Tecnológico de Monterrey

Diseño de Compiladores

"A+"

Mayra Sofía Ruiz Rodriguez	A00812918
Adriana Elizabeth Valenzuela Flores	A01195331

_____ Mayra Ruiz

_____ Adriana Valenzuela

A miércoles 23 de noviembre de 2016

Índice

1. Descripción del Proyecto:

1.1 Visión, Objetivos y Alcance del Proyecto.

Nuestro lenguaje, A+, consistirá en un lenguaje de output visual para que los jóvenes de secundaria aprendan a programar. La interfaz estará compuesta por tres partes: la primera parte contendrá un lienzo interactivo en el cual se podrán dibujar paredes con un clic para formar laberintos, habrá un personaje, Doggy, quien solo podrá moverse en los espacios vacíos; la segunda parte de la pantalla consiste en un espacio en blanco donde se podrán teclear el código requerido para poder mover al personaje, por último la tercera parte consiste en un espacio en blanco en el cuál se podrá visualizar el output escrito del programa.

El propósito de dicha aplicación es que los jóvenes aprendan a programar de una forma divertida e interactiva, nuestra aplicación está diseñada para que los jóvenes aprendan la lógica de la programación, mediante juegos. Los jóvenes podrán interactuar con los elementos básicos de los lenguajes de programación, para así formar bases sólidas en esta área.

El objetivo principal es realizar un lenguaje gráfico de tipo output, el cual ayudará a los niños a aprender un lenguaje de forma interactiva. Los niños utilizarán la pantalla como lienzo para poder dibujar laberintos y mover su personaje a través de las paredes. Los jóvenes comenzaran aprendiendo a mover el personaje en el lienzo sin utilizar paredes, y poco a poco irán dibujando laberintos e incrementando la dificultad de los mismos dependiendo de qué tan avanzados vayan en su nivel de programación. Cuando lleguen a otro nivel de programación podrán interactuar con los elementos básicos de todos los lenguajes como lo son ciclos, condiciones, desarrollo de funciones, arreglos unidimensionales y recursividad.

1.2 Análisis de requerimientos

1.2.1 Requerimientos funcionales

Identificación	Aplus_Func_1
Título	Crear y borrar Walls
Descripción	El usuario podrá modificar la interfaz para crear muros e interactuar con ellos.

Identificación	Aplus_Func_2
Título	Uso de variables
Descripción	El usuario podrá declarar diferentes variables para interactuar con el código y/o la interfaz, estas pueden ser afectadas por operaciones aritméticas y por asignación.

Identificación	Aplus_Func_3

Título	Uso de funciones
Descripción	El usuario podrá declarar diferentes funciones en el programa para utilizarlas cuando sea necesario.

Identificación	Aplus_Func_4
Título	Uso de funciones de un lenguaje: ciclos, condiciones, operaciones aritméticas
Descripción	El usuario podrá utilizar funciones conocidas de los lenguajes más utilizados como while, if y operaciones aritméticas.

Identificación	Aplus_Func_5
Título	Uso de funciones de A+: move, turnLeft, turnRight
Descripción	El usuario podrá interactuar con la interfaz al llamar a estas funciones, en cualquier cambio se modificará al personaje sobre la pantalla

Identificación	Aplus_Func_6
Título	Uso de funciones de A+: pickBeeper
Descripción	El usuario podrá interactuar con la interfaz al llamar a pickBeeper cuando ya haya posicionado a su personaje en la casilla donde el hueso se encuentre.

Identificación	Aplus_Func_7
Título	Uso de funciones de A+: putBeeper
Descripción	El usuario podrá interactuar con la interfaz al llamar a putBeeper para dejar una marca donde se posiciona actualmente el personaje.

Identificación	Aplus_Func_8
Título	Print
Descripción	El usuario podrá desplegar valores en la interfaz.

1.2.2 Requerimientos no funcionales

Identificación	Aplus_NoFunc_1

Título	Instalación de Python 3
Descripción	La computadora donde corra el programa A+ debe contar con Python 3

Identificación	Aplus_NoFunc_2
Título	Copia del lenguaje A+
Descripción	La computadora debe tener guardada una copia del lenguaje A+ en alguna carpeta.

