Tecnológico de Monterrey

3 de junio del 2020

Diseño de Compiladores

!Jitters

Alejandro Lara Cuevas
A0080153

Cristina De León Martínez
A01282017

Índice

1. Descripción del Proyecto

1.1. Propósito, Objetivos y Alcance del Proyecto.

El propósito de este proyecto es es demostrar los conocimientos adquiridos en las clase de de *Diseño de Compiladores 2020* a través de la realización de un lenguaje básico que puede realizar operaciones entre variables de dimensiones. El objetivo es poder realizar de manera correcta el compilador, con los requerimientos sugeridos y de manera limpia. Se espera que el compilador tenga las operaciones básicas y como plus, que pueda facilitar el realizamiento de operaciones con arreglos y matrices.

1.2. Análisis de Requerimientos y Casos de Uso generales.

El lenguaje debe contar con expresiones aritméticas/lógicas y relacionales, estatutos de interacción (LEER y ESCRIBIR), estatutos de Control de Flujo (Ciclos y Decisiones), elementos de cambio de contexto (Funciones parametrizables) y manejo de elementos uni y bidimensionales. Se debe poder realizar algunas operaciones entre variables dimensionadas (+, -, *), así como operaciones especiales para obtener la determinante y poder asignar una variable dimensionada a otra.

El caso de uso principal de este lenguaje es leer el programa dentro de un editor de texto con terminación .jitt , este programa tendrá que seguir una sintaxis especificada en el manual de usuario para así poder ser compilable.

1.3. Descripción de los principales Test Cases.

Se desarrollaron distintos casos de prueba que tienen como propósito demostrar el funcionamiento correcto de nuestra máquina virtual y la generación de código intermedio. Estos son lo principales archivos de prueba que creamos en conjunto con el funcionamiento que se espera demostrar:

- helloWorld.jitt : Demostrar que el programa compila de manera adecuada, se demuestra el funcionamiento de LEER y ESCRIBIR.
- arrays.jitt : Demostrar que el programa es capaz de manejar arreglos; declarar, asignar y conseguir su valor.
- matrices.jitt : Demostrar que el programa es capaz de manejar matrices; declarar, asignar y conseguir su valor.
- funciones.jitt : Demostrar la capacidad de ejecutar funciones void y de retorno.
- if-ciclos.jitt : Demostrar la capacidad de ejecutar ciclos y decisiones de pasada.
- fibonacci.jitt : Demostrar que el programa es capaz manejar funciones parametrizadas recursivas.
-

1.4. Proceso general para el desarrollo del proyecto

Este lenguaje se fue desarrollando a lo largo del semestre Febrero - Junio 2020, se comenzó todo el proceso en el mes de Marzo. A continuación el calendario que estuvimos siguiendo para la elaboración de este proyecto.

Semana	L	Mi	J	AVANCE	Contenido esperado de la entrega
Mrzo-Abril (30 - 3)	30	**1**	3	**#1**	Análisis de Léxico y Sintaxis *(**solo entregan quienes cuenten ya con propuesta de proyecto APROBADA **)*
Abril (6 - 10)	6	**8**	10	**--**	**SEMANA SANTA**
Abril (13 - 17)	13	**15**	17	**#2**	Semántica Básica de Variables: Directorio de Procedimientos y Tablas de Variables
Abril (20- 24)	25	**27**	28	**#3**	Semántica Básica de Expresiones: Tabla de Consideraciones semánticas (Cubo Semántico) Generacion de Código de Expresiones Aritméticas y estatutos secuenciales: Asignación, Lectura, etc.
Abril (27 - 1)	27	**29**	1	**#4**	Generacion de Código de Estatutos Condicionales: Decisiones/Ciclos
Mayo (4 – 8)	4	**6**	8	**#5**	Generacion de Código de Funciones
Mayo (11 – 15)	11	**13**	15	**#6**	Mapa de Memoria de Ejecución para la Máquina Virtual Máquina Virtual: Ejecución de Expresiones Aritméticas y Estatutos Secuenciales
Mayo (18 – 22)	18	**20**	22	**#7**	Generacion de Código de Arreglos /Tipos estructurados Máquina Virtual: Ejecución de Estatutos Condicionales
Mayo (25 – 29)	25	**27**	29	**#8**	1era versión de la Documentación Generacion de Código y Máquina Virtual para una parte de la aplicación particular
Junio (1– 5)	1	**3**	5	**FINAL**	***ENTREGA FINAL DEL PROYECTO JUNIO 3, 12:00pm***

Diseño de Compiladores--PROGRAMACIÓN DE AVANCES
Semestre Febrero-Junio 2020

Estuvimos al día en casi todas las entregas, en la sección de anexos, podrán visualizar más a detalle que se entregó cada semana de Marzo a Junio. Asimismo, para mantener control de versiones de código utilizamos git: https://github.com/LiteMasteralex/-Jitters.git . En este repositorio se pueden ver la elaboración del código a lo largo del proyecto y también ahí se encuentran las bitácoras. En general, el calendario ejemplifica lo que se obtuvo por avance, también tuvimos sesiones de *pair programming* a través de google meets mínimo 2 sesiones por cada avance. También utilizamos whatsapp y drive como medios para organización y establecimiento de datos.

