXP
G_EXP
G_EXP
G_EXP
<EXP>
popsSumaResta: Verifica que la pilaPoper no tenga un fondo falso y en caso de que su ultimo simbolo sea "+" o "-", crea un cuadruplo con este y las ultimas dos direcciones de la pilaOp
TERMINO
T
PLUS
EXP
MINUS
EXP
meterOperador: Añade el operador utilizado a la pilaPoper

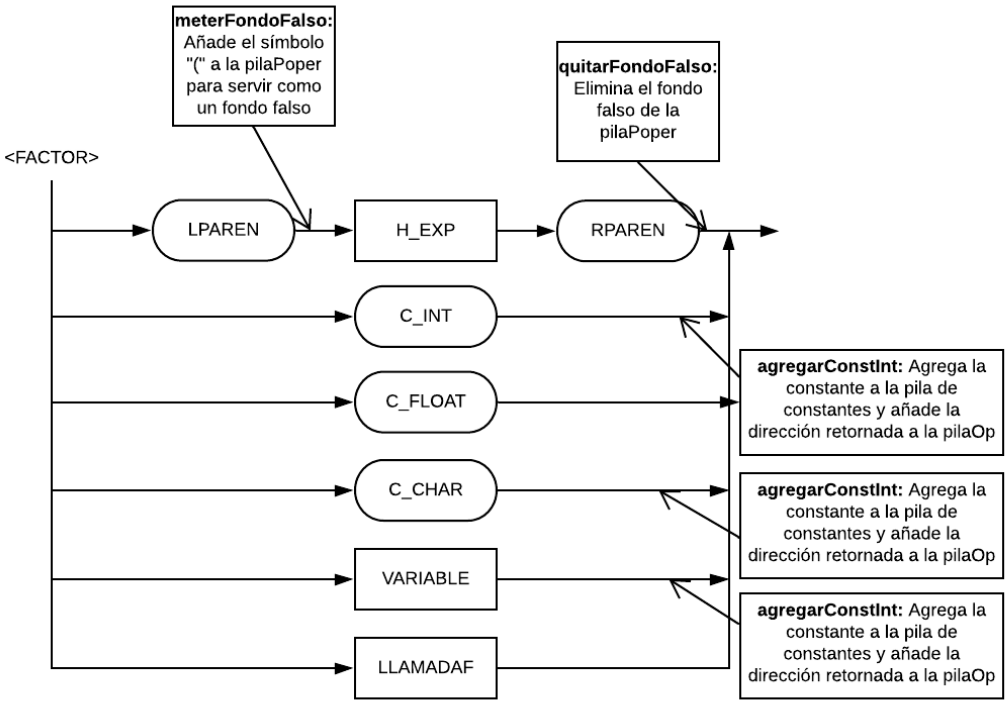


Tabla de consideraciones semánticas

LOp	ROp	+	-	*	/	==	¡=	>=	<=	>	<	||	&&	=
int	int	int	int	int	float	bool	bool	bool	bool	bool	bool	error	error	int
	float	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error	float
	char	error	error	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	error	error	error	error	error	error	error	error	error
float	int	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error	error
	float	float	float	float	float	bool	bool	bool	bool	bool	bool	error	error	float
	char	error	error	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	error	error	error	error	error	error	error	error	error
char	int	error	error	error	error	error	error	error	error	error	error	error	error	error
	float	error	error	error	error	error	error	error	error	error	error	error	error	error
	char	error	error	error	error	bool	bool	error	error	error	error	error	error	char
	bool	error	error	error	error	error	error	error	error	error	error	error	error	error
bool	int	error	error	error	error	error	error	error	error	error	error	error	error	error
	float	error	error	error	error	error	error	error	error	error	error	error	error	error
	char	error	error	error	error	error	error	error	error	error	error	error	error	error
	bool	error	error	error	error	bool	bool	error	error	error	error	bool	bool	bool

Proceso de Administración de Memoria

Directorio de Funciones: La clase de DirFunciones es un diccionario de diccionarios, siendo la primera llave el nombre de la función y las demás llaves los siguientes strings:

- returnType
- varTable
- paramTable
- numberParams
- numLocalInt
- numLocalFloat
- numLocalChar
- numTempInt
- numTempFloat
- numTempChar
- numTempBool
- memoriaTam
- actualCuad

Tabla de Variables: La clase de TablaVariables es un diccionario de diccionario, siendo la primera llave el nombre de la variable y las demás llaves los siguientes strings:

- type
- dir
- isArray
- arraySize

Cuádruplos: Para la implementación de cuádruplos se creó la clase Cuadruplo la cual es un arreglo de arreglos, los arreglos que se añaden al arreglo principal contienen los elementos:

- operador
- op1
- op2
- res

Tabla de Constantes: Para esta implementación se utilizo una clase llamada TablaConstantes la cual es un diccionario de diccionarios, la principal llave siendo la constante y las llaves secundarios son los strings “type” y “dir”.