1.3 Casos de uso generales

Identificador	Caso de uso 1
Nombre	Crear programa
Descripción	El usuario podrá crear programas con el lenguaje A+ como con cualquier otro lenguaje de programación y recibir su respectivo output en la interfaz.
Actor	Usuario
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá ver el output generado de su lenguaje.
Flujo normal	1. El usuario pensará un problema a resolver usando un lenguaje de programación o en utilizar estatutos del lenguaje para interactuar con el personaje (como usar while y dentro hacer que el personaje se mueva). 2. El usuario puede o no dibujar paredes en la cuadrícula de la interfaz 3. El usuario ingresará su código en el cuadro de texto a la izquierda 4. El usuario da click en el botón “Compile” para ejecutar su código. 5. El resultado se ve en el cuadro de texto a la derecha y, si utilizó funciones de la interfaz, verá que el personaje cambió de posición o giró dependiendo de la operación que se hizo.

Identificador	Caso de uso 2
Nombre	Crear muros
Descripción	El usuario podrá crear sus propios muros dentro de la interfaz
Actor	Usuario
Pre-Condiciones	Ninguno
Post-Condiciones	El usuario podrá ver el muro creado
Flujo normal	1. El usuario dará un click al primer cuadro donde se desea crear un muro

	2. El usuario dará otro click al siguiente cuadro deseado para hacer un muro entre los dos.
Identificador	Caso de uso 3
Nombre	Borrar muros
Descripción	El usuario podrá borrar los muros creados en la interfaz
Actor	Usuario
Pre-Condiciones	Ninguno
Post-Condiciones	El usuario podrá ver que los muros fueron borrados
Flujo normal	1. El usuario dará doble clic en cualquier lugar del cuadro superior derecho, donde se encuentra el personaje y la cuadrícula.

Identificador	Caso de uso 4
Nombre	PickBeeper
Descripción	El usuario podrá hacer que el personaje recoja un hueso de la interfaz
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	Hueso desaparece de la interfaz
Flujo normal	1. El usuario guía al personaje hacia donde se encuentra el hueso y utiliza la función pickBeeper() para recogerlo,

Identificador	Caso de uso 5
Nombre	PutBeeper
Descripción	El usuario podrá hacer que el personaje deje un rastro en un punto específico.
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	Se crea una nueva imagen en la interfaz

Flujo normal	1. El usuario guía al personaje hasta dónde quiere dejar una marca. 2. El usuario ingresa la función putBeeper()

Identificador	Caso de uso 6
Nombre	CheckWall
Descripción	El usuario podrá verificar si existe un muro cercano al personaje
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	Si existe un muro, manda un mensaje a la línea de comandos: “Hayun muro cercano”.
Flujo normal	1. El usuario guía, o no, al personaje hasta dónde quiere checar si hayun muro cercano. 2. El usuario ingresa la función checkWall() 3. En la línea de comandos se muestra el mensaje si es que haymuros cercanos.

Identificador	Caso de uso 7
Nombre	Uso de funciones de un lenguaje: ciclos, condiciones, operaciones
Descripción	El usuario podrá hacer uso de funciones creadas por el mismo, declaración de variables y operaciones aritméticas (+,-,*,/) y lógicas (& ,|).
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá utilizar la lógica de programación del lenguaje A+
Flujo normal	1. El usuario utiliza el lenguaje para crear variables, funciones o hacer cálculos aritméticos o lógicos.

Identificador	Caso de uso 8
Nombre	Crear funciones
Descripción	El usuario podrá crear sus propias funciones
Actor	Usuario, lenguaje de programación

Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá utilizar la función creada a lo largo del programa
Flujo normal	1. El usuario declarará la función iniciando con un “def” 2. Seguirá su valor de retorno 3. El usuario escribirá los parámetros de la función si es que requiere. 4. Para finalizar, tecleara el signo de dos puntos (:) y empezará a describir en el lenguaje A+ la funcionalidad. 5. Se termina la declaración de la funcionalidad usando un “end_def”