Aún siendo un proyecto "no original" (ya que la idea fue dada por los maestros) siento que fue un proyecto interesante, retador y enriquecedor. Lo interesante fue en general como se codifica, funciona e interactúa con un lenguaje para la generación de un lenguaje propio. En mi vida hubiera investigado cómo hacerlo y es por eso que me interesó la realización de este proyecto. Lo retador fue comprender TODO esto a la par con lo visto en clase y lograr aplicarlo a nuestro proyecto. La verdad es que mi compañero siempre iba un paso adelante de mi en la comprensión del funcionamiento, pero el uso de las sesiones colaborativas me ayudaba para comprender mejor las cosas e incluso lograr corregir y apoyar como equipo. Luego, por último, pero no menos importante, para mi el proyecto fue muy enriquecedor por los resultados. Definitivamente se ve lo visto en clase aplicado en este proyecto, logré aprender Python y nuevas cosas de este mismo lenguaje. También practicamos de una manera inmnesa el uso de pair

programming y siento que eso es algo muy importante para la vida laboral, lograr trabajar en conjunto con alguien en un mismo código y poder comprender el código de uno y el otro para así poder mejorar el producto. Disfruté mucho esta clase y me hizo darme cuenta que verdaderamente estoy donde debo estar, como Ingeniera en Tecnología Computacionales.

Cristina Nohemí De León Martínez
A01282017

Este proyecto fue retador en el sentido que nos pidió pensar fuera del entorno de programación que siempre hacemos. Esto me refiero a que siempre que tratamos de resolver un problema en una situación normal como la del trabajo o otros proyectos, tenemos una idea general de lo que se tienen que hacer y cómo podría enfrentarme al problema. A la hora de que se nos presentó la problemática de definir el 'lenguaje de programación' que tendríamos que desarrollar la verdad me quede perplejo, no tenia la minima idea de cómo proceder. Esto debido en parte a que no estaba seguro de las implicaciones y el alcance del proyecto, y la verdad en mi mente era mucho más de lo que pensaba. Una vez que fuimos viendo los temas en clase y que fui entendido lo sencillo que era cada parte individual del compilador pude entender com ir haciendo el compilador. Esta clase siempre tenía preocupación por lo abrumador que sonaba tener que construir un compilador para un lenguaje en específico, pero una vez que los profesores nos dieron una descripción específica a desarrollar y el alcance del proyecto se dio a entender , mis preocupaciones se fueron disolviendo. Esto pasó en muchas de las entregas, cada cosa sonaba difícil de resolver pero una vez que entiendas que tienes que hacer todo se volvía sencillo, y no es que el proyecto en su totalidad es fácil, sino más bien por cómo se desarrolla la clase las partes individuales son fáciles de entender.

Javier Alejandro Lara Cuevas
A00080153

2. Descripción del Lenguaje

2.1. Nombre del Lenguaje

!Jitters (NOT Jitters)

2.2. Descripción genérica de las carácteristicas del lenguaje

Nuestro lenguaje es un lenguaje expresivo e basado en compilador básico, con la característica especial de poder manejar variables dimensionadas (hasta 2), y operaciones especiales sobre estas variables dimensionadas. Aunque nuestro código está construido en python este utiliza la técnica de compilar y crear un archivo que se corre para la ejecución del programa. Esto nos permitió hacerlo un lenguaje de alto-nivel que es sencillo de entender y utilizar. Otra característica importante es que el lenguaje distingue entre mayusculas y minusculas y las llamadas se tienen que tener la puntuación correcta para funcionar.

2.3. Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución.

Compilación

- Syntax Error
- Out of Memory
- Undefined Variable
- Semantics Error
- Already Declared Variable
- Already Declared Function
- Return Type Mismatch
- Type Mismatch
- Index Out of Range
- Non Indexable Variables
- Index Type Mismatch
- Wrong Number of Parameters
- Parameter Type Mismatch
- Wrong Function Return

Ejecución

- Input Type Error
- Index Out of Range
- Out of Memory

3. Descripción del compilador

3.1. Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.

Se utilizaron dos sistemas operativos: Windows y MacOS, pero el proyecto puede ser utilizado en cualquier SO que pueda correr Python3.
El lenguaje que utilizamos (Python3) nos permitió la generación de Código intermedio utilizando su librería de PLY, que es una implementación de herramientas de análisis lex y yacc para Python.

En resumen:

Sistema Operativo: Windows, MacOS, Linux
Lenguaje: Python3
Utilerias: ply.yacc, ply.LEX

3.2. Descripción del Análisis de Léxico.

3.2.1. Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales.

```
ID = r'[a-zA-Z_][a-zA-Z_0-9]*'
CTEF = r'[0-9]+.[0-9]+'
CTEI = r'[0-9]+'
CTECH = r"\'[A-Za-z]\'"
comment = r'\#.*'
```

