

**Diseño de Compiladores**



Ing. Elda Guadalupe Quiroga

Dr. Héctor Gibrán Ceballos

     \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                            \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

        Martha Lisett Benavides Martínez                                   José González Ayerdi

                       A01280115                                                                A01036121

Mayo 3, 2017

Contenido

[1. Descripción y Documentación del Proyecto 4](#_Toc481526899)

[1.1. Descripción del Proyecto 4](#_Toc481526900)

[1.1.1. Visión, Objetivo y Alcance del Proyecto 4](#_Toc481526901)

[1.1.2. Análisis de Requerimientos y Casos de Uso Generales 4](#_Toc481526902)

[1.1.3. Descripción de los Principales Test Cases 6](#_Toc481526903)

[1.1.4. Descripción del Proceso General Seguido para el Desarrollo del Proyecto 6](#_Toc481526904)

[1.2. Descripción del Lenguaje 13](#_Toc481526905)

[1.2.1. Nombre del Lenguaje 13](#_Toc481526906)

[1.2.2. Descripción Genérica de las Principales Características del Lenguaje 13](#_Toc481526907)

[1.2.3. Descripción de los Errores que Pueden Ocurrir (Compilación y Ejecución) 16](#_Toc481526908)

[1.3. Descripción del Compilador 16](#_Toc481526909)

[1.3.1. Equipo de Cómputo, Lenguaje y Utilerías Usadas en el Desarrollo del Proyecto 16](#_Toc481526910)

[1.3.2. Descripción del Análisis de Léxico 17](#_Toc481526911)

[1.3.3. Descripción del Análisis de Sintaxis 18](#_Toc481526912)

[1.3.4. Descripción de Generación de Código Intermedio y Análisis Semántico 20](#_Toc481526913)

[1.3.5. Descripción del Proceso de Administración de Memoria usado en Compilación 34](#_Toc481526914)

[1.4. Descripción de la Máquina Virtual 35](#_Toc481526915)

[1.4.1. Equipo de Cómputo, Lenguaje y Utilerías Usadas 35](#_Toc481526916)

[1.4.2. Proceso de Administración de Memoria en Ejecución (Arquitectura) 36](#_Toc481526917)

[1.5. Pruebas del Funcionamiento del Lenguaje 37](#_Toc481526918)

[1.5.1. Pruebas que Comprueban el Funcionamiento del Proyecto 37](#_Toc481526919)

[1.6. Listados Documentados del Proyecto 59](#_Toc481526920)

[2. Manual de Usuario 60](#_Toc481526921)

[2.1. Introducción al Lenguaje APODORAX 60](#_Toc481526922)

[2.2. Características del Lenguaje APODORAX 60](#_Toc481526923)

[2.2.1. Tipos de Datos 60](#_Toc481526924)

[2.2.2. Palabras Reservadas 60](#_Toc481526925)

[2.2.3. Operadores 61](#_Toc481526926)

[2.2.4. Variables 62](#_Toc481526927)

[2.2.5. Funciones 63](#_Toc481526928)

[2.2.6. Ciclos 63](#_Toc481526929)

[2.2.7. Condiciones 64](#_Toc481526930)

[2.2.8. Entrada de Información 64](#_Toc481526931)

[2.2.9. Asignación 64](#_Toc481526932)

[2.2.10. Desplegar Información 64](#_Toc481526933)

[2.2.11. Funciones Predefinidas (Gráficas) 64](#_Toc481526934)

[2.3. Ejemplos 66](#_Toc481526935)

[2.3.1. General 66](#_Toc481526936)

[2.3.2. Gráfico 67](#_Toc481526937)

# Descripción y Documentación del Proyecto

## Descripción del Proyecto

### Visión, Objetivo y Alcance del Proyecto

Hoy en día el uso de las Tecnologías de Información se encuentra en auge en distintas áreas de la vida diaria, tal como lo son los casos de grandes empresas de distintos giros, escuelas, aplicaciones web y móviles, así como videojuegos en 2D y 3D. Dado lo anterior, el propósito del presente proyecto es el de fomentar en jóvenes y adultos el aprendizaje de conceptos básicos de programación de manera sencilla y didáctica, en donde puedan tener la oportunidad de ver la perspectiva de desarrollo de los sistemas/programas que utilizan diariamente, de modo que obtengan una herramienta que les sirva como base para desarrollar sus propias ideas, que incluso, quizá en un futuro, este aprendizaje sea suficiente para buscar estudiar una carrera universitaria o técnica relacionada con las Tecnologías de Información, en donde la creatividad de cada persona es el límite.

El principal objetivo de APODORAX es el producir, como ya se mencionó con anterioridad, un lenguaje sencillo de utilizar, en donde las personas puedan aprender de forma simplificada y didáctica la lógica de la programación que se encuentra detrás de todas las aplicaciones que usan, dando paso a que con un poco de esfuerzo ellos mismos puedan generar instrucciones que les permitan observar de manera gráfica el resultado de las mismas (output gráfico).

De igual manera, dentro del alcance del proyecto se encuentra que el lenguaje tendrá como característica principal que estará basado en español, de modo que sea más sencillo para las personas que no tienen conocimiento del idioma inglés el declarar variables, ciclos, condiciones y funciones, ya que como es conocido, la mayoría de los lenguajes de programación usados a nivel mundial se encuentran escritos en inglés, logrando intimidar y limitar a aquellas personas con un bajo entendimiento del idioma.

### Análisis de Requerimientos y Casos de Uso Generales

Debido al hecho de que se mencionó que APODORAX estará desarrollado en español, la estructura del mismo, así como sus palabras clave, será representada en el mismo idioma.

#### Requerimientos Funcionales

Entre las principales funcionalidades que debe tener el compilador son las siguientes:

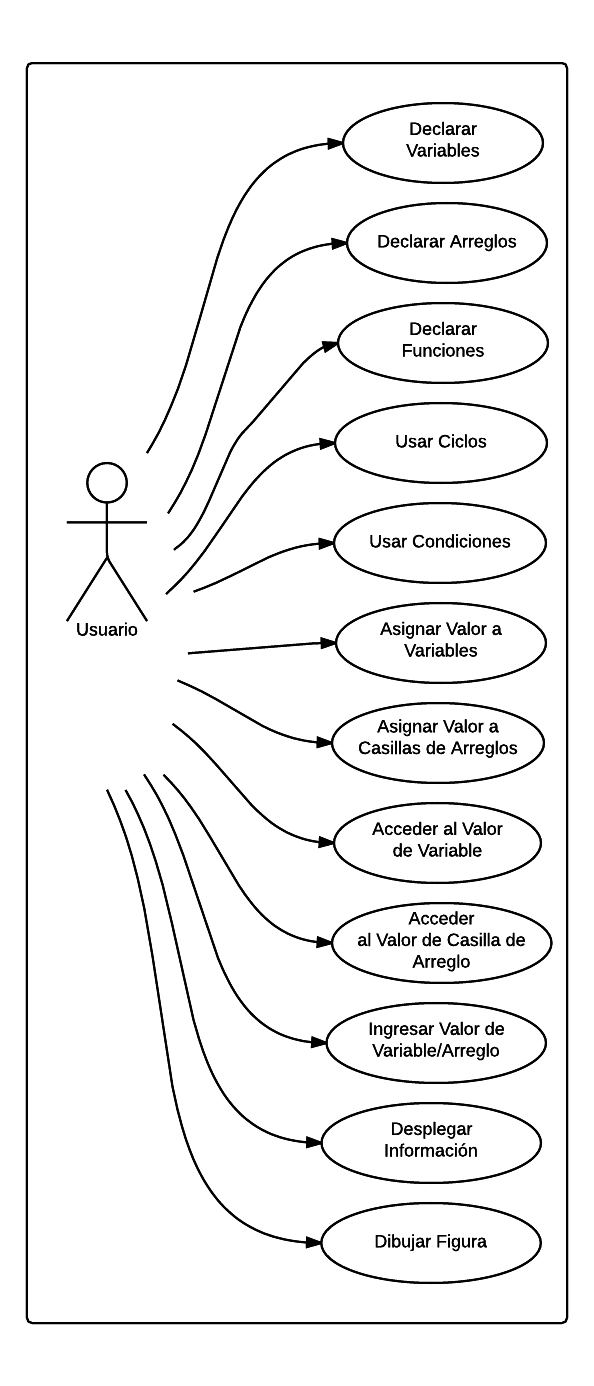
* RF.01. El sistema aceptará la declaración de variables globales.
* RF.02. El sistema aceptará la declaración de variables locales.
* RF.03. El sistema aceptará la declaración de arreglos globales.
* RF.04. El sistema aceptará la declaración de arreglos locales.
* RF.05. El sistema aceptará la declaración de funciones.
* RF.06. El sistema aceptará la declaración/uso de ciclos.
* RF.07. El sistema aceptará la declaración/uso de condiciones.
* RF.08. El sistema aceptará el uso de diversos tipos de datos (entero, flotante, cadena, carácter, bool).
* RF.09. El sistema permitirá asignarles valor a las variables.
* RF.10. El sistema permitirá asignarle valor a cada casilla de los arreglos.
* RF.11. El sistema permitirá tener acceso al valor de cada variable.
* RF.11. El sistema soportará recursividad.
* RF.12. El sistema soportará el acceso al valor de cada casilla de los arreglos.
* RF.13. El sistema mostrará en pantalla los datos solicitados por el usuario cuando él lo desee a través de la instrucción “desplegar”.
* RF.14. El sistema permitirá que el usuario ingrese en consola los datos que desee a través de la instrucción “entrada”.
* RF.15. El sistema permitirá mostrar de manera gráfica las figuras especificadas por el usuario.
* RF.16. El sistema deberá mostrar los errores que encuentre en compilación para notificarle al usuario.

#### Requerimientos No Funcionales

Los Requerimientos No Funcionales esperados del compilador son los siguientes.

* RNF.01. El sistema deberá compilar sin ningún error.
* RNF.02. La extensión oficial del compilador será .apdx (APODORAX).
* RNF.03. El sistema deberá funcionar en sistemas operativos Windows, Mac OS y Ubuntu.

#### Casos de Uso

Los casos de uso que se esperan por parte del usuario están ilustrados a continuación:

### Descripción de los Principales Test Cases

Los principales casos de prueba realizados fueron aquellos que permitían comprobar el funcionamiento correcto de expresiones, ciclos, funciones, recursividad y arreglos.

*Arreglos*

|  |  |  |
| --- | --- | --- |
| Find de Posición | Buscar la posición de un valor dentro de un arreglo. | Prueba Exitosa |
| Find de Valor | Buscar el valor de un elemento dada una posición. | Prueba Exitosa |
| BubbleSort | Ordenar un arreglo acomodando sus valores de menor a mayor. | Prueba Exitosa |
| Valor en Casilla | Hacer que el valor de una casilla sea el valor de otra casilla. Ejemplo: x[y[5]] | Prueba Exitosa |
| Producto Punto | Hacer el producto punto entre dos vectores. | Prueba Exitosa |

*Ciclos*

|  |  |  |
| --- | --- | --- |
| Factorial Iterativo | Obtener el factorial de un número haciendo uso de un ciclo while (mientras). | Prueba Exitosa |
| Fibonacci Iterativo | Obtener el resultado final de la serie Fibonacci dado un número haciendo uso de un ciclo while (mientras). | Prueba Exitosa |

*Funciones*

|  |  |  |
| --- | --- | --- |
| Factorial Recursivo | Obtener el factorial de un número haciendo uso de recursividad | Prueba Exitosa |
| Fibonacci Recursivo | Obtener el resultado final de la serie Fibonacci dado un número haciendo uso de recursividad. | Prueba Exitosa |

### Descripción del Proceso General Seguido para el Desarrollo del Proyecto

De manera general, los integrantes del equipo estuvieron trabajando de manera conjunta para desarrollar todo el proyecto, por lo que se hizo uso de herramientas como Google Drive, Google Hangouts, Facebook y GitHub (<https://github.com/MarthaLisett/APODORAX>) para estar en constante contacto y tener un control de versiones.

#### Bitácoras de Avance

Para desarrollar el proyecto, el equipo de APODORAX se estuvo reuniendo semanalmente, incluyendo Semana Santa, para trabajar en los avances, por lo que en cada bitácora que se muestra a continuación se encuentra el resultado de los avances que se entregaron, incluyendo el progreso y las dificultades obtenidas.

