Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Manchas

Proyecto de Diseño de Compiladores
Profesores:
M.C. Elda G. Quiroga
Dr. Héctor Ceballos

A01154891 Raúl Castellanos
24 de noviembre, 2021

Índice

Descripción del Proyecto

Propósitos, objetivos y alcance

Propósito

El lenguaje Manchas, es un lenguaje totalmente imperativo con el propósito de poder aplicar todos los conocimientos aprendidos a lo largo del semestre y de muchas otras materias. A su vez, este lenguaje tiene el propósito de simplificar algunas operaciones básicas que se utilizan comúnmente en el análisis estadístico.

Objetivos

El principal objetivo de este proyecto es poder crear un lenguaje de programación que pase por el proceso de compilación y ejecución.

El lenguaje está orientado a jóvenes que buscan aprender los fundamentos de programación, a través del manejo y manipulación de conjuntos de datos simples para realizar un análisis estadístico básico (Descripción de MyRLike). Por lo que se emplean estructuras y palabras reservadas muy similares a las de otros lenguajes muy populares.

Alcance

El proyecto busca cumplir con los requerimientos establecidos por los profesores que imparten la materia de Diseño de compiladores del semestre agosto-diciembre 2021, haciendo un compilador funcional.

Para el proyecto se busca generar un compilador simple, con una sintaxis no muy extensa como se puede tener en los lenguajes de programación más conocidos. Se considera que en el proyecto se deben implementar las funcionalidades más básicas de un lenguaje. Como declaración de variables (de diferentes tipos), ejecutar operaciones aritméticas y lógicas, ciclos, condicionales, crear y llamar a funciones entre otras.

Análisis de requerimientos

Requerimientos funcionales

ID	REQ001
Nombre	Función main
Descripción	El sistema debe contar con la función main. Desde esta función comenzará la ejecución del programa, y deberá contar con acceso a las variables de su contexto y del contexto global.

ID	REQ002
Nombre	Declaración de Funciones
Descripción	El sistema debe ser capaz de permitir la declaración de funciones. Esto conlleva que se puedan tener declaraciones para el contexto de la función. En las funciones se deben poder ejecutar las mismas instrucciones en la función principal main. Además, se debe permitir que las funciones reciban parámetros y puedan retornar un valor de tipo entero, flotante o carácter (Booleanas quedan fuera del alcance, pero es deseable).

ID	REQ003
Nombre	Declaración de variables
Descripción	El sistema debe permitir la declaración de variables de tipo: enteras, flotantes y de caracteres (Booleanas quedan fuera del alcance, sin embargo, es deseable).

ID	REQ004
Nombre	Declaración de arreglos
Descripción	El sistema de permitir la declaración de arreglos de una dimensión de los mismos tipos mencionados en la declaración de variables. Se deben poder declarar a un contexto local o global, sin embargo, no es necesario que sean argumentos o parámetros de alguna función

ID	REQ005
Nombre	Asignación de variables
Descripción	El sistema debe permitir la asignación a variables de acuerdo al tipo con el que se definieron.

ID	REQ006
Nombre	Asignación de arreglos
Descripción	El sistema debe permitir la asignación de valores de acuerdo al tipo de su declaración en la posición que se indica para algún arreglo.

ID	REQ007
Nombre	Lectura de variables
Descripción	El sistema deberá ser capaz de leer variables previamente declaradas para hacer su uso dentro de los programas.

ID	REQ008
Nombre	Lectura de arreglos
Descripción	El sistema debe ser capaz de leer una posición de los arreglos previamente declarados para hacer su uso dentro de los programas.

ID	REQ009
Nombre	Operaciones aritméticas
Descripción	El sistema debe permitir realizar operaciones aritméticas con los operadores: '+' como sumar, '-' como resta, '*' como multiplicación, '/' como división.

ID	REQ010
Nombre	Operaciones lógicas
Descripción	El sistema debe permitir el uso de operaciones lógicas con los operadores: '&&' como `and`, "||" como `or`, '!=' como `not`.

ID	REQ011
Nombre	Operaciones relacionales
Descripción	El sistema debe permitir el uso de operaciones relaciones con los operadores: '<' como menor que, '<=' como menor igual que, '>' como mayor que, '>=' como mayor igual que, '==' como igual que.

ID	REQ012
Nombre	Uso de ciclos condicionales y no condicionales
Descripción	El sistema deberá ser contar con ciclos condicionales (`while`) y no condiciones (`for`), que podrán ser usados dentro de los contextos de funciones declaradas o de la función principal, `main`.

ID	REQ013
Nombre	Uso de condicionales
Descripción	El sistema debe permitir llamar a la función condicional `if`, que podrá o no tener un estatuto `else`. Este condicional evaluará una condición booleana y ejecutará el flujo deseado por el usuario.

ID	REQ014
Nombre	Función de escritura
Descripción	El sistema debe contar con una función que permita la escritura de cadenas de texto y/o variables (Arreglos no se consideran).

ID	REQ015
Nombre	Función de lectura
Descripción	El sistema deberá contar una función que permita al usuario poder ingresar valores al momento de ejecución y permita utilizar estos valores.

ID	REQ016
Nombre	Llamada de funciones declaradas
Descripción	El sistema debe ser capaz de llamar a las funciones que han sido declaradas en el mismo programa. Estas funciones podrán ser llamadas en cualquier contexto (aun si es el mismo).

ID	REQ017
Nombre	Uso de funciones especiales

Descripción	El sistema debe contar con funciones particulares que simplifiquen algunas operaciones básicas de estadística. Puede o no incluir: - Una función para calcular la media de un arreglo de datos numéricos. - Una función para calcular la mediana de un arreglo de datos numéricos. - Una función para calcular la varianza de un arreglo de datos numéricos. - Una función para calcular la desviación estándar de un arreglo de datos numéricos. Se deberá contar por lo menos con dos funciones especiales.

ID	REQ018
Nombre	Verificación e impresión de errores
Descripción	El sistema deberá de ser capaz de detectar posibles errores que se puedan dar al momento de crear y ejecutar un programa.

Requerimientos no funcionales

ID	REQ-NF001
Nombre	Uso de Memoria
Descripción	El sistema deberá contar con una cantidad de memoria limitada, y se le deberá notificar al usuario si esta se consume en su totalidad. Además, se deberá hacer un uso eficiente de la misma.

ID	REQ-NF002
Nombre	Expresiones
Descripción	El sistema deberá contar con las expresiones tradicionales (tomando como posible referencia a C, Java o C++). Al igual que se deberá manejar las prioridades tradicionales.

Principales Casos de prueba

ID	TC-001
Nombre	Operaciones básicas
ID de Req	REQ001, REQ003, REQ005, REQ007, REQ009, REQ014
Descripción	El usuario generará un programa en el cual definirá por lo menos dos variables numéricas y realizara operaciones aritméticas con ellas, imprimirá el resultado en consola.
Criterios de aceptación	- El programa se ejecuta de manera correcta - Se imprime el resultado esperado - Los cálculos aritméticos realizados son correctos

ID	TC-002
Nombre	Operaciones booleanas
ID de Req	REQ001, REQ003, REQ005, REQ007, REQ009, REQ010, REQ011, REQ013
Descripción	El usuario generará un programa en el cual definirá por lo menos dos variables numéricas y ejecutara operaciones aritméticas, lógicas y relacionales donde dependiendo de los resultados usara un condicional para obtener diferentes resultados impresos
Criterios de aceptación	- El programa se ejecuta de manera correcta - Se imprime el resultado esperado - Los cálculos aritméticos y booleanos ejecutados son correctos

ID	TC-003
Nombre	Uso de funciones
ID de Req	REQ001, REQ002, REQ003, REQ005, REQ007, REQ009, REQ016
Descripción	El usuario declara un programa dónde cree por lo menos una función la cual recibe parámetros y tiene algún valor de retorno. Este es impreso en consola.
Criterios de aceptación	- El programa se ejecuta de manera correcta - Se imprime el resultado esperado - Se llama a la función con ciertos parámetros y estos son correctos en el uso interno de la función.

ID	TC-004
Nombre	Uso de arreglos

ID de Req	REQ001, REQ004, REQ006, REQ008, REQ012
Descripción	El usuario declara un programa, en el cual declara un arreglo de cualquier de los tipos existentes, llena el arreglo con valores y los imprime usando ciclos.
Criterios de aceptación	- El programa se ejecuta de manera correcta - Se imprime el resultado esperado - Los valores que se insertan en el arreglo son los correctos al momento de leerlos - En caso de tratar de acceder a un espacio diferente al definido del arreglo, se arroja un error.

ID	TC-006
Nombre	Input y condicionales
ID de Req	REQ001, REQ003, REQ005, REQ007, REQ009, REQ011, REQ013, REQ013
Descripción	El usuario crea un programa el cual recibe valores por el usuario, y realiza operaciones con ellos. Validando valores que recibe con condicionales.
Criterios de aceptación	- El programa se ejecuta de manera correcta - Se imprime el resultado esperado - Los valores ingresados por el usuario son los correctos en el uso del programa. - Los flujos de los condicionales se ejecutan de manera correcta.

Proceso del proyecto

Proceso de desarrollo

El proyecto se comenzó a desarrollar el 25 de septiembre del 2021, y para administrar todos los cambios se hizo uso de `git` como herramienta de control de versiones. Además, se creó un repositorio en `GitHub` para tener respaldados los cambios que se fueron realizando.

Para el desarrollo del proyecto se estuvieron manejando entregas semanales de avances, en la sección de bitácoras del proyecto se puede ver el avance que se tuvo en cada semana. Cada semana se iba agregando más funcionalidad hasta tener el producto final.

Bitácoras del proyecto

A lo largo del proyecto se estuvieron creando archivos individuales con cada una de estas entradas a la bitácora del proyecto, estas pueden ser encontradas en la carpeta de documentación del proyecto.

Como se mencionó cada semana se estuvo efectuando una entrega, estos son los cambios que se fueron ejecutando:

Avance 1 - 1° de octubre, 2021

Durante este primer avance se diseñó la sintaxis del lenguaje de programación a trabajar. Además, se generó la primera versión de los diagramas de sintaxis. Esto se implementó ya en código haciendo utilizo de la librería `PLY`, como analizador sintáctico.

Como resultado de este primer avance, se tiene ya el programa que puede recibir como input un archivo de texto y este es validado que tenga la sintaxis correcta.

Para este primer avance se diseñó él la sintaxis del lenguaje. Se crearon los diagramas y gramática y se realizó el analizador sintáctico usando PLY, este comprueba la sintaxis de un archivo `txt` como input y válida que sea correcta o incorrecta.

Avance 2 - 8 de octubre, 2021

Este segundo avance implico crear estructuras que son fundamentales para el desarrollo de la ejecución del código. Por ello, se agregaron tres principales clases con las cuales se estará trabajando todo el proyecto.

- `Directory`
- `Scopes_directory`
- `Vars`

Las clases de `Scopes_directory` y `Vars`, heredan todas las propiedades y métodos de `Directory`, aunque en ambos casos se sobreescribe el constructor para poder inicializar con los valores correspondientes al uso que se le dará. Para el caso de `Scopes_directory` se sobreescribe también la función de `print()`, dado que en su uso tendrá instancias de `Vars`. Ambas clases son usadas para poder tener los diccionarios de procedimientos y de variables.

Por otro lado, se agregaron los primeros puntos neurálgicos necesarios para poder agregar los contextos con los que tenga el programa en cuestión. Por ejemplo, el contexto global, contexto del main, y los contextos específicos por cada función. Una vez eso completo que añadieron otros puntos neurálgicos para añadir las variables que se declaren en el contexto correspondiente.

Finalmente, en esta entrega se añade también la clase `Operation`, que esta será utilizada más adelante para los cuádruplos. El objetivo de esta clase es saber si alguna operación que se está intentando realizar es válida o no.

Avance 3 - 16 de octubre, 2021

En este tercer avance se trabaja en la generación de código intermedio. Comenzando con las expresiones aritméticas y asignaciones y otros estatutos linéales.

Se crea la clase `Quadruples`, la cual sirve para representar las producciones de código. Esta cuenta con algunos métodos básicos para facilitar su uso a lo largo del proyecto.

Usando esta clase, se añaden los puntos neurálgicos correspondientes para ir produciendo los cuádruplos de lo mencionado anteriormente. Para poder lograr esto, se agregan cuatro principales pilas con las que se estará trabajando para todas las operaciones y control de flujos.

Las pilas son:

- `operands`
- `operators`
- `types`
- `jumps`

Con estas pilas es como se estará controlando que las operaciones tengan una correcta precedencia y ayudaran a poder tener validaciones de los posibles errores que se podrían están generando al momento de crear un programa.

Finalmente, se hacen modificaciones en los archivos de test, y se añade uno, para seguir probando estos avances.

Avance 4 - 22 de octubre, 2021

Para este cuarto avance, se trabajó con la generación de código intermedio de los estatutos ciclos y condicionales. El ciclo el `while` ahora ya genera correctamente sus cuádruplos correspondientes. Sin embargo, para el ciclo `for` me encontré con varias complicaciones, que implicaron algunos cambios en la gramática del lenguaje, por lo que no se logró tener listos los cuádruplos de este estatuto para este avance.

Por otro lado, se refactorizaron algunas funciones del código que ya estaban complicándose con el código actual. Por otro lado se creó un constructor, y unos métodos para modificar y acceder más fácil a los atributos en los cuádruplos.

Para el próximo avance, se trabajará en agregar los puntos neurálgicos de todos los estatutos restantes y se comenzará a diseñar como se trabajara con la memoria para ya hacer su implementación en la generación de código intermedio. Además de revisar el código para la generación de código de funciones

Avance 5 - 6 de noviembre, 2021

Este quinto avances tiene un gran progreso en cuando a la generación del código intermedio. Se añadieron los puntos neurálgicos para algunos estatutos lineales pendientes como el del `main`, `return`, entre otros.

Durante este avance se enfocaron esfuerzos en todas las implicaciones que tuvo generar el código intermedio de funciones. Para poder lograr esto, se creó una lista específica para poder llevar un control de los parámetros y se hicieron modificaciones al directorio de funciones. Modificaciones, que sirven para llevar la cuenta de cuantos parámetros y de qué tipo son los que tiene una función.

Se añadieron todos los puntos neurálgicos necesarios para declarar y llamar a funciones. Durante este desarrollo, hubo unos problemas con la gramática lo cual llevó a que se fuera una cantidad de tiempo en refactorizar cosas ya existentes.

Por otro lado, se definieron e implementaron los códigos que se estarán usando para poder identificar todas las operaciones que existan en los cuádruplos generados.

Finalmente, durante la semana se comenzó a definir como estará estructurada la memoria, para ya implementar esta idea en la creación de código intermedio. Quedó pendiente empezar a trabajar con la máquina virtual para la ejecución de expresiones aritméticas.

Avance 6 - 15 de noviembre, 2021

Durante este sexto avance, hubo muchas complicaciones de conflictos que ya había en entregas pasadas, pero no habían sido detectadas. Por lo que se tuvieron que cambiar algunos flujos importantes que ya estaban en la generación de código intermedio. Además de corregir estos errores desde la gramática, lo cual implico hacer unas pruebas extensas sobre lo que se asumía se tenía listo.

Para este avance se definió como se estaría trabajando con la memoria y se hicieron los cambios necesarios para que el código intermedio fuera totalmente escrito con direcciones de memoria. Cambios como, la tabla de constantes y validaciones de la memoria a la que se accedía.

Además, se añadieron los puntos neurálgicos para la definición de y uso de arreglos, y se crearon sus cuádruplos correspondientes.

Se hicieron esfuerzos para poder ya tener la máquina virtual trabajando. Está ya se logró que ejecutara programas sencillos que implicaran operaciones aritméticas, booleanas y relacionales. Así como también la ejecución de condicionales y de ciclos. Con este avance, se pudo validar que la generación de código intermedio iba por el camino correcto.

Finalmente, se revisó como se estará trabajando con arreglos, y se añadieron los puntos neurálgicos para su generación de código intermedio. Esto a su vez implico reordenar como estaban sus declaraciones y llamadas en la gramática del lenguaje. Para el siguiente avance se espera completar la generación de cuádruplos

Avance 7 - 21 de noviembre 2021

Con este avance se comenzaron a unir todas las piezas que estaban pendientes. Primeramente, se arreglaron unos errores detectados al llamar una función. Se corrigieron los problemas que estaban causando una llamada de una función que esperaba un valor de retorno y una función que podía ser sin ningún valor de retorno.