3.2.2. Enumeración de los "tokens" del lenguaje y su código asociado.

```
reservadas = {
   'Programa': 'PROGRAMA',
   'funcion': 'FUNCION',
   'principal': 'PRINCIPAL',
   'var': 'VAR',
   'si': 'SI',
   'sino': 'SINO',
   'entonces': 'ENTONCES',
   'regresa': 'REGRESA',
   'desde': 'DESDE',
   'mientras': 'MIENTRAS',
   'haz': 'HAZ',
   'hasta': 'HASTA',
   'hacer': 'HACER',
   'escribe': 'ESCRIBE',
   'lee': 'LEE',
   'int': 'INT',
   'float': 'FLOAT',
   'char': 'CHAR',
   'void': 'VOID',
}
```

```
tokens = (
   'PROGRAMA', 'FUNCION', 'PRINCIPAL', 'VAR', #reservados
   'SI', 'SINO', 'ENTONCES', 'REGRESA', 'DESDE', 'MIENTRAS','HAZ', 'HASTA', 'HACER',
   # para ciclos for, while, if else
```

```
  'ESCRIBE', 'LEE', # funciones internas (print, read)
  'INT', 'FLOAT', 'CHAR', 'VOID', # relacionado con var y tipos de retorno
  'SEMICOLON', 'LCORCHETE', 'RCORCHETE', 'LPAREN', 'RPAREN', 'COLON', 'COMMA',
'LBRACKET', 'RBRACKET', # Delimitadores
  'DETERMINANTE', 'TRANSPUESTA', 'INVERSA', # operadores especiales
  'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'ASIGNAR', # operadores aritméticos
  'GREATER', 'LESS', 'EQUALS', 'NOTEQUAL', 'AND', 'OR', # operadores lógicos
  'ID', 'CTEF', 'CTEI','CTECH', 'STRING'  #constantes
  )
```

3.3. Descripción del Análisis de Sintaxis

3.3.1. Gramática Formal empleada para representar las estructuras sintácticas (Sin "codificar").

<Programa> → **Programa** id **;** <Variables> <Funciones> **principal () {** <Estatuto> **}**

<Variables> → **var** <Tipo> **:** <Lista_Ids>**;** <Variables_1> | ε
<Variables_1> → <Tipo> **:** <Lista_Ids>**;** <Variables_1> | ε

<Lista_Ids> → <Identificadores> <Lista>
<Lista> → , <Identificadores> <Lista> | ε

<Funciones> → **funcion** <Tipo_retorno> id (<Parametros>) **;** <Variables>
{ <Estatuto> } <Funciones> | ε

<Tipo_retorno> → **INT** | **FLOAT** | **CHAR** | **VOID**
<Tipo> → **INT** | **FLOAT** | **CHAR**

<Parametros> → <Tipo> **:** id <Variables_2> | ε
<Variables_2> → , <Tipo> **:** id <Variables_2> | ε

<Estatuto> → <Estatuto_1><Estatuto> | ε
<Estatuto_1> → <Asignacion> | <Funcion_Void> | <Retorno> | <Lectura> |
<Escritura> | <Decision> | <Repeticion>
<Asignacion> → <IdentificadoreExp> **=** <Expresiones> **;**

<Identificadores> → id **[**CTEI**]** | id **[**CTEI**][**CTEI**]** | id
<IdentificadorExp> → id **[**<Expresiones>**]** | id **[**<Expresiones>**][**<Expresiones>**]** | id

<Terminos> → <IdentificadorExp> | <Funcion_Retorno>| <Var_CTE> |
(<Expresiones>)

<Especiales> → <Terminos> | <Terminos> <Especiales_1>
<Especiales_1> → **$** | **¡** | **?**

<Factores> → <Especiales> | <Especiales> <Factores_1> <Factores>
<Factores_1> → * | /

<Aritmeticos> → <Factores> | <Factores> <Aritmeticos_1> <Aritmeticos>
<Aritmeticos_1> → + | -

<Logicos> → <Aritmeticos> | <Aritmeticos> <Logicos_1> <Logicos>
<Logicos_1> → < | > | == | !=

<Expresiones> → <Logicos> | <Logicos> <Expresiones_1> <Expresiones>
<Expresiones_1> → & | |

<Funcion_Retorno> → id(lista_exp);
<Funcion_Void> → id(lista_exp);

<lista_exp> → <Expresiones> <lista_exp_1> | ε
<lista_exp_1> → , <Expresiones> <lista_exp_1> | ε

<Retorno> → **regresa(**<Expresiones>**);**

<Lectura> → **lee(**<Lectura_1>**);**
<Lectura_1> → <IdentificadorExp> <Lectura_2>
<Lectura_2> → , <Lectura_1> | ε

<Escritura> → **escribe(** <Escritura_1>**);**
<Escritura_1> → <Imprimir> <Escritura_2>
<Escritura_2> → , <Escritura_1> | ε
<Imprimir> → **STRING** | <Expresiones>

<Decision> → **si (**<Expresiones>**) entonces {** <Estatuto> **}** <Decision_1>
<Decision_1> → **sino {** <Estatuto> **}** | ε

<Repeticion> → <Repeticion_Cond> | <Repeticion_No_Cond>
<Repeticion_Cond> → **mientras (**<Expresiones>**) haz {** <Estatuto> **}**
<Repeticion_No_Cond> → **desde** <Asignacion> **hasta** <Expresiones> **hacer {**
<Estatuto> **}**