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 6 Marzo 2017 | |
| **Avance** | 1. Léxico y Sintaxis | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Léxico | Se incluyeron los tokens y palabras reservadas. | Funciona al 100% |
| Sintaxis | Se hizo la gramática correspondiente ya haciendo los cambios marcados a realizar en la propuesta. | Funciona al 100% |
| Pruebas | Se hicieron pruebas con todas las gramáticas, incluyendo declaraciones, asignaciones, ciclos, condiciones, comparaciones y uso de distintos tipos de datos. | Se hicieron pruebas tratando de afectar a la gramática, dando paso a que efectivamente lo detectara como error. |
| Ambigüedades | Solamente muestra 1 Warning de conflicto Shift-Reduce | No parece afectar al lenguaje. |
| **Cambios** | Se hicieron cambios a distintos diagramas de la gramática ya que habían elementos repetitivos que ocasionaban muchas ambigüedades (conflictos Reduce Reduce y Shift Reduce). | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 10 Marzo 2017 | |
| **Avance** | 1. Directorio de Funciones y Tablas de Variables | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Directorio de Funciones | Se creó un diccionario de funciones en donde se puedan agregar nuevas funciones si no existen previamente y se marque un error si existen funciones repetidas. Cada diccionario de funciones tiene su propio diccionario de variables locales. | Funciona al 100% |
| Tabla de Variables | Se creó un diccionario de variables con distinto scope, ya sea local y global, de modo que se asegure un acceso correcto a las mismas. | Funciona al 100% |
| Pruebas | Se hicieron pruebas de todos los distintos casos posibles: variables que no existen, funciones que no existen, variables globales/locales repetidas, funciones repetidas, acceso incorrecto a variables locales/globales. | Se pasaron todas las pruebas. |
| Ambigüedades | Ya no se muestra el Warning del conflicto Shift-Reduce, pero ahora sale 1 conflicto Reduce Reduce. | No parece afectar al lenguaje, ya que las pruebas funcionan al 100%. |
| **Cambios** | No se hicieron cambios a las gramáticas en general, sólo se agregaron nuevas reglas entre las gramáticas (intermedias) para poner hacer el directorio de funciones y la tabla de variables. | |
| **Dificultades** | Tuvimos dificultades para mostrar los errores, pudimos desplegarlos usando el “raise keyError”, por lo que con esto si despliega los errores pertinentes, pero además muestra un mensaje de error propio del raise keyError (no afecta al programa).  Al principio tuvimos el problema de que no registraba el scope global, pero se arregló después al insertar las reglas intermedias. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 19 Marzo 2017 | |
| **Avance** | 3. Semántica Básica de Expresiones: Tabla de Consideraciones Semánticas (Cubo Semántico), Generación de Código de Expresiones Aritméticas y estatutos secuenciales: Asignación, Lectura. | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Cubo Semántico | Se hizo una tupla de tuplas con operaciones lógicas/ aritméticas y los diferentes tipos de datos (enteros, flotantes, cadenas, caracteres, booleanos) para saber cuáles eran válidos y cuáles deberían mostrar error. | Se terminó el cubo semántico al 100%. |
| Código Asignación | Se codificaron las instrucciones para generar los cuádruplos de los estatutos. | Faltaron hacer las pruebas de su funcionamiento. Se declaró el orden de precedencias. |
| Pruebas | Se hicieron pruebas de operaciones entre diferentes tipos de operandos, los cuales mostraron resultados correctos. | Por el momento solamente se probó el cubo semántico con constantes, las cuales pasaron las pruebas de aceptación. |
| **Pendientes** | Código de Escritura y Lectura.  Aún no se manejan Direcciones Virtuales. | |
| **Adicionales** | Se agregaron las clases stack y queue para usarlas en el proyecto. | |
| **Dificultades** | Se tuvo la dificultad en saber en qué momento agregar un id a la pila de operandos y el asignar valores a los elementos del cubo. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 24 Marzo 2017 | |
| **Avance** | 4. Generación de Código de Estatutos Condicionales: Decisiones/Ciclos | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Cuádruplos Decisiones | Se recreó el código visto en clase para generar los cuádruplos de decisiones, por lo que ya se generan los mismos para if (si) y los else (sino). | Funciona al 100%, se hace uso de los GotoF y Goto. |
| Cuádruplos Ciclos | Se recreó el código visto en clase para generar los cuádruplos de ciclos, por lo que ya se generan los cuádruplos para los while (mientras). | Funciona al 100%, se hace uso de los GotoF y Goto. |
| Cuádruplos Entrada y Salida | Se aplicó el código de generación de cuádruplos para obtener los input (ingreso) y output (escritura). | Funciona al 100%, genera los cuádruplos de manera correcta. |
| Pruebas | Se hicieron pruebas para saber que se estuvieran generando los resultados correctos de expresiones (incluyendo paréntesis), que se hiciera una validación semántica correcta entre tipos, que el scope de variables fuera el correcto, que se estuvieran generando de manera correcta los cuádruplos. | Se pasaron todas las pruebas. Se pueden generar de manera correcta todos los cuádruplos descritos con anterioridad. |
| Ambigüedades | Sigue apareciendo el conflicto Reduce Reduce. | No parece afectar al lenguaje, ya que las pruebas funcionan al 100%. |
| **Cambios** | Se añadió una regla de Lógica (&&, ||) en la gramática (se removieron de Comparación para moverlos a otro nivel) y se decidió que para hacer los input y output solamente sería de un valor a la vez (no múltiples separados por comas). | |
| **Dificultades** | Se tomó mucho tiempo en desarrollar las pruebas para saber que se estuvieran generando de manera correcta las operaciones, validaciones de semántica y cuádruplos. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 2 Abril 2017 | |
| **Avance** | 5. Generación de Código de Funciones | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Cuádruplos Funciones | Se recreó el código visto en clase para generar los cuádruplos de funciones (void). | Funciona al 100% el ejercicio que está en la hoja que se nos dio en clase, las cuales son funciones void, sin embargo, aún no se implementan las funciones con retorno. |
| Pruebas | Se hicieron pruebas para saber que se estuvieran generando los cuádruplos correctos de funciones void. | Se pasaron todas las pruebas. Se pueden generar de manera correcta todos los cuádruplos descritos con anterioridad (sólo funciones void). Aún falta implementar las funciones con retorno. |
| Ambigüedades | Sigue apareciendo el conflicto Reduce Reduce. | No parece afectar al lenguaje, ya que las pruebas funcionan al 100%. |
| **Cambios** | No se hicieron cambios. | |
| **Dificultades** | Se estaban haciendo mal los cuádruplos de las funciones, pero se logró detectar que el problema estaba en una constante y se hizo la modificación. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 10 Abril 2017 | |
| **Avance** | 6. Mapa de Memoria de Ejecución para la Máquina Virtual. Máquina Virtual: Ejecución de Expresiones Aritméticas y Estatutos Secuenciales. | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Cuádruplos Funciones | Se recreó el código visto en clase para generar los cuádruplos de funciones tanto void, así como funciones con retorno. También se hicieron los cuádruplos de las funciones predefinidas. | Funciona al 100% el ejercicio que está en la hoja que se nos dio en clase. |
| Memoria | Se hicieron las clases memoria, temporal, local, global y constante, las cuales tienen direcciones para enteros, flotantes, booleanos, cadenas y caracteres. | Existen algunos problemas con la asignación correcta de los valores temporales, al parecer se están sobre-escribiendo. Se trabajará para resolverlo. |
| Pruebas | Se hicieron pruebas para saber que se estuvieran generando los cuádruplos correctos de funciones.  También se hicieron pruebas con la librería. | Se pasaron todas las pruebas. Se pueden generar de manera correcta todos los cuádruplos descritos con anterioridad.  Se generó un algoritmo para crear curvas usando puntos. |
| Ambigüedades | Se removió el conflicto Reduce Reduce. | Se hicieron pequeñas modificaciones en la gramática y se arregló el conflicto Reduce Reduce que provocaba tener la instrucción id[] o id(). |
| **Cambios** | Se hicieron cambios en la gramática para remover el conflicto Reduce Reduce, además a las funciones predefinidas se les cambió el “Args” por “llamadapar”, de forma que aceptara los argumentos específicos que deben recibir cada una de ellas.  Se agregaron los colores rosa y naranja. | |
| **Dificultades** | Había un error en el conteo de los cuádruplos, por lo que no encontrábamos el error, finalmente se resolvió notando que faltaba incrementar en 1 el contador en las condiciones. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 23 Abril 2017 | |
| **Avance** | 7. Generación de Código de Arreglos/Tipos Estructurados. Máquina Virtual: Ejecución de Estatutos Condicionales | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Cuádruplos Arreglos | Se recreó el código visto en clase para generar los cuádruplos de arreglos, por lo que ahora se pueden declarar y acceder a ellos. | Se probó con un BubbleSort y si funcionó. |
| Memoria | Se hicieron las clases memoria, temporal, local, global y constante, las cuales tienen direcciones para enteros, flotantes, booleanos, cadenas y caracteres. | Todas las variables se están asignando en sus respectivas secciones de memoria de manera correcta. |
| Máquina Virtual | Ya se tiene la máquina virtual para expresiones aritméticas, estatutos secuenciales, condiciones, ciclos, funciones, y funciones predefinidas (librería de output). | Recursividad ya funciona, se probó con un Fibonnaci.  También ya se puede dibujar el output gráfico. |
| Interfaz | Se creó una interfaz en donde se pueda escribir código, modificarlo y compilarlo. Además, tiene una sección en donde la persona puede ejecutar funciones de la librería (dibujar figuras) de modo que pueda observar cómo se dibujan las figuras haciendo cambios en colores, tamaños, etc. | La interfaz funciona de manera correcta, también hay una sección de debug en donde después de compilar se podrá ver si hay errores en el programa. |
| **Cambios** | A las funciones predefinidas (librería) se les hizo el cambio de que ahora en lugar de sólo mostrar que pediría Args, ahora se establecen el orden y el tipo de dato que tendrán los parámetros para llamarlas. | |
| **Dificultades** | Se tuvo mucha dificultad en poder generar los arreglos y llamadas recursivas, pero después de mucho tiempo y correcciones ya funcionaron. | |

|  |  |  |
| --- | --- | --- |
| **Participantes** | Martha Benavides | José González |
| **Fecha** | 23 Abril 2017 | |
| **Avance** | 8. Primera Versión de la Documentación. Generación de Código y Máquina Virtual para una Parte de la Aplicación en Particular. | |
|  | | |
| ***Componente*** | ***Descripción*** | ***Estatus*** |
| Documentación | Se comenzó a desarrollar la documentación del proyecto. | Se tienen hechas las secciones: a (completa), b (incompleta), c (incompleta), d (incompleta). |
| Máquina Virtual | La máquina virtual ya está completa. | Solamente falta la implementación del not. |
| Output Gráfico | Ya se tiene el funcionamiento completo para dibujar todas las figuras establecidas en la propuesta. | Funciona al 90%, se cierra la ventana cuando hay recursividad. |
| **Cambios** | Ahora los arreglos pueden ser anidados y se pueden asignar valores a los subíndices de arreglos a través del usuario (entrada).  Los colores ahora son puestos en la máquina virtual, ya no se hace uso de los implementados en la gramática. | |
| **Dificultades** | Se tuvo la dificultad en establecer la relación entre un return y un endproc.  También se tuvo la dificultar para usar arreglos de distintos tipos. | |

#### Reflexiones

##### Martha Benavides

El desarrollar este proyecto me dio la oportunidad de aplicar todo el conocimiento que he aprendido a través de toda la carrera, incluyendo estructura de datos, algoritmos, objetos, ingeniería de software (documentación), entre muchos otros, brindándome la posibilidad de desarrollar un lenguaje/compilador propio, siendo un sueño que no pensé que podría cumplir al ingresar a la carrera.

Fue muy interesante el poder generar un compilador desde cero, en donde se tenía que crear desde el léxico y la gramática, hasta generar los cuádruplos y la máquina virtual para que funcionara, teniendo que pasar, junto con mi compañero de equipo, por dificultades que nos costaron desveladas, estancias tardías en la escuela y muchas horas de trabajo, pero que finalmente nos deja una gran satisfacción el tener un producto 100% funcional que puede ser usado por jóvenes y adultos para iniciar su camino en la programación, y que mejor que hacerlo mediante el output gráfico. Me quedo con una grata experiencia con la realización de este proyecto, ya que incluso pudimos volvernos mejores testers.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Martha Lisett Benavides Martínez

##### José González

Sin duda, este ha sido uno de los proyectos más retadores que me han asignado durante mi carrera universitaria, ya que además de ser sumamente demandante, implicaba un amplio conocimiento de materias de otros semestres, sobre todo de Estructuras de Datos y Programación Orientada a Objetos.

Una de las cosas que encuentro interesantes en la elaboración de un Compilador es que a diferencia de otras disciplinas (Matemáticas abstractas, Física, Filosofía, etc) la construcción de un Compilador es especialmente difícil puesto que es necesario entender todas las estructuras implicadas en el programa con la máxima cantidad de claridad posible, no se puede ignorar algún aspecto de éste o dar por hecha su correcta funcionalidad.

Aunque en un principio este proyecto comenzó siendo parte importante de la ponderación necesaria para acreditar el curso, terminó volviéndose el proyecto más valioso del semestre en cuanto al valor aportado por el mismo, ya que requirió mucho compromiso por mi parte y de mi compañera de equipo para poderlo sacar adelante, además de que se volvió muy significativo por el hecho de que puede ser de utilidad para aquellas personas principiantes en el área de programación que quieran aprender conceptos básicos de generación de código, y más aún, que incluso después ya de adquirir un conocimiento más profundo, puedan tener la oportunidad de crear su propio Compilador al igual que nosotros.

En general, aprendí mucho con la realización de este proyecto, desde trabajar mejor en equipo, hasta profundizar mis conocimientos en el Lenguaje Python.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

José González Ayerdi

## Descripción del Lenguaje

### Nombre del Lenguaje

El nombre asignado para el compilador fue APODORAX, ya que Apodora es una especie de serpiente relacionada con el pitón (Python), por lo que debido a que usamos Python para desarrollar nuestro lenguaje, el equipo decidió darle un nombre similar.

### Descripción Genérica de las Principales Características del Lenguaje

APODORAX le da la oportunidad al usuario de declarar variables, arreglos y funciones; hacer uso de ciclos y condiciones; dibujar figuras con la ayuda de una librería gráfica (output gráfico), por lo que se le da la oportunidad de usar un lenguaje que tenga las herramientas básicas y esenciales para entender los conocimientos importantes sobre programación.

#### Elementos del Lenguaje

* *Tipos de Datos*: APODORAX soporta enteros, flotantes, cadenas, caracteres y booleanos.
* *Operaciones Aritméticas*: Se pueden usar sumas, restas, multiplicaciones y divisiones.
* *Operaciones Lógicas*: Se pueden usar && y ||.
* *Operaciones Relacionales*: Se pueden usar <, >, >=, <=, = =, !=.
* *Variables*: Se pueden declarar variables globales al inicio del programa y variables locales dentro de las funciones. Para la declaración de variables se necesita seguir la siguiente sintaxis: var tipo\_de\_dato nombre\_variable;

*var entero numero;*

* *Funciones*: Se pueden declarar funciones después de declarar todas las variables globales y antes de la función principal inicio. La sintaxis para declarar funciones es: funcion tipo\_de\_dato nombre\_funcion(declaración de parámetros){ }. Las funciones pueden tener o no retorno.

*funcion entero factorial (var entero x) {*

*resultado = 10;*

*regresar resultado;*

*}*

* *Arreglos*: Se pueden declarar arreglos siguiendo la siguiente sintaxis: var tipo\_de\_dato nombre\_arreglo[cantidad\_casillas];

*var entero clases[10];*

* *Condiciones*: Se pueden declarar condiciones para tener instrucciones más complejas: Condición simple: si (condición) entonces{ instrucciones }

Condición alternativa: si (condición) entonces{ inst } sino entonces { inst }

*si (x == 10) entonces {*

*y = 5;*

*} sino entonces {*

*y = 3;*

*}*

* *Ciclos*: Se pueden declarar ciclos para cuando se quiere hacer un procedimiento de manera repetitiva siguiendo la siguiente sintaxis: mientras (condición){ instrucciones }

*mientras (x < 10) {*

*x = x + 1;*

*}*

* *Escritura*: El usuario puede mostrar información en consola usando la instrucción “desplegar”:

*desplegar(“Hola Mundo”);*

* *Ingreso*: El usuario puede ingresar información en consola usando la instrucción “entrada”, la cual permite asignarle un valor a una variable:

*entrada(num);*

* Asignación: Se pueden asignar valores a las variables o subíndices de arreglos siguiendo la sintaxis: variable = valor;

*saludo = “hola”;*

*clases[3] = 5;*

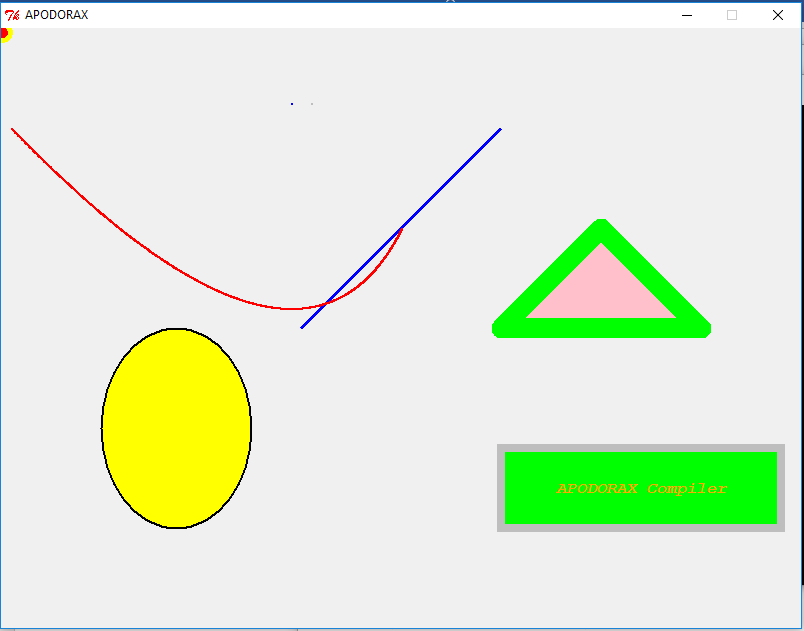
#### Elementos Gráficos

Asimismo, la librería a usar permite dibujar ciertas figuras, por lo que los datos necesarios para dibujar cada una de ellas son los siguientes:

* *Punto*: Permite dibujar un punto teniendo como datos la coordenadaX, coordenadaY, y el color del punto.
* *Texto*: Permite dibujar una línea teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2, el texto a desplegar y el tamaño del texto.
* *Rectángulo*: Permite dibujar un rectángulo teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2, el color del relleno, el color del contorno y el tamaño del grosor del contorno.
* *Triángulo*: Permite dibujar un triángulo teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2, coordenadaX3, coordenadaY3, el color del relleno, el color del contorno y el tamaño del grosor del contorno.
* *Círculo*: Permite dibujar un círculo teniendo como datos la coordenadaX, coordenadaY, el tamaño del radio, el color del relleno y el grosor del contorno.
* *Óvalo*: Permite dibujar un óvalo teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2, el color del relleno, el color del contorno y el tamaño del grosor del contorno.
* *Línea*: Permite dibujar una línea teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2, el color del relleno y el tamaño del grosor del contorno.
* *Curva*: Permite dibujar una curva teniendo como datos la coordenadaX1, coordenadaY1, coordenadaX2, coordenadaY2 y el color de la curva.

Cabe destacar que los colores disponibles son: rojo, amarillo, azul, rosa, naranja, verde, negro y gris.

A continuación se muestra una imagen en donde se aplicaron cambios en los colores, grosores y figuras, de forma que se puedan apreciar los posibles resultados.



***Curva***

***Punto***

***Línea***

***Óvalo***

***Triángulo***

***Rectángulo***

***Texto***

### Descripción de los Errores que Pueden Ocurrir (Compilación y Ejecución)

Algunos de los errores que podrían ocurrir tanto en compilación y ejecución por parte del usuario, así como los mensajes que le aparecerán al mismo son:

|  |  |
| --- | --- |
| Error | Respuesta (Mensaje del Compilador) |
| No se codifica siguiendo la gramática al pie de la letra. | Error de sintaxis. |
| Se trata de acceder al subíndice de una variable no dimensionada. | La variable \_ no es dimensionada. |
| Se trata de acceder a un subíndice de un arreglo mediante un valor no entero. | El índice el arreglo debe ser entero. |
| Se quiere obtener un valor (retorno) de una función vacía. | No se puede obtener un valor de la función \_. |
| Se crea una función con un tipo y un tipo de retorno diferente. | Vacía: La función \_ no puede regresar ningún valor.  Otro: La función \_ debe regresar \_. |
| Se manda llamar una función con la cantidad incorrecta de argumentos. | La función recibe una cantidad distinta de argumentos. |
| Se llama a una función con argumentos de distintos tipos que los parámetros. | Los tipos en la llamada y en la función no coinciden. |
| Se quiere asignar a una variable un valor de diferente tipo. | Tipos incompatibles. |
| Se quiere hacer una operación entre valores no compatibles en el cubo semántico. | Tipos incompatibles. |
| Se quiere usar una función que no está declarada. | La función \_ no está declarada. |
| Se quiere usar una variable que no está declarada. | La variable \_ no ha sido declarada. |
| Se quiere usar una variable local en un scope global. | La variable \_ no ha sido declarada. |
| Se declaran dos o más variables repetidas dentro del mismo scope. | Variable \_ repetida. |
| Se declaran dos o más funciones con el mismo nombre. | Función \_ repetida. |
| Se intenta acceder a un subíndice de un arreglo fuera de los límites. | Se intentó acceder a un espacio fuera del arreglo. |

## Descripción del Compilador

### Equipo de Cómputo, Lenguaje y Utilerías Usadas en el Desarrollo del Proyecto

#### Equipo de Cómputo

Los Sistemas Operativos de las computadoras usados para la implementación y uso del lenguaje son Windows 10, Ubuntu Linux, Mac OS Sierra.