Identificador	Caso de uso 9
Nombre	Declaración de variables
Descripción	El usuario podrá declarar las variables que necesite
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá utilizar las variables a lo largo del programa.
Flujo normal	1. El usuario declara el tipo de variable 2. El usuario ingresa el nombre de la variable 3. El usuario puede seguir declarando nuevas variables de ese tipo mientras que estén separadas por una coma. 4. El usuario termina la declaración de variables con un punto y coma (;)

Identificador	Caso de uso 10
Nombre	Uso de arreglos
Descripción	El usuario podrá crear arreglos para manejar datos que así lo requieran
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá utilizar y asignar valores a los arreglos
Flujo normal	1. El usuario declarará la variable antes del main como una variable normal, excepto que se añadirán corchetes después de su nombre y dentro de ellos se colocará el tamaño del arreglo (en enteros). 2. Cada arreglo debe ser declarado en un renglón distinto.

Identificador	Caso de uso 11
Nombre	Print
Descripción	El usuario podrá imprimir en la interfaz alguna variable, string o dato.
Actor	Usuario, lenguaje de programación
Pre-Condiciones	Conocer el léxico, sintaxis y gramática de A+.
Post-Condiciones	El usuario podrá ver en pantalla lo que escriba en el programa
Flujo normal	1. El usuario inicia con la función print seguido de los signos de paréntesis 2. Dentro de los paréntesis, el usuario escribirá lo que quiera imprimir. 3. Termina escribiendo un punto y coma (;)

1.4 Descripción de los principales "Test Cases"

Pruebas básicas del lenguaje:

- Probar el funcionamiento de declaración de variables, arreglos y funciones
- Comprobar que la sintaxis, gramática y léxico funcionan de manera adecuada según nuestros diagramas de sintaxis.
- Probar el funcionamiento de funciones declaradas por el usuario
- Probar que los arreglos puedan recibir un dato como valor de esa casilla.
- Probar que los arreglos saquen información obtenida
- Probar el uso de expresiones aritméticas y lógicas

Pruebas de la interfaz:

- Probar el uso de funciones de la interfaz como move(), turnLeft(), turnRight
- Probar el botón "Compile" el cual ejecuta el código detrás para procesar lo que el usuario ha tecleado en el campo para programar el lenguaje.

1.5 Descripción del proceso general seguido para el desarrollo del proyecto

1.5.1 Proyecto

El proyecto desarrollado es un lenguaje de programación similar a Python en funcionalidad, fue creado para que personas que apenas inician en el mundo de la programación puedan entender fácilmente instrucciones comunes en los lenguajes de programación más usados como ciclos, condiciones, operaciones aritméticas o de lógica. Para complementar este aprendizaje, las personas que utilicen este lenguaje pueden complementar sus conocimientos del lenguaje con la interfaz gráfica, que funciona en conjunto con el código del lenguaje como una opción para que los usuarios puedan aprender más sobre programación

1.5.2 Bitácoras

Avance #1: Léxico y Sintaxis

24 de Septiembre
En cuanto a la sintaxis se presentaron problemas de jerarquía de lenguaje, tuvimos que repasar varias veces los diagramas y tuvimos problemas al limpiar la gramática, aunque no tuvimos problemas de recursión izquierda. Una de las cosas que se nos dificultó fue que al crear el léxico no nos aceptaban las líneas de comentarios, en

cuanto entraba un comentario el programa tronaba. Al final vimos el problema y se arregló. Al final se terminó completamente el avance, el léxico y sintaxis funcionan completamente.

Avance #2: Dir de Funciones y Tablas de Variables

7 de octubre
En esta entrega se nos complicó el pensar cómo implementar la tabla de variables, si usar hashing o diccionarios, al final implementamos diccionarios y obtuvimos los datos en la función sintáctica donde detecta que se realiza una asignación, se crea un objeto de tipo tablaVar y se asigna a su respectivo diccionario. Decidimos implementar diccionarios por la facilidad de Python al manejar diccionarios y por su rapidez. Encontramos errores en léxico y sintaxis que fueron arreglados en esta misma fecha. La tabla de variables y directorio de procedimientos ya funcionan, todas las variables declaradas se guardan en la tabla.