<Var_CTE> → CTEI | CTEF | CTECH

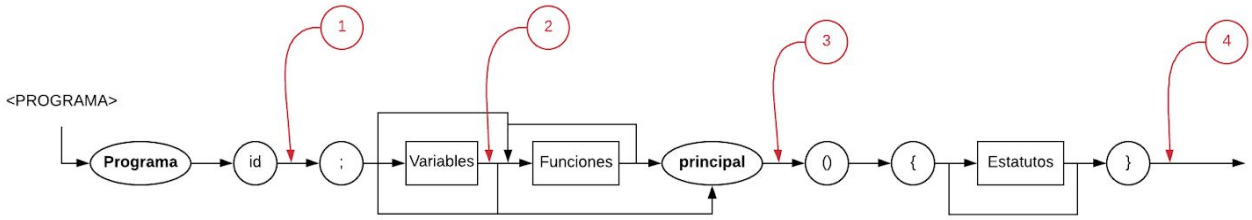
id → [a-zA-Z]+([a-zA-Z0-9_])*
CTEI → [0-9]+
CTEF → [0-9]+(.[0-9]+)*

CTECH → ‘[a-zA-z]’
STRING → \".*?\"

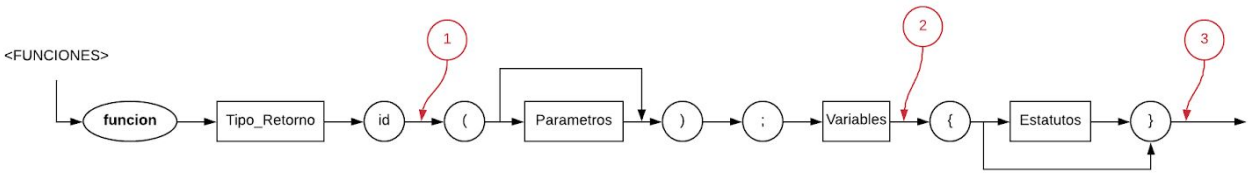
3.4. Descripción de Generación de Código Intermedio y Análisis Semántico

3.4.1. Código de operación y direcciones virtuales asociadas a los elementos del código.

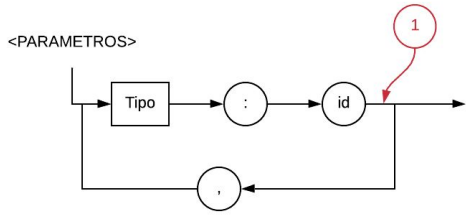
3.4.2. Diagramas de Sintaxis con las acciones correspondientes



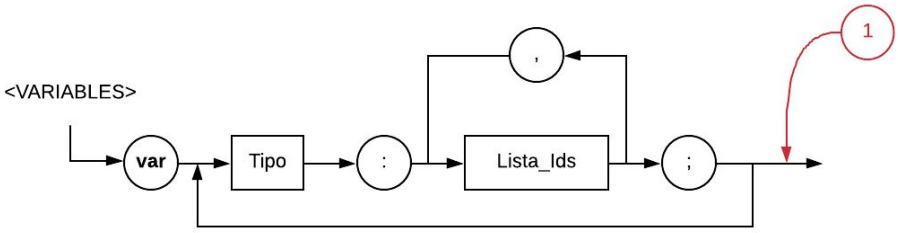
1. Define Función global y agrega quad de jump para ir a principal.
2. Asigna la Tabla de Variables a la Tabla de Globales.
3. Resuelve el salto a main pendiente en los quads.
4. Cuenta el espacio de memoria usado.



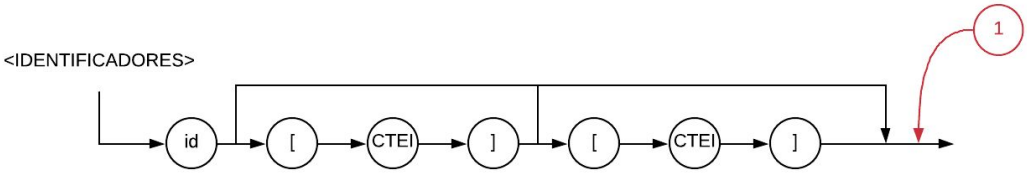
1. Verifica que el nombre no exista previamente como variable o función, en caso de que no define Función local y lo se agrega a la Tabla de Funciones, en caso de que si, arroja error.
2. Asigna el valor en tabla de funciones en que número de Cuádruplo se encuentra esta función.
3. Borra tabla de variables, define en Cuadruplos la terminación de la función. Verifica que haya un "regresa" en caso de que la función sea tipo retorno, sino lo hay arroja error. Cuenta el espacio de memoria usado. Borra las variables locales.



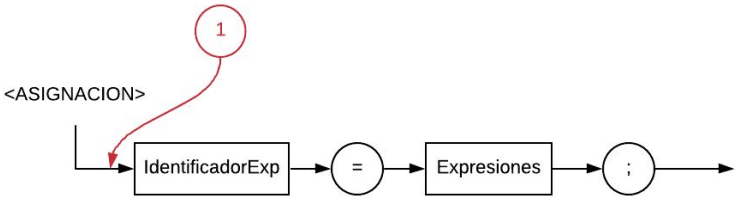
1. Verifica que la variable no exista en la tabla de Variables, en caso de que si exista, arroja error, sino asigna memoria y la agrega a la tabla de variables y a parámetros en la tabla de funciones en su función correspondiente.