Cubo Semántico: El cubo semántico se implemento por medio de una clase llamada CuboSemantico la cual es un diccionario de diccionario de diccionarios donde las primeras llaves son int, float, char y bool; las llaves secundarias son int, float, char y bool; y las ultimas llaves son los operadores que el lenguaje utiliza: +, -, *, /, ==, ¡=, >=, <=, >, <, ||, && y =.

Descripción de la Máquina Virtual

Proceso de Administración de Memoria en ejecución

La asociación entre las direcciones virtuales de compilación y de ejecución se realizó por medio de una clase Memoria la cual contiene 4 diccionarios, una para cada tipo de variable posible (int, float, char, bool). En la máquina virtual se instancian tres memorias, la memoria global, memoria local y memoria temporal; a estas memorias se les asignan las direcciones en base a la dirección previa que tenían en compilación, es decir, si tuviera la dirección 19003 entonces se creará un espacio en la memoria temporal en su diccionario de int y creará una llave con el valor 3 ya que la declaración de temporales enteras en compilación es a partir de la dirección 19000. Para traducir estas direcciones se crearon las funciones regresarVal y asignarVal; las cuales comparan la dirección de memoria con los rangos que se utilizaron en compilación y en base a estos regresa el valor de ese espacio o asigna un nuevo valor en ese espacio respectivamente.

Pruebas del Funcionamiento del Lenguaje

Prueba Fibonnaci Iterativo

La primera prueba realizada fue la implementación del calculo de la serie Fibonacci en base a un número ingresado por el usuario y el resultado es desplegado en la consola, de tal

forma que se pueda verificar la utilización de los estatutos print y read al mismo tiempo que se prueba el ciclo from-to. Para ejecutar esta prueba se utilizó el siguiente código:

```
program Fibonacci;
var int numero, fibo1, fibo2, i;

main(){
    numero = 0;
    while(numero <= 1) do{
        print("Introduce un numero mayor que 1");
        read(numero);
    }
    print("Los", numero, "primeros numeros de la seria de Fibonacci son:");
    fibo1 = 0;
    fibo2 = 1;
    print(fibo1);
    from i=2 to numero+1 do{
        print(fibo2);
        fibo2 = fibo1 + fibo2;
        fibo1 = fibo2 - fibo1;
    }
}
```

La ejecución y el resultado fueron los siguientes:

```
C:\Users\chuch\Desktop\PracticasWeb\Compilador>C:/Users/chuch/Anaconda3/python.exe ForeverAlone.py pruebaFib.txt
Introduce un numero mayor que 1
10
Los
10
primeros numeros de la seria de Fibonacci son:
0
1
1
2
3
5
8
13
21
34
```

Prueba Factorial Recursivo

Para la segunda prueba se programó el código para que en base a un número que escriba el usuario en consola se despliegue el factorial resultante de ese valor y fue elaborado de manera recursiva para probar las llamadas a funciones dentro de una misma función así como el estatuto de return y los operandos boléanos de and y or. El código utilizado fue el siguiente:

```
program factorial;
var int num;

func int fact(int x){
    if((x == 0) || (x==1)) then {
        return(1);
    }
    else{
        return(x * fact(x - 1));
    }
}


main(){
    print("Ingresa el numero a evaluar con factorial");
    read(num);
    fact(num);
    print(fact(num));
}
```

El resultado de la ejecución se muestra en la siguiente imagen:

```
C:\Users\chuch\Desktop\PracticasWeb\Compilador>C:/Users/chuch/Anaconda3/python.exe ForeverAlone.py pruebaFactRec.txt
Ingresa el numero a evaluar con factorial
5
120
```

Prueba Find en Arreglo

La última prueba consiste en encontrar un entero ingresado por el usuario dentro de los valores de un arreglo y en caso de encontrarlo imprimir un mensaje de éxito y en lo contrario mostrar un error. Con esta prueba se verifica las funciones de tipo char y el funcionamiento correcto en la asignación y el acceso a los valores dentro de arreglos, así como el funcionamiento del estatuto condicional if-else. Para esta prueba se utilizo el siguiente código:

```
program find;
var int i, Arreglo[100], z;

func char find(int y, int lon)
var char boolVal;
{
    boolVal = 'F';
    from i=0 to lon do{
        if(Arreglo[i] == y)then{
            boolVal = 'T';
```

```
        }
    }
    return(boolVal);
}

main(){
    from i=0 to 100 do{
        Arreglo[i] = 1000 - i;
    }
    print("Ingresa entero a encontrar");
    read(z);
    if(find(z, 100) == 'T') then{
        print("Se encontra el entero");
    }
    else{
        print("No se encontro el entero");
    }

}
```

El resultado de la última prueba fue el siguiente:

```
C:\Users\chuch\Desktop\PracticasWeb\Compilador>C:/Users/chuch/Anaconda3/python.exe ForeverAlone.py find.txt
Ingresa entero a encontrar
910
Se encontra el entero
```