Avance #3 Semántica y Expresiones

15 de Octubre
En esta entrega decidimos usar literalmente un cubo "cubo[x][y][z]" donde en cada casilla del cubo se asigne el valor de salida de ese cálculo, en otras palabras: "cubo[int][int][operacion] = resultado". Hicimos un diccionario de operaciones y otro diccionario de tipos para facilitar la búsqueda. Se asignó un -1 a las casillas donde se deben marcar errores, por ejemplo: "cubo[bool][string][<] = -1 // error". Los cuádruplos de aritmética se entregaron completamente.

Avance #4: Generación de Código para Estatutos

23 de Octubre
Se realizaron ciclos, decisiones, asignaciones y prints en cuádruplos. Se probaron ciclos anidados, ifs anidados, ifs anidados a whiles y todo funciona. Empezamos a investigar sobre la interfaz gráfica y que librerías podríamos usar, empezamos a utilizar graphics.py.

Avance #5: Código Intermedio de Funciones y Diseño del Mapa de Memoria para Ejecución

30 de Octubre
Se avanzó a la interfaz gráfica un 40%. Corregimos alguna funcionalidad en el cubo. Realizamos la memoria virtual pensando en alguna forma de que no fuera "monolito". La dividimos en 4, global, temporal, local y constantes tal como lo habíamos visto en clase. Esta se divide para cada uno de los tipos de dato. Y es manejada puramente por una serie de contadores.

Avance #6: Máquina Virtual: Ejecución de Expresiones Aritméticas y Estatutos Secuenciales

5 de Noviembre
Esta semana fue dedicada a realizar correcciones del código que teníamos ya limpiar dicho código, borramos una serie de estructuras que ya no se necesitaban. En cuanto a la parte gráfica, encontramos la librería de Python para gráficas, tkinter, que procesa más cosas que necesitamos como botones, textbox y divisiones de pantalla. Se dedicó esta semana a pasar el avance ya hecho en graphics.py a cambiarlo a como se manejan los objetos en tkinter. En cuanto al compilador, la parte más difícil de esta semana fue decidir qué clase de estructura usaríamos para memoria real, no pudimos avanzar mucho algunos días en lo que planeábamos la estructura, pero finalmente decidimos hacer una serie de diccionarios cuya llave sería la memoria virtual, esto para evitar hacer un "monolito" y la conversión de memoria virtual a física. Luego de esto proseguimos a codificar parte de la máquina virtual a pesar de que ya tenemos la parte de aritmética, no pudimos avanzar en exportar los cuádruplos a un archivo nuevo que la máquina virtual procesará. El día de hoy tuvimos problemas con github y desaparecieron todos los commits que se hicieron antes del merge del jueves, pero sin afectar los avances que ya teníamos, solo desaparecieron en github los commits pero no se vieron afectados los archivos que tenemos. Nos falta aún exportar cuádruplos a archivo .obj.

Avance #7: Generación de Código de Arreglos/Tipos estructurados. Máquina Virtual: Ejecución de Estatutos Condicionales

15 de Noviembre
En esta semana se vieron muchos avances en el proyecto, la parte gráfica está casi terminada, el personaje ya se mueve y ya se pueden pintar paredes en la cuadrícula, además de que el texto que se escriba en automático se guarda al archivo donde tenemos el código fuente que lee el compilador. En cuanto al compilador, se hicieron correcciones cruciales en algunos cuádruplos, además de correcciones de memoria virtual; se agregó sintácticamente "read" y todo lo relacionado a arreglos (sintaxis y cuádruplos). Ahora ya tenemos archivos obj, y una tabla de constantes y la máquina virtual está corriendo la memoria real de manera correcta, ya se implementaron los saltos condicionales (goto, y gotof) en la máquina virtual. Solo queda realizar funciones (era, gosub) y realizar los cuádruplos de funciones de nuestro lenguaje propio en la máquina virtual y conectar todo.

Avance #8: Generación de Código de Arreglos y Máquina Virtual para una parte de la aplicación particular

19 de Noviembre
Implementamos funciones en la máquina virtual, aunque aún faltan arreglos y read de consola. Esta semana hicimos un gran avance en la documentación, juntamos la información necesaria en la documentación que se encuentra en la propuesta e hicimos algunas pruebas. En cuanto a la interfaz gráfica, nuestro personaje ya puede moverse por la cuadrícula, girar, poner beepers, recoger huesos y checar si hay un muro cercano. Los huesos se ponen en un lugar aleatorio de la cuadrícula. Aún falta ligar estas opciones con la máquina virtual.