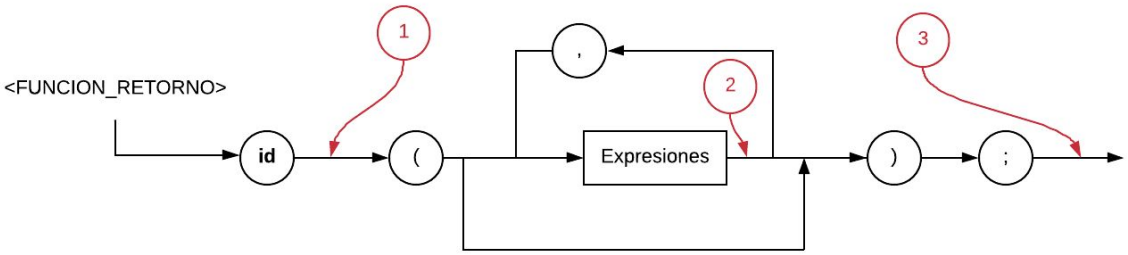
1. Para cada ID que se encontró verifica que no exista en la Tabla global o en la de variables y agregalo a la tabla de variables.



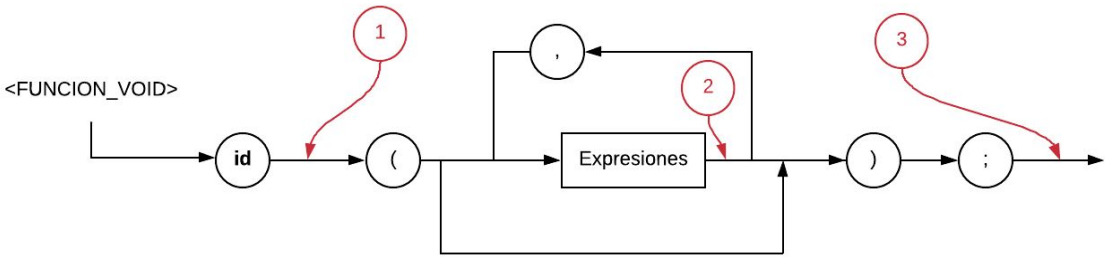
1. Verifica que no se creen arreglos de tamaños negativos y crea la estructura de dimensiones que se guardará como atributo de la variable.



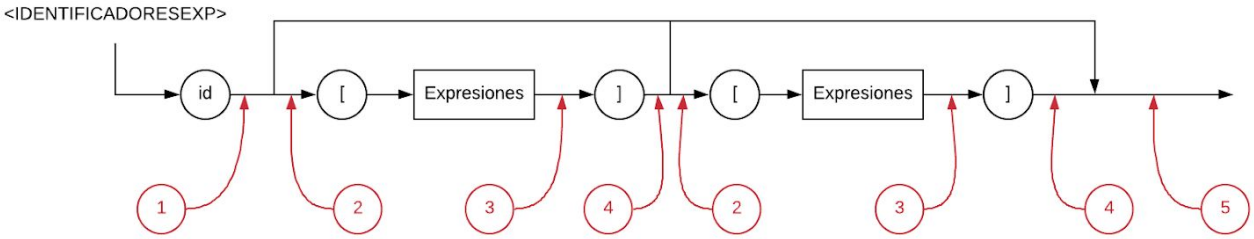
1. Verifica que sean variables del mismo tipo, en caso de que no, arroja error. Genera cuádruplo de asignación en la lista de cuádruplos.



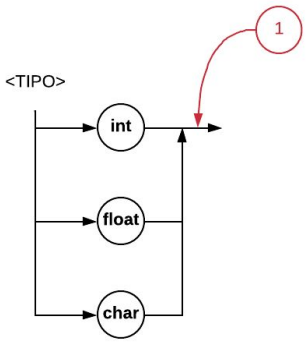
1. Verifica que la variable exista en la tabla de globales o en la actual y obtiene sus valores.
2. Verifica que el tipo de parámetro que se mandó sea correcto, en caso de que la función requiera parámetros.
3. Checa si la función contiene los parámetros requeridos. Asigna memoria y genera Cuádruplo GOSUB. Se genera el cuádruplo para obtener el valor de retorno de la función.



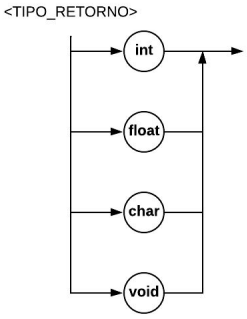
1. Verifica que la variable exista en la tabla de globales o en la actual y obtiene sus valores.
2. Verifica que el tipo de parámetro que se mandó sea correcto, en caso de que la función requiera parámetros.
3. Checa si la función contiene los parámetros requeridos. Asigna memoria y genera Cuádruplo GOSUB.

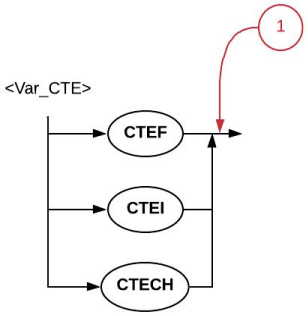


1. Verifica que la variable exista y agrega sus dimensiones al Stack de dimensiones.
2. Verifica que haya una siguiente dimensión.
3. Calcula la posición en memoria que se necesita hacer para indexar la memoria en el espacio correcto, ya sea una dimensión o una matriz.
4. Calcula la dirección actual basada en la dirección base de la variable dimensionada
5. Verifica que no falten dimensiones por indexar.