Después, se creó la clase `Memory`, que sería la base para poder tener la memoria en la máquina virtual. Con esto, se podrían agregar validaciones que estaban pendientes sobre los límites de memoria. Una vez que esto se logró, se implementó la ejecución de funciones en la máquina virtual.

Además, se implementó el manejo de apuntadores para poder hacer uso de las direcciones virtuales que son apuntadores. Con esto, se implementó también la ejecución de código intermedio que utilizará arreglos. La ejecución de arreglos implico de nuevo varios cambios en los puntos neurálgicos ya existentes y en la gramática porque estaban creándose unas inconsistencias al momento de indexar en los arreglos.

Por otro lado, se completó la generación de cuádruplos para los estatutos pendientes, aquí incluidas las funciones particulares del programa. Con esto también se trabajó con su ejecución.

Finalmente, se realizaron varios de los programas que se estarán usando como pruebas para validar el funcionamiento del programa y se comenzó a trabajar en la documentación del proyecto.

Commits

Dentro de la carpeta de documetanción se puede encontrar un archivo con todos los commits, con toda la información completa.

A continuación se muestran todos los commits del proyecto formateados para poder verlos mejor y ordenados por fecha (al día 21 de noviembre del 2021):

2021-09-25 13:43:47 -0500 e5c39af by Raul, Initial commit
2021-09-27 10:08:10 -0500 9ebec99 by Raul, Add grammar diagrams
2021-09-28 12:34:18 -0500 fbe6d6c by Raul, Add grammar diagrams
2021-09-28 12:43:51 -0500 f9a8c5c by Raul, Created files for parser and lexer
2021-09-30 00:46:50 -0500 a1c5ae3 by Raul, Add tests files
2021-10-01 11:34:05 -0500 5e21960 by Raul, Add tokens, Fixes on diagrams
2021-10-01 16:38:41 -0500 e936022 by Raul, Add parser
2021-10-01 19:12:59 -0500 b478509 by Raul, Fix parser grammars
2021-10-01 19:13:35 -0500 61cc640 by Raul, Add gitignore
2021-10-02 20:36:16 -0500 ff34cbb by Raul, Add README file
2021-10-03 14:14:07 -0500 7874710 by Raul, Fix statements grammar
2021-10-07 15:30:39 -0500 7a9d676 by Raul, Add debugging on parser
2021-10-08 12:36:01 -0500 5f1edb7 by Raul, Add doc files
2021-10-08 18:56:57 -0500 b9803fc by Raul, Add getType for a operation, Add getType tests
2021-10-09 13:38:55 -0500 8cbbfc1 by Raul, Add firectories for vars and scopes
2021-10-09 21:46:00 -0500 8416cb4 by Raul, Add Vars Class, Add Directory father class, Fix Scopes class
2021-10-09 21:46:34 -0500 2cd9e04 by Raul, Add neuralgic points and add scopes and vars
2021-10-09 21:46:52 -0500 d205ce6 by Raul, Update grammar diagrams
2021-10-09 21:57:48 -0500 0714d34 by Raul, Update readme and remove unused file
2021-10-16 14:44:23 -0500 ce37042 by Raul, Update readme
2021-10-16 16:31:42 -0500 0fa596b by Raul, Add Quadruples class
2021-10-16 20:33:56 -0500 c0aebe4 by Raul, Add quadruples
2021-10-16 21:48:43 -0500 ac14754 by Raul, Add quadruples for logic exprs
2021-10-16 22:17:09 -0500 ff1fe22 by Raul, Add asignments quadruples
2021-10-16 23:15:34 -0500 5af5cd5 by Raul, Add conditional quadruples
2021-10-16 23:16:01 -0500 e4aa7e6 by Raul, Update readme
2021-10-17 00:58:00 -0500 a6287c9 by Raul, Add doc file
2021-10-17 11:14:20 -0500 6ef24d0 by Raul, Move test files
2021-10-22 13:23:57 -0500 cf10361 by Raul, Add documentation 4th
2021-10-23 07:53:12 -0500 638b73a by Raul, Add while neuralgic point
2021-10-23 23:43:00 -0500 3d3e803 by Raul, Add neural points for non linear statements
2021-10-24 13:28:38 -0500 0a8d98e by Raul, Add file 4
2021-10-31 23:43:19 -0600 400553e by Raul, Add for quadruples
2021-11-02 17:35:14 -0600 1c99262 by Raul, Fix print grammar
2021-11-02 21:55:07 -0600 feec432 by Raul, Add print neuralgic points
2021-11-06 19:39:29 -0600 d41c0d3 by Raul, Add goto main, add params tables on scopes, add funtions neuralig points
2021-11-07 09:09:33 -0600 e70f8f4 by Raul, Add first neuralgic point on function call
2021-11-07 09:35:32 -0600 c0507ee by Raul, Add return statement quadruple
2021-11-07 11:25:32 -0600 1b0c82d by Raul, Add function call np
2021-11-07 12:50:26 -0600 ae3dc41 by Raul, Add memory class, first idea
2021-11-07 17:50:41 -0600 bd28908 by Raul, Add deliver summary
2021-11-07 17:51:02 -0600 76917e9 by Raul, Add 5th deliver summary
2021-11-07 18:05:17 -0600 43ed46e by Raul, Add documentation
2021-11-13 14:39:28 -0600 15d7739 by Raul, Add integer, float and char constants to constants table and memory address
2021-11-13 14:44:57 -0600 f2421ca by Raul, Add bools on constants
2021-11-13 19:45:36 -0600 1678b6f by Raul, Fix grammar - operations hierarchy

2021-11-13 20:20:31 -0600 9238024 by Raul, Add addresses for temporal vars
2021-11-13 23:33:48 -0600 65d2ba7 by Raul, Order files, and virtual machine creation
2021-11-14 22:22:43 -0600 78b1f99 by Raul, Change how info is print
2021-11-15 00:27:20 -0600 1ca3fa4 by Raul, Add array declaration
2021-11-15 17:47:35 -0600 3d105dd by Raul, Add arrays neralgic points for definición and access
2021-11-15 17:54:18 -0600 ed9d240 by Raul, Fix array access
2021-11-15 19:56:58 -0600 7694a0e by Raul, Add basic math operations to virtual machine
2021-11-15 22:18:32 -0600 988b88f by Raul, Add execution for conditionals and loops
2021-11-15 22:46:11 -0600 9f54a21 by Raul, Add documentation
2021-11-17 21:39:30 -0600 b9fc883 by Raul, Start project documentation
2021-11-18 00:44:53 -0600 a3d9650 by Raul, Add asignment for functions on parser
2021-11-19 18:39:43 -0600 9f201b7 by Raul, Add memory class on virtual machine, add operations for function on virtual machine
2021-11-19 20:25:30 -0600 093671d by Raul, Fix calling anidated function
2021-11-19 21:27:48 -0600 20825de by Raul, Add arrays execution on vm
2021-11-19 22:40:40 -0600 63d7647 by Raul, Fix function call for recursive functions, add factorial tests
2021-11-20 14:44:17 -0600 457faf3 by Raul, Fix arrays operands stack, add mean neuralgin point
2021-11-20 15:39:35 -0600 2f86a15 by Raul, Add mean functionality on vm
2021-11-20 16:26:44 -0600 a928798 by Raul, Add median function
2021-11-20 17:31:49 -0600 0d7bf11 by Raul, Add special functions variance and stdev
2021-11-20 23:23:17 -0600 4e5f4a8 by Raul, Add find program, Fix negatives on parser
2021-11-20 23:57:59 -0600 9fe1f1a by Raul, Add test for array operations
2021-11-21 14:34:10 -0600 c057722 by Raul, Add commits log file. Add docs for project

Reflexiones personales

Uno escucha muchos comentarios sobre el proyecto en el que se tiene que trabajar para esta materia y se habla mucho sobre su complejidad y dificultad. Por un lado puedo decir que estos comentarios sí tienen una razón de ser, y por otro lado debo admitir que estos comentarios crean miedo al momento de ir trabajando en el proyecto.

De forma general debo mencionar que este proyecto efectivamente es complejo, porque todo lo que se va aprendiendo en las clases se deberá aplicar, y se deberá comprender correctamente para poder darle el uso adecuado que uno quiere. Aunque más que complejo es muy demandante, en el sentido de que siempre durante su desarrollo debes estar trabajando en él, y si se llega uno a atrasar deberá pagar un precio muy alto más adelante.

El ir realizando el proyecto me cambio la manera de ver todos los estatutos de los lenguajes de programación. Me hizo ser más consciente sobre como el definir nada más lo necesario puede tener grandes implicaciones. Al inicio del proyecto todo el proceso de compilación y ejecución de cualquier lenguaje era una gran incógnita para mí. Pero ahora creo que puedo entender de una mejor manera como algunos lenguajes se desempeñan diferente y ahora aprecio más las buenas prácticas al momento de programar.

Finalmente, debo decir que el proyecto se me hizo muy pesado, y a la vez muy interesante. Fue pesado porque efectivamente implicó mucho de mi tiempo, y mucha perseverancia de no darse por vencido cuando uno tenía arreglar cosas que implicaban mucho retrabajo, y eso me costó mucho mentalmente dado que había que balancear la carga de trabajo de todas mis responsabilidades. Aunque, a pesar de todo eso, ahora que estoy concluyendo el proyecto, veo el gran progreso que he tenido como programador desde que empece la carrera a hoy en día.

Raúl Castellanos Herrero
21, noviembre 2021

Descripción del Lenguaje

Nombre del lenguaje

El nombre del lenguaje es: Manchas (haciendo referencia a una de mis mascotas).

Descripción general

El lenguaje Manchas es un lenguaje imperativo con algunas características muy similares a lenguajes muy populares. Este lenguaje cuenta con la capacidad de declarar variables y arreglos de tipo entero, booleano, carácter y flotantes. Estas variables pueden ser usadas dentro de un mismo contexto, aunque también se cuenta con la posibilidad de crear y utilizar variables a un nivel de contexto global. Por otro lado, el lenguaje cuenta soporta operaciones aritméticas como suma, resta, multiplicación y división. También las operaciones relacionales y lógicas más comunes.

Con Manchas se pueden crear funciones que regresan o no algún valor de los tipos ya mencionados y que pueden o no recibir parámetros. Es importante mencionar que no se pueden declarar arreglos como parámetros y las funciones tampoco pueden regresar un arreglo.

Finalmente, el lenguaje cuenta con algunas funciones ya incluidas que pueden ser de mucha utilidad, funciones que tienen un enfoque estadístico como la mediana, promedio, varianza y desviación estándar de una muestra o población. Además de las funciones de escritura y lectura en la consola.

Listado de Errores

Errores en compilación

ID	Mensaje de error
C-01	You are trying to create an array of size: {array_size}\n Arrays must be defined with a value grater than 1
C-02	Error, not (in operators stack
C-03	Invalid operation, type mismatch on {right_type}, and {left_type} with a {operator}
C-04	Error, type mismatch on if statement, it must be a boolean
C-05	Type-mismatch on While expresion, it must be a boolean on a while expression
C-06	Conditional variable of "For" must be an integer
C-07	Invalid type in "For" statement (stop condition), must be boolean
C-08	Invalid type in "For" statement in delta, must be integer

C-09	Invalid type in "For" expression\n type mismatch on {for_var_type} and {delta_type} with a +
C-10	Function {current_scope}, has a return type of {func_return_type}, and you are trying to return a {v_type}
C-11	Function {current_function_call_id} is not defined
C-12	The {params_count + 1}° argument of function {current_function_call_id} should be of type {function_call_params[params_count]} and you are giving a {argument_type}
C-13	The function {current_function_call_id}, expected {size_of_params} arguments, you gave {params_count} arguments
C-14	{array_id} is not defined as an array.
C-15	You are trying to access {array_id} with an {accessing_array_type} value. This must be an integer
C-16	Invalid operation, type mismatch on {right_type} and {left_type} with a {operator}
C-17	{var_id} not found in current or global scope \n in get_var function
C-18	The {quadruple_str} function only accepts an array of floats or integers.
C-19	Invalid assignment operation, trying to asign a {type1} to a {type2}
C-20	Invalid integers operation, {type1} {symbol} {type2}
C-21	Invalid integer with float operation, {type1} {symbol} {type2}
C-22	Invalid character with integer operation, {type1} {symbol} {type2}
C-23	Invalid character with float operation, {type1} {symbol} {type2}
C-24	Invalid boleans operation, {type1} {symbol} {type2}
C-25	Invalid operation, {type1} {symbol} {type2}
C-26	Invalid float value
C-27	Invalid Int value
C-28	Illegal character {t.value[0]}
C-29	{id} already exists on this scope
C-30	Return type: {return_type} ais an invalid type for {id}
C-31	Cannot add this variable {id} is already declared on this scope
C-32	Return type: {type} is an invalid type for {id}
C-33	For a plot function, both variables must be arrays
C-34	For a plot function, both variables must be integers or float arrays
C-35	For a plot function, {x_array_id} and {y_array_id} must be of the same length.

Errores en ejecución

ID	Mensaje de error
E-01	This program cannot be executed due to lack of available memory (On global memory is created)
E-02	This program cannot be executed due to lack of available memory (On start main memory)
E-03	This program cannot be executed due to lack of available memory (On ERA of a new function)
E-04	On searching a value, this address: {pointer} hasnt been assigned
E-05	Instruction pointer is trying to access {new_position}, that doesnt exits
E-06	{function_id} is not defined on this program
E-07	Function {function_id}, must have a return statement, with a value of type {func_return_type}
E-08	Function {function_id} shoud be returning a {func_return_type}, instead it is being returned a {types_values[str(return_value_address)[1]]}
E-09	The input value is not a valid type, for a character is must be of length 1.
E-10	For boolean values, you must provide "false" or "true"
E-11	The read statement expected a {types_values[type_to_read]}.
E-12	Out of bounds\n Trying to acces value {value}, but limits are {inferior_limit} and {upper_limit}
E-13	Oh no, there is a division by zero, and we cannot perform this operation.

Descripción del Compilador

Herramientas usadas para el desarrollo

Para el desarrollo del compilador se trabajó en una computadora con sistema operativo Mac OS - Big Sur / Monterrey.

Todo el código fue desarrollando en Python, versión 3.7.10

Además, para la fase de compilación se utilizó de PLY, como herramienta para ser analizador sintáctico y para generar el parser.

Finalmente, se emplearon librerías adicionales para manejar estructuras de datos y realizar cálculos.

- sys
- collections (sólo deque), para manejar pilas y filas

Análisis Léxico

Patrones de construcción y Tokens del lenguaje

No.	Token	Descripción	Expresión Regular
1	PROGRAM	inicio del programa	'program'
2	LET	inicio de declaración de variables	'let'
3	INT	de tipo entero	'int'
4	FLOAT	de tipo flotante	'float'
5	CHAR	de tipo caracter	'char'
6	BOOL	de tipo booleano	'bool'
7	TRUE	contante verdadera	'true'
8	FALSE	constante falsa	'false'
9	FUNCTION	inicio de declaración de una función	'function'
10	MAIN	inicio de función main	'main'
11	VOID	de tipo vacio	'void'
12	RETURN	retorno de un valor	'return'
13	IF	inicio de un estatuto de condición	'if'
14	ELSE	inicio de la condición falta de un estatuto de condición	'else'
15	WHILE	inicio de un ciclo condicional	'while'

16	DO	inicio del ciclo condicional	'do'
17	FOR	inicio de un ciclo no condicional	'for'
18	TO	utilizado en ciclo for, para indicar límite	'to'
19	BY	utilizado en ciclo for, para indicar el valor delta	'by'
20	PRINT	función para escribir un valor en la consola	'print'
21	PRINTLN	función para escribir un valor en la consola con un salto de línea	'println'
22	READ	función para leer un valor de la consola	'read'
23	MEAN	función para calcular el promedio de un conjunto de datos	'mean'
24	MEDIAN	función para calcular la mediana de un conjunto de datos	'median'
25	PVARIANCE	función para calcular la varianza (población) de un conjunto de datos	'pvariance'
26	PSTDEV	función para calcular la desviación estandar (población) de un conjunto de datos	'pstdev'
27	VARIANCE	función para calcular la varianzaa (muestra) de un conjunto de datos	'variance'
28	STDEV	función para calcular la desviación estandar (muestra) de un conjunto de datos	'stdev'
29	RANDOM	función para obtener un valor aleatroeo	'random'
30	COMMA	coma	','
31	COLON	dos puntos	':'
32	SEMI	punto y coma	';'
33	LPAREN	abre parentésis	'\('
34	RPAREN	cierra parentésis	'\)'
35	LBRACKET	abre corchete	'\['
36	RBRACKET	cierra corchete	'\]'
37	LBRACE	abre llave	'\{'
38	RBRACE	cierra llave	'\}'
39	LT	menor que	'<'
40	LE	menor igual que	'<='
41	GT	mayor que	'>'
42	GE	mayor igual que	'>='
43	EQ	igual (operador relacional)	'=='
44	NE	difernte	'!='
45	EQUALS	igual (operador de asignación)	'='
46	OR	operador lógico: o	'||'

47	AND	operador lógico: and	'&&'
48	PLUS	suma	'\+'
49	MINUS	resta	'-'
50	TIMES	multiplicación	'*'
51	DIVIDE	división	'/'
52	CTESTRING	cadena de texto	'\".*\"'
53	CTEC	constante caracter	'\'(.{1})\''
54	ID	identficador de una variable/función	'[A-Za-z]([A-Za-z]|[0-9])*'
55	CTEF	constante flotante	'[0-9]+(\.[0-9]+)'
56	CTEI	constante entera	'[0-9]+'

Análisis Sintáctico

Gramática

A continuación se muestra la gramática libre de contexto que se creó para el lenguaje Manchas.

Regla sintáctica	Gramática libre de contexto
program	PROGRAM ID SEMI **vars program_1** | PROGRAM ID SEMI **program_1**
program_1	**function program_1** **main_block**
vars	LET **vars_prima_1 vars** | LET **vars_prima_1**
vars_prima_1	ID COLON **type** SEMI | ID COMMA **vars_prima_1**
type	INT **type_1** | FLOAT **type_1** | CHAR **type_1** | BOOL **type_1**
type_1	LBRACKET CTEI RBRACKET | **epsilon**
function	FUNCTION ID COLON **return_type** LPAREN RPAREN **block** **|** FUNCTION ID COLON **return_type** LPAREN **params** RPAREN **block** **|** FUNCTION ID COLON **return_type** LPAREN RPAREN **vars block** **|** FUNCTION ID COLON **return_type** LPAREN **params** RPAREN **vars block**
main_block	MIAN LPAREN RPAREN **block** **|** MIAN LPAREN RPAREN **vars block**

block	LBRACE **statements** RBRACE
return_type	VOID | **type**
params	ID COLON **type** COMMA **params** | ID COLON **type**
statements	void_function_call **statements1** | assignment **statements1** | condition **statements1** | writing **statements1** | reading **statements1** | repetition **statements1** | return **statements1** | plot **statements1**
special_functions	**mean** | **median** | **random** | **variance** | **p_variance** | **standard_deviation** | **p_standard_deviation**
statements1	**statements** | **epsilon**
assignment	ID LBRACKET **expression** RBRACKET EQUALS **expression** SEMI | ID EQUALS **expression** SEMI
condition	ID LPAREN **expression** RPAREN **block** | IF LPAREN **expression** RPAREN **block** ELSE **block**
expression	**exp** | **expression0** | **expression1**
expression0	**expression** AND **expression** | **expression** OR **expression**
expression1	**exp** LT **exp** | **exp** LE **exp** | **exp** GT **exp** | **exp** GE **exp** | **exp** EQ **exp** | **exp** NE **exp**
exp	**term** | **term exp_1**
exp_1	PLUS **exp** | MINUS **exp**
term	**factor** | **factor term_2**
temr_2	TIMES **term**

	| DIVIDE **term**
factor	LPAREN **expression** RPAREN | ID LBRACKET **expression** RBRACKET | **factor_prima_1** | **function_call** | **special_functions**
factor_prima_1	PLUS **varcte** | MINUS **varcte** | **varcte**
varcte	ID | CTEI | CTEF | CTEC | TRUE | FALSE
writing	PRINT LPAREN **writing_1** RPAREN SEMI
writing_1	**expression** COMMA **writing_1** | CTESTRING COMMA **writing_1** | **expression** | CTESTRING
reading	READ LPAREN **reading_1** RPAREN SEMI
reading_1	ID | ID LBRACKET **expression** RBRACKET
repetition	**non_conditional_loop** | **conditional_loop**
conditional_loop	WHILE LPAREN **expression** RPAREN DO **block**
non_conditional_loop	FOR LPAREN ID EQUALS **expression** TO **expression** BY RPAREN **block**
return	RETURN **expression** SEMI
function_call	ID LPAREN RPAREN | ID LPAREN **function_call_1** RPAREN
void_function_call	ID LPAREN RPAREN SEMI | ID LPAREN **function_call_1** RPAREN
function_call_1	**expression** | **expression** COMMA **function_call_1**
mean	MEAN LPAREN ID RPAREN
median	MEDIAN LPAREN ID RPAREN
random	RANDOM LPAREN CTEI COMMA CTEI RPAREN
variance	VARIANCE LPAREN ID RPAREN

p_variance	PVARIANCE LPAREN ID RPAREN
standard_deviation	STDEV LPAREN ID RPAREN
p_standard_deviation	PSTDEV LPAREN ID RPAREN
plot	PLOT LPAREN ID COMMA ID RPAREN SEMI
epsilon	-

Generación de código intermedio y análisis semántico

Códigos de operación

Para crear los cuádruplos, e identificar más adelante las operaciones se definieron los siguientes código de operación:
Nota: del 1 al 19 serían código para operaciones aritméticas, relacionales o lógicas (y asignación). Del 20 al 29 para operaciones de funciones, del 30 al 39 para otras operaciones de apoyo del lenguaje. Finalmente del 40 en adelante para funciones especificas.

Código	Operación	Descripción
1	+	Para realizar sumas entre dos operadores
2	-	Para realizar restas entre dos operadores
3	/	Para realizar divisiones entre dos operadores
4	*	Para realizar multiplicaciones entre dos operadores
5	<	Checar la relación de menor que entre dos operadores
6	<=	Checar la relación de menor igual que entre dos operadores
7	>	Checar la relación de mayor que entre dos operadores
8	>=	Checar la relación de mayor igual que entre dos operadores
9	==	Checar la relación de igual que entre dos operadores
10	!=	Checar la relación de diferente que entre dos operadores
11	&&	Comparar dos operadores lógicos (and)
12	||	Comparar dos operadores lógicos (or)
13	=	Símbolo de asignación
20	GOTO	Realizar un salto
21	GOTOV	Realizar un salto si el valor proporcionado es verdadero
22	GOTOF	Realizar un salto si el valor proporcionado es false
23	GOSUB	Salto a una función
24	ERA	Inicio de llamada de una función
25	PARAM	Se da un parámetro de una función
26	ENDFUNC	Finaliza una función
27	END	Finaliza el programa
30	RETURN	Se regresa un valor de una función

31	PRINT	Imprimir una expresión (sin salto de línea)
32	PRINTLN	Imprimir una expresión (con salto de línea)
33	VERIFY	Para verificar el acceso a un arreglo
34	READ	Leer una entrada de la consola
40	MEAN	Calcular la media de un arreglo
41	MEDIAN	Calcular la mediana de un arreglo
42	PVARIANCE	Calcular la varianza (para población) de un arreglo
43	PSTDEV	Calcular la desviación estándar (para población) de un arreglo
44	VARIANCE	Calcular la varianza (para muestra) de un arreglo
45	STDEV	Calcular la desviación estándar (para muestra) de un arreglo
46	RANDOM	Obtener un número aleatorio
47	PLOT	Crear una gráfica de puntos de dos arreglos

Direcciones virtuales

Para la generación de código intermedio, se definieron ciertos rangos de números para trabajarlos como direcciones virtuales, facilitar validaciones y poder identificar con que datos se estaba trabajando.

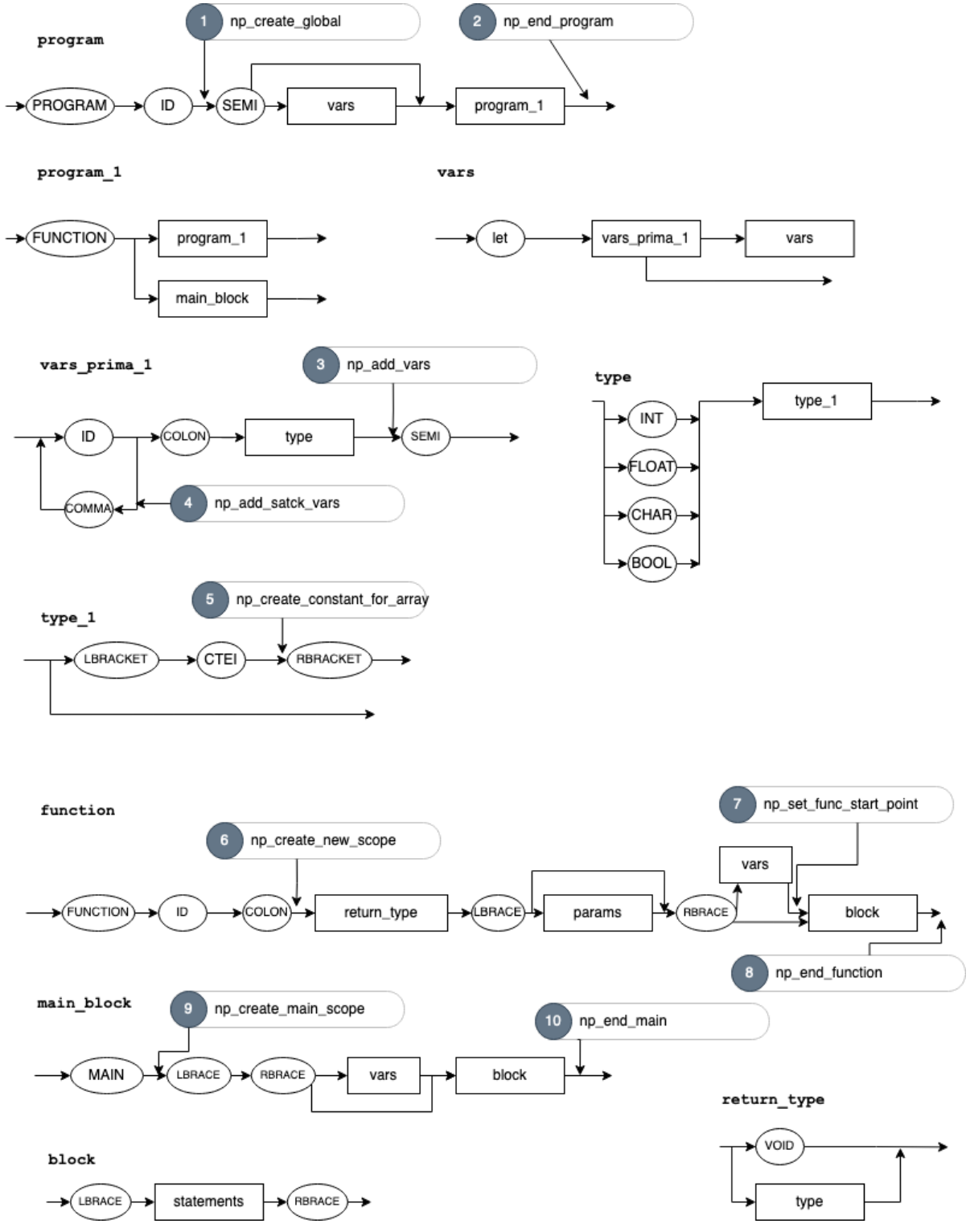
Para manejar estas direcciones, se implementó una clase memory.py en la cual se tienen definidos estos rangos al igual que algunos métodos para simplificar su uso a lo largo de la construcción del código intermedio.

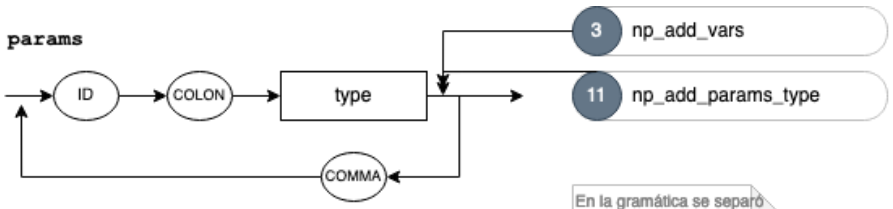
Tipo de memoria	Dirección de inicio
Globales enteras	110000
Globales flotantes	120000
Globales caracteres	130000
Globales booleanas	140000
Locales enteras	210000
Locales flotantes	220000
Locales caracteres	230000
Locales booleanas	240000
Temporales enteras	310000
Temporales flotantes	320000
Temporales caracteres	330000
Temporales booleanas	340000
Constantes enteras	410000
Constantes flotantes	420000
Constantes caracteres	430000
Constantes booleanas	440000
Apuntadores	500000

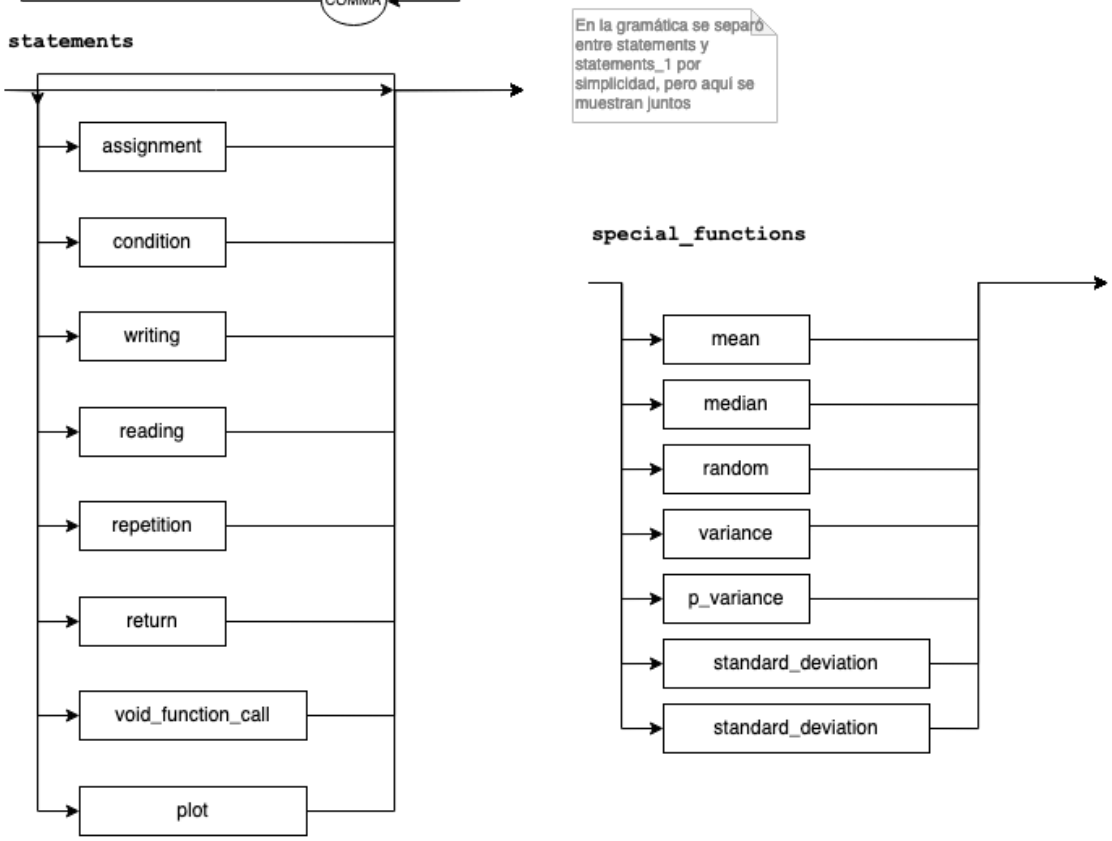
Se manejan estos rangos para que visualmente sea fácil de saber con qué tipo de dato se está tratando.

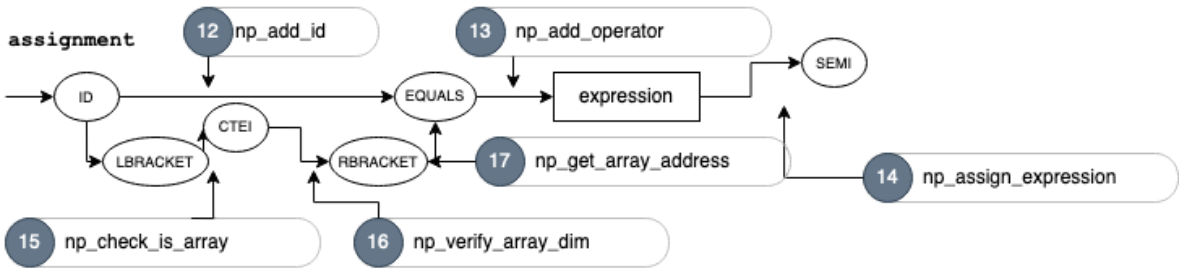
Todas las que son globales empiezan con 1, las locales con 2, las temporales con 3 y las constantes con 4. Por otro lado, las decenas de millar indican el tipo de dato, 1 para enteras, 2 para flotantes, 3 para caracteres y 4 para booleanas. Las direcciones que comienzan con 5 son apuntadores.