1.5.3 Reflexiones

Reflexión Adriana:
Este proyecto resultó ser un reto para mí ya que tuvimos que realizar un compilador en un lenguaje que no conocía. Al principio me era muy difícil trabajar con Python, pero a lo largo de las entregas logré manejar bien el lenguaje. Con este proyecto logré desarrollar buenas prácticas de programación y además aprendí a manejar temas como gramáticas y expresiones regulares, los cuales me habían parecido bastante difíciles en otras clases. Gracias a este proyecto he reflexionado que tan complejos son los lenguajes cotidianos y algo que parece una simple operación implica muchas horas de trabajo y esfuerzo. Con esto me he dado cuenta que debo cuidar la forma en la que debo codificar ya que ahora he logrado comprender la gran carga que se genera para el compilador y la memoria, para procesar un programa simple.

_____
Firma

Reflexión Mayra:
Este proyecto significó un gran reto ya que nunca había trabajado con Python, pero la investigación que había hecho me decía que era la mejor opción para tratar con la creación de un nuevo lenguaje. Al inicio frustraba tener tantos errores de tabulación que no sabía ni que existían, pero al paso del tiempo empezó a agradarme Python. Con este proyecto aprendí todo lo que hay detrás de un lenguaje como el manejo de memoria o el código intermedio, todo lenguaje maneja diferente estas cosas y es interesante saber cómo puede ayudarte el usar un lenguaje sobre otro teniendo en cuenta el cómo están construidos. Creo que el mayor problema que se presentó fue el pensar cómo desarrollar la máquina virtual y el arreglar los errores en contadores que se presentaron en la generación de cuádruplos ya que eso hacía que nuestro código no respondiera como esperábamos.

_____
Firma

2. Descripción del lenguaje

2.1 Nombre del lenguaje

El nombre de nuestro lenguaje es A+.

2.2 Descripción genérica de las principales características del lenguaje

A+ se caracteriza por su gran similitud a la sintaxis del lenguaje Python, pero sin el uso estricto de tabulación, las cuales solo se usan opcionalmente para organización del código. Además de que toma elementos de lenguajes como C++, como lo son el ";", la declaración de variables y funciones entre otros. Además de elementos Visual Basic como que marcan el fin de un tipo de estatuto como el "end_if", "end_else", "end_while".

El lenguaje puede comenzar o no declarando variables globales, seguido por la declaración opcional de funciones de usuario. Después, se declara el programa principal, el cual se identifica por la instrucción "main:" al cual sigue una serie de estatutos. Para indicar fin de una función se usa la palabra reservada "end_*nombre de estatuto*", por ejemplo, para terminar un estatuto if se necesitaría añadir la instrucción "end_if". Otra característica es que al final de cada instrucción se debe de ingresar un punto y coma, ej "int x = 1;". La asignación de variables no se puede realizar en la declaración de las mismas, y los arreglos deben ir declarados en líneas diferentes.

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

Durante compilación los errores que pueden ocurrir son los siguientes:

- Variable ya declarada:
    - Este error se genera cada vez que se declara una variable con el mismo nombre, no debe haber dos variables globales con el mismo nombre, y ninguna variable local puede llamarse igual que una local, pero dos locales pueden tener el mismo nombre siempre y cuando estén dentro de diferentes funciones.
- Uso incorrecto de tipos:
    - Este error se genera cada vez que tratas de igualar a una variable de un tipo de dato a una expresión regular que no sea del mismo tipo que esa variable. O bien puede ocurrir cuando se hacen operaciones de distintos tipos de dato, o bien que produzcan un tipo de dato diferente. Por ejemplo, int a; a = 2.0; genera este error ya que 2.0 es una expresión regular de tipo float y la variable a es int. int a; string b; a = a + b; genera un error ya que no se pueden hacer operaciones entre tipos de datos incorrectos.
- Variable no declarada:
    - Este error se genera cada vez que se trata de usar una variable que no haya estado previamente declarada dentro del main o bien de la función actual.
- Función previamente definida:
    - Este error ocurre cuando haya dos o más funciones con el mismo nombre.
- Se definió una función que debe retornar un valor y no lo retorna
    - Este error ocurre cuando se declara una función de algún tipo de dato como int, float, string o bool y que no haya un return dentro de la función.
- Función no definida:

    - Este error ocurre cuando se trata de usar una función que no fue previamente declarada.
- Tipo de parámetros no concuerda con los parámetros de la función:
    - Este error ocurre cuando los tipos de los parámetros que se mandan a la función no son iguales a los declarados.
- La cantidad de parámetros no concuerda con los parámetros de la función
    - Este error ocurre cuando se mandan una cantidad distinta de parámetros a los establecidos en la declaración de la función.
- Error de sintaxis
    - Este error ocurre cuando se recibe un token o una expresión no esperada.
- Error de léxico
    - Este error ocurre cuando el token recibido no coincide con la expresión regular establecida para ese token.
- El índice del arreglo solo debe contener expresiones enteras
    - Este error ocurre cuando el arreglo se indexa con una variable que no sea de tipo entera.

Durante ejecución los siguientes errores pueden ocurrir:

- Debes guardar una variable de tipo
    - Este error ocurre cuando se hace un read y el tipo de dato introducido no coincide con el tipo de dato de la variable a la cual se le asigna dicho valor.
- Índice no válido
    - Este error ocurre cuando el índice al cual se trata de acceder no está dentro de los rangos establecidos del arreglo.
- Valor no asignado:
    - Este valor ocurre cuando a la variable a la cual se trata de acceder su valor nunca fue asignada.

3. Descripción del compilador:

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

Para desarrollar el lenguaje trabajamos usando Python 3.5.2 en conjunto con la herramienta PLY (Python Lex-Yacc) 3.9 para desarrollar todo el compilador. Usamos Tk interface, la interfaz estándar de Python para realizar la parte gráfica donde el usuario añade su código y se despliega el resultado.

3.2 Descripción del Análisis de Léxico.

3.2.1 Palabras Reservadas

- int
- float
- string
- bool
- void
- return
- while
- end_while
- checkWall
- move
- turnLeft
- turnRight
- pickBeeper

- putBeeper
- if
- end_if
- else
- end_else
- print
- read
- def
- end_def
- main
- true
- false
- and
- or

3.2.2 Tokens del lenguaje con código asociado

SUMA	'+'
RESTA	'-'
MULTIPLICACION	'*'
DIVISION	'/'
PARENTESIS_IZQ	'('
PARENTESIS_DER	')'
MENOR_QUE	'<'
MAYOR_QUE	'>'
MENOR_IGUAL	'<='
MAYOR_IGUAL	'>='
DIFERENTE	'<>'
EQUIVALE	'='
IGUAL_A	'=='
PUNTO_Y_COMA	';'
DOS_PUNTOS	':'
COMA	','
CORCHETE_IZQ	'['
CORCHETE_DER	']'
AND	'&'
OR	'|'

3.2.3 Expresiones Regulares del lenguaje

- cte_float: [0-9]+\.[0-9]+
- cte_int: \d (se declara de esta forma en PLY a la expresión regular [0-9])
- id: [a-zA-Z_][a-zA-Z_0-9]*
- cte_string: "([a-zA-Z]|[0-9]|[*\[\]\\\^\-\.\?\+\|\(\)\$\/\{\}\%\<\>=&;,_:\[\]\'!$#@])*\"
- cte_bool : “true” | “false”

3.3 Descripción del Análisis de Sintaxis.

3.3.1 Gramáticas formales

program := declaracion_aux funcion_aux MAIN : opciones_aux

declaracion_aux := declaracion_var declaracion_aux | empty

funcion_aux := funcion funcion_aux | empty

opciones_aux := opciones opciones_aux | empty

declaracion_var := declaracion | declaracion_arreglo
declaracion := tipo ID declaracion_1 ;
declaracion_1 := , ID delcaracion_1 | empty
declaracion_arreglo := tipo ID [CTE_INT] ;

var_cte:= ID arreglo | CTE_INT | CTE_FLOAT | CTE_BOOL | CTE_STRING

comparador := exp comparador_aux
comparador_aux := & exp | | exp | empty

exp := expresion signos expresion
signos := < | > | <= | >= | <> | ==

expresion := termino expresion_2
expresion_2 := + expresion | - expresion | empty

termino := factor termino_2
termino_2:= * termino | / termino| empty

factor:= (expresion) | var_cte

tipo := INT | BOOL | FLOAT | STRING

asignacion := ID array = asignacion_aux ;
asignacion_aux := expresion | funcionUsuario

print := PRINT (print_aux) ;
print_aux := CTE_STRING | expresion

read := READ (ID) ;

condicion:= IF (comparador) : opciones_aux END_IF condicion_2
condicion_2 := ELSE : opciones_aux END_ELSE

ciclo := WHILE (comparador) : opciones_aux END_WHILE

tipoFuncion := INT | FLOAT | BOOL | STRING | VOID

funcion:= DEF tipoFuncion ID (funcion_param) : declaracion_aux opciones_aux END_DEF

funcionUsuario := ID (params) ;
params := expresion params_aux
params_aux := , expresion params_aux | empty

array := [expresion] | empty

opciones := asignacion | condicion | funcionUsuario | escritura | ciclo | return | turnLeft | turnRight | move | checkWall | pickBeeper | putBeeper | read

pickBeeper := PICKBEEPER ( ) ;

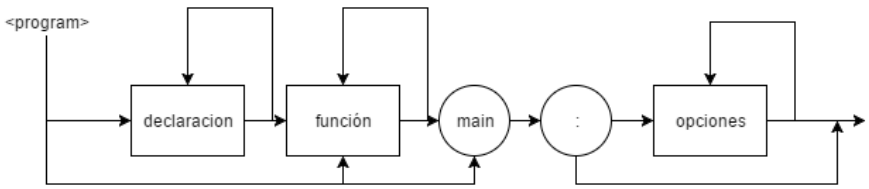
putBeeper := PUTBEEPER ( ) ;

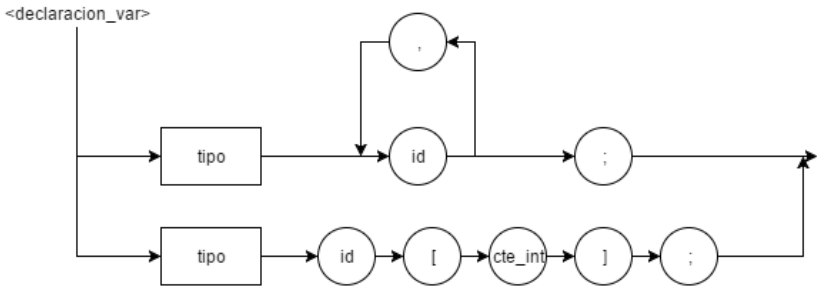
move := MOVE ( ) ;

turnLeft := TURNLEFT ( ) ;
turnRight := TURNRIGHT ( ) ;

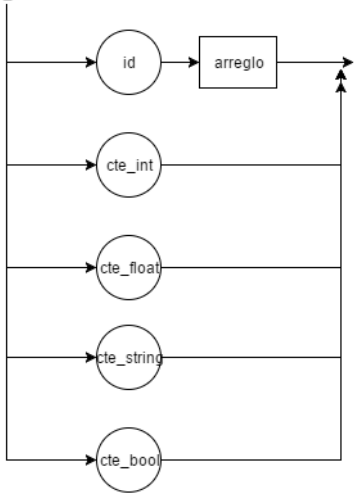
checkWall := CHECKWALL ( ) ;

3.3.2 Diagramas de sintaxis

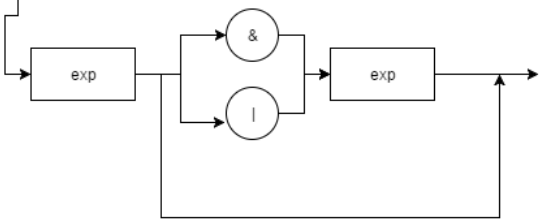