#### Lenguaje

El lenguaje en el que fue desarrollado el compilador es Python 2.7, usando la herramienta PLY como analizador de léxico y de sintaxis.

#### Utilerías

De igual manera, se hizo uso de la librería graphics.py como base para el desarrollo del output gráfico, con la cual se pueden mostrar puntos, líneas, círculos, óvalos, rectángulos, polígonos, texto e imágenes de manera gráfica.

### Descripción del Análisis de Léxico

#### Patrones de Construcción (Expresiones Regulares)

APODORAX mantiene un registro de tokens y palabras reservadas, las cuales se encuentran establecidas en el escáner del proyecto y se enlistan a continuación.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Token | Expresión Regular | Token | Expresión Regular | Token | Expresión Regular |
| PUNTOYCOMA | r'\;' | CORCHETEIZQ | r'\[' | MAYORQUE | r'\>' |
| DOSPUNTOS | r'\:' | CORCHETEDER | r'\]' | MAYORIGUAL | r'\<\=' |
| COMA | r'\,' | SUMA | r'\+' | MENORIGUAL | r'\>\=' |
| PARENIZQUIERDO | r'\(' | RESTA | r'\-' | ASIGNACION | r'\=' |
| PARENDERECHO | r'\)' | MULTIPLICACION | r'\\*' | IGUAL | r'\=\=' |
| LLAVEIZQUIERDO | r'\{' | DIVISION | r'\/' | DIFERENTE | r'\!\=' |
| LLAVEDERECHO | r'\}' | MENORQUE | r'\<' |  | |
| CONJUNCION | r'\&\&' | DISYUNCION | r'\|\|' |
| CCADENA | r'\"(\\.|[^"])\*\"' | | CCARACTER | r'\'[a-z]\'|\'[A-Z]\'' | |
| ID | r'[a-zA-Z](\_?[a-zA-Z0-9]+)\*' | | CFLOTANTE | r'([\+|-]?[0-9]+[.])[0-9]+' | |
| CENTERO | r'[-]?[0-9]+' | | COMMENT | r'\#.\*' | |

|  |  |  |  |
| --- | --- | --- | --- |
| Tokens | | | |
| PROGRAMA | ID | INICIO | FIN |
| FUNCION | VAR | CONJUNCION | DISYUNCION |
| CENTERO | ENTERO | CFLOTANTE | FLOTANTE |
| CCADENA | CADENA | CCARACTER | CARACTER |
| BOOL | SI | NO | SINO |
| ENTONCES | MIENTRAS | ENTRADA | DESPLEGAR |
| VERDADERO | FALSO | VACIO | REGRESAR |
| MENORQUE | MAYORQUE | MAYORIGUAL | MENORIGUAL |
| DIFERENTE | IGUAL | SUMA | RESTA |
| MULTIPLICACION | DIVISION | PARENIZQUIERDO | PARENDERECHO |
| CORCHETEIZQ | CORCHETEDER | LLAVEIZQUIERDO | LLAVEDERECHO |
| PUNTOYCOMA | COMA | DOSPUNTOS | ASIGNACION |

|  |  |  |  |
| --- | --- | --- | --- |
| Palabras Reservadas | | | |
| programa | entero | flotante | cadena |
| caracter | bool | verdadero | falso |
| inicio | fin | var | funcion |
| entrada | desplegar | mientras | regresar |
| si | sino | no | entonces |
| vacio | insertaTexto | insertaRectangulo | insertaTriangulo |
| insertaCirculo | insertaOvalo | insertaPunto | insertaCurva |
| insertaLinea |  |  |  |

#### Tokens del Lenguaje y su Código Asociado

De igual manera, para la formación de los cuádruplos se hizo uso de los tokens en lugar del código de operación, por lo que a continuación se muestra cada token con su respectivo código aunque estos últimos no se hayan usado debido a que se prefirió trabajar con los cuádruplos de la misma manera en la que se vio en clase.

|  |  |  |  |
| --- | --- | --- | --- |
| Tokens | | | |
| PROGRAMA | 1 | ID | 2 |
| INICIO | 3 | FIN | 4 |
| FUNCION | 5 | VAR | 6 |
| CONJUNCIÓN | 7 | DISYUNCIÓN | 8 |
| CENTERO | 9 | ENTERO | 10 |
| CFLOTANTE | 11 | FLOTANTE | 12 |
| CCADENA | 13 | CADENA | 14 |
| CCARACTER | 15 | CARACTER | 16 |
| BOOL | 17 | SI | 18 |
| NO | 19 | SINO | 20 |
| ENTONCES | 21 | MIENTRAS | 22 |
| ENTRADA | 23 | DESPLEGAR | 24 |
| VERDADERO | 25 | FALSO | 26 |
| VACIO | 27 | REGRESAR | 28 |
| MENORQUE | 29 | MAYORQUE | 30 |
| MENORIGUAL | 31 | MAYORIGUAL | 32 |
| DIFERENTE | 33 | IGUAL | 34 |
| SUMA | 35 | RESTA | 36 |
| MULTIPLICACION | 37 | DIVISION | 38 |
| PARENIZQUIERDO | 39 | PARENDERECHO | 40 |
| CORCHETEIZQ | 41 | CORCHETEDER | 42 |
| LLAVEIZQUIERDO | 43 | LLAVEDERECHO | 44 |
| PUNTOYCOMA | 45 | COMA | 46 |
| DOSPUNTOS | 47 | ASIGNACIÓN | 48 |
| GOTOF | 49 | GOTO | 50 |
| ERA | 51 | PARAMETER | 52 |
| GOSUB | 53 | ENDPROC | 54 |
| RETURN | 55 | VER | 56 |

### Descripción del Análisis de Sintaxis

#### Gramática Formal Empleada para Representar las Estructuras Sintácticas

La gramática formal empleada por APODORAX para representar las estructuras sintácticas se muestran a continuación, incluyendo todos los aspectos importantes para el desarrollo idóneo y correcto del lenguaje.

PROGRAMA ::= 'programa' 'id' ':' DECLARACION FUNCION 'inicio' BLOQUE 'fin'

DECLARACION ::= ('var' TIPO CTEID\_DECLARACION ';')\*

CTEID\_DECLARACION ::= 'id'

FUNCTION ::= 'funcion' TIPO\_REGRESO 'id' '(' FUNCTIONPAM ')' BLOQUEFUN

FUNCTIONPAM ::= ('var' TIPO 'id' (',' FUNCTIONPAM)\* | )

TIPO\_REGRESO ::= TIPO | 'vacio'

TIPO ::= 'cadena' | 'entero' | 'flotante' | 'bool' | 'caracter'

BLOQUEFUN ::= '{' DECLARACION\* ESTATUTO\* REGRESO '}'

ESTATUTO ::= INGRESO | ESCRITURA | TEXTO | RECTANGULO | CIRCULO | OVALO | TRIANGULO | PUNTO | LINEA | CURVA | 'id' LLAMADA ';' | ASIGNACION | CONDICION | CICLO

REGRESO ::= ('regresar' EXP ';' | 'regresar' 'vacio' ';' | )

BLOQUE ::= '{' ESTATUTO\* '}'

LLAMADAPAR ::= (EXP LLAMADAPARAUX)\*

LLAMADAPARAUX ::= (',' LLAMADAPAR)\*

INGRESO ::= 'entrada' '(' CTE ')' ';'

ESCRITURA ::= 'desplegar' '(' EXPRESION ')' ';'

CTE ::= 'id' SYMBOL | 'c\_entero' | 'c\_flotante' | 'c\_cadena' | 'c\_caracter' | 'verdadero' | 'falso'

SYMBOL ::= CTE\_ID | LLAMADA

CTE\_ID ::= (('[' EXP ']') |)

EXPRESION ::= EXP ((COMPARACION EXP)|)

COMPARACION ::= '<' | '>' | '<=' | '>=' | '!=' | '=='

EXP ::= TERMINO | TERMINO '+' EXP | TERMINO '-' EXP

TERMINO ::= FACTOR | FACTOR '\*' TERMINO | FACTOR '/' TERMINO

FACTOR ::= '(' LOGICO ')' | CTE

CONDICION ::= 'si' '(' LOGICO ')' 'entonces' BLOQUE ( 'sino' 'entonces' BLOQUE | )

FACTOR ::= '(' LOGICO ')' | CTE

LOGICO ::= EXPRESION ('&&' LOGICO | '||' LOGICO |)

CICLO ::= 'mientras' '(' LOGICO ')' BLOQUE

LLAMADA ::= '(' LLAMADAPAR ')'

FUNCTIONPAM ::= ('var' TIPO 'id' (',' FUNCTIONPAM)\* |)

ASIGNACION ::= ('id' ('[' EXP ']' |)) '=' EXP ';'

TEXTO ::= 'insertaTexto' '(' LLAMADAPAR ')' ';'

RECTANGULO ::= 'insertaRectangulo' '(' LLAMADAPAR ')' ';'

CIRCULO ::= 'insertaCirculo' '(' LLAMADAPAR ')' ';'

OVALO ::= 'insertaOvalo' '(' LLAMADAPAR ')' ';'

TRIANGULO ::= 'insertaTriangulo' '(' LLAMADAPAR ')' ';'

PUNTO ::= 'insertaPunto' '(' LLAMADAPAR ')' ';'

LINEA ::= 'insertaLinea' '(' LLAMADAPAR ')' ';'

CURVA ::= 'insertaCurva' '(' LLAMADAPAR ')' ';'

### Descripción de Generación de Código Intermedio y Análisis Semántico

#### Código de Operación y Direcciones Virtuales Asociadas a los Elementos del Código

De igual forma, enseguida se enlistan los códigos de operación asociados a los elementos del código más importantes, es decir, aquellos que se usan en los cuádruplos.

|  |  |  |
| --- | --- | --- |
| Elemento | Código de Operación | Descripción |
| Asignación | = | Asigna el valor de la derecha a la variable de la izquierda. |
| Suma | + | Suma dos números. |
| Resta | - | Resta dos números. |
| Multiplicación | \* | Multiplica dos números. |
| División | / | Divide dos números. |
| Escritura | escritura | Despliega la variable o valor pasado como parámetro. |
| Ingreso | entrada | Permite que el usuario ingrese un valor para una variable desde la consola. |
| And | && | Checa que ambas condiciones sean verdaderas. |
| Or | | | | Checa que por lo menos una de las condiciones sea verdadera. |
| Diferente | ! = | Checa que las condiciones sean diferentes. |
| Igual Igual | = = | Checa que las condiciones sean iguales. |
| Mayor | > | Compara si un número es mayor a otro. |
| Menor | < | Compara si un número es menor a otro. |
| Mayor Igual | > = | Compara si un número es mayor o igual a otro. |
| Menor Igual | < = | Compara si un número es menor o igual a otro. |
| GOTOF | GotoF | Checa cuando la condición sea falsa para saltar de cuádruplo. |
| GOTO | Goto | Checa a que cuádruplo saltar. |
| ERA | ERA | Permite saber que se ha llamado a una función. |
| PARAMETER | PARAMETER | Permite saber que las variables pertinentes son los parámetros de la función llamada. |
| GOSUB | GOSUB | Permite acceder a la información de la función llamada. |
| ENDPROC | ENDPROC | Notifica que el bloque de una función ha terminado. |
| RETURN | RETURN | Checa el valor de retorno de una función. |
| VER | VER | Verifica los límites de un arreglo. |

Por otro lado, las direcciones de memoria virtuales asignadas para cada tipo de dato se muestran a continuación:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Enteros | Flotantes | Booleanos | Caracteres | Cadenas | *Total* |
| Global | 0 - 999 | 1,000 -1,999 | 2,000 -2,999 | 3,000 - 3,999 | 4,000 - 4,999 | 0 – 4,999 |
| Local | 5,000 - 5,999 | 6,000 - 6,999 | 7,000 - 7,999 | 8,000 - 8,999 | 9,000 - 9,999 | 5,000 - 9,999 |
| Temporal | 10,000 - 10,999 | 11,000 - 11,999 | 12,000 - 12,999 | 13,000 - 13,999 | 14,000 - 14,999 | 10,000 - 14,999 |
| Constantes | 15,000 - 15,999 | 16,000 - 16,999 | 17,000 - 17,999 | 18,000 - 18,999 | 19,000 - 19,999 | 15,000 - 19,999 |

#### Diagramas de Sintaxis con las Acciones Semánticas y de Código

**1**

**<PROGRAMA>**

**6**

**4 5**

**2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCacheContent.Word\PROGRAMA.PNG

**1)** Agrega a la tabla de símbolos las funciones predefinidas (figuras) y sus variables.

**2)** Inicializa la cantidad de variables de una función en cero.

**3)** Genera cuádruplo de Goto al main.

**4)** Llena el cuádruplo del Goto al main con el número del cuádruplo donde se encuentra.

**5)** Agrega la función inicio a la tabla de símbolos junto con su scope.

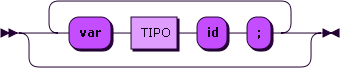
**6)** Generar los cuádruplos del programa y mandar a ejecutar la máquina virtual.

**<DECLARACION>**

**5**

**1**

**2 3 4**

Imagen relacionadaImagen relacionadaImagen relacionadaImagen relacionada

**1)** Obtener el tipo de la variable.

**2)** Obtener ID.

**3)** Guardar el ID y su tipo.

**4)** Checar si es variable normal o dimensionada.

**5)** Agregar 1 a la cantidad de variables.

**<FUNCION>**

**5**

**4**

**2 3**

**1**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\FUNCTION.PNG

**1)** Resetear al scope a global.

**2)** Checar si la función aún no está en la tabla de símbolos, y si no está agrega su ID y el tipo de regreso.

**3)** Agregar función con scope global.

**4)** Agregar cantidad de parámetros de la función.

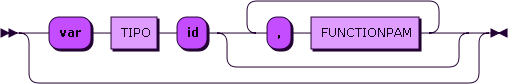
**5)** Generar cuádruplo de ENDPROC.

**3**

**2**

**1**

**<FUNCIONPAM>**

Imagen relacionadaImagen relacionadaImagen relacionadaImagen relacionadaImagen relacionadaImagen relacionada

**1)** Buscar en la tabla de variables (dado un scope) si la variable existe, y si no existe la agrega a la tabla junto con su tipo.

**2)** Incrementar la cantidad de argumentos de una función en 1.

**3)** Incrementar la cantidad de argumentos de una función en 1.

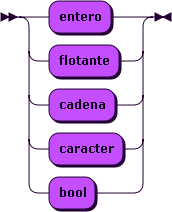
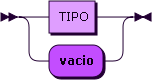
**<TIPO\_REGRESO> <TIPO>**

Imagen relacionadaImagen relacionadaImagen relacionadaImagen relacionada

**<BLOQUEFUN>**

**1 2**

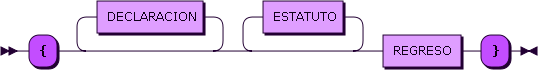


Imagen relacionadaImagen relacionadaImagen relacionadaImagen relacionada

**1)** Inserta en la tabla de símbolos la cantidad de variables locales que tiene la función.

**2)** Insertar el valor actual del contador de los cuádruplos en la tabla de símbolos.

**3**

**2**

**1**

**<REGRESO>**

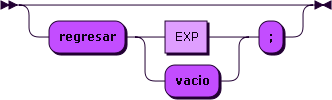
Imagen relacionadaImagen relacionada

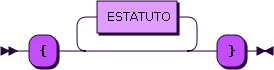
Imagen relacionadaImagen relacionada

**1)** Validar el tipo de retorno de la función.

**2)** Generar el cuádruplo RETURN.

**3)** Checar que si la función no es de tipo “vacío” entonces regresar el tipo de dato correcto.

**<BLOQUE>**

Imagen relacionadaImagen relacionada

**<LLAMADAPAR>**

**1 2**

**3**

Imagen relacionadaImagen relacionadaC:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LLAMADAPAR.PNG

Imagen relacionadaC:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LLAMADAPARAUX.PNG

**1)** Validar que argumentos mandados en la llamada a una función correspondan al tipo de parámetros que espera la función y genera el cuádruplo PARAMETER.

**2)** Incrementar apuntador hacia los argumentos de una función e ir validando a la par los argumentos con los parámetros.