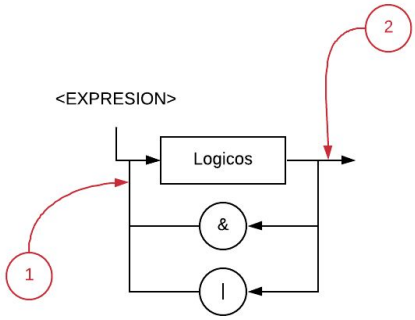


1. Asigna el valor de current_type

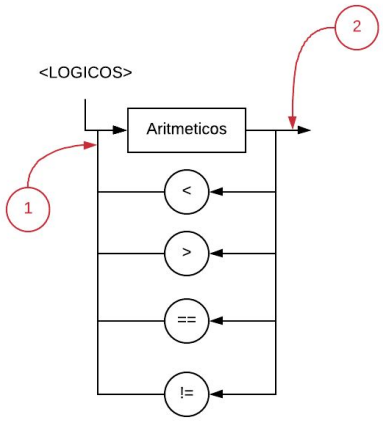




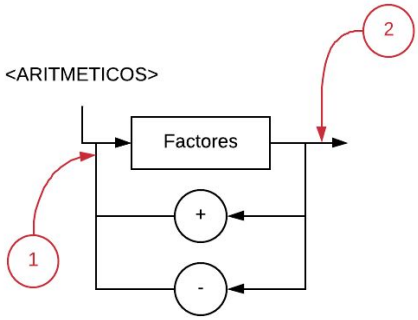
1. asigna el valor a current_type, y obten o crea la constantes en la Tabla de Constantes



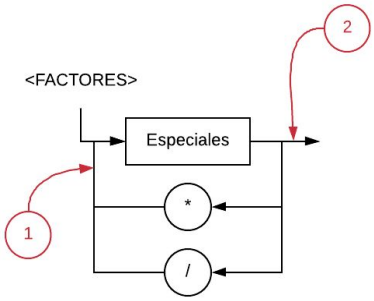
1. Agrega el operador a el Stack de Operadores
2. Verifica que la semántica está correcta y genera el quadruplo



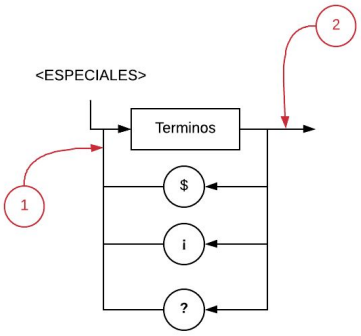
1. Agrega el operador a el Stack de Operadores
2. Verifica que la semántica está correcta y genera el quadruplo



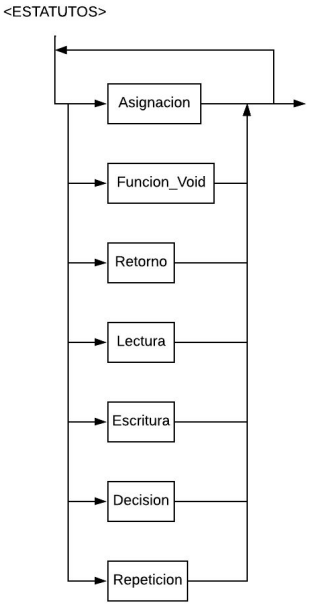
1. Agrega el operador a el Stack de Operadores
2. Verifica que la semántica está correcta y genera el quadruplo

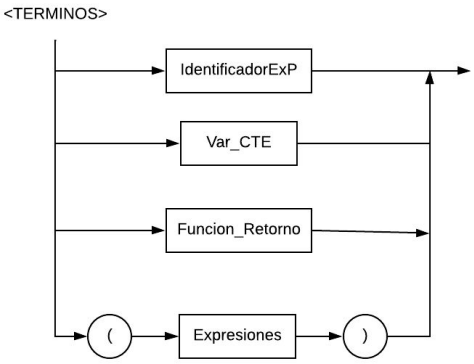


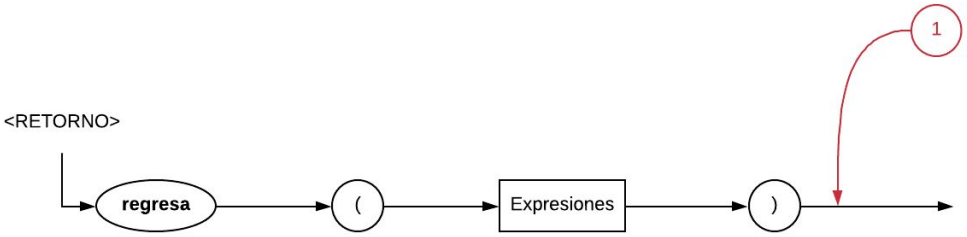
1. Agrega el operador a el Stack de Operadores
2. Verifica que la semántica está correcta y genera el quadruplo



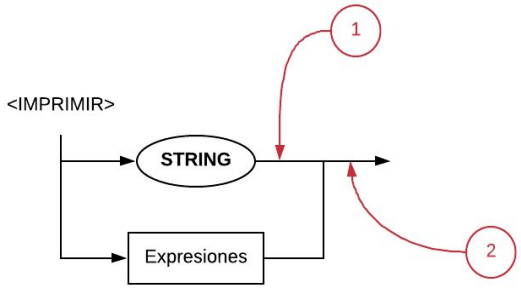
1. PENDIENTE



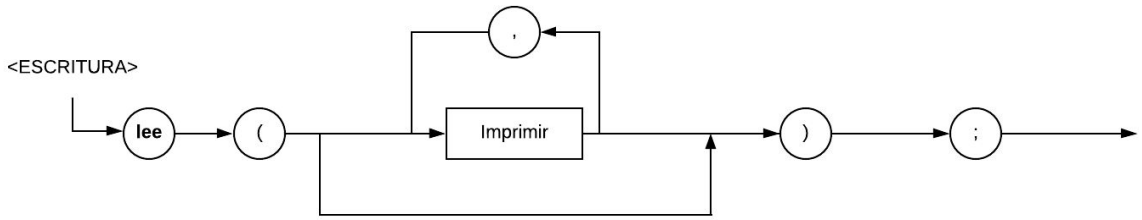


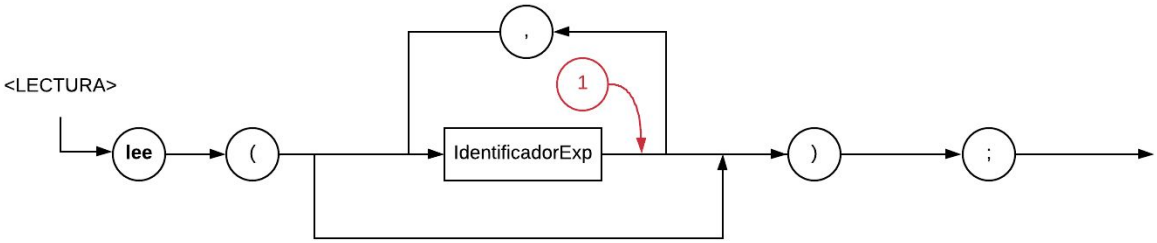


1. Verifica que la función actual requiere un valor de retorno del tipo que se está tratando de regresar y guarda su valor en la dirección correspondiente de la Tabla Global

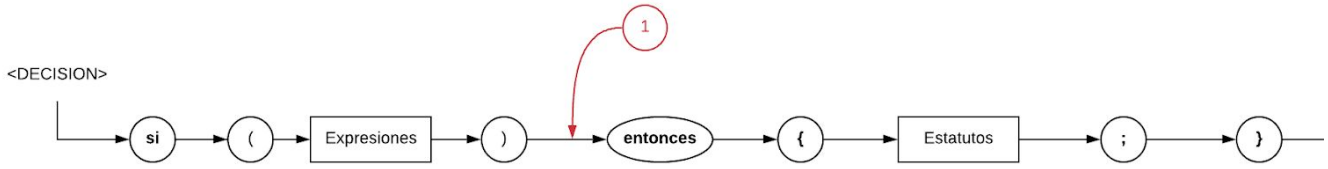


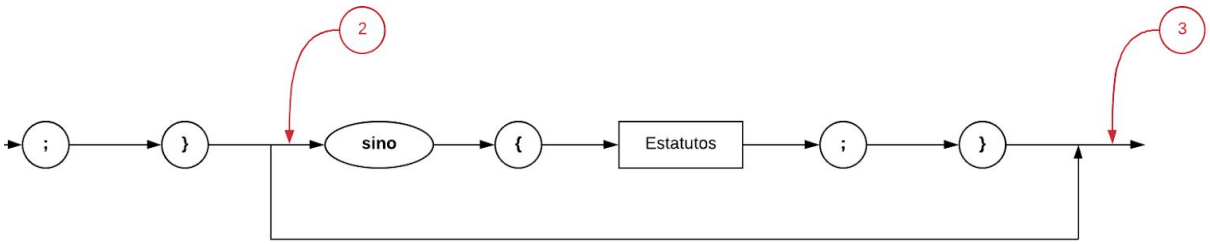
1. Crea o obtener el string como constante en la Tabla de constantes y agregalo al Stack de Operadores
2. Genera los cuádruplos de impresión