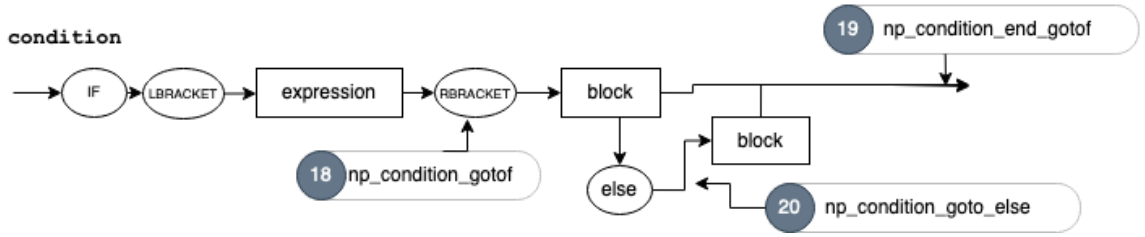
Diagramas de Sintaxis

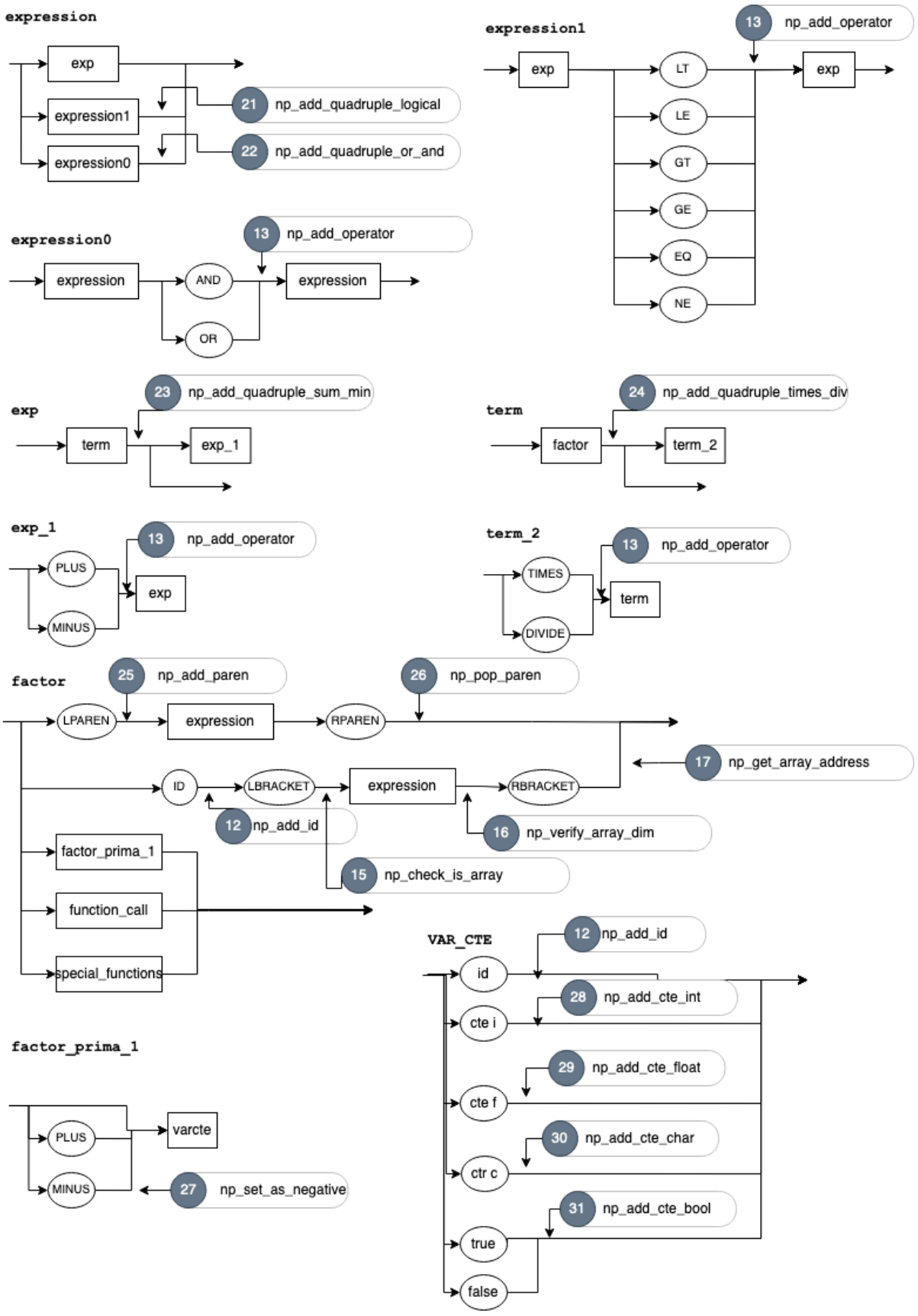



params
ID
COLON
type
COMMA
3
np_add_vars
11
np_add_params_type
Estan separados por la reutilización de estas funciones, pero podrían ser uno solo.

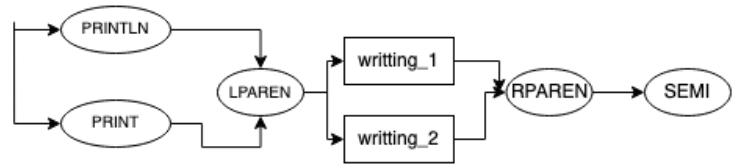

statements
En la gramática se separó entre statements y statements_1 por simplicidad, pero aquí se muestran juntos
assignment
condition
writing
reading
repetition
return
void_function_call
plot
special_functions
mean
median
random
variance
p_variance
standard_deviation
standard_deviation


assignment
12
np_add_id
13
np_add_operator
SEMI
ID
EQUALS
expression
CTEI
LBRACKET
RBRACKET
17
np_get_array_address
14
np_assign_expression
15
np_check_is_array
16
np_verify_array_dim

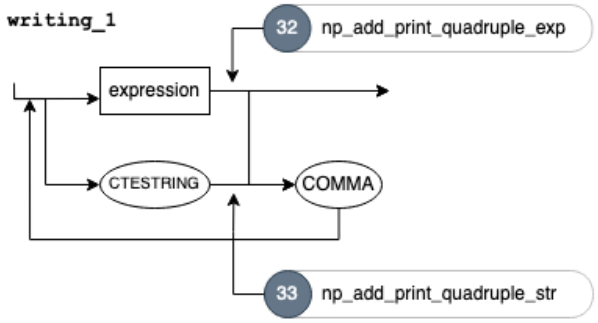

condition
19
np_condition_end_gotof
IF
LBRACKET
expression
RBRACKET
block
block
18
np_condition_gotof
else
20
np_condition_goto_else


expression
exp
expression1
expression0
21 np_add_quadruple_logical
22 np_add_quadruple_or_and
expression1
13 np_add_operator
exp
LT
LE
GT
GE
EQ
NE
exp
expression0
13 np_add_operator
expression
AND
OR
expression
exp
23 np_add_quadruple_sum_min
term
exp_1
term
24 np_add_quadruple_times_div
factor
term_2
exp_1
13 np_add_operator
PLUS
MINUS
exp
term_2
13 np_add_operator
TIMES
DIVIDE
term
factor
25 np_add_paren
26 np_pop_paren
LPAREN
expression
RPAREN
17 np_get_array_address
ID
LBRACKET
expression
RBRACKET
12 np_add_id
16 np_verify_array_dim
15 np_check_is_array
factor_prima_1
function_call
special_functions
VAR_CTE
12 np_add_id
id
28 np_add_cte_int
cte i
29 np_add_cte_float
cte f
30 np_add_cte_char
ctr c
31 np_add_cte_bool
true
false
factor_prima_1
PLUS
MINUS
varcte
27 np_set_as_negative

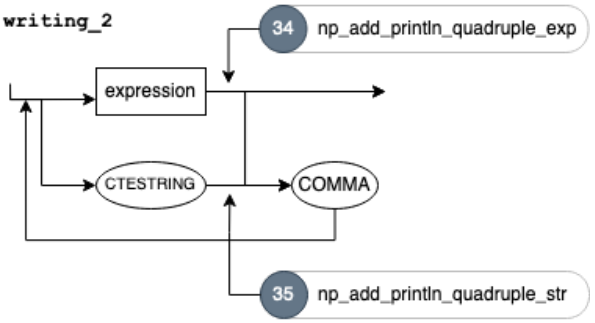
writing



writing_1

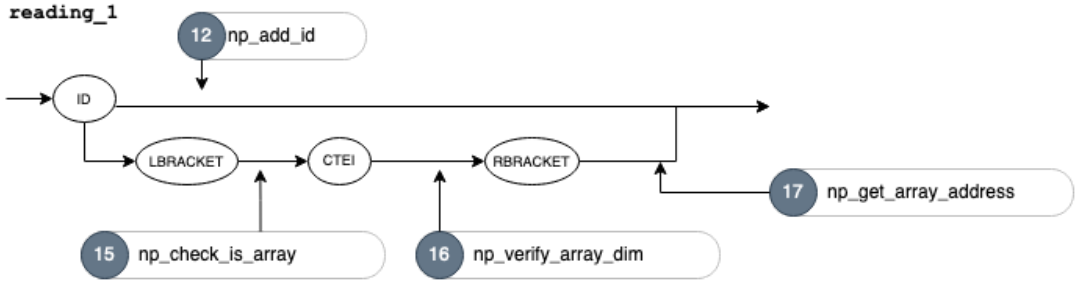


writing_2

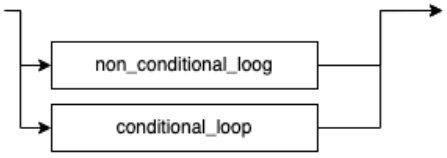


reading

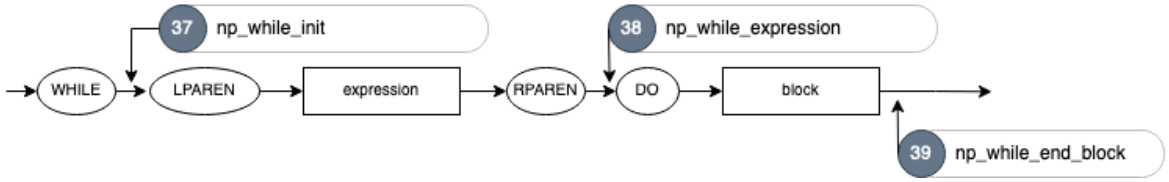
reading_1



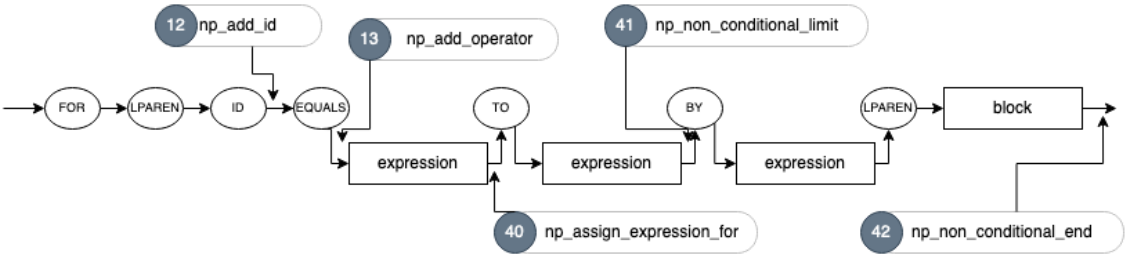
repetition



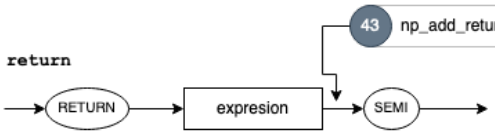
conditional_loop



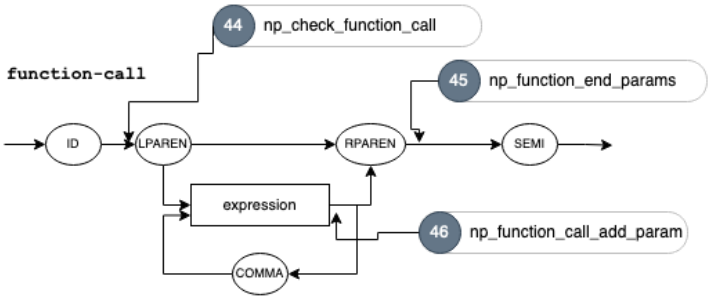
non_conditional_loop



return



function-call

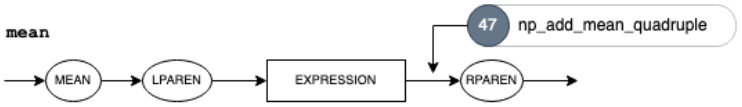


En la gramatica se añadio a function_call_1 que contiene la exoression y la coma, por razones de legibilidad

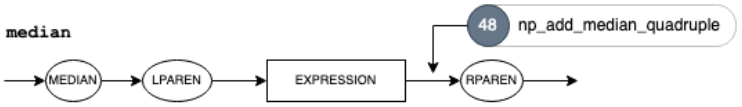
Este mismo este es el mismo diagrama para void_function_call

A excepción que no tiene "SEMI" (punto y coma)

mean

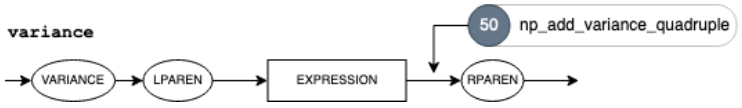


median



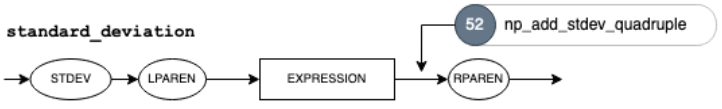
random

variance



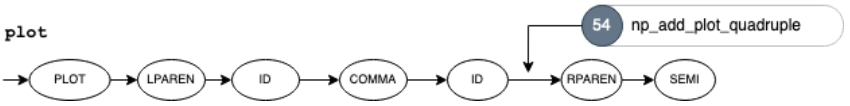
pvariance

standard_deviation



p_standard_deviation

plot



Descripción de acciones semánticas

No.	Punto neurálgico	Descripción
1	np_create_global	Se crea un nuevo scope en el directorio de scopes, con el id de 'program' y se crea el cuádruplo de GOTO para ir al main
2	np_end_program	Se crea un cuádruplo de operación END
3	np_add_vars	Al crear variables, se sacan todas las pendientes del stack y se van creando junto con sus direcciones virtuales y guardando la información en el diccionario de vars del scope correspondiente.
4	np_add_satck_vars	Cuando se están declarando varias variables de un tipo, separadas por coma, se van agregando a un stack, el cual se vacía en np_add_vars.
5	np_create_constant_for_array	Se revisa si la constante numérica existe o no en el directorio de constantes, en caso de que no, se añade.
6	np_create_new_scope	Al crear una nueva función, se crea un nuevo scope en el directorio de funciones, creando también una variable global con el mismo nombre (solo si no es void).
7	np_set_func_start_point	Se guarda en el directorio de funciones, la posición de donde comienza la función actual. Valor que será usado en el cuádruplo GOSUB.
8	np_end_function	Se crea un nuevo cuádruplo de tipo ENDFUNC, se hacen los cálculos del espacio que requiere la función y se guardan en el directorio de funciones. Finalmente se resetean los contadores de memoria.
9	np_create_main_scope	Se crea un nuevo scope en el directorio de scopes, con el id de 'main' y se completa el primer cuádruplo del goto
10	np_end_main	Se hacen los cálculos correspondientes de cuanta memoria requerirá el main.
11	np_add_params_type	Se actualiza la lista de parámetros de la función actual en el directorio de funciones.
12	np_add_id	Al encontrar un ID, se busca en el directorio, validando que exista y este se añade a la pila de operandos, junto con su tipo a la pila de tipos.
13	np_add_operator	Se añade un operador a la pila de operadores
14	np_assign_expression	Se sacan los elementos de la asignación de las pilas de tipos, operandos y operadores. Se valida que la asignación sea del mismo tipo de variables y se crea el cuádruplo para asignación
15	np_check_is_array	Se sacan los elementos de las pilas, se busca el id en el directorio de variables de la función (o de forma global) y se valida que sea arreglo. Después se guarda la dirección virtual en la pila de operandos junto con su tipo y se añade un fondo falso.
16	np_verify_array_dim	Se sacan los elementos de las pilas de operandos y tipos, se saca del directorio de variables el tamaño con el que se definió el arreglo y se crea el cuádruplo de VERIFY con estos datos.
17	np_get_array_address	Se crea el cuádruplo de suma con la dirección base del arreglo (dirección que se agrega al directorio de constantes si es necesario) y el valor al que se quiere acceder. Se genera una dirección de un apuntador y esta se usa para crear el cuádruplo.

18	np_condition_gotof	Se valida que el dato de la pila de tipos sea booleano, y en caso correcto, se crea un nuevo cuádruplo de operación GOTOF con el resultado que se sacó de la pila de operandos.
19	np_condition_end_gotof	Al terminar la condición, se saca de la pila de saltos la posición que se había guardado del inicio de la condición y se actualiza el cuádruplo correspondiente.
20	np_condition_goto_else	Se crea un nuevo cuádruplo de operación GOTO, y se añade la posición a la pila de saltos. Por otro lado se actualiza el cuádruplo del inicio de la condición con la posición actual.
21	np_add_quadruple_logical	Se genera un cuádruplo nuevo con la operación de mayor que, menor que, igual etc. Sacando los operandos y el operador de las pilas. A la par se crea una variable temporal del tipo que lo indique la clase Operation, y esta variable temporal se añade a la pila junto con su tipo.
22	np_add_quadruple_or_and	Se genera un cuádruplo nuevo con la operación de or/and, sacando los operandos y el operador de las pilas. A la par se crea una variable temporal del tipo que lo indique la clase Operation, y esta variable temporal se añade a la pila junto con su tipo.
23	np_add_quadruple_sum_min	Se genera un cuádruplo nuevo con la operación de suma/resta, sacando los operandos y el operador de las pilas. A la par se crea una variable temporal del tipo que lo indique la clase Operation, y esta variable temporal se añade a la pila junto con su tipo.
24	np_add_quadruple_times_div	Se genera un cuádruplo nuevo con la operación de multiplicación/división sacando los operandos y el operador de las pilas. A la par se crea una variable temporal del tipo que lo indique la clase Operation, y esta variable temporal se añade a la pila junto con su tipo.
25	np_add_paren	Se añade el fondo falso (un parentesis) a la pila de operadores.
26	np_pop_paren	Se saca de la pila de operadores un elemento y se valida que sea el fondo falso.
27	np_set_as_negative	Se añade un nuevo cuádruplo que es una multiplicación de -1, para convertir a negativo el número siguiente.
28	np_add_cte_int	Se valida si existe el entero encontrado en el directorio de constantes, si no existe, se crea. Además se añade a la pila de operandos y tipos.
29	np_add_cte_float	Se valida si existe el entero encontrado en el directorio de constantes, si no existe, se crea. Además se añade a la pila de operandos y tipos.
30	np_add_cte_char	Se valida si existe el entero encontrado en el directorio de constantes, si no existe, se crea. Además se añade a la pila de operandos y tipos.
31	np_add_cte_bool	Se valida si existe el entero encontrado en el directorio de constantes, si no existe, se crea. Además se añade a la pila de operandos y tipos.
32	np_add_print_quadruple_exp	Se crea un nuevo cuádruplo PRINT con el valor que se saca de la pila de operandos.
33	np_add_print_quadruple_str	Se crea un nuevo cuádruplo PRINT con el valor
34	np_add_println_quadruple_exp	Se crea un nuevo cuádruplo PRINTLN con el valor que se saca de la pila de operandos.
35	np_add_println_quadruple_str	Se crea un nuevo cuádruplo PRINTLN con el valor

36	np_add_read_quadruple	Se crea un nuevo cuádruplo READ, pasandole el tipo del valor a leer y la dirección el cual se deberá guardar el valor.
37	np_while_init	Se añade la posición actual a la pila de saltos
38	np_while_expression	Se valida que el resultado de la operación sea booleano, sacando el valor de la pila de tipos y operandos y se crea un nuevo cuádruplo de operación GOTOF. Finalmente se añade la posición a la pila de saltos, para actualizarla al final del while.
39	np_while_end_block	Al finalizar el while, se añade un nuevo cuádruplo a la posición que quedó pendiente en la pila de saltos. Y se actualiza el cuádruplo del inicio del while.
40	np_assign_expression_for	Después de la primer expresión del for, se valida que la variable que se esta usando y el valor inicial sean enteros. Después se crea un nuevo cuádruplo con la asignación que se esta haciendo y se guarda la posición en la pila de saltos.
41	np_non_conditional_limit	Se valida que la condición del for sea booleana, y se crea el cuádruplo de GOTOV con este resultado. Tambien se añade esta posición a la pila de saltos
42	np_non_conditional_end	Al finalizar el for, se sacan de las pilas el valor delta y su tipo, se validan, y se crea el cuádruplo de suma para la variable de control del ciclo y delta. Después se crea otro cuádruplo de operación GOTO con la posición que se había guardado del inicio del for, justo antes de la validación. Finalmente se actualiza el cuádruplo pendiente con la posición actual.
43	np_add_return_quadruple	Se sacan los valores de las pilas, y se valida que el valor que se quiere retornar es del tipo que la función indicó. FInalmente se crea el cuádruplo de tipo return con este valor.
44	np_check_function_call	Se verifica que la función que se esta llamando este declarada. Se añade el cuádruplo ERA indicando el id de la función y se añade un fondo falso.
45	np_function_end_params	Al terminar los argumentos de una llamada de función, se valida que sea la cantidad que se tiene en el directorio de funciones. En caso de que la función que se esta llamando no sea void, se crea un nuevo cuádruplo con la asignación de la variable global con el nombre de la función y una nueva temporal del mismo tipo (parche guadalupano).
46	np_function_call_add_param	Se valida que el tipo del parametro corresponda con la firma de la función que se esta llamando, y se crea un cuádruplo de tipo PARAM con el argumento y el número de parametro.
47	np_add_mean_quadruple	Se crea el cuádruplo de MEAN, pasandole el tamaño y dirección del arreglo a calcular junto con una dirección de un temporal en la cual se guardará el resultado.
48	np_add_median_quadruple	Se crea el cuádruplo de MEDIAN, pasandole el tamaño y dirección del arreglo a calcular junto con una dirección de un temporal en la cual se guardará el resultado.
49	np_add_random_quadruple	Se crea el cuádruplo de RANDOM, pasandole el rango de valores para obtener el valor random junto con una dirección de un temporal en la cual se guardará el resultado.
50	np_add_variance_quadruple	Se crea el cuádruplo de VARIANCE, pasandole el tamaño y dirección del arrelgo a calcular junto con una dirección de un temproal en la cual se guardará el resultado.
51	np_add_p_variance_quadruple	Se crea el cuádruplo de PVARIANCE, pasandole el tamaño y

		dirección del arrelgo a calcular junto con una dirección de un temproal en la cual se guardará el resultado.
52	np_add_stdev_quadruple	Se crea el cuádruplo de STDEV, pasandole el tamaño y dirección del arrelgo a calcular junto con una dirección de un temproal en la cual se guardará el resultado.
53	np_add_p_stdev_quadruple	Se crea el cuádruplo de PSTDEV, pasandole el tamaño y dirección del arrelgo a calcular junto con una dirección de un temproal en la cual se guardará el resultado.
54	np_add_plot_quadruple	Se crea el cuádruplo de PLOT, pasandole la dirección base de ambos los arreglos pasados, junto con el tamaño de estos.

Consideraciones semánticas

Algunas consideraciones semánticas importantes son:

- Las variables tienen que ser claras al inicio de su contexto o en un contexto global.
- Los nombres de las variables y funciones son únicos.
- No puede haber una variable global con el mismo nombre que de alguna función.
- Los arreglos sólo son de una dimensión, y sólo pueden contener un tipo de elemento.
- Se aceptan las combinaciones de operaciones más comunes, estas se definen a continuación.

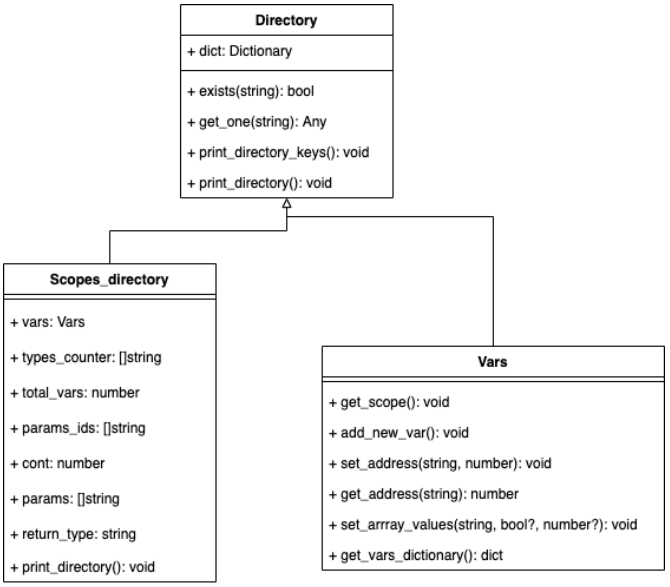
Operando	Operador	Operando	Resultado
int/float/bool/char	=	int/float/bool/char	int/float/bool/char (deben ser iguales)
int	+, -, *	int	int
int	/	int	float
int	<, >, ==, <=, >=	int	boolean
int	+, -, *, /	float	float
float	+, -, *, /	float	float
int	<, >, ==, <=, >=	float	boolean
float	<, >, ==, <=, >=	float	boolean
int	+, *, -	char	int
int	/	char	float
int	<, >, ==, <=, >=	char	boolean
char	+, *, -	char	int
char	/	char	float
char	<, >, ==, <=, >=	char	boolean

float	+, -, *, /	char	float
float	<, >, ==, <=, >=	char	boolean
boolean	&&, ||, !=	boolean	boolean

Administración de Memoria

Para el diseño del compilador se trabajó con diferentes estructuras de datos para poder organizar mejor la información y que a lo largo del desarrollo, fuera fácil de acceder y modificar la misma.

Antes de mostrar la estructuras que se usaron, hay que saber que se definieron tres clases principales. Las tres clases tienen el propósito de ser de apoyo, por lo que principalmente tienen puros métodos.



Estructuras utilizadas

Directorio de contextos

El directorio de contextos o directorio de funciones, es él más se usa en el proceso de compilación. Para este, se implementó como una instancia de la clase Scopes_directory, que hereda de Directory.

Cada contexto es una llave en el directorio, con esto nos aseguramos también que no puedan haber dos funciones con el mismo nombre.

Los principales atributos que tienen cada uno de los elementos del diccionario son:

- El numero total de variables que utitliza
- Un arreglo que indica cuantas variables de que tipo son cada una
- El tipo de retorno de la función
- El orden de los tipos de parámetros que recibe
- Un arreglo de ids en orden de como se definen los de los parámetros.
- El atributo de vars, es otro diccionario que se describe más adelante
- La posición para el apuntador de instrucciones de donde comienza la función

Para una llave de este diccionario sería:

ID	return_type	params	params_ids	cont	types_counter	total_vars	vars
sumTwoNumbers	int	[int, int]	[numberA, numberB]	11	[3,0,0,0]	5	{...}

Tabla de Variables

La tabla de variables es la siguiente estructura más usada de todo el proyecto, ya que un elemento de este directorio, contiene los atributos de una variable, y esto se usa en muchas validaciones.

Esta tabla de variables es del tipo Vars, por lo que se cuentan con varios métodos específicos, como lo es el de add_new_var(), este método se utiliza en la creación de todas las variables del programa, y también de los temporales. A su vez en él se valida que no se hayan excedido los límites de los rangos de las variables, saber si ya existía o no, entre otras cosas.

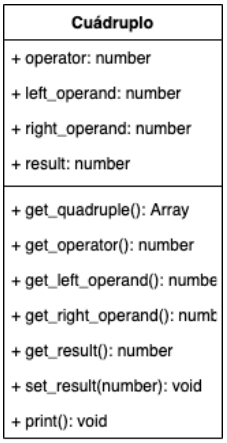
Los principales atributos que tienen cada uno de los elementos del diccionario son:
- El tipo de la variable
- La dirección de memoria de la variable (o inicio, si es arreglo)
- Un booleano que indica si es o no arreglo
- El tamaño del arreglo (indefinido si no es arreglo)

Para una llave de este diccionario sería:

ID	type	adddress	is_array	array_size
result	int	220004	True	11

Cuádruplos

Para la generación de código intermedio se trabajó con cuádruplos. Estos cuádruplos se guardan en un arreglo. Donde cada uno de los elementos es de tipo Cuadruplo.



La estructura de un cuádruplo es la siguiente: [operator, left_operand, right_operand, result]
Donde:

- operator: es el identificador de la operación que se realiza, estos se encuentran definidos en esta sección.
- left_operand: la dirección de memoria del operador izquierdo
- right_operand: la dirección de memoria del operador derecho
- result: la dirección de memoria donde se deberá guardar el resultado de la operación.

El orden recién mencionado no necesariamente es el que se usa en algunos casos, pero esta definición es la más constante a lo largo de todas las operaciones. Existirían ligeras variaciones entre result, left y right operands, sin embargo son casos que dependen de la operación a ejecutar, por lo que hay que revisar su implementación en el código en particular.

Por ejemplo, un cuádruplo podría verse:
[1, 210005, 120005, 320000]

Interpretándolo con los códigos anteriores y las direcciones de memoria definidas también anteriormente se vería:
[
    +,
    dir de memoria de variable local entera,
    dir de memoria de una global flotante,
    dir de una temporal flotante
]

Tabla de Constantes

La tabla de constantes al igual que los anteriores son diccionarios en los que la llave es el valor de la constante y su valor es la dirección virtual donde se está haciendo referencia a ese valor.

Las constantes solo se van agregando conforme se van usando en el programa, por lo que solo se tienen las necesarias para su ejecución más adelante.

```
0               410000
1               410001
10              410002
-1              410003
210000          410004
14              410005
```

Pilas

Estas pilas son parte de los cimientos de todo el proyecto, ya que con estas se van resolviendo todas las operaciones, y asignaciones. Por lo que se utilizan desde una suma, una asignación a una variable, hasta para asignar valores de retorno de las funciones y poder hacer el cambio de contexto.

Un ejemplo de como se verían las pilas sería:
Operandos:
Para mayor legibilidad sería:

```
[..., a, i, var1, resultado]
```

Pero en realidad son puras direcciones virtuales, por ejemplo:

```
[... 210005, 210001, 320000, 320023]
```

Operadores:
En el caso de los operadores, estos sí se mantienen con los caracteres. Aunque cuando se sacan de las pilas y se evalúa para guardar en los cuádruplos, ahí se intercambian por sus identificadores.

```
[..., +, -, +, (, *]
```

Tipo:
En el caso de los tipos, estos sí se mantienen con el valor del token, ya que se tienen definidos en unos enums a lo largo del código. Un ejemplo de como se observarían es:

```
[..., int, int, float, bool]
```

Saltos
Para los saltos, estos guardaban la ubicación del cuádruplo que quedaba pendiente actualizar o las ubicaciones a actualizar.

```
[..., 1, 20, 23]
```

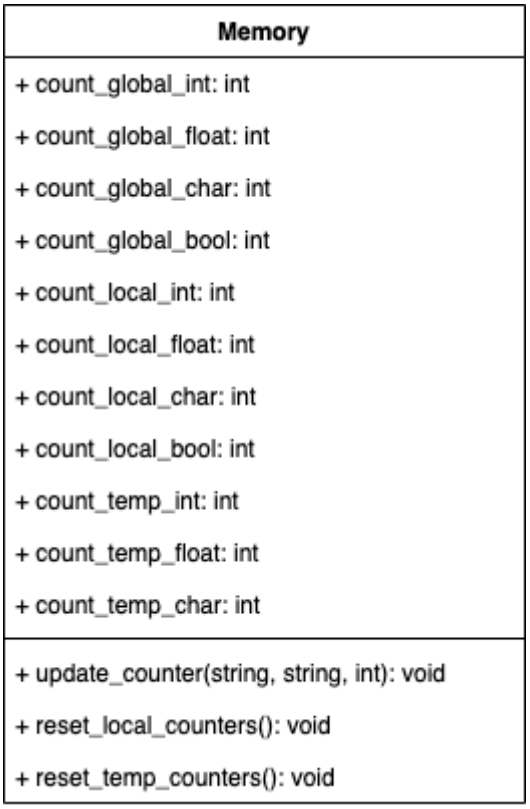
Fila de variables
La fila de variables, se utilizan al momento de ir definiendo variables, ya que puede haber una declaración con varias variables de un tipo, por ejemplo:

```
let var1, var2, var: int;
let arr1, arr2: float[10];
```

Por esto mismo se necesita ir guardando cuales son las que se van a declarar,para esto lo manejamos mejor en una fila para ya que se tiene toda la información de un estatuto se crean todas las variables del tipo.

Clase Memoria

Para llevar a cabo el manejo de las direcciones virtuales en compilación se creó una clase Memoria, la cual tiene como atributos las direcciones de inicio de cada una de las secciones.



Como se puede observar en el diagrama de clases, se ve como existen tres métodos. El de update_couter se encargaba de hacer las validaciones necesarias para verificar que hubiera espacio en la sección que quería reservar. Además de actualizar los datos correspondientes para todos los datos.

Como algo adicional a saber, es que cada que terminaba la declaración de una función se llamaba a estos métodos de reset counters. Ya que en la máquina virtual, cada vez que se inicialice un arreglo, se define su espacio y al terminal se elimina.

Descripción de la Máquina Virtual

Herramientas usadas para el desarrollo

Para el desarrollo de la máquina virtual se usó una computadora con sistema operativo Mac OS - Big Sur / Monterrey.

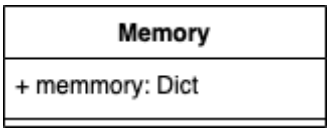
Todo el código fue desarrollando en Python, versión 3.7.10

Como librerías adicionales para manejar estructuras de datos y realizar cálculos se utilizaron:

- sys
- collections (sólo deque), para manejar pilas y filas
- operator, para identificar la operación a realzar
- bisect (solo insort), para ir creando una lista ordenada
- statistics, para calcular funciones estadísticas del lenguaje, como media, varianza, etc.
- random (solo randint) para obtener un numero entero aleatorio

Proceso de administración de memoria

La máquina virtual comienza iniciando la memoria con un espacio definido (para este caso particular, se estableció un límite de 10000 registros, pero se puede ajustar según el caso). Se comienza instanciando una Memoria, que se le llama super_memmory. Esta es la memoria de contexto global, y una vez que se tiene esta, se llena con todas las constantes que se crearon en la fase de compilación. Esto para que al momento de comenzar la ejecución, sea más rápido solo acceder a la dirección de memoria que le corresponde.

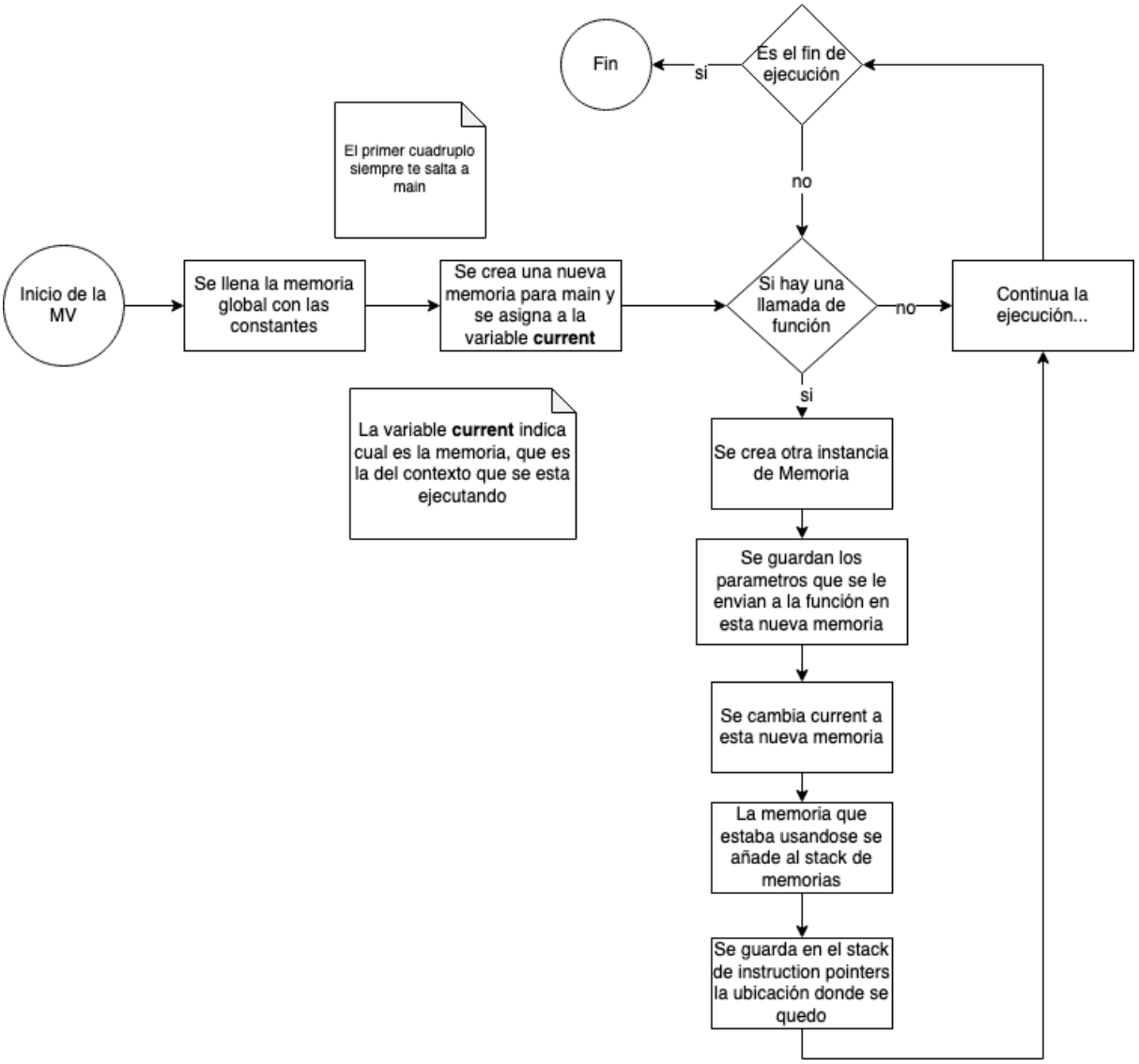


La memoria de la máquina virtual, tiene como límite que se estableció, por lo que en cuanto se llega a estimar que se quedaría sin memoria o se intente agregar un registro más se detendría la ejecución y se marcaría un error por límite de memoria.

El asociar las direcciones con las que se definieron en compilación fue muy transparente porque las direcciones se usaron como llaves, por lo que no hubo algún transformación compleja. Pero los números de estas direcciones virtuales sí se utilizan para realizar validaciones de tipos y contextos.

Para trabajar con las llamadas de funciones, se definió un stack de memorias y el stack de instruction pointers. Ambos son usados para poder mantener las memorias para llamadas recursivas o una llamada sencilla a una función.

Para explicar más visualmente el funcionamiento de de la máquina virtual en cuanto a la administración de memoria se realizó este diagrama de flujo que ejemplifica el flujo.



Pruebas de Funcionamiento

Quick Sort

El código de un quick sort en el lenguaje manchas es el siguiente, se omiten las funciones de fillData y print para poder ver mejor el código:

```
program sort;
let data : int[20];
let arraySize : int;

function partition: int(start: int, end:
int)
let i, pivotIndex, pivot, temp: int;
{
  pivotIndex = start;
  pivot = data[pivotIndex];

  while (start < end) do {
    while (start < arraySize && data[start]
<= pivot) do {
      start = start + 1;
    }

    while (data[end] > pivot) do {
      end = end - 1;
    }

    if(start < end) {
      temp = data[start];
      data[start] = data[end];
      data[end] = temp;
    }
  }

  temp = data[end];
  data[end] = data[pivotIndex];
  data[pivotIndex] = temp;

  return end;
}

function quickSort: void (start : int, end:
int)
let p : int;
{
  if(start < end) {
    p = partition(start, end);

    quickSort(start, p-1);
    quickSort(p+1, end);
  }
}

main ()
{
  arraySize = 20;
  fillArrayData();
  println("Array to be sorted");
  printData();
  println(" ");
  quickSort(0, arraySize - 1);
  println("Array sorted");
  printData();
}
```

El código intermedio generado es:

```
================================================
               Quadruples
================================================
0       20      -1      -1      118
1       13      410000          -1      210000
2       9       210000          110020          340000
3       21      340000          -1      10
4       33      210000          410000          410001
5       1       210000          410003          500000
6       1       210000          410004          310000
7       13      310000          -1      500000
8       1       210000          410002          210000
9       20      -1      -1      2
10      33      410000          410000          410001
11      1       410000          410003          500001
12      13      410005          -1      500001
13      33      410006          410000          410001
14      1       410006          410003          500002
15      13      410007          -1      500002
16      33      410008          410000          410001
17      1       410008          410003          500003
18      13      410009          -1      500003
19      33      410010          410000          410001
20      1       410010          410003          500004
21      13      410002          -1      500004
22      33      410011          410000          410001
23      1       410011          410003          500005
24      13      410012          -1      500005
25      33      410013          410000          410001
26      1       410013          410003          500006
27      13      410014          -1      500006
28      33      410015          410000          410001
29      1       410015          410003          500007
30      13      410016          -1      500007
31      33      410017          410000          410001
32      1       410017          410003          500008
33      13      410018          -1      500008
34      33      410019          410000          410001
35      1       410019          410003          500009
36      13      410000          -1      500009
37      26      -1      -1      -1
38      32      -1      -1      ""
39      13      410000          -1      210000
40      9       210000          110020          340000
41      21      340000          -1      48
42      33      210000          410000          410001
43      1       210000          410003          500010
44      31      -1      -1      500010
```

```
45      31      -1      -1      " "
46      1       210000          410002          210000
47      20      -1      -1      40
48      32      -1      -1      ""
49      26      -1      -1      -1
50      13      210000          -1      210003
51      33      210003          410000          410001
52      1       210003          410003          500011
53      13      500011          -1      210004
54      5       210000          210001          340000
55      22      340000          -1      86
56      5       210000          110020          340001
57      33      210000          410000          410001
58      1       210000          410003          500012
59      6       500012          210004          340002
60      11      340001          340002          340003
61      22      340003          -1      65
62      1       210000          410002          310000
63      13      310000          -1      210000
64      20      -1      -1      56
65      33      210001          410000          410001
66      1       210001          410003          500013
67      7       500013          210004          340004
68      22      340004          -1      72
69      2       210001          410002          310001
70      13      310001          -1      210001
71      20      -1      -1      65
72      5       210000          210001          340005
73      22      340005          -1      85
74      33      210000          410000          410001
75      1       210000          410003          500014
76      13      500014          -1      210005
77      33      210000          410000          410001
78      1       210000          410003          500015
79      33      210001          410000          410001
80      1       210001          410003          500016
81      13      500016          -1      500015
82      33      210001          410000          410001
83      1       210001          410003          500017
84      13      210005          -1      500017
85      20      -1      -1      54
86      33      210001          410000          410001
87      1       210001          410003          500018
88      13      500018          -1      210005
89      33      210001          410000          410001
90      1       210001          410003          500019
91      33      210003          410000          410001
92      1       210003          410003          500020
93      13      500020          -1      500019
94      33      210003          410000          410001
95      1       210003          410003          500021
96      13      210005          -1      500021
97      30      -1      -1      210001
98      26      -1      -1      -1
99      5       210000          210001          340000
100     22      340000          -1      117
101     24      -1      -1      partition
102     25      210000          -1      _param_0
103     25      210001          -1      _param_1
104     23      partition       -1      50
105     13      110021          -1      310000
106     13      310000          -1      210002
107     24      -1      -1      quickSort
108     25      210000          -1      _param_0
109     2       210002          410002          310001
110     25      310001          -1      _param_1
111     23      quickSort       -1      99
112     24      -1      -1      quickSort
113     1       210002          410002          310002
114     25      310002          -1      _param_0
115     25      210001          -1      _param_1
116     23      quickSort       -1      99
117     26      -1      -1      -1
118     13      410001          -1      110020
119     24      -1      -1      fillArrayData
120     23      fillArrayData   -1      1
121     32      -1      -1      "Array to be sorted"
122     24      -1      -1      printData
123     23      printData       -1      38
124     32      -1      -1      " "
125     24      -1      -1      quickSort
126     25      410000          -1      _param_0
127     2       110020          410002          310000
128     25      310000          -1      _param_1
129     23      quickSort       -1      99
130     32      -1      -1      "Array sorted"
131     24      -1      -1      printData
132     23      printData       -1      38
133     27      -1      -1      -1
```

El resultado de ejecución es:

```
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos
ython3 main.py compiler/test_files/quickSort.manchas
correct
==================================================
                Starts virtual machine
==================================================
-Execution-starts-

Array to be sorted

15 6 30 8 19 10 0 12 13 1 15 8 7 18 19 20 21 100 78 24

Array sorted

0 1 6 7 8 8 10 12 13 15 15 18 19 19 20 21 24 30 78 100


-Execution-ended-
```

Factorial (cíclico)

El código en para encontrar el factorial (de forma cíclica) de un número en el lenguaje manchas es:

```
program factorialCyclic;

function clacFactorial: int (value:int)
let sum, i: int;
{
  sum = 1;
  for ( i = 1 to i > value by 1) {
```

```
    sum = sum * i;
  }
  return sum;
}

function pelos: void()
let x, res: int;
{
  read(x);
  res = clacFactorial(x);
  print("El resultado: ", res);
}

main()
let sum, i, x :int;
{
  pelos();
}
```

El código intermedio generado es:

```
================================================
                  Quadruples
================================================
0       20      -1      -1      20
1       13      410001          -1      210001
2       13      410001          -1      210002
3       7       210002          210000          340000
4       21      340000          -1      9
5       4       210001          210002          310000
6       13      310000          -1      210001
7       1       210002          410001          210002
8       20      -1      -1      3
9       30      -1      -1      210001
10      26      -1      -1      -1
11      34      -1      1       210000
12      24      -1      -1      clacFactorial
13      25      210000          -1      _param_0
14      23      clacFactorial   -1      1
15      13      110000          -1      310000
16      13      310000          -1      210001
17      31      -1      -1      "El resultado: "
18      31      -1      -1      210001
19      26      -1      -1      -1
20      24      -1      -1      pelos
21      23      pelos   -1      11
22      27      -1      -1      -1
```

El resultado de su ejecución es:

```
ython3 main.py compiler/test_files/factorialCyclic.manchas
correct
========================================================
                Starts virtual machine
========================================================
-Execution-starts-

15
El resultado: 1307674368000

-Execution-ended-
```

Factorial Recursivo

El código en para encontrar el factorial (de forma recursiva) de un número en el lenguaje manchas es:

```
program factorialRecursive;

function multiply:int (number : int)
{
  if (number <= 1) {
    return 1;
  }else {
    return (number * multiply(number - 1));
```

```
  }
}

main()
let sum, x: int;
{
  x = 5;

  sum = multiply(x);
  print("Resultado ", sum);
}
```

El código intermedio generado es:

```
=====================================================
                    Quadruples
=====================================================
0       20      -1      -1      13
1       6       210000          410001          340000
2       22      340000          -1      5
3       30      -1      -1      410001
4       20      -1      -1      12
5       24      -1      -1      multiply
6       2       210000          410001          310000
7       25      310000          -1      _param_0
8       23      multiply        -1      1
9       13      110000          -1      310001
10      4       210000          310001          310002
11      30      -1      -1      310002
12      26      -1      -1      -1
13      13      410002          -1      210001
14      24      -1      -1      multiply
15      25      210001          -1      _param_0
16      23      multiply        -1      1
17      13      110000          -1      310000
18      13      310000          -1      210000
19      31      -1      -1      "Resultado "
20      31      -1      -1      210000
21      27      -1      -1      -1
```

El resultado de su ejecución es:

```
ython3 main.py compiler/test_files/factorialRecursive.manchas
correct
=====================================================
                Starts virtual machine
=====================================================
-Execution-starts-

Resultado 120

-Execution-ended-
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellan
```

Fibonacci (cíclico)

El código en para encontrar un valor en la sucesión de fibonacci (de forma cíclica) en el lenguaje manchas es:

```
program fibonacciCyclic;

main()
let n, i, value, temp: int;
{
  value = 0;
  temp = 1;

  println("Ingrese el numero de valores a calcular para la sucesión de Fibonacci");
```

```
  read(n);
  if (n > 0){
    for(i = 1 to i > n by 1) {
      temp = temp + value;
      value = temp - value;
    }
    print("El valor de la serie de Fibonacci en la posición ", n);
    print(" es: ", value);
  }else {
    println("El número debe ser mayor a 0");
  }

}
```

El código intermedio generado es:

```
=====================================================
                    Quadruples
=====================================================
0       20      -1      -1      1
1       13      410000          -1      210002
2       13      410001          -1      210003
3       32      -1      -1      "Ingrese el numero de valores a calcular para la sucesión de Fibonacci"
4       34      -1      1       210000
5       7       210000          410000          340000
6       22      340000          -1      21
7       13      410001          -1      210001
8       7       210001          210000          340001
9       21      340001          -1      16
10      1       210003          210002          310000
11      13      310000          -1      210003
12      2       210003          210002          310001
13      13      310001          -1      210002
14      1       210001          410001          210001
15      20      -1      -1      8
16      31      -1      -1      "El valor de la serie de Fibonacci en la posición "
17      31      -1      -1      210000
18      31      -1      -1      " es: "
19      31      -1      -1      210002
20      20      -1      -1      22
21      32      -1      -1      "El número debe ser mayor a 0"
22      27      -1      -1      -1
```

El resultado de su ejecución es:

```
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos/Google
ython3 main.py compiler/test_files/fibonacciCyclic.manchas
correct
=====================================================
               Starts virtual machine
=====================================================
-Execution-starts-

Ingrese el numero de valores a calcular para la sucesión de Fibonacci
15
El valor de la serie de Fibonacci en la posición 15 es: 610

-Execution-ended-
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos/Google
```

Fibonacci Recursivo

El código en para encontrar un valor en la sucesión de fibonacci (de forma recursiva) en el lenguaje manchas es:

```
program fibonacciRecursive;

function fibonacci: int(n : int) {
  if(n == 0 || n == 1) {
```

```
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}

main()
let n, value: int;
{
  value = 0;
  println("Ingrese el numero de valores a calcular para la sucesión de Fibonacci");
  read(n);

  if (n > 0){
    value = fibonacci(n);
    print("El valor de la serie de Fibonacci en la posición ", n);
    print(" es: ", value);
  }else {
    println("El número debe ser mayor a 0");
  }

}
```

El código intermedio generado es:

```
=====================================================
                    Quadruples
=====================================================
0       20      -1      -1      20
1       9       210000          410000          340000
2       9       210000          410001          340001
3       12      340000          340001          340002
4       22      340002          -1      7
5       30      -1      -1      210000
6       20      -1      -1      19
7       24      -1      -1      fibonacci
8       2       210000          410001          310000
9       25      310000          -1      _param_0
10      23      fibonacci       -1      1
11      13      110000          -1      310001
12      24      -1      -1      fibonacci
13      2       210000          410002          310002
14      25      310002          -1      _param_0
15      23      fibonacci       -1      1
16      13      110000          -1      310003
17      1       310001          310003          310004
18      30      -1      -1      310004
19      26      -1      -1      -1
20      13      410000          -1      210001
21      32      -1      -1      "Ingrese el numero de
valores a calcular para la sucesión de Fibonacci"
22      34      -1      1       210000
23      7       210000          410000          340000
24      22      340000          -1      35
25      24      -1      -1      fibonacci
26      25      210000          -1      _param_0
27      23      fibonacci       -1      1
28      13      110000          -1      310000
29      13      310000          -1      210001
30      31      -1      -1      "El valor de la serie de
Fibonacci en la posición "
31      31      -1      -1      210000
32      31      -1      -1      " es: "
33      31      -1      -1      210001
34      20      -1      -1      36
35      32      -1      -1      "El número debe ser
mayor a 0"
36      27      -1      -1      -1
```

El resultado de su ejecución es:

```
ython3 main.py compiler/test_files/fibonacciRecursive.manchas
correct
=====================================================
                Starts virtual machine
=====================================================
-Execution-starts-

Ingrese el numero de valores a calcular para la sucesión de Fibonacci
18
El valor de la serie de Fibonacci en la posición 18 es: 2584

-Execution-ended-
```

Find

Para encontrar un valor de un arreglo se crea el siguiente código:

Nota: Se omiten las funciones de fillArrayData y PrintData por legibilidad en este documento.

```
program search;
let data : int[20];
let arraySize : int;

function findValue:int(target: int)
let i: int;
{
  for(i = 0 to i == arraySize by 1) {
    if(data[i] == target) {
      return i;
    }
  }
  return -1;
}

main ()
let valueToFind, valuePosition: int;
{
  arraySize = 20;

  fillArrayData();
  println("El arreglo actual es: ");
  printData();

  println("¿Cuál es el valor que quiere buscar?");
  read(valueToFind);

  valuePosition = findValue(valueToFind);
  if(valuePosition < 0) {
    println("El valor no se encontró en el arreglo");
  }else {
    print("El valor se encuentra en la posición ", valuePosition);
  }
}
```

El código intermedio generado es:

```
=================================================
                  Quadruples
=================================================
0       20      -1      -1      63
1       13      410000          -1      210000
2       9       210000          110020          340000
3       21      340000          -1      10
4       33      210000          410000          410001
5       1       210000          410003          500000
6       1       210000          410004          310000
7       13      310000          -1      500000
8       1       210000          410002          210000
9       20      -1      -1      2
10      33      410000          410000          410001
11      1       410000          410003          500001
12      13      410005          -1      500001
13      33      410006          410000          410001
14      1       410006          410003          500002
15      13      410007          -1      500002
16      33      410008          410000          410001
17      1       410008          410003          500003
18      13      410009          -1      500003
19      33      410010          410000          410001
20      1       410010          410003          500004
21      13      410002          -1      500004
22      33      410011          410000          410001
23      1       410011          410003          500005
24      13      410012          -1      500005
25      33      410013          410000          410001
26      1       410013          410003          500006
27      13      410014          -1      500006
28      33      410015          410000          410001
29      1       410015          410003          500007
30      13      410016          -1      500007
31      33      410017          410000          410001
32      1       410017          410003          500008
33      13      410018          -1      500008
34      33      410019          410000          410001
35      1       410019          410003          500009
36      13      410000          -1      500009
37      26      -1      -1      -1
38      32      -1      -1      ""
39      13      410000          -1      210000
40      9       210000          110020          340000
41      21      340000          -1      48
42      33      210000          410000          410001
43      1       210000          410003          500010
44      31      -1      -1      500010
45      31      -1      -1      " "
46      1       210000          410002          210000
47      20      -1      -1      40
48      32      -1      -1      ""
49      26      -1      -1      -1
50      13      410000          -1      210001
51      9       210001          110020          340000
52      21      340000          -1      60
53      33      210001          410000          410001
54      1       210001          410003          500011
55      9       500011          210000          340001
56      22      340001          -1      58
57      30      -1      -1      210001
58      1       210001          410002          210001
```

```
59      20      -1      -1      51
60      4       410020          410002          310000
61      30      -1      -1      310000
62      26      -1      -1      -1
63      13      410001          -1      110020
64      24      -1      -1      fillArrayData
65      23      fillArrayData   -1      1
66      32      -1      -1      "El arreglo actual es: "
67      24      -1      -1      printData
68      23      printData       -1      38
69      32      -1      -1      "¿Cuál es el valor que quiere
buscar?"
70      34      -1      1       210000
71      24      -1      -1      findValue
72      25      210000          -1      _param_0
73      23      findValue       -1      50
74      13      110021          -1      310000
75      13      310000          -1      210001
76      5       210001          410000          340000
77      22      340000          -1      80
78      32      -1      -1      "El valor no se encontró en el
arreglo"
79      20      -1      -1      82
80      31      -1      -1      "El valor se encuentra en la
posición "
81      31      -1      -1      210001
82      27      -1      -1      -1
```

El resultado de su ejecución es:

```
ython3 main.py compiler/test_files/find.manchas
correct
=================================================
                Starts virtual machine
=================================================
-Execution-starts-

El arreglo actual es:

15 6 30 8 19 10 0 12 13 1 15 8 7 18 19 20 21 100 78 24
¿Cuál es el valor que quiere buscar?
7
El valor se encuentra en la posición 12

-Execution-ended-
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastel
ython3 main.py compiler/test_files/find.manchas
correct
=================================================
                Starts virtual machine
=================================================
-Execution-starts-

El arreglo actual es:

15 6 30 8 19 10 0 12 13 1 15 8 7 18 19 20 21 100 78 24
¿Cuál es el valor que quiere buscar?
122
El valor no se encontró en el arreglo


-Execution-ended-
```

Operaciones en arreglos

Un programa para realizar operaciones básicas sobre un arreglo en el lenguaje manchas es:
Nota: Se omiten las funciones de fillArrayData y PrintData por legibilidad en este documento.

```
program arrayOperations;
let data : float[20];
let arraySize: int;

function getOperation : int()
let option: int;
{
  println("================== Opciones
de operaciones ==================");
  println("0 -> Salir");
  println("1 -> Suma un valor a todos
los elementos del arreglo");
  println("2 -> Multiplica un valor a
todos los elementos del arreglo");
  println("3 -> Divide un valor a todos
los elementos del arreglo");
  println("4 -> Suma todos los
elementos del arreglo");
  println("5 -> Multiplica todos los
elementos del arreglo");

println("==============================
================================");
  print("Ingrese la opción:  ");
  read(option);
  return option;
}

function sumValueToArray: void()
let value: float;
let i: int;
{
  print("Ingrese el valor que se sumara
a todos los elementos:  ");
  read(value);
```

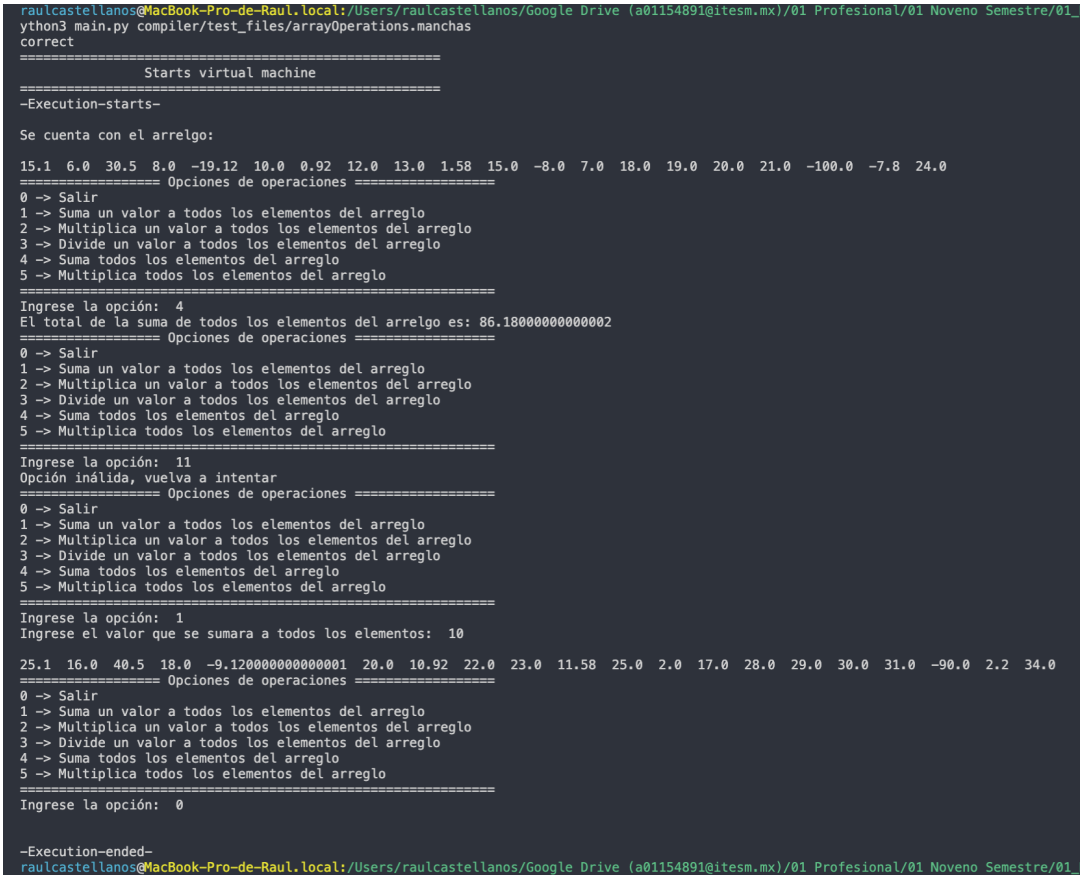
```
  for(i = 0 to i == arraySize by 1) {
    data[i] = data[i] + value;
  }
}

function multiplyValueToArray: void()
let value: float;
let i: int;
{
  print("Ingrese el valor que con el
que se multiplicara a todos los
elementos:  ");
  read(value);
  for(i = 0 to i == arraySize by 1) {
    data[i] = data[i] * value;
  }
}

function divideValueToArray: void()
let value: float;
let i: int;
{
  print("Ingrese el valor que con el
que se dividira a todos los elementos:
");
  read(value);
  for(i = 0 to i == arraySize by 1) {
    data[i] = data[i] / value;
  }
}

function sumAllElements: void()
let total: float;
let i: int;
{
  total = 0.0;
  for(i = 0 to i == arraySize by 1) {
    total = total + data[i];
  }
  print("El total de la suma de todos
los elementos del arrelgo es: ",
total);
  println("");
}

function multiplyAllElements: void()
let total: float;
let i: int;
{
  total = 1.0;
  for(i = 0 to i == arraySize by 1) {
    total = total * data[i];
  }
  print("El total de la multiplicación
de todos los elementos del arrelgo es:
", total);
  println("");
}

main ()
let option: int;
{
  arraySize = 20;
  option = -1;
  fillArrayData();
  println("Se cuenta con el arrelgo:");
  printData();


  while (option != 0) do {
    option = getOperation();
    if (option < 0 || option > 5) {
      println("Opción inálida, vuelva a
intentar");
    }else {

      if (option == 1) {
        sumValueToArray();
        printData();
      }
      if (option == 2) {
        multiplyValueToArray();
        printData();
      }
      if (option == 3) {
        divideValueToArray();
        printData();
      }
      if (option == 4) {
        sumAllElements();
      }
      if (option == 5) {
        multiplyAllElements();
      }


    }
  }

}
```

El código intermedio generado es:

```
=====================================================
                    Quadruples
=====================================================
0       20      -1      -1      136
1       13      410000          -1      210000
2       9       210000          110000          340000
3       21      340000          -1      10
4       33      210000          410000          410001
5       1       210000          410003          500000
6       1       210000          420000          320000
7       13      320000          -1      500000
8       1       210000          410002          210000
```

```
9       20      -1      -1      2
10      33      410000          410000          410001
11      1       410000          410003          500001
12      13      420001          -1      500001
13      33      410004          410000          410001
14      1       410004          410003          500002
15      4       410005          420002          320001
16      13      320001          -1      500002
17      33      410006          410000          410001
18      1       410006          410003          500003
19      13      420003          -1      500003
20      33      410007          410000          410001
21      1       410007          410003          500004
22      13      420004          -1      500004
23      33      410008          410000          410001
24      1       410008          410003          500005
25      4       410005          420005          320002
26      13      320002          -1      500005
27      33      410009          410000          410001
28      1       410009          410003          500006
29      4       410005          420006          320003
30      13      320003          -1      500006
31      33      410010          410000          410001
32      1       410010          410003          500007
33      4       410005          420007          320004
34      13      320004          -1      500007
35      33      410011          410000          410001
36      1       410011          410003          500008
37      13      420008          -1      500008
38      33      410012          410000          410001
39      1       410012          410003          500009
40      13      420009          -1      500009
41      26      -1      -1      -1
42      32      -1      -1      ""
43      13      410000          -1      210000
44      9       210000          110000          340000
45      21      340000          -1      52
46      33      210000          410000          410001
47      1       210000          410003          500010
48      31      -1      -1      500010
49      31      -1      -1      "  "
50      1       210000          410002          210000
51      20      -1      -1      44
52      32      -1      -1      ""
53      26      -1      -1      -1
54      32      -1      -1      "================= Opciones de
operaciones ================="
55      32      -1      -1      "0 -> Salir"
56      32      -1      -1      "1 -> Suma un valor a todos los
elementos del arreglo"
57      32      -1      -1      "2 -> Multiplica un valor a
todos los elementos del arreglo"
58      32      -1      -1      "3 -> Divide un valor a todos
los elementos del arreglo"
59      32      -1      -1      "4 -> Suma todos los elementos
del arreglo"
60      32      -1      -1      "5 -> Multiplica todos los
elementos del arreglo"
61      32      -1      -1
"============================================================"
62      31      -1      -1      "Ingrese la opción:  "
63      34      -1      1       210000
64      30      -1      -1      210000
65      26      -1      -1      -1
66      31      -1      -1      "Ingrese el valor que se sumara
a todos los elementos:  "
67      34      -1      2       220000
68      13      410000          -1      210000
69      9       210000          110000          340000
70      21      340000          -1      79
71      33      210000          410000          410001
72      1       210000          410003          500011
73      33      210000          410000          410001
74      1       210000          410003          500012
75      1       500012          220000          320000
76      13      320000          -1      500011
77      1       210000          410002          210000
78      20      -1      -1      69
79      26      -1      -1      -1
80      31      -1      -1      "Ingrese el valor que con el que
se multiplicara a todos los elementos:  "
81      34      -1      2       220000
82      13      410000          -1      210000
83      9       210000          110000          340000
84      21      340000          -1      93
85      33      210000          410000          410001
86      1       210000          410003          500013
87      33      210000          410000          410001
88      1       210000          410003          500014
89      4       500014          220000          320000
90      13      320000          -1      500013
91      1       210000          410002          210000
92      20      -1      -1      83
93      26      -1      -1      -1
94      31      -1      -1      "Ingrese el valor que con el que
se dividira a todos los elementos:  "
95      34      -1      2       220000
96      13      410000          -1      210000
97      9       210000          110000          340000
98      21      340000          -1      107
99      33      210000          410000          410001
100     1       210000          410003          500015
101     33      210000          410000          410001
102     1       210000          410003          500016
103     3       500016          220000          320000
104     13      320000          -1      500015
105     1       210000          410002          210000
106     20      -1      -1      97
107     26      -1      -1      -1
108     13      410000          -1      220000
109     13      410000          -1      210000
110     9       210000          110000          340000
111     21      340000          -1      118
112     33      210000          410000          410001
113     1       210000          410003          500017
114     1       220000          500017          320000
115     13      320000          -1      220000
116     1       210000          410002          210000
117     20      -1      -1      110
118     31      -1      -1      "El total de la suma de todos
los elementos del arrelgo es: "
119     31      -1      -1      220000
120     32      -1      -1      ""
121     26      -1      -1      -1
122     13      410002          -1      220000
123     13      410000          -1      210000
124     9       210000          110000          340000
125     21      340000          -1      132
126     33      210000          410000          410001
127     1       210000          410003          500018
128     4       220000          500018          320000
129     13      320000          -1      220000
130     1       210000          410002          210000
131     20      -1      -1      124
132     31      -1      -1      "El total de la multiplicación
de todos los elementos del arrelgo es: "
133     31      -1      -1      220000
134     32      -1      -1      ""
135     26      -1      -1      -1
136     13      410001          -1      110000
137     4       410005          410002          310000
138     13      310000          -1      210000
139     24      -1      -1      fillArrayData
140     23      fillArrayData   -1      1
141     32      -1      -1      "Se cuenta con el arrelgo:"
142     24      -1      -1      printData
143     23      printData       -1      42
144     10      210000          410000          340000
145     22      340000          -1      183
146     24      -1      -1      getOperation
147     23      getOperation    -1      54
148     13      110001          -1      310001
149     13      310001          -1      210000
150     5       210000          410000          340001
151     7       210000          420000          340002
152     12      340001          340002          340003
153     22      340003          -1      156
154     32      -1      -1      "Opción inálida, vuelva a
intentar"
155     20      -1      -1      182
156     9       210000          410002          340004
157     22      340004          -1      162
158     24      -1      -1      sumValueToArray
159     23      sumValueToArray         -1      66
160     24      -1      -1      printData
161     23      printData       -1      42
162     9       210000          410006          340005
163     22      340005          -1      168
164     24      -1      -1      multiplyValueToArray
165     23      multiplyValueToArray    -1      80
166     24      -1      -1      printData
167     23      printData       -1      42
168     9       210000          410013          340006
169     22      340006          -1      174
170     24      -1      -1      divideValueToArray
171     23      divideValueToArray      -1      94
172     24      -1      -1      printData
173     23      printData       -1      42
174     9       210000          410004          340007
175     22      340007          -1      178
176     24      -1      -1      sumAllElements
177     23      sumAllElements          -1      108
178     9       210000          420000          340008
179     22      340008          -1      182
180     24      -1      -1      multiplyAllElements
181     23      multiplyAllElements     -1      122
182     20      -1      -1      144
183     27      -1      -1      -1
```

El resultado de su ejecución es:



myRLike

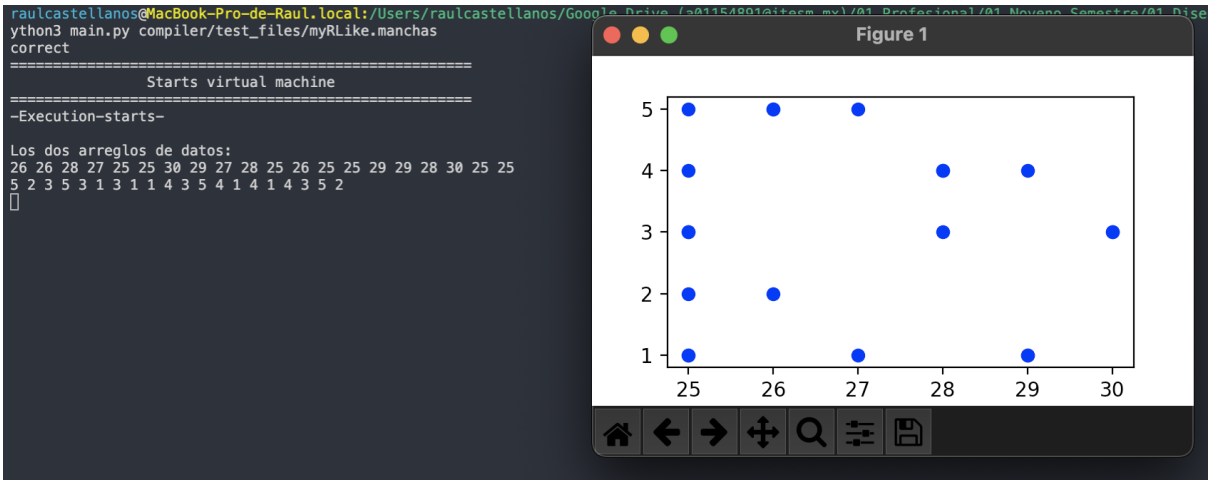
Se creó este pequeño programa en el lenguaje manchas para ilustrar cómo es la gráfica de puntos.

```
program myRLike;
let datay: int[20];
let datax: int[20];
let arraySize: int;

function fillArrayData: void()
let i: int;
{
  for(i = 0 to i == arraySize by 1) {
    datay[i] = random(25, 30);
    print(datay[i], " ");
  }
  println("");
}

function fillArrayData2: void()
let i: int;
{
  for(i = 0 to i == arraySize by 1) {
    datax[i] = random(1, 5);
    print(datax[i], " ");
  }
  println("");
}

main()
{
  arraySize = 20;
  println("Los dos arreglos de
datos:");
  fillArrayData();
  fillArrayData2();

  plot(datay, datax);

}
```

El código intermedio generado es:

```
==================================================
                    Quadruples
==================================================
0       20      -1      -1      31
1       13      410000          -1      210000
2       9       210000          110040          340000
3       21      340000          -1      14
4       33      210000          410000          410001
5       1       210000          410003          500000
6       46      25      30      310000
7       13      310000          -1      500000
8       33      210000          410000          410001
9       1       210000          410003          500001
10      31      -1      -1      500001
11      31      -1      -1      " "
12      1       210000          410002          210000
13      20      -1      -1      2
14      32      -1      -1      ""
15      26      -1      -1      -1
16      13      410000          -1      210000
17      9       210000          110040          340000
18      21      340000          -1      29
19      33      210000          410000          410001
20      1       210000          410004          500002
21      46      1       5       310000
22      13      310000          -1      500002
23      33      210000          410000          410001
24      1       210000          410004          500003
25      31      -1      -1      500003
26      31      -1      -1      " "
27      1       210000          410002          210000
28      20      -1      -1      17
29      32      -1      -1      ""
30      26      -1      -1      -1
31      13      410001          -1      110040
32      32      -1      -1      "Los dos arreglos de datos:"
33      24      -1      -1      fillArrayData
34      23      fillArrayData   -1      1
35      24      -1      -1      fillArrayData2
36      23      fillArrayData2          -1      16
37      47      110000          110020          20
38      27      -1      -1      -1
```

El resultado de ejecución es:



Operaciones de estadística

Con el lenguje Manchas, se creo este programa para visualizar algunas funciones particulares del lenguaje:

```
program myRLike;
let data: int[20];
let arraySize: int;

function fillArrayData: void()
let i: int;
{
  for(i = 0 to i == arraySize by 1) {
    data[i] = random(19, 25);
    print(data[i], " ");
  }
}

function printData: void()
let i: int;
{
  println("");
  for(i = 0 to i == arraySize by 1) {
    print(data[i], " ");
  }
  println("");
}
```

```
main()
{
  arraySize = 20;
  println("Las edades de un grupo de 20
personas es:");
  fillArrayData();
  println("");
  print("El promedio de estos datos es:
", mean(data));
  println("");
  print("La mediana de estos datos es:
", median(data));
  println("");
  print("La varianza de la muestra de
estos datos es: ", variance(data));
  println("");
  print("La desviación estandar de la
muestra de estos datos es: ",
stdev(data));
}
```

El código intermedio generado es:

```
=====================================================
                    Quadruples
=====================================================
0       20      -1      -1      27
1       13      410000          -1      210000
2       9       210000          110020          340000
3       21      340000          -1      14
4       33      210000          410000          410001
5       1       210000          410003          500000
6       46      19      25      310000
7       13      310000          -1      500000
8       33      210000          410000          410001
9       1       210000          410003          500001
10      31      -1      -1      500001
11      31      -1      -1      " "
12      1       210000          410002          210000
13      20      -1      -1      2
14      26      -1      -1      -1
15      32      -1      -1      ""
16      13      410000          -1      210000
17      9       210000          110020          340000
18      21      340000          -1      25
19      33      210000          410000          410001
20      1       210000          410003          500002
21      31      -1      -1      500002
22      31      -1      -1      " "
23      1       210000          410002          210000
24      20      -1      -1      17
25      32      -1      -1      ""
26      26      -1      -1      -1
27      13      410001          -1      110020
28      32      -1      -1      "Las edades de un grupo
de 20 personas es:"
29      24      -1      -1      fillArrayData
30      23      fillArrayData   -1      1
31      32      -1      -1      ""
32      31      -1      -1      "El promedio de estos
datos es: "
33      40      20      110000          320000
34      31      -1      -1      320000
35      32      -1      -1      ""
36      31      -1      -1      "La mediana de estos
datos es: "
37      41      20      110000          320001
38      31      -1      -1      320001
39      32      -1      -1      ""
40      31      -1      -1      "La varianza de la
muestra de estos datos es: "
41      44      20      110000          320002
42      31      -1      -1      320002
43      32      -1      -1      ""
44      31      -1      -1      "La desviación estandar
de la muestra de estos datos es: "
45      45      20      110000          320003
46      31      -1      -1      320003
47      27      -1      -1      -1
```

El resultado de ejecución es:

```
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos/Google Drive (a011548
ython3 main.py compiler/test_files/statistics.manchas
correct
=====================================================
                Starts virtual machine
=====================================================
-Execution-starts-

Las edades de un grupo de 20 personas es:
20 20 20 20 19 24 24 22 23 21 23 19 19 25 24 22 19 20 22 20
El promedio de estos datos es: 21.3
La mediana de estos datos es: 20.5
La varianza de la muestra de estos datos es: 3.905263157894737
La desviación estandar de la muestra de estos datos es: 1.9761738683361687

-Execution-ended-
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos/Google Drive (a011548
ython3 main.py compiler/test_files/statistics.manchas
correct
=====================================================
                Starts virtual machine
=====================================================
-Execution-starts-

Las edades de un grupo de 20 personas es:
24 23 23 19 19 23 21 20 24 22 21 21 25 19 25 20 21 19 19 23
El promedio de estos datos es: 21.55
La mediana de estos datos es: 21.0
La varianza de la muestra de estos datos es: 4.36578947368421
La desviación estandar de la muestra de estos datos es: 2.0894471693929497

-Execution-ended-
raulcastellanos@MacBook-Pro-de-Raul.local:/Users/raulcastellanos/Google Drive (a011548
```

Documentación del código del proyecto

En general todo el código que se encuentra en el repositorio esta documento y se busco que fuera muy funcional, por lo que los nombres de las funciones y comentarios ayudan mucho a entender el código.

A continuación se muestran algunos fragmentos de código que se crearon:

Este fragmento de código son los puntos neurálgicos que se utilizan para crear los cuádruplos de las principales operaciones, suma, resta, comparación etc. Como se puede ver, todos comparten la misma función, solo se le pasa que tipo de cuádruplo es el que se agregaría, pero se valida que sea el que corresponde para respetar la jerarquía de operaciones.

```
def p_np_add_quadruple_sum_min(p):
        '''np_add_quadruple_sum_min : '''
        generate_new_quadruple(['+', '-'])

def p_np_add_quadruple_times_div(p):
        '''np_add_quadruple_times_div : '''
        generate_new_quadruple(['*', '/'])

def p_np_add_quadruple_logical(p):
        '''np_add_quadruple_logical : '''
        generate_new_quadruple(['<', '<=', '>', '>=', '==', '!='])

def p_np_add_quadruple_or_and(p):
        '''np_add_quadruple_or_and : '''
        generate_new_quadruple(['||', '&&'])
```

La definición de la función mostrada es:

```
def generate_new_quadruple(operator_to_check):
        global quadruples, operands, operators, types, program_scopes, current_scope,
tempsCount
        if len(operators) > 0 and (operators[-1] in operator_to_check):
        # Get operator and operands from stacks
        # For example:
        # operator = + right_operand = 10.5  right_type = FLOAT
        #              left_operand = 10     left_type = INTEGER
        # res_type should be FLOAT
        operator = operators.pop()
        right_operand = operands.pop()
        right_type = types.pop()
        left_operand = operands.pop()
        left_type = types.pop()
        # Get the resulting type of the operation
        res_type = Operation.getType(operator, right_type, left_type)

        if res_type == 'Error':
```

```
                create_error(f'Invalid operation, type mismatch on {right_type} and
{left_type} with a {operator}', 'C-16')
        # generate new temporal of type rest_type
        current_scope_vars = program_scopes.get_vars_table(current_scope)
        temp_var_name = f"_temp{tempsCount}"
        tempsCount += 1
        # Create temp var on table of vars
        current_scope_vars.add_new_var(temp_var_name, res_type)
        # Ger address of this temporal var, and set it on the vars table of temp_var_name
        new_address = get_vars_new_address(res_type, True)
        current_scope_vars.set_address(temp_var_name, new_address)
        # Append a new quadruple to the quadruples list
        # set_new_quadruple(operator, left_operand, right_operand, new_address)
        set_new_quadruple(operator, left_operand, right_operand, new_address)
        # add to operands and types stacks the result
        operands.append(new_address)
        types.append(res_type)
```

Otro ejemplo de una función imporante del código es con la que se obtiene una variable (en compilación), esta se usa a lo largo de todo el código para poder revisar los atributos que tiene una variable de un cierto contexto.

```
'''
Get the vars directory given the scope id
First the directory is searched on the current scope
If it's not found, then it is searched on the global scope
'''
def get_var(var_id):
        global program_scopes, current_scope
        scope_vars = program_scopes.get_vars_table(current_scope)
        directory_var = scope_vars.get_one(var_id)
        # if 'not_in_directory' received, check the global scope
        if (directory_var == 'not_in_directory'):
            program_vars = program_scopes.get_vars_table('program')
            directory_var = program_vars.get_one(var_id)

        if (directory_var == 'not_in_directory'):
           create_error(f'{var_id} not found in current or global scope \n in get_var
function', 'C-17')

        return directory_var
```

Anexos

1. Liga de acceso al repositorio de `GitHub`: https://github.com/RCH010/manchas
2. Librería utilizada para el analizador sintáctico, PLY. Documentación: https://www.dabeaz.com/ply/
3. Liga de acceso a la documentación del proyecto: https://github.com/RCH010/manchas/tree/main/documentacion
4. El Manual de usuario se encuentra en el repositorio del proyecto junto con una liga a un vídeo demostrativo. https://github.com/RCH010/manchas