**3)** Incrementar apuntador hacia los argumentos de una función e ir validando a la par los argumentos con los parámetros.

**<ESTATUTO>**



Imagen relacionadaImagen relacionada

**1**

**1)** Guardar en una variable el último id encontrado.

**<INGRESO>**

**1**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\INGRESO.PNG

**1)** Generar el código de entrada.

**1**

**<ESCRITURA>**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ESCRITURA.PNG

**1)** Generar el código de escritura.

**<CTE>**

**1**

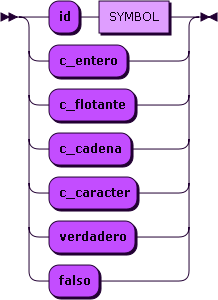


Imagen relacionadaImagen relacionada

**1)** Guardar el ID en una variable.

**<SYMBOL>**

**1**

Imagen relacionadaImagen relacionada

**1)** Verificar que cuando se haga una asignación a una variable, la variable o función que se quiera asignar no esté o regrese vacío.

**<CTE\_ID>**

**4**

**2 3**

**1**

Imagen relacionadaImagen relacionadaC:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CTE_ID.PNG

**1)** Checar que la variable exista en la tabla de símbolos.

**2)** Guardar el ID del arreglo y meterlo a la pila de dimensiones.

**3)** Validar que la variable a la que se está tratando de acceder sea dimensionada.

**4)** Generar el cuádruplo de acceso a los elementos de un arreglo VER.

**2**

**1**

**<EXPRESION>**

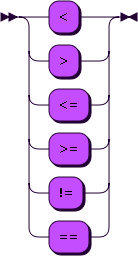
Imagen relacionada

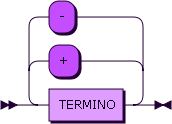
Imagen relacionada

**1)** Agregar operadores relacionales a la pila de operadores.

**2)** Genera cuádruplos de expresiones con operaciones de suma y resta.

**<COMPARACION>**

Imagen relacionadaImagen relacionada

**<EXP>**

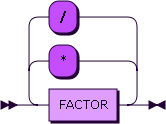
**1**

**Imagen relacionada**

**Imagen relacionada**

**1)** Genera cuádruplos de expresiones con operaciones de suma y resta.

**<TERMINO>**

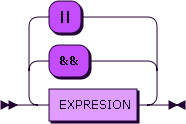


**1**

**Imagen relacionadaImagen relacionada**

**1)** Genera cuádruplos de expresiones con operaciones de multiplicación y división.

**<LOGICO>**



**1**

**Imagen relacionadaImagen relacionada**

**2**

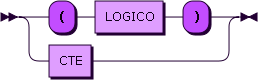
**1)** Generar cuádruplo correspondiente a expresiones con operaciones lógicas && y ||.

**2)** Agregar operador lógico a la pila de operadores.

**<FACTOR>**

**22**

**1**



**Imagen relacionadaImagen relacionada**

**1)** Crear fondo falso para las operaciones anidadas.

**2)** Quitar fondo falso para terminar con operaciones anidadas.

**1**

**<CICLO>**

**3**

**2**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CICLO.PNG

**1)** Insertar contador a la pila de saltos.

**2)** Generar cuádruplo GotoF.

**3)** Genera cuádruplo Goto.

**<CONDICION>**

**3**

**2**

**1**

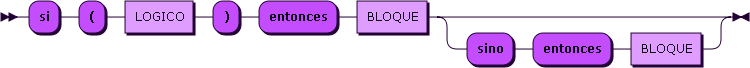


Imagen relacionada

Imagen relacionada

**1)** Generar cuádruplo GotoF.

**2)** Generar cuádruplo Goto.

**3)** Obtener de la pila de saltos el número del cuádruplo a rellenar.

**<LLAMADA>**

**3**

**4 5**

**1 2**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LLAMADA.PNG

**1)** Verificar en la tabla de símbolos que la función que se está llamando exista.

**2)** Guardar el nombre de la última función que se leyó.

**3)** Generar cuádruplo ERA.

**4)** Validar cantidad de argumentos que son enviados a una función cuando se llama.

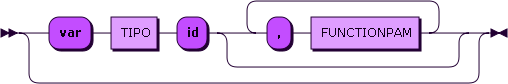
**5)** Generar cuádruplo GOSUB.

**<FUNCTIONPAM>**

**3**

**2**

**1**

**Imagen relacionadaImagen relacionadaImagen relacionadaImagen relacionada**

**1)** Insertar en la tabla de variables de la funciónel valor del parámetro.

**2)** Incrementar en la cantidad de parámetros que debe recibir una función.

**3)** Incrementar en la cantidad de parámetros que debe recibir una función.

**<ASIGNACIÓN>**

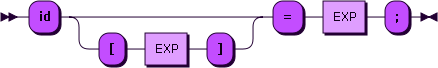
**8**

**6**

**7**

**4 5**

**1 2 3**



**Imagen relacionadaImagen relacionada**

**1)** Guardar en una variable el último ID encontrado.

**2)** Checar en la tabla de símbolos que el ID exista.

**3)** Checar que el ID vaya a ser utilizado posteriormente, de otra manera vaciar las pilas.

**4)** Guardar el ID del arreglo, agregarlo a la pila de dimensiones y eliminar ID de la pila de operadores.

**5)** Validar que el ID sea una variable dimensionada.

**6)** Generar cuádruplo VER.

**7)** Insertar operador de asignación ‘=’ en la pila de operadores.

**8)** Generar cuádruplo de asignación ‘=’ para darle valor a una variable.

**<TRIANGULO>**

**5 6**

**4**

**1 2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\TRIANGULO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<TEXTO>**

**5 6**

**4**

**1 2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\TEXTO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<RECTANGULO>**

**1 2 3**

**4**

**5 6**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\RECTANGULO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<PUNTO>**

**5 6**

**4**

**1 2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PUNTO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<OVALO>**

**4**

**1 2 3**

**5 6**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\OVALO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<LÍNEA>**

**1 2 3**

**4**

**5 6**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\LINEA.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<CURVA>**

**5 6**

**4**

**1 2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CURVA.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

**<CIRCULO>**

**5 6**

**4**

**1 2 3**

C:\Users\César Benavides\AppData\Local\Microsoft\Windows\INetCache\Content.Word\CIRCULO.PNG

**1)** Obtiene el ID de la función actual.

**2)** Buscar que la función esté en la tabla de símbolos.

**3)** Guardar el ID de la función actual para hacer las llamadas posteriores.

**4)** Generar el cuádruplo de ERA.

**5)** Checar que la cantidad de argumentos sea la misma con la que está guardada esa función.

**6)** Generar el cuádruplo GOSUB.

#### Tabla de Consideraciones Semánticas

APODORAX toma en consideración algunos aspectos semánticos para ver la compatibilidad entre tipos, por lo que se formó un Cubo Semántico para mantener la relación entre los distintos tipos de datos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **var1** | **var2** | **-** | **+** | **\*** | **/** | **&&** | **| |** |
| entero | entero | ENT | ENT | ENT | FLOAT | ERROR | ERROR |
| entero | flotante | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| entero | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | entero | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| flotante | flotante | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| flotante | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | bool | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **var1** | **var2** | **>** | **<** | **>=** | **<=** | **!=** | **==** | **=** |
| entero | entero | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | ENT |
| entero | flotante | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | ENT |
| entero | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | entero | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | FLOAT |
| flotante | flotante | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | FLOAT |
| flotante | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | cadena | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | CADENA |
| cadena | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | caracter | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | CHAR |
| caracter | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | bool | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | BOOL |

### Descripción del Proceso de Administración de Memoria usado en Compilación

Para la administración de memoria usada en compilación se hizo uso de una “tabla de símbolos”, la cual es un diccionario que tiene como llaves los nombres de las funciones y como valores tiene tuplas con dos elementos, el primero representa el tipo de dato que regresa la función mientras que en el segundo están diccionarios que contienen las variables pertenecientes a cada función. Para estos diccionarios de variables las llaves corresponden al nombre de las variables y los valores corresponden a una lista con las propiedades de esta variable que son: nombre de la variable, tipo de la variable, *scope* de la variable, valor de la variable, la dirección en donde está almacenado el valor, una bandera booleana que indica si la variable es o no dimensionada.

La lista de cuádruplos es generada almacenando los elementos de cada uno (token, instrucción, símbolo) en una lista que a su vez es agregada a una lista de listas que contiene todos los cuádruplos generados. Esta lista imbricada se usa finalmente para generar el archivo de salida con los cuádruplos. Eventualmente la máquina virtual lee este archivo y genera una nueva lista imbricada con los cuádruplos (que es la que se recorre durante la ejecución).

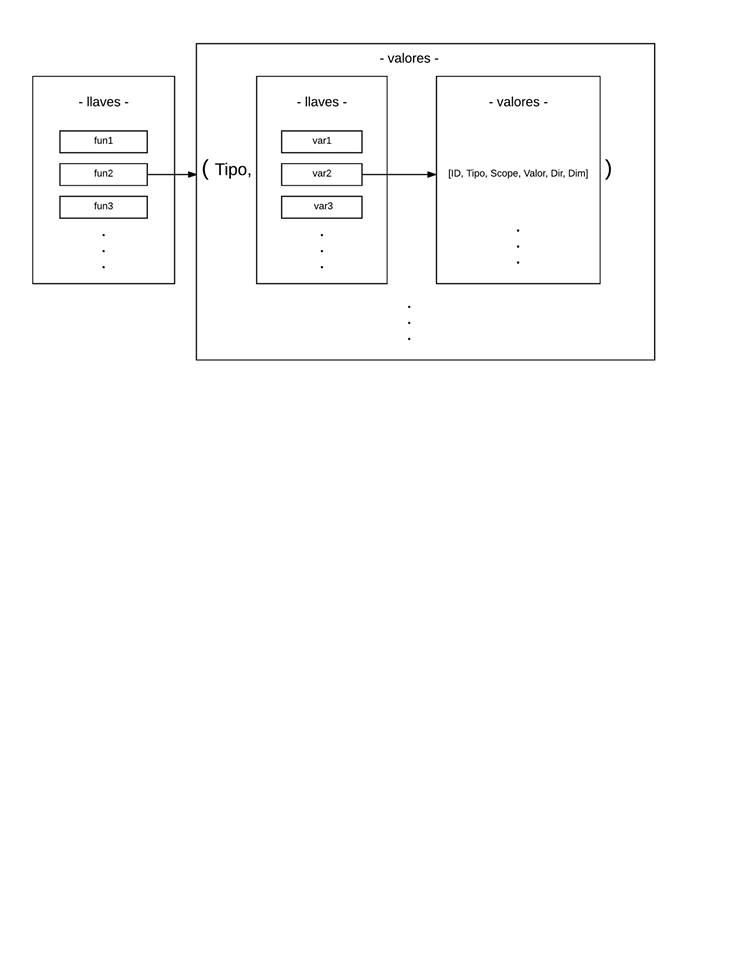
Se tiene también un diccionario en donde las llaves son los nombres de las funciones y los valores son la cantidad de parámetros.

De igual manera, hay un diccionario en donde las llaves son los nombres de las funciones y los valores la cantidad de variables en total que tiene esa función.

Hay otro diccionario en donde las llaves son los nombres de las funciones y los valores el cuádruplo en donde se encuentran, esto para uso posterior en la máquina virtual en los saltos.

Del mismo modo, hay un diccionario en donde las llaves son los nombres de las variables dimensionadas y los valores son las listas con sus características como variables.

Para el manejo de memoria se utiliza una clase “memory\_manager” encargada de administrar la comunicación entre la “tabla de símbolos” y las estructuras de memoria, las cuales consisten en una clase Memory que contiene todos los métodos necesarios para incrementar los apuntadores a la memoria virtual dependiendo de su tipo y *scope* y obtener las direcciones para los cuádruplos. Esta clase Memory hereda sus métodos a las clases correspondientes a los distintos *scopes,* las cuales tienen especificados sus propios límites dentro de la memoria.



## Descripción de la Máquina Virtual

### Equipo de Cómputo, Lenguaje y Utilerías Usadas

#### Equipo de Cómputo

Los Sistemas Operativos de las computadoras usados para la implementación y uso de la máquina virtual son Windows 8.1, Windows 10, Ubuntu Linux, OS X.

#### Lenguaje

El lenguaje en el que fue desarrollada la máquina virtual es Python 2.7, usando la herramienta PLY como analizador de léxico y de sintaxis.

#### Utilerías

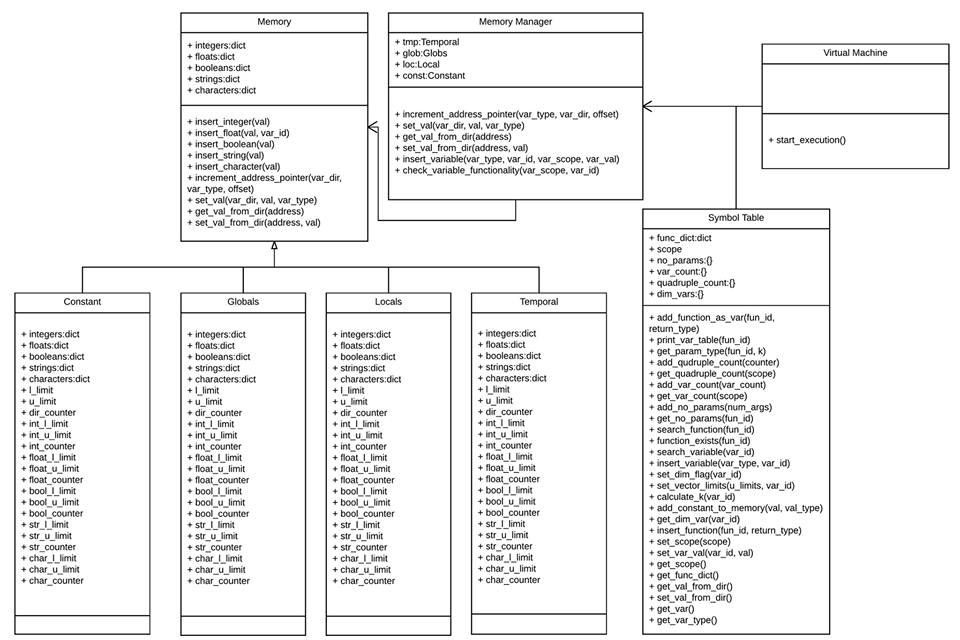
De igual manera, se hizo uso de la librería graphics.py como base para el desarrollo del output gráfico, con la cual se pueden mostrar puntos, líneas, círculos, óvalos, rectángulos, polígonos, texto e imágenes de manera gráfica.

### Proceso de Administración de Memoria en Ejecución (Arquitectura)

#### Especificación Gráfica de Cada Estructura de Datos Usada

La clase Memory tiene todos los métodos necesarios para insertar y obtener valores en la memoria virtual y de memoria real. Esta clase hereda sus métodos a las clases correspondientes a las variables constantes, globales, locales y temporales. Cada una de estas clases tiene diccionarios correspondientes a todos los tipos de datos disponibles (enteros, flotantes, caracteres, cadenas, booleanos) donde las llaves son las direcciones y los valores son los valores que se asignan a ese espacio.

Durante la ejecución se leen en los cuádruplos los identificadores de las variables y las constantes junto con las direcciones de memoria a donde se van a asignar. En la máquina virtual se llaman a los métodos de inserción y recuperación de la clase “memory\_manager” para insertar los valores especificados en los cuádruplos en las direcciones reales especificadas también en los cuádruplos.



#### Asociación Hecha entre las Direcciones Virtuales y las Reales

Las direcciones virtuales se distribuyen de tal forma que cada *scope* (local, global, temporal, constante) tenga asignado 5,000 direcciones, las cuales se van a distribuir entre los 5 tipos de datos (entero, flotante, cadena, caracter, bool), dando paso a que el acomodo de las direcciones, así como los límites de cada *scope*, queden de la siguiente manera:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Enteros | Flotantes | Booleanos | Caracteres | Cadenas | *Total* |
| Global | 0 - 999 | 1,000 -1,999 | 2,000 -2,999 | 3,000 - 3,999 | 4,000 - 4,999 | 0 – 4,999 |
| Local | 5,000 - 5,999 | 6,000 - 6,999 | 7,000 - 7,999 | 8,000 - 8,999 | 9,000 - 9,999 | 5,000 - 9,999 |
| Temporal | 10,000 - 10,999 | 11,000 - 11,999 | 12,000 - 12,999 | 13,000 - 13,999 | 14,000 - 14,999 | 10,000 - 14,999 |
| Constantes | 15,000 - 15,999 | 16,000 - 16,999 | 17,000 - 17,999 | 18,000 - 18,999 | 19,000 - 19,999 | 15,000 - 19,999 |

Dado lo anterior, cada vez que el compilador encuentre una variable, intentará colocarla en su *scope* y tipo correspondiente, dando paso a que se tenga que validar que exista el espacio suficiente (memoria) para guardar los valores de las variables o funciones, de otra forma debe marcar error.