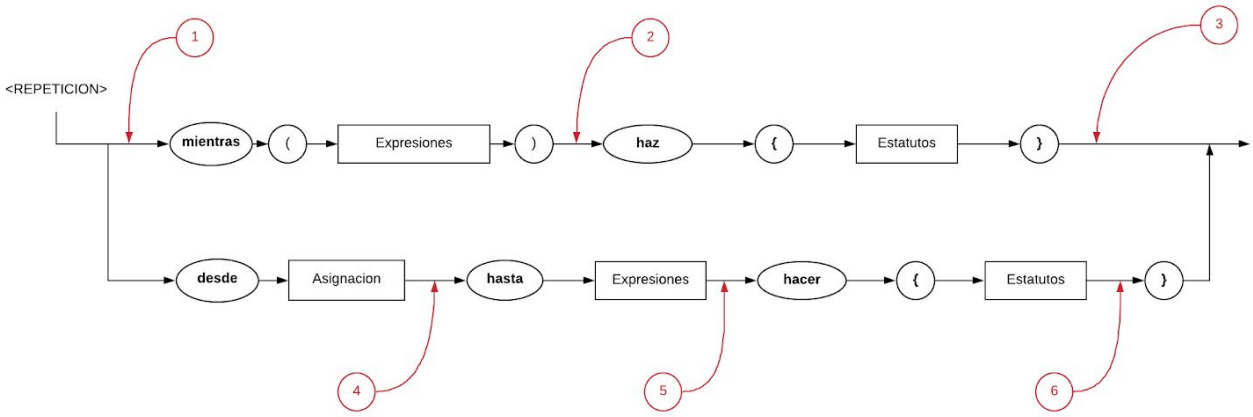




1. Genera el cuádruplo de lectura







1. Verifica que el resultado de la expresión sea bool y genera el cuádruplo de salto con el salto pendiente
2. Genera el cuádruplo de salto pendiente y resuelve el salto pendiente en el Stack de saltos
3. Resuelve el salto pendiente en el Stack de saltos

1. Agrega al stack de saltos la posición actual
2. Verifica que el resultado de la expresión sea bool y genera el cuádruplo de salto con el salto pendiente
3. Genera el cuádruplo de salto al inicio del ciclo y resuelve el salto pendiente en Stack de saltos
4. Genera el cuádruplo de salto y genera los cuádruplos de incrementación para la variable de control verificando que esta sea de tipo int
5. Genera los cuádruplos de salto y genera los cuádruplos de comparación, resolviendo también el salto pendiente en el Stack de Saltos
6. Genera el cuádruplo de salto al inicio del ciclo y resuelve el salto pendiente en Stack de saltos

3.4.3. Breve descripción de cada una de las acciones semánticas y de código (no más de 2 líneas).

3.4.4. Tabla de consideraciones semánticas.

3.5. Descripción detallada del proceso de Administración de Memoria usado en la compilación.

3.5.1. Especificación gráfica de CADA estructura de datos usada.

TablaFunciones : es un diccionario de diccionarios que utilizamos para la administración de las funciones generadas o utilizadas por el programa. Cada programa es una entrada nueva al diccionario y contiene los siguientes datos: tipo, parametros, num_parametros, num_temporales, start. int, float, char, tint, tfloat, tchar, bool y addr.

TablaVariables: es un diccionario de diccionarios que utilizamos para la administración de las variables generadas por el programa. Cada nombre de una variables es una entrada nueva al diccionario y contiene los siguientes datos: tipo, nxt y loc. El tipo contiene el tipo de variable que es, nxt nos dice si tiene

dimensión o no y loc nos guarda la asignación de memoria. Esta tabla nos permite poner las locs correspondientes al momento de generar los cuádruplos.

TablaGlobales: es un diccionario de diccionarios que utilizamos para la administración de las variables globales, cada entrada es el nombre de la variable y lo que contienen es lo siguiente: tipo, nxt y loc. En el caso de ser una variable proveniente de lo que regresará una función tipo retorno, la entrada es el nombre de la función y lo que contiene es: nombre, los, nxt y size. Esta tabla nos permite poner las locs correspondientes al momento de generar los cuádruplos.

TablaConstantes: es un diccionario de diccionarios que utilizamos para la administración de variables constantes en el programa. Cada entrada es el valor de la variable y lo que contienen es lo siguiente: loc y tipo. Esta tabla nos permite poner las locs correspondientes al momento de generar los cuádruplos.

Stack:

Quad:

4. Máquina Virtual

4.1. Equipo de cómputo, lenguaje y utilerías especiales usadas.

Mismo cómputo que el compilador, al igual que lenguaje. Solo varía en las utilerías especiales usadas, se la librería Numpy para facilitar el cálculo de la determinante.

Sistema Operativo: Windows, MacOS, Linux
Lenguaje: Python3
Utilerias: numpy

4.2 Descripción detallada del proceso de Administración de Memoria en ejecución

Hicimos uso de diccionarios para el manejo de la memoria global, local y temporal, para la tabla de funciones y tabla de constantes.

4.2.1. Especificación gráfica de CADA estructura de datos usada para manejo de scopes

```
Def Memoria = {
   'Global': {
       'int': 1000,
       'float': 2000,
       'char': 3000
   },
   'Local': {
       'int': 4000,
       'float': 5000,
       'char': 6000
   },
   'Temporal': {
       'int': 7000,
```

```
        'float': 8000,
        'char': 9000,
        'bool': 10000,
        'addr': 11000
    }
}
```

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

5. Pruebas del Funcionamiento

5.1. Incluir pruebas que "comprueben" el funcionamiento del proyecto

- codificación y resultados por generación de código intermedio y por la ejecución.

6. Listados perfectamente documentados

6.1. Comentarios de Documentación (para cada módulo, una pequeña explicación de qué hace, qué parámetros recibe, qué genera como salida y cuáles son los módulos más importantes que hacen uso de él)

6.2. Dentro de los módulos principales se esperan comentarios de Implementación, es decir: pequeña descripción de cuál es la función de algún estatuto que sea importante de ese módulo