## Pruebas del Funcionamiento del Lenguaje

### Pruebas que Comprueban el Funcionamiento del Proyecto

#### Codificación de la Prueba (APODORAX)

##### Fibonacci Iterativo

programa fibonacciIterativo:

var entero result;

var entero term1;

var entero term2;

var entero i;

var entero num;

inicio {

result = 1;

term1 = 0;

term2 = 1;

i = 1;

entrada(num);

mientras (i < num){

result = term1 + term2;

term1 = term2;

term2 = result;

i = i + 1 ;

}

desplegar (result);

}

fin

##### Fibonacci Recursivo

programa fibonacciRecursivo:

var entero res;

var entero num;

funcion entero fibonacci(var entero x) {

var entero resultado;

var entero aux1;

var entero aux2;

si (x == 0) entonces {

resultado = 0;

}

si (x == 1) entonces {

resultado = 1;

}

si (x > 1) entonces {

aux1 = fibonacci(x - 1);

aux2 = fibonacci(x - 2);

resultado = aux1 + aux2;

}

regresar resultado;

}

inicio {

entrada(num);

res = fibonacci(num);

desplegar(res);

}

fin

##### Factorial Iterativo

programa factorialIterativo:

var entero num;

var entero cont;

var entero factorial;

inicio {

factorial = 1;

cont = 1;

entrada(num);

mientras (cont <= num) {

factorial = factorial \* cont;

cont = cont + 1;

}

desplegar(factorial);

}

fin

##### Factorial Recursivo

programa factorialRecursivo:

var entero res;

var entero num;

funcion entero factorial(var entero x) {

var entero resultado;

var entero aux;

si (x <= 1) entonces {

resultado = 1;

} sino entonces {

aux = factorial(x - 1);

resultado = aux \* x;

}

regresar resultado;

}

inicio {

entrada(num);

res = factorial(num);

desplegar(res);

}

fin

##### Bubble Sort

programa bubble\_sort:

var entero i[9];

var entero va;

var bool swapped;

var entero swap;

var entero j;

var entero w;

var entero z;

inicio {

# Inicialización de variables

w = 0;

z = 8;

# se piden los valores para las casillas del arreglo

mientras(z >= w) {

entrada(i[w]);

w = w + 1;

}

va = 0;

swapped = verdadero;

mientras (swapped == verdadero) {

swapped = falso;

va = 0;

mientras (va < 8) {

si (i[va] > i[va + 1]) entonces {

swap = i[va];

i[va] = i[va + 1];

i[va + 1] = swap;

swapped = verdadero;

}

va = va + 1;

}

}

j = 0;

mientras (j < 9) {

desplegar(i[j]);

j = j + 1;

}

}

Fin

##### Find

programa busqueda:

var entero lista[3];

var caracter opcion;

var entero i;

var entero objetivo;

var entero pos;

var bool encontrado;

inicio {

i = 0;

encontrado = falso;

lista[0] = 1;

lista[1] = 2;

lista[2] = 3;

opcion = 'h';

mientras(opcion != 'v' && opcion != 'i') {

desplegar("¿Quiere buscar valor o indice?[v/i]");

entrada(opcion);

si (opcion != 'v' && opcion != 'i') entonces {

desplegar("Ingrese un valor valido.");

}

}

si (opcion == 'i') entonces {

desplegar("Ingrese el valor que desea buscar:");

entrada(objetivo);

mientras(i < 3) {

si(lista[i] == objetivo) entonces {

encontrado = verdadero;

pos = i;

i = 3;

}

i = i + 1;

}

si (encontrado == verdadero) entonces {

desplegar(pos);

} sino entonces {

desplegar("No se encontró.");

}

} sino entonces {

desplegar("Ingrese la posición que desea buscar:");

entrada(objetivo);

si (objetivo > 2 || objetivo < 0) entonces {

desplegar("Esa posición no existe.");

} sino entonces {

desplegar(lista[objetivo]);

}

}

}

fin

##### Producto Punto

programa producto\_punto:

var entero vec1[5];

var entero vec2[5];

var entero res[5];

var entero acumulado;

var entero i;

var entero j;

inicio {

i = 1;

j = 1;

acumulado = 0;

mientras (i < 5) {

vec1[i - 1] = i;

i = i + 1;

}

Mientras (i < 10) {

vec2[i + -5] = i;

i = i + 1;

}

i = 1;

mientras (i < 5) {

res[i - 1] = vec1[i - 1] \* vec2[i - 1];

i = i + 1;

}

i = 1;

mientras (i < 5) {

acumulado = acumulado + res[i - 1];

i = i + 1;

}

desplegar(acumulado);

}

Fin

##### Output Gráfico

programa dibujarEscena:

var flotante radio;

var flotante sol;

var cadena cTexto;

var entero arrGrosor[5];

var flotante lago;

var entero cont;

var flotante coordx;

var flotante coordy;

var flotante radioF;

funcion vacio solRecursivo(var flotante radio) {

si (radio > 0) entonces {

insertaCirculo(700.0, 100.0, radio, "amarillo", "naranja", 4);

solRecursivo(radio - 10.0);

}

insertaCirculo(700.0, 100.0, radio, "amarillo", "naranja", 4);

}

funcion vacio flor (var flotante coordx, var flotante coordy, var flotante radio){

# Rama

insertaLinea(coordx, coordy, coordx, coordy + 40, "negro", 5);

# Petalos Diagonal

insertaCirculo(coordx + 10, coordy + 10, radio, "rosa", "rojo", 2);

insertaCirculo(coordx + 10, coordy - 10, radio, "rosa", "rojo", 2);

insertaCirculo(coordx - 10, coordy + 10, radio, "rosa", "rojo", 2);

insertaCirculo(coordx - 10, coordy - 10, radio, "rosa", "rojo", 2);

# Centro

insertaCirculo(coordx, coordy, radio, "rojo", "rosa", 2);

}

inicio {

cTexto = "TEC";

radio = 50.0;

arrGrosor[0] = 5;

arrGrosor[1] = 10;

arrGrosor[2] = 3;

arrGrosor[3] = 2;

arrGrosor[4] = 12;

#lago = 420.0;

sol = 60.0;

cont = 1;

coordx = 700.0;

coordy = 500.0;

radioF = 10.0;

# Cielo

insertaRectangulo(0.0, 0.0, 1600 / 2.0, 200.0 \* 2, "azul", "azul", arrGrosor[2]);

# Cesped

insertaRectangulo(0.0, 400.0, 700.0 + 100.0, 600.0, "verde", "verde", arrGrosor[2]);

# Casa

insertaRectangulo(250.0, 400.0, 550.0, 200.0, "rosa", "rojo", arrGrosor[2]);

# Techo

insertaTriangulo(250.0, 200.0, 550.0, 200.0, 40 \* 10.0, arrGrosor[1] \* 10.0, "rojo", "negro", arrGrosor[4]);

# Puerta

insertaRectangulo(250.0 + 100, 400.0, 450.0, 300.0, "naranja", "gris", arrGrosor[0]);

# Camino Izquierdo

insertaCurva(450.0, 150.0 \* 4, 700.0 / 2, 600.0 - 200, "negro");

# Camino Derecha

insertaCurva(600.0, 150.0 \* 4, 450.0, 600.0 - 200, "negro");

# Nube1

insertaOvalo(30.0 , 150.0 , 80.0 \* 2, 200.0, "gris", "gris", arrGrosor[1]);

# Nube2

insertaOvalo(100.0 , 100.0 , 80.0 \* 3, 150.0, "gris", "gris", arrGrosor[1]);

# Texto

insertaTexto(400.0, 250.0, "azul", cTexto, 12);

# Puntos Cielo

mientras (cont < 5000) {

insertaPunto(100.0 + cont, 40.0, "gris");

cont = cont \* 2;

}

# Flor

flor (700.0, 500.0, 10.0);

flor (700.0 - 600, 500.0, 10.0);

# Sol

solRecursivo(sol);

# Perilla Puerta

insertaCirculo(430.0, 350.0, 4.0, "amarillo", "gris", 6);

}

fin

#### Resultados Arrojados por la Generación de Código Intermedio y por la Ejecución

##### Fibonacci Iterativo

0:['Goto', '', '', 1]

1:['=', 15000, '', 0]

2:['=', 15001, '', 1]

3:['=', 15002, '', 2]

4:['=', 15003, '', 3]

5:['entrada', '', '', 4]

6:['<', 3, 4, 12000]

7:['GotoF', 12000, '', 15]

8:['+', 1, 2, 10000]

9:['=', 10000, '', 0]

10:['=', 2, '', 1]

11:['=', 0, '', 2]

12:['+', 3, 15004, 10001]

13:['=', 10001, '', 3]

14:['Goto', '', '', 6]

15:['escritura', '', '', 0]

##### Fibonacci Recursivo

0:['Goto', '', '', 23]

1:['==', 5006, 15000, 12000]

2:['GotoF', 12000, '', 4]

3:['=', 15001, '', 5007]

4:['==', 5006, 15002, 12001]

5:['GotoF', 12001, '', 7]

6:['=', 15003, '', 5007]

7:['>', 5006, 15004, 12002]

8:['GotoF', 12002, '', 21]

9:['ERA', 'fibonacci', '', '']

10:['-', 5006, 15005, 10000]

11:['PARAMETER', 10000, '', 'arg #0']

12:['GOSUB', 'fibonacci', '', 'dir']

13:['=', 2, '', 5008]

14:['ERA', 'fibonacci', '', '']

15:['-', 5006, 15006, 10001]

16:['PARAMETER', 10001, '', 'arg #0']

17:['GOSUB', 'fibonacci', '', 'dir']

18:['=', 2, '', 5009]

19:['+', 5008, 5009, 10002]

20:['=', 10002, '', 5007]

21:['RETURN', 5007, '', '']

22:['ENDPROC', '', '', '']

23:['entrada', '', '', 1]

24:['ERA', 'fibonacci', '', '']

25:['PARAMETER', 1, '', 'arg #0']

26:['GOSUB', 'fibonacci', '', 'dir']

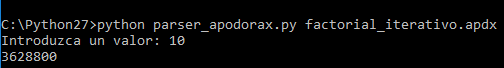
27:['=', 2, '', 0]

28:['escritura', '', '', 0]

##### Factorial Iterativo

0:['Goto', '', '', 1]

1:['=', 15000, '', 2]

2:['=', 15001, '', 1]

3:['entrada', '', '', 0]

4:['<=', 1, 0, 12000]

5:['GotoF', 12000, '', 11]

6:['\*', 2, 1, 10000]

7:['=', 10000, '', 2]

8:['+', 1, 15002, 10001]

9:['=', 10001, '', 1]

10:['Goto', '', '', 4]

11:['escritura', '', '', 2]

##### Factorial Recursivo

0:['Goto', '', '', 14]

1:['<=', 5006, 15000, 12000]

2:['GotoF', 12000, '', 5]

3:['=', 15001, '', 5007]

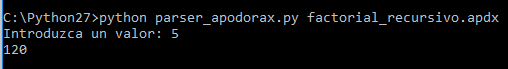
4:['Goto', '', '', 12]

5:['ERA', 'factorial', '', '']

6:['-', 5006, 15002, 10000]

7:['PARAMETER', 10000, '', 'arg #0']

8:['GOSUB', 'factorial', '', 'dir']

9:['=', 2, '', 5008]

10:['\*', 5008, 5006, 10001]

11:['=', 10001, '', 5007]

12:['RETURN', 5007, '', '']

13:['ENDPROC', '', '', '']

14:['entrada', '', '', 1]

15:['ERA', 'factorial', '', '']

16:['PARAMETER', 1, '', 'arg #0']

17:['GOSUB', 'factorial', '', 'dir']

18:['=', 2, '', 0]

19:['escritura', '', '', 0]

##### Bubble Sort

0:['Goto', '', '', 1]

1:['=', 15001, '', 12]

2:['=', 15002, '', 13]

3:['>=', 13, 12, 12000]

4:['GotoF', 12000, '', 12]

5:['VER', 12, 0, 8]

6:['+', 12, '0\_', 10000]

7:['+', 10000, '0\_', 10000]

8:['entrada', '', '', '\_10000']

9:['+', 12, 15003, 10001]

10:['=', 10001, '', 12]

11:['Goto', '', '', 3]

12:['=', 15004, '', 9]

13:['=', 17000, '', 2000]

14:['==', 2000, 17001, 12001]

15:['GotoF', 12001, '', 51]

16:['=', 17002, '', 2000]

17:['=', 15005, '', 9]

18:['<', 9, 15006, 12002]

19:['GotoF', 12002, '', 50]

20:['VER', 9, 0, 8]

21:['+', 9, '0\_', 10002]

22:['+', 10002, '0\_', 10002]

23:['+', 9, 15007, 10003]

24:['VER', 10003, 0, 8]

25:['+', 10003, '0\_', 10004]

26:['+', 10004, '0\_', 10004]

27:['>', '\_10002', '\_10004', 12003]

28:['GotoF', 12003, '', 47]

29:['VER', 9, 0, 8]

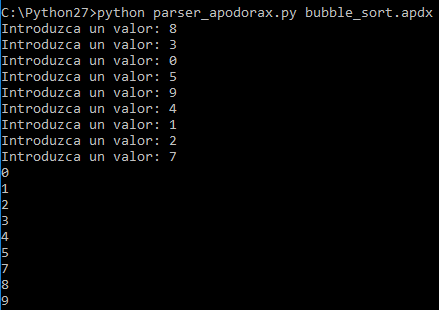
30:['+', 9, '0\_', 10005]

31:['+', 10005, '0\_', 10005]

32:['=', '\_10005', '', 10]

33:['VER', 9, 0, 8]

34:['+', 9, '0\_', 10006]

35:['+', 10006, '0\_', 10006]

36:['+', 9, 15008, 10007]

37:['VER', 10007, 0, 8]

38:['+', 10007, '0\_', 10008]

39:['+', 10008, '0\_', 10008]

40:['=', '\_10008', '', '\_10006']

41:['+', 9, 15009, 10009]

42:['VER', 10009, 0, 8]

43:['+', 10009, '0\_', 10010]

44:['+', 10010, '0\_', 10010]

45:['=', 10, '', '\_10010']

46:['=', 17003, '', 2000]

47:['+', 9, 15010, 10011]

48:['=', 10011, '', 9]

49:['Goto', '', '', 18]

50:['Goto', '', '', 14]

51:['=', 15011, '', 11]

52:['<', 11, 15012, 12004]

53:['GotoF', 12004, '', 61]

54:['VER', 11, 0, 8]

55:['+', 11, '0\_', 10012]

56:['+', 10012, '0\_', 10012]

57:['escritura', '', '', '\_10012']

58:['+', 11, 15013, 10013]

59:['=', 10013, '', 11]

60:['Goto', '', '', 52]

##### Find

0:['Goto', '', '', 1]

1:['=', 15001, '', 3]

2:['=', 17000, '', 2000]

3:['VER', 15002, 0, 2]

4:['+', 15002, '0\_', 10000]

5:['+', 10000, '0\_', 10000]

6:['=', 15003, '', '\_10000']

7:['VER', 15004, 0, 2]

8:['+', 15004, '0\_', 10001]

9:['+', 10001, '0\_', 10001]

10:['=', 15005, '', '\_10001']

11:['VER', 15006, 0, 2]

12:['+', 15006, '0\_', 10002]

13:['+', 10002, '0\_', 10002]

14:['=', 15007, '', '\_10002']

15:['=', 19000, '', 4000]

16:['!=', 4000, 19001, 12000]

17:['!=', 4000, 19002, 12001]

18:['&&', 12000, 12001, 12002]

19:['GotoF', 12002, '', 28]

20:['escritura', '', '', 18002]

21:['entrada', '', '', 4000]

22:['!=', 4000, 19003, 12003]

23:['!=', 4000, 19004, 12004]

24:['&&', 12003, 12004, 12005]

25:['GotoF', 12005, '', 27]

26:['escritura', '', '', 18006]

27:['Goto', '', '', 16]

28:['==', 4000, 19005, 12006]

29:['GotoF', 12006, '', 51]

30:['escritura', '', '', 18008]

31:['entrada', '', '', 4]

32:['<', 3, 15008, 12007]

33:['GotoF', 12007, '', 45]

34:['VER', 3, 0, 2]

35:['+', 3, '0\_', 10003]

36:['+', 10003, '0\_', 10003]

37:['==', '\_10003', 4, 12008]

38:['GotoF', 12008, '', 42]

39:['=', 17001, '', 2000]

40:['=', 3, '', 5]

41:['=', 15009, '', 3]

42:['+', 3, 15010, 10004]

43:['=', 10004, '', 3]

44:['Goto', '', '', 32]

45:['==', 2000, 17002, 12009]

46:['GotoF', 12009, '', 49]

47:['escritura', '', '', 5]

48:['Goto', '', '', 50]

49:['escritura', '', '', 18017]

50:['Goto', '', '', 63]

51:['escritura', '', '', 18018]

52:['entrada', '', '', 4]

53:['>', 4, 15011, 12010]

54:['<', 4, 15012, 12011]

55:['||', 12010, 12011, 12012]

56:['GotoF', 12012, '', 59]

57:['escritura', '', '', 18022]

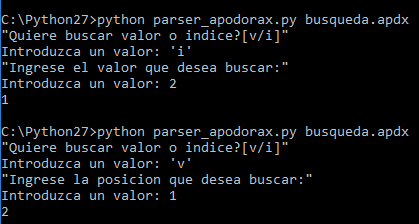
58:['Goto', '', '', 63]

59:['VER', 4, 0, 2]

60:['+', 4, '0\_', 10005]

61:['+', 10005, '0\_', 10005]

62:['escritura', '', '', '\_10005']

**

0:['Goto', '', '', 1]

1:['=', 15001, '', 3]

2:['=', 17000, '', 2000]

3:['VER', 15002, 0, 2]

4:['+', 15002, '0\_', 10000]

5:['+', 10000, '0\_', 10000]

6:['=', 15003, '', '\_10000']

7:['VER', 15004, 0, 2]

8:['+', 15004, '0\_', 10001]

9:['+', 10001, '0\_', 10001]

10:['=', 15005, '', '\_10001']

11:['VER', 15006, 0, 2]

12:['+', 15006, '0\_', 10002]

13:['+', 10002, '0\_', 10002]

14:['=', 15007, '', '\_10002']

15:['=', 19000, '', 4000]

16:['!=', 4000, 19001, 12000]

17:['!=', 4000, 19002, 12001]

18:['&&', 12000, 12001, 12002]

19:['GotoF', 12002, '', 28]

20:['escritura', '', '', 18002]

21:['entrada', '', '', 4000]

22:['!=', 4000, 19003, 12003]

23:['!=', 4000, 19004, 12004]

24:['&&', 12003, 12004, 12005]

25:['GotoF', 12005, '', 27]

26:['escritura', '', '', 18006]

27:['Goto', '', '', 16]

28:['==', 4000, 19005, 12006]

29:['GotoF', 12006, '', 51]

30:['escritura', '', '', 18008]

31:['entrada', '', '', 4]

32:['<', 3, 15008, 12007]

33:['GotoF', 12007, '', 45]

34:['VER', 3, 0, 2]

35:['+', 3, '0\_', 10003]

36:['+', 10003, '0\_', 10003]

37:['==', '\_10003', 4, 12008]

38:['GotoF', 12008, '', 42]

39:['=', 17001, '', 2000]

40:['=', 3, '', 5]

41:['=', 15009, '', 3]

42:['+', 3, 15010, 10004]

43:['=', 10004, '', 3]

44:['Goto', '', '', 32]

45:['==', 2000, 17002, 12009]

46:['GotoF', 12009, '', 49]

47:['escritura', '', '', 5]

48:['Goto', '', '', 50]

49:['escritura', '', '', 18017]

50:['Goto', '', '', 63]

51:['escritura', '', '', 18018]

52:['entrada', '', '', 4]

53:['>', 4, 15011, 12010]

54:['<', 4, 15012, 12011]

55:['||', 12010, 12011, 12012]

56:['GotoF', 12012, '', 59]

57:['escritura', '', '', 18022]

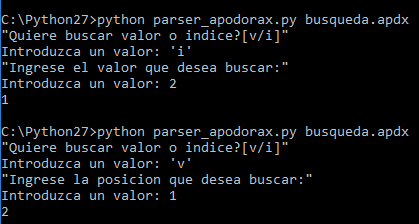
58:['Goto', '', '', 63]

59:['VER', 4, 0, 2]

60:['+', 4, '0\_', 10005]

61:['+', 10005, '0\_', 10005]

62:['escritura', '', '', '\_10005']

**

##### Producto Punto

0:['Goto', '', '', 1]

1:['=', 15003, '', 16]

2:['=', 15004, '', 17]

3:['=', 15005, '', 15]

4:['<', 16, 15006, 12000]

5:['GotoF', 12000, '', 14]

6:['-', 16, 15007, 10000]

7:['VER', 10000, 0, 4]

8:['+', 10000, '0\_', 10001]

9:['+', 10001, '0\_', 10001]

10:['=', 16, '', '\_10001']

11:['+', 16, 15008, 10002]

12:['=', 10002, '', 16]

13:['Goto', '', '', 4]

14:['<', 16, 15009, 12001]

15:['GotoF', 12001, '', 24]

16:['+', 16, 15010, 10003]

17:['VER', 10003, 0, 4]

18:['+', 10003, '0\_', 10004]

19:['+', 10004, '5\_', 10004]

20:['=', 16, '', '\_10004']

21:['+', 16, 15011, 10005]

22:['=', 10005, '', 16]

23:['Goto', '', '', 14]

24:['=', 15012, '', 16]

25:['<', 16, 15013, 12002]

26:['GotoF', 12002, '', 44]

27:['-', 16, 15014, 10006]

28:['VER', 10006, 0, 4]

29:['+', 10006, '0\_', 10007]

30:['+', 10007, '10\_', 10007]

31:['-', 16, 15015, 10008]

32:['VER', 10008, 0, 4]

33:['+', 10008, '0\_', 10009]

34:['+', 10009, '0\_', 10009]

35:['-', 16, 15016, 10010]

36:['VER', 10010, 0, 4]

37:['+', 10010, '0\_', 10011]

38:['+', 10011, '5\_', 10011]

39:['\*', '\_10009', '\_10011', 10012]

40:['=', 10012, '', '\_10007']

41:['+', 16, 15017, 10013]

42:['=', 10013, '', 16]

43:['Goto', '', '', 25]

44:['=', 15018, '', 16]

45:['<', 16, 15019, 12003]

46:['GotoF', 12003, '', 56]

47:['-', 16, 15020, 10014]

48:['VER', 10014, 0, 4]

49:['+', 10014, '0\_', 10015]

50:['+', 10015, '10\_', 10015]

51:['+', 15, '\_10015', 10016]

52:['=', 10016, '', 15]

53:['+', 16, 15021, 10017]

54:['=', 10017, '', 16]

55:['Goto', '', '', 45]

56:['escritura', '', '', 15]



##### Output Gráfico

0:['Goto', '', '', 82]

1:['>', 6029, 15001, 12000]

2:['GotoF', 12000, '', 15]

3:['ERA', 'insertaCirculo', '', '']

4:['PARAMETER', 16000, '', 'arg #0']

5:['PARAMETER', 16001, '', 'arg #1']

6:['PARAMETER', 6029, '', 'arg #2']

7:['PARAMETER', 18002, '', 'arg #3']

8:['PARAMETER', 18003, '', 'arg #4']

9:['PARAMETER', 15002, '', 'arg #5']

10:['GOSUB', 'insertaCirculo', '', 'dir']

11:['ERA', 'solRecursivo', '', '']

12:['-', 6029, 16002, 11000]

13:['PARAMETER', 11000, '', 'arg #0']

14:['GOSUB', 'solRecursivo', '', 'dir']

15:['ERA', 'insertaCirculo', '', '']

16:['PARAMETER', 16003, '', 'arg #0']

17:['PARAMETER', 16004, '', 'arg #1']

18:['PARAMETER', 6029, '', 'arg #2']

19:['PARAMETER', 18006, '', 'arg #3']

20:['PARAMETER', 18007, '', 'arg #4']

21:['PARAMETER', 15003, '', 'arg #5']

22:['GOSUB', 'insertaCirculo', '', 'dir']

23:['ENDPROC', '', '', '']

24:['ERA', 'insertaLinea', '', '']

25:['PARAMETER', 6030, '', 'arg #0']

26:['PARAMETER', 6031, '', 'arg #1']

27:['PARAMETER', 6030, '', 'arg #2']

28:['+', 6031, 15004, 11001]

29:['PARAMETER', 11001, '', 'arg #3']

30:['PARAMETER', 18012, '', 'arg #4']

31:['PARAMETER', 15005, '', 'arg #5']

32:['GOSUB', 'insertaLinea', '', 'dir']

33:['ERA', 'insertaCirculo', '', '']

34:['+', 6030, 15006, 11002]

35:['PARAMETER', 11002, '', 'arg #0']

36:['+', 6031, 15007, 11003]

37:['PARAMETER', 11003, '', 'arg #1']

38:['PARAMETER', 6032, '', 'arg #2']

39:['PARAMETER', 18016, '', 'arg #3']

40:['PARAMETER', 18017, '', 'arg #4']

41:['PARAMETER', 15008, '', 'arg #5']

42:['GOSUB', 'insertaCirculo', '', 'dir']

43:['ERA', 'insertaCirculo', '', '']

44:['+', 6030, 15009, 11004]

45:['PARAMETER', 11004, '', 'arg #0']

46:['-', 6031, 15010, 11005]

47:['PARAMETER', 11005, '', 'arg #1']

48:['PARAMETER', 6032, '', 'arg #2']

49:['PARAMETER', 18021, '', 'arg #3']

50:['PARAMETER', 18022, '', 'arg #4']

51:['PARAMETER', 15011, '', 'arg #5']

52:['GOSUB', 'insertaCirculo', '', 'dir']

53:['ERA', 'insertaCirculo', '', '']

54:['-', 6030, 15012, 11006]

55:['PARAMETER', 11006, '', 'arg #0']

56:['+', 6031, 15013, 11007]

57:['PARAMETER', 11007, '', 'arg #1']

58:['PARAMETER', 6032, '', 'arg #2']

59:['PARAMETER', 18026, '', 'arg #3']

60:['PARAMETER', 18027, '', 'arg #4']

61:['PARAMETER', 15014, '', 'arg #5']

62:['GOSUB', 'insertaCirculo', '', 'dir']

63:['ERA', 'insertaCirculo', '', '']

64:['-', 6030, 15015, 11008]

65:['PARAMETER', 11008, '', 'arg #0']

66:['-', 6031, 15016, 11009]

67:['PARAMETER', 11009, '', 'arg #1']

68:['PARAMETER', 6032, '', 'arg #2']

69:['PARAMETER', 18031, '', 'arg #3']

70:['PARAMETER', 18032, '', 'arg #4']

71:['PARAMETER', 15017, '', 'arg #5']

72:['GOSUB', 'insertaCirculo', '', 'dir']

73:['ERA', 'insertaCirculo', '', '']

74:['PARAMETER', 6030, '', 'arg #0']

75:['PARAMETER', 6031, '', 'arg #1']

76:['PARAMETER', 6032, '', 'arg #2']

77:['PARAMETER', 18036, '', 'arg #3']

78:['PARAMETER', 18037, '', 'arg #4']

79:['PARAMETER', 15018, '', 'arg #5']

80:['GOSUB', 'insertaCirculo', '', 'dir']

81:['ENDPROC', '', '', '']

82:['=', 18038, '', 3000]

83:['=', 16005, '', 1000]

84:['VER', 15019, 0, 4]

85:['+', 15019, '0\_', 10000]

86:['+', 10000, '0\_', 10000]

87:['=', 15020, '', '\_10000']

88:['VER', 15021, 0, 4]

89:['+', 15021, '0\_', 10001]

90:['+', 10001, '0\_', 10001]

91:['=', 15022, '', '\_10001']

92:['VER', 15023, 0, 4]

93:['+', 15023, '0\_', 10002]

94:['+', 10002, '0\_', 10002]

95:['=', 15024, '', '\_10002']

96:['VER', 15025, 0, 4]

97:['+', 15025, '0\_', 10003]

98:['+', 10003, '0\_', 10003]

99:['=', 15026, '', '\_10003']

100:['VER', 15027, 0, 4]

101:['+', 15027, '0\_', 10004]

102:['+', 10004, '0\_', 10004]

103:['=', 15028, '', '\_10004']

104:['=', 16006, '', 1001]

105:['=', 15029, '', 5]

106:['=', 16007, '', 1003]

107:['=', 16008, '', 1004]

108:['=', 16009, '', 1005]

109:['ERA', 'insertaRectangulo', '', '']

110:['PARAMETER', 16010, '', 'arg #0']

111:['PARAMETER', 16011, '', 'arg #1']

112:['/', 15030, 16012, 11010]

113:['PARAMETER', 11010, '', 'arg #2']

114:['\*', 16013, 15031, 11011]

115:['PARAMETER', 11011, '', 'arg #3']

116:['PARAMETER', 18039, '', 'arg #4']

117:['PARAMETER', 18040, '', 'arg #5']

118:['VER', 15032, 0, 4]

119:['+', 15032, '0\_', 10005]

120:['+', 10005, '0\_', 10005]

121:['PARAMETER', '\_10005', '', 'arg #6']

122:['GOSUB', 'insertaRectangulo', '', 'dir']

123:['ERA', 'insertaRectangulo', '', '']

124:['PARAMETER', 16014, '', 'arg #0']

125:['PARAMETER', 16015, '', 'arg #1']

126:['+', 16016, 16017, 11012]

127:['PARAMETER', 11012, '', 'arg #2']

128:['PARAMETER', 16018, '', 'arg #3']

129:['PARAMETER', 18042, '', 'arg #4']

130:['PARAMETER', 18043, '', 'arg #5']

131:['VER', 15033, 0, 4]

132:['+', 15033, '0\_', 10006]

133:['+', 10006, '0\_', 10006]

134:['PARAMETER', '\_10006', '', 'arg #6']

135:['GOSUB', 'insertaRectangulo', '', 'dir']

136:['ERA', 'insertaRectangulo', '', '']

137:['PARAMETER', 16019, '', 'arg #0']

138:['PARAMETER', 16020, '', 'arg #1']

139:['PARAMETER', 16021, '', 'arg #2']

140:['PARAMETER', 16022, '', 'arg #3']

141:['PARAMETER', 18045, '', 'arg #4']

142:['PARAMETER', 18046, '', 'arg #5']

143:['VER', 15034, 0, 4]

144:['+', 15034, '0\_', 10007]

145:['+', 10007, '0\_', 10007]

146:['PARAMETER', '\_10007', '', 'arg #6']

147:['GOSUB', 'insertaRectangulo', '', 'dir']

148:['ERA', 'insertaTriangulo', '', '']

149:['PARAMETER', 16023, '', 'arg #0']

150:['PARAMETER', 16024, '', 'arg #1']

151:['PARAMETER', 16025, '', 'arg #2']

152:['PARAMETER', 16026, '', 'arg #3']

153:['\*', 15035, 16027, 11013]

154:['PARAMETER', 11013, '', 'arg #4']

155:['VER', 15036, 0, 4]

156:['+', 15036, '0\_', 10008]

157:['+', 10008, '0\_', 10008]

158:['\*', '\_10008', 16028, 11014]

159:['PARAMETER', 11014, '', 'arg #5']

160:['PARAMETER', 18049, '', 'arg #6']

161:['PARAMETER', 18050, '', 'arg #7']

162:['VER', 15037, 0, 4]

163:['+', 15037, '0\_', 10009]

164:['+', 10009, '0\_', 10009]

165:['PARAMETER', '\_10009', '', 'arg #8']

166:['GOSUB', 'insertaTriangulo', '', 'dir']

167:['ERA', 'insertaRectangulo', '', '']

168:['+', 16029, 15038, 11015]

169:['PARAMETER', 11015, '', 'arg #0']

170:['PARAMETER', 16030, '', 'arg #1']

171:['PARAMETER', 16031, '', 'arg #2']

172:['PARAMETER', 16032, '', 'arg #3']

173:['PARAMETER', 18052, '', 'arg #4']

174:['PARAMETER', 18053, '', 'arg #5']

175:['VER', 15039, 0, 4]

176:['+', 15039, '0\_', 10010]

177:['+', 10010, '0\_', 10010]

178:['PARAMETER', '\_10010', '', 'arg #6']

179:['GOSUB', 'insertaRectangulo', '', 'dir']

180:['ERA', 'insertaCurva', '', '']

181:['PARAMETER', 16033, '', 'arg #0']

182:['\*', 16034, 15040, 11016]

183:['PARAMETER', 11016, '', 'arg #1']

184:['/', 16035, 15041, 11017]

185:['PARAMETER', 11017, '', 'arg #2']

186:['-', 16036, 15042, 11018]

187:['PARAMETER', 11018, '', 'arg #3']

188:['PARAMETER', 18055, '', 'arg #4']

189:['GOSUB', 'insertaCurva', '', 'dir']

190:['ERA', 'insertaCurva', '', '']

191:['PARAMETER', 16037, '', 'arg #0']

192:['\*', 16038, 15043, 11019]

193:['PARAMETER', 11019, '', 'arg #1']

194:['PARAMETER', 16039, '', 'arg #2']

195:['-', 16040, 15044, 11020]

196:['PARAMETER', 11020, '', 'arg #3']

197:['PARAMETER', 18056, '', 'arg #4']

198:['GOSUB', 'insertaCurva', '', 'dir']

199:['ERA', 'insertaOvalo', '', '']

200:['PARAMETER', 16041, '', 'arg #0']

201:['PARAMETER', 16042, '', 'arg #1']

202:['\*', 16043, 15045, 11021]

203:['PARAMETER', 11021, '', 'arg #2']

204:['PARAMETER', 16044, '', 'arg #3']

205:['PARAMETER', 18057, '', 'arg #4']

206:['PARAMETER', 18058, '', 'arg #5']

207:['VER', 15046, 0, 4]

208:['+', 15046, '0\_', 10011]

209:['+', 10011, '0\_', 10011]

210:['PARAMETER', '\_10011', '', 'arg #6']

211:['GOSUB', 'insertaOvalo', '', 'dir']

212:['ERA', 'insertaOvalo', '', '']

213:['PARAMETER', 16045, '', 'arg #0']

214:['PARAMETER', 16046, '', 'arg #1']

215:['\*', 16047, 15047, 11022]

216:['PARAMETER', 11022, '', 'arg #2']

217:['PARAMETER', 16048, '', 'arg #3']

218:['PARAMETER', 18060, '', 'arg #4']

219:['PARAMETER', 18061, '', 'arg #5']

220:['VER', 15048, 0, 4]

221:['+', 15048, '0\_', 10012]

222:['+', 10012, '0\_', 10012]

223:['PARAMETER', '\_10012', '', 'arg #6']

224:['GOSUB', 'insertaOvalo', '', 'dir']

225:['ERA', 'insertaTexto', '', '']

226:['PARAMETER', 16049, '', 'arg #0']

227:['PARAMETER', 16050, '', 'arg #1']

228:['PARAMETER', 18063, '', 'arg #2']

229:['PARAMETER', 3000, '', 'arg #3']

230:['PARAMETER', 15049, '', 'arg #4']

231:['GOSUB', 'insertaTexto', '', 'dir']

232:['<', 5, 15050, 12001]

233:['GotoF', 12001, '', 243]

234:['ERA', 'insertaPunto', '', '']

235:['+', 16051, 5, 11023]

236:['PARAMETER', 11023, '', 'arg #0']

237:['PARAMETER', 16052, '', 'arg #1']

238:['PARAMETER', 18067, '', 'arg #2']

239:['GOSUB', 'insertaPunto', '', 'dir']

240:['\*', 5, 15051, 10013]

241:['=', 10013, '', 5]

242:['Goto', '', '', 232]

243:['ERA', 'flor', '', '']

244:['PARAMETER', 16053, '', 'arg #0']

245:['PARAMETER', 16054, '', 'arg #1']

246:['PARAMETER', 16055, '', 'arg #2']

247:['GOSUB', 'flor', '', 'dir']

248:['ERA', 'flor', '', '']

249:['-', 16056, 15052, 11024]

250:['PARAMETER', 11024, '', 'arg #0']

251:['PARAMETER', 16057, '', 'arg #1']

252:['PARAMETER', 16058, '', 'arg #2']

253:['GOSUB', 'flor', '', 'dir']

254:['ERA', 'solRecursivo', '', '']

255:['PARAMETER', 1001, '', 'arg #0']

256:['GOSUB', 'solRecursivo', '', 'dir']

257:['ERA', 'insertaCirculo', '', '']

258:['PARAMETER', 16059, '', 'arg #0']

259:['PARAMETER', 16060, '', 'arg #1']

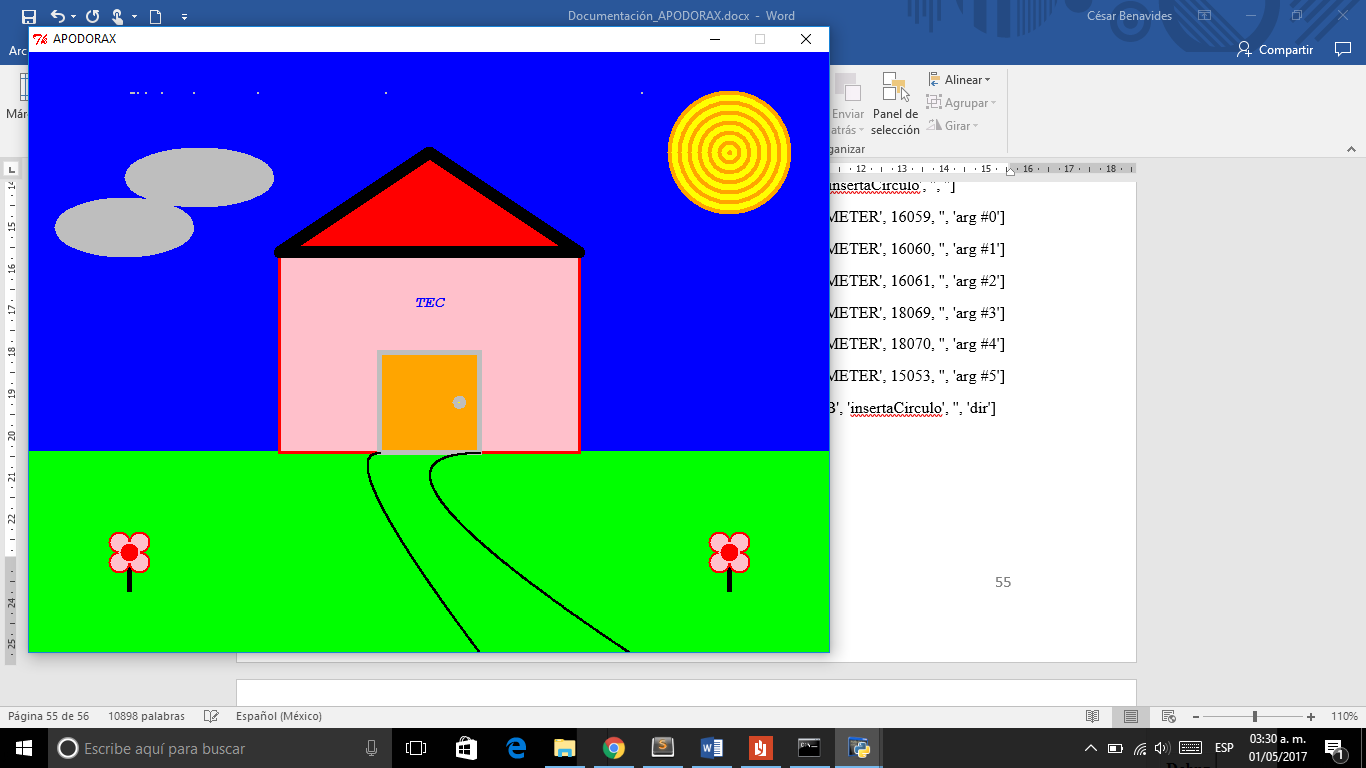
260:['PARAMETER', 16061, '', 'arg #2']

261:['PARAMETER', 18069, '', 'arg #3']

262:['PARAMETER', 18070, '', 'arg #4']

263:['PARAMETER', 15053, '', 'arg #5']

264:['GOSUB', 'insertaCirculo', '', 'dir']



## Listados Documentados del Proyecto

### Compilador

""" Clase symbol\_table

Implementa funcionalidad para el manejo de las tablas

de variables y funciones para el compilador APODORAX.

José González Ayerdi - A01036121

Martha Benavides - A01280115

3/05/2017 """

from collections import OrderedDict

from memory\_manager import memory\_manager

mm = memory\_manager()

const\_counter = 0

debug = False

class symbol\_table:

""" Constructor de la clase inicializa diccionario (que contiene las tablas)

y el scope inicial. En el diccionario las llaves son los id de las variables

mientras que los valores son diccionarios de variables, cuyas llaves son los

id de las variables y sus valores son sus propiedades. """

def \_\_init\_\_(self, func\_dic={}, scope='global'):

self.\_\_func\_dic = func\_dic

self.\_\_func\_dic['global'] = (None, OrderedDict())

self.\_\_scope = scope

self.\_\_no\_params = {}

self.\_\_var\_count = {}

self.\_\_quadruple\_count = {}

self.\_\_dim\_vars = {}

def add\_function\_as\_var(self, fun\_id, return\_type):

"""Funcion que agrega a la tabla una variable global que representa a una funcion.

Args:

fun\_id: el identificador de la funcion.

return\_type: el tipo de retorno de la funcion"""

global mm

new\_dir = mm.insert\_variable(return\_type, fun\_id, 'global', None)

self.\_\_func\_dic['global'][1][fun\_id] = [fun\_id, return\_type, 'global', None, new\_dir, False]

def print\_var\_table(self, fun\_id):

"""Funcion para debugueo que imprime la tabla de variables de una funcion.

Args:

fun\_id: El identificador de la funcion."""

table = self.\_\_func\_dic[fun\_id]

for var\_table in table:

print(var\_table)

def get\_param\_type(self, fun\_id, k):

"""Funcion que obtiene el tipo de dato que corresponde

al parametro k de una funcion.

Args:

fund\_id: Identificador de una funcion.

k: posicion del parametro

Retorno:

El tipo de dato del parametro buscado."""

key = self.\_\_func\_dic[fun\_id][1].keys()[k]

return self.\_\_func\_dic[fun\_id][1][key][1]

def add\_quadruple\_count(self, counter):

"""Funcion que acumula la numeracion de los cuadruplos.

Args:

counter: numero de cuadruplo hasta el momento."""

self.\_\_quadruple\_count[self.\_\_scope] = counter

def get\_quadruple\_count(self, scope):

"""Funcion para obtener el numero de cuadruplo de una funcion.

Args:

scope: Nombre de la funcion.

Regreso:

El cuadruplo donde se encuentra especificada la funcion"""

return self.\_\_quadruple\_count.get(scope)

def add\_var\_count(self, var\_count):

"""Suma la cantidad de variables que tiene una funcion

Args:

var\_count: Cantidad de variables."""

global debug

if debug : print("la funcion:", self.\_\_scope, "tiene:", var\_count, "locales !=")

param\_count = 0

if self.\_\_scope != "main" and self.\_\_scope != "global":

param\_count = self.get\_no\_params(self.\_\_scope)

self.\_\_var\_count[self.\_\_scope] = var\_count + param\_count

def get\_var\_count(self, scope):

"""Obtiene la cantidad de variables de una funcion.

Args:

scope: Nombre de la funcion que se busca.

Regreso:

Cantidad de variables locales que tiene esa funcion"""

return self.\_\_var\_count.get(scope)

def add\_no\_params(self, num\_args):

"""Especifica la cantidad de parametros de una funcion

Args:

num\_args: Cantidad de parametros.

Regreso:

La cantidad de parametros."""

self.\_\_no\_params[self.\_\_scope] = num\_args

def get\_no\_params(self, fun\_id):

"""Obtiene la cantidad de parametros de una funcion.

Args:

fun\_id: El nombre de la funcion.

Regreso:

La cantidad de parametros."""

return self.\_\_no\_params.get(fun\_id)

def search\_function(self, fun\_id):

""" search\_function busca el id de una función en la tabla de funciones,

si no la encuentra despliega un error

Args:

fun\_id: nombre de la funcion"""

if fun\_id not in self.\_\_func\_dic:

raise KeyError("La funcion '" + fun\_id + "' no esta declarada.")

def function\_exists(self, fun\_id):

"""Funcion que verifica si una funcion existe.

Args:

fun\_id: El nombre de la funcion.

Regreso:

Booleano que depende de si existe o no."""

return fun\_id in self.\_\_func\_dic

def search\_variable(self, var\_id):

""" search\_variable busca el id de una vairable primero dentro del scope actual,

si no la encuentra busca en el scope global, si no la encuentra despliega un error.

Args:

var\_id: Nombre de la variable."""

if self. \_\_func\_dic.get(self.\_\_scope)[1].get(var\_id) is None:

if self.\_\_func\_dic.get('global')[1].get(var\_id) is None:

raise KeyError('La variable: ' + var\_id + ' no ha sido declarada.')

def insert\_variable(self, var\_type, var\_id):

""" insert\_variable busca enla tabla de variables (dado un scope) si la variable existe,

si no existe la agrega a la tabla, de lo contrario despliega un error.

Args:

var\_type: Tipo de la variable.

var\_id: Nombre de la variable."""

if self.\_\_func\_dic.get(self.\_\_scope) is not None:

if self.\_\_func\_dic.get(self.\_\_scope)[1].get(var\_id) is None:

global mm

global debug

new\_dir = mm.insert\_variable(var\_type, var\_id, self.\_\_scope, None)

if debug : print("a",var\_id,"se le asigno",new\_dir)

self.\_\_func\_dic[self.\_\_scope][1][var\_id] = [var\_id, var\_type, self.\_\_scope, None, new\_dir, False]

else:

raise KeyError("Variable repetida: " + "'" + var\_id + "'")

else:

raise KeyError('Error en estructura de funciones/scope.')

def set\_dim\_flag(self, var\_id):

"""Funcion que determina que una variable es dimensionada y asigna su direccion base.

Args:

var\_id: Nombre de la variable."""

self.\_\_func\_dic[self.\_\_scope][1][var\_id][5] = True

# dim, l\_inf, l\_sup, k, r, aux,

base\_dir = self.\_\_func\_dic[self.\_\_scope][1].get(var\_id)[4]

self.\_\_dim\_vars[var\_id] = [1, 0, None, None, 1, None, base\_dir]

def set\_vector\_limits(self, u\_limit, var\_id):

"""Funcion que establece los limites en los espacios de un arreglo.

Args:

u\_limit: Limite superior.

var\_id: nombre de la variable."""

# calculamos los limites inferior y superior

self.\_\_dim\_vars[var\_id][2] = u\_limit - 1

u\_limit = self.\_\_dim\_vars.get(var\_id)[2]

l\_limit = self.\_\_dim\_vars.get(var\_id)[1]

# calculamos el valor r

r = self.\_\_dim\_vars.get(var\_id)[4]

r = (u\_limit - l\_limit + 1) \* r

self.\_\_dim\_vars[var\_id][4] = r

self.\_\_dim\_vars[var\_id][5] = r

global mm

global debug

# obtenemos la direccion base y el tipo para guardar la variable

base\_dir = self.\_\_func\_dic[self.\_\_scope][1].get(var\_id)[4]

var\_type = self.\_\_func\_dic[self.\_\_scope][1].get(var\_id)[1]

aux = self.\_\_dim\_vars.get(var\_id)[5]

mm.increment\_address\_pointer(var\_type, base\_dir, aux)

if debug : print("el arreglo cubre desde:", base\_dir, "hasta", base\_dir+aux)

def calculate\_k(self, var\_id):

"""Calcula el valor de la constante K.

Args:

var\_id: Es el nombre de la variable dimensionada"""

global debug

k = 0

# obtenemos los limites inferior y superior

u\_limit = self.\_\_dim\_vars.get(var\_id)[2]

l\_limit = self.\_\_dim\_vars.get(var\_id)[1]

# calculamos la r

r = self.\_\_dim\_vars.get(var\_id)[4]

r = r / (u\_limit - l\_limit + 1)

k += l\_limit \* r

# calculamos el valor final de k

self.\_\_dim\_vars[var\_id][3] = k \* (-1)

if debug : print("el valor de k es:", k)

def add\_constant\_to\_memory(self, val, val\_type):

"""Agrega el valor de una constante a la memoria.

Args:

val: Valor.

val\_type: Tipo.

Regresa:

La direccion donde se guardo la constante."""

global mm

global const\_counter

new\_dir = mm.insert\_variable(val\_type, "const", "const\_" + str(const\_counter), val)

const\_counter += 1

return new\_dir

def get\_dim\_var(self, var\_id):

"""Obtiene el diccionario de variables dimensionadas.

Args:

var\_id: Nombre de la variable.

Regreso:

Diccionario de variables dimensionadas."""

return self.\_\_dim\_vars.get(var\_id)

def insert\_function(self, fun\_id, return\_type):

""" insert\_function revisa si el id de esta funciónya existe en la tabla de funciones,

si no existe la inserta y la inicializa con vacío como valor.

Args:

fun\_id: Nombre de la funcion.

return\_type: Tipo de retorno."""

if self.\_\_func\_dic.get(fun\_id) is None:

self.\_\_func\_dic[fun\_id] = (return\_type, OrderedDict())

self.set\_scope(fun\_id)

else:

raise KeyError("Funcion '" + fun\_id + "' repetida.")

""" Sección de setters """

def set\_scope(self, scope):

self.\_\_scope = scope

def set\_func\_dic(self, func\_dic):

self.\_\_func\_dic = func\_dic

def set\_var\_val(self, var\_id, val):

"""Agrega un valor a la variable.

Args:

var\_id: Nombre de la variable.

val: Valor para la variable."""

global mm

global debug

# Se busca el scope de la variable y se le asigna el valor en memoria.

if self. \_\_func\_dic.get(self.\_\_scope)[1].get(var\_id) is not None:

self.\_\_func\_dic[self.\_\_scope][1][var\_id][3] = val

if debug : print("Estoy buscando", var\_id)

var\_dir = self.\_\_func\_dic[self.\_\_scope][1][var\_id][4]

var\_type = self.\_\_func\_dic[self.\_\_scope][1][var\_id][1]

if debug : print("dentro de if id:",var\_id,var\_dir)

mm.set\_val(var\_dir, val, var\_type)

else:

self.\_\_func\_dic['global'][1][var\_id][3] = val

var\_dir = self.\_\_func\_dic['global'][1][var\_id][4]

if debug : print("dentro de else id:",var\_id,var\_dir)

var\_type = self.\_\_func\_dic['global'][1][var\_id][1]

mm.set\_val(var\_dir, val, var\_type)

""" Sección de getters """

def get\_scope(self):

return self.\_\_scope

def get\_func\_dic(self):

return self.\_\_func\_dic

def get\_val\_from\_dir(self, address):

global mm

return mm.get\_val\_from\_dir(address)

def set\_val\_from\_dir(self, address, val):

global mm

mm.set\_val\_from\_dir(address, val)

def get\_var(self, var\_id):

"""Obtiene la lista de caracteristicas de una variable.

Args:

var\_id: Nombre de la variable.

Regreso:

Lista de la variable."""

if var\_id in self.\_\_func\_dic.get(self.\_\_scope)[1]:

return self.\_\_func\_dic.get(self.\_\_scope)[1].get(var\_id)

elif var\_id in self.\_\_func\_dic.get('global')[1]:

return self.\_\_func\_dic.get('global')[1].get(var\_id)

def get\_var\_type(self, var\_id):

if var\_id in self.\_\_func\_dic.get(self.\_\_scope)[1]:

return self.\_\_func\_dic.get(self.\_\_scope)[1].get(var\_id)[1]

elif var\_id in self.\_\_func\_dic.get('global')[1]:

return self.\_\_func\_dic.get('global')[1].get(var\_id)[1]

def check\_available\_memory(self, no\_vars):

""" Pide la memoria que sera utilizada por una llamada a funcion.

Args:

no\_vars: Cantidad de variables de una funcion. """

global mm

mm.check\_available\_memory(no\_vars)

def free\_memory(self, no\_vars):

""" Libera la memoria que ha sido utilizada por una llamada a funcion.

Args:

no\_vars: Cantidad de variables de una funcion. """

global mm

mm.free\_memory(no\_vars);

""" Sección para declarar propiedades de la clase. """

scope = property(get\_scope, set\_scope)

func\_dic = property(get\_func\_dic, set\_func\_dic)

### Máquina Virtual

# *Manual de Usuario*

## Introducción al Lenguaje APODORAX

APODORAX es un lenguaje nacido como un proyecto de la materia de Diseño de Compiladores de la carrera de Ingeniería en Tecnologías Computacionales del Tecnológico de Monterrey Campus Monterrey.

Este proyecto fue desarrollado por dos estudiantes de 8° semestre de la carrera, José González Ayerdi y Martha Lisett Benavides Martínez, con el propósito de generar un lenguaje que ofreciera la facilidad de estar escrito en español, de modo que sea más fácil y práctico programar para aquellas personas que no tienen un dominio de la lengua inglesa, con la cual la mayoría de los lenguajes de programación están desarrollados.

De igual manera, APODORAX ofrece la ventaja de poder experimentar las propiedades de la programación haciendo uso de output gráfico, el cual permite dibujar figuras con distintos colores, formas y tamaños.

## Características del Lenguaje APODORAX

### 2.2.1. Tipos de Datos

Los tipos de datos con los que se puede trabajar en APODORAX (hacer cálculos, asignar valores a variables, entre otros) son:

* *Entero*: Toma los valores entre –2.147.483.648 a 2.147.483.647. Se usa frecuentemente en operaciones aritméticas, lógicas y relacionales. Ejemplo: 5.
* *Flotante*: Toma los valores establecidos por el estándar IEEE 754, cuyos valores están entre 1.17549435×1038 – 3.402823466×1038. Se usa frecuentemente en operaciones aritméticas, lógicas y relacionales. Ejemplo: 6.5.
* *Cadena*: Toma los valores de palabras con minúsculas y/o mayúsculas, números y símbolos. Debe escribirse entre comillas dobles. Ejemplo: “holaM95”.
* *Caracter*: Toma solamente una letra, mayúscula o minúscula escrita entre comillas simples. Ejemplo ‘a’.
* *Bool*: Toma los valores de ‘verdadero’ y ‘falso’. Se usa frecuentemente en ciclos y condiciones. Ejemplo: verdadero.

### 2.2.2. Palabras Reservadas

APODORAX cuenta con una serie de palabras reservadas, es decir, que tienen una función especial en el lenguaje, por lo que no podrán ser usadas como nombres de variables o funciones (que no sean las predefinidas por el lenguaje para graficar).

|  |  |  |  |
| --- | --- | --- | --- |
| **Palabras Reservadas** | | | |
| inicio | fin | funcion | var |
| desplegar | entrada | entero | flotante |
| cadena | bool | caracter | programa |
| mientras | verdadero | falso | vacio |
| regresar | si | sino | entonces |
| c\_entero | c\_flotante | c\_cadena | c\_caracter |
| c\_bool |  | | |

### 2.2.3. Operadores

#### 2.2.3.1. Operadores Aritméticos

Los operadores aritméticos son aquellos que permiten hacer operaciones matemáticas entre dos operandos, por lo que los operadores de este tipo aceptados por APODORAX son:

* **+** : Permite realizar la suma entre dos valores.
* **-** : Permite realizar la resta entre dos valores.
* / : Permite realizar la división entre dos valores.
* \* : Permite realizar la multiplicación entre dos valores.

#### 2.2.3.2. Operadores Relacionales

Los operadores relacionales son aquellos que permiten hacer comparaciones entre dos operandos, por lo que los operadores de este tipo aceptados por APODORAX son:

* > : Permite comparar si el valor de la izquierda es mayor que el de la derecha.
* < : Permite comparar si el valor de la izquierda es menor que el de la derecha.
* >= : Permite comparar si el valor de la izquierda es mayor o igual que el de la derecha.
* <= : Permite comparar si el valor de la izquierda es menor o igual que el de la derecha.
* = = : Permite comparar si el valor de la izquierda es igual que el de la derecha.
* ! = : Permite comparar si el valor de la izquierda es diferente que el de la derecha.

#### 2.2.3.3. Operadores Lógicos

Los operadores lógicos son aquellos que permiten hacer comparaciones entre condiciones de dos operandos, por lo que los operadores de este tipo aceptados por APODORAX son:

* && : Permite comparar si dos condiciones son verdaderas, si lo son, se obtiene el valor de “verdadero”, de otra forma se obtiene “falso”.
* | | : Permite comparar si por lo menos una de las dos condiciones es verdadera, si lo es, se obtiene el valor de “verdadero”, de otra forma se obtiene “falso”.

Dado lo anterior, a continuación se muestra el tipo del valor esperado al realizar las diversas operaciones entre variables de distintos tipos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **var1** | **var2** | **-** | **+** | **\*** | **/** | **&&** | **| |** |
| entero | entero | ENT | ENT | ENT | FLOAT | ERROR | ERROR |
| entero | flotante | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| entero | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | entero | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| flotante | flotante | FLOAT | FLOAT | FLOAT | FLOAT | ERROR | ERROR |
| flotante | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | bool | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **var1** | **var2** | **>** | **<** | **>=** | **<=** | **!=** | **==** | **=** |
| entero | entero | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | ENT |
| entero | flotante | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | ENT |
| entero | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| entero | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | entero | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | FLOAT |
| flotante | flotante | BOOL | BOOL | BOOL | BOOL | BOOL | BOOL | FLOAT |
| flotante | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| flotante | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | cadena | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | CADENA |
| cadena | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| cadena | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| caracter | caracter | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | CHAR |
| caracter | bool | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | entero | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | flotante | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | cadena | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | caracter | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| bool | bool | ERROR | ERROR | ERROR | ERROR | BOOL | BOOL | BOOL |

### 2.2.4. Variables

#### 2.2.4.1. Variables Simples

Las variables simples son aquellas de una sola dimensión, a las cuales se les puede asignar un valor dependiendo de su tipo de dato.

Se pueden declarar variables globales al inicio del programa y variables locales dentro de las funciones. Para la declaración de variables se necesita seguir la siguiente sintaxis: var tipo\_de\_dato nombre\_variable;

*var entero numero;*

#### 2.2.4.2. Variables Dimensionadas

Las variables dimensionadas son aquellas que guardan mediante “casillas” diversas cantidades de datos del mismo tipo, dando paso a que se les conozca como vectores o arreglos.

Se pueden declarar arreglos siguiendo la siguiente sintaxis: var tipo\_de\_dato nombre\_arreglo[cantidad\_casillas];

*var entero clases[10];*

Como en el ejemplo anterior, se asignó que la variable clases tendrá 10 datos, los cuales podrán ser accedidos con los subíndices 0 – 9.

Por ejemplo, si se quiere asignar un valor a una casilla en específico se tiene que hacer lo siguiente:

*clases[4] = 20;*

Por otro lado, si se quiere acceder al valor de una casilla en específico se tiene que hacer lo siguiente:

*valor = clases[4];*

### 2.2.5. Funciones

Las funciones permiten realizar procedimientos sin tener que estar escribiendo el mismo código múltiples veces, por lo que al hacer una función se puede hacer un mismo procedimiento pasándole diversos valores, obteniendo valores diferentes.

Se pueden declarar funciones después de declarar todas las variables globales y antes de la función principal inicio. La sintaxis para declarar funciones es: funcion tipo\_de\_dato nombre\_funcion(declaración de parámetros){ }. Las funciones pueden tener o no retorno.

*funcion entero factorial (var entero x) {*

*resultado = 10;*

*regresar resultado;*

*}*

Si la función tiene como tipo de dato “vacio”, entonces puede tener o no un retorno, es decir, puede escribirse un *regresar vacio* o no poner el retorno.

### 2.2.6. Ciclos

Los ciclos permiten realizar de manera repetitiva ciertas acciones/procedimientos mientras que se esté cumpliendo alguna condición.

Se pueden declarar ciclos siguiendo la sintaxis: mientras (condición){ instrucciones }

*mientras (x < 10) {*

*x = x + 1;*

*}*

### 2.2.7. Condiciones

Las condiciones son aquellas que permiten tener instrucciones más complejas al hacer comparaciones u otros procedimientos de modo que se puedan ejecutar ciertas instrucciones cuando se cumpla alguna condición en específico.

Condición simple: si (condición) entonces{ instrucciones }

*si (x < 10) entonces {*

*y = 5;*

*x = x + 1;*

*}*

Condición alternativa: si (condición) entonces{ inst } sino entonces { inst }

*si (x == 10) entonces {*

*y = 5;*

*} sino entonces {*

*y = 3;*

*}*

### 2.2.8. Entrada de Información

El usuario puede ingresar información en consola usando la instrucción “entrada”, la cual permite asignarle un valor a una variable:

*entrada(num);*

### 2.2.9. Asignación

Se pueden asignar valores a las variables o subíndices de arreglos: variable = valor;

*saludo = “hola”;*

*clases[3] = 5;*

### 2.2.10. Desplegar Información

El usuario puede mostrar información en consola usando la instrucción “desplegar”:

*desplegar(“Hola Mundo”);*

La información que está entre paréntesis puede ser una variable, número, palabra, etc.

### 2.2.11. Funciones Predefinidas (Gráficas)

Las funciones predefinidas permiten graficar figuras en la pantalla, por lo que dependiendo de cada una de las formas se necesitarán diversas cantidades de coordenadas, colores, entre otros.

Las coordenadas en las que se graficarán las figuras necesitan ser del tipo flotante, por lo que se pueden poner variables, operaciones, valores de casillas de arreglos, entre otros.

Los colores que tendrán las figuras necesitan ponerse entre comillas dobles. Los colores disponibles son: negro, gris, azul, amarillo, verde, rojo, rosa y naranja.

El grosor de la línea del contorno, así como el tamaño del texto, deberán ser enteros (variables, operaciones, valores de casillas de arreglos).

Si se trata de un círculo, el radio puede ser cualquier número entero o flotante variables, operaciones, valores de casillas de arreglos).

#### 2.2.11.1. Texto

*insertaTexto(coord1, coord2, color, "texto", tamaño)*

*insertaTexto(400.0, 250.0, "azul", “APODORAX”, 12);*

#### 2.2.11.2. Rectángulo

*insertaRectangulo(coord1, coord2, coord3, coord4, colorRelleno, colorOutline, grosor)*

*insertaRectangulo(250.0, 400.0, 550.0, 200.0, "rosa", "rojo", arrGrosor[2]);*

#### 2.2.11.3. Triángulo

*insertaTriangulo(coord1, coord2, coord3, coord4, coord5, coord6, colorRelleno, colorOutline, grosor)*

*insertaTriangulo(250.0, 200.0, 550.0, 200.0, 40 \* 10.0, arrGrosor[1] \* 10.0, "rojo", "negro", arrGrosor[4]);*

#### 2.2.11.4. Círculo

*insertaCirculo(coord1, coord2, radio, colorRelleno, colorOutline, grosor)*

*insertaCirculo(coordx + 10, coordy + 10, radio, "rosa", "rojo", 2);*

#### 2.2.11.5. Óvalo

*insertaOvalo(coord1, coord2, coord3, coord4, colorRelleno, colorOutline, grosor)*

*insertaOvalo(30.0 , 150.0 , 80.0 \* 2, 200.0, "gris", "gris", arrGrosor[1]);*

#### 2.2.11.6. Punto

*insertaPunto(coord1, coord2, colorRelleno)*

*insertaPunto(100.0 + cont, 40.0, "gris");*

#### 2.2.11.7. Línea

*insertaLinea(coord1, coord2, coord3, coord4, colorRelleno, grosor)*

*insertaLinea(400.0, coordy, 15.0, coordy + 40, "negro", 5);*

#### 2.2.11.8. Curva

*insertaCurva(coord1, coord2, coord3, coord4, colorRelleno)*

*insertaCurva(450.0, 150.0 \* 4, 700.0 / 2, 600.0 - 200, "negro");*

## Ejemplos

### General

programa fibonacciIterativo:

var entero result;

var entero term1;

var entero term2;

var entero i;

var entero num;

inicio {

result = 1;

term1 = 0;

term2 = 1;

i = 1;

entrada(num);

mientras (i < num){

result = term1 + term2;

term1 = term2;

term2 = result;

i = i + 1 ;

}

desplegar (result);

}

fin

### Gráfico

programa dibujarEscena:

var flotante radio;

var flotante sol;

var cadena cTexto;

var entero arrGrosor[5];

var flotante lago;

var entero cont;

var flotante coordx;

var flotante coordy;

var flotante radioF;

funcion vacio solRecursivo(var flotante radio) {

si (radio > 0) entonces {

insertaCirculo(700.0, 100.0, radio, "amarillo", "naranja", 4);

solRecursivo(radio - 10.0);

}

insertaCirculo(700.0, 100.0, radio, "amarillo", "naranja", 4);

}

funcion vacio ola(var flotante c1, var flotante c2, var flotante c3, var flotante c4){

insertaCurva(c1, c2, c3, c4, "negro");

}

inicio {

cTexto = "Compiladores";

radio = 50.0;

arrGrosor[0] = 5;

arrGrosor[1] = 10;

arrGrosor[2] = 3;

arrGrosor[3] = 2;

arrGrosor[4] = 12;

sol = 60.0;

cont = 1;

# Cielo

insertaRectangulo(0.0, 0.0, 1600 / 2.0, 200.0 \* 2, "gris", "gris", arrGrosor[2]);

# Agua

insertaRectangulo(0.0, 400.0, 700.0 + 100.0, 600.0, "azul", "azul", arrGrosor[2]);

# Mastil

insertaLinea(400.0, 400.0, 400.0, 300.0 - 100, "negro", 10);

# Bote

insertaRectangulo(200.0, 400.0, 600.0, 300.0, "verde", "negro", arrGrosor[2]);

# Bandera

insertaTriangulo(400.0, 200.0, 550.0, 200.0, 40 \* 10.0, arrGrosor[1] \* 10.0, "naranja", "rosa", arrGrosor[4]);

# Texto

insertaTexto(450.0, 180.0, "rojo", cTexto, 8);

# Ola

ola(0.0, 550.0 - 100, 200.0, 500.0);

ola(0.0 + 200, 550.0 - 100, 300.0 + 100, 500.0);

ola(0.0 + 300, 550.0 - 100, 400.0 + 200, 500.0);

ola(500.0, 480.0, 780.0, 410.0);

# Salvavidas

insertaOvalo(300.0 - 80 , 320.0 , 420.0 - 80, 370.0, "rojo", "gris", arrGrosor[1]);

# Aves

mientras (cont < 5000) {

insertaPunto(100.0 + cont, 80.0, "negro");

cont = cont \* 3;

}

# Sol

solRecursivo(sol);

}

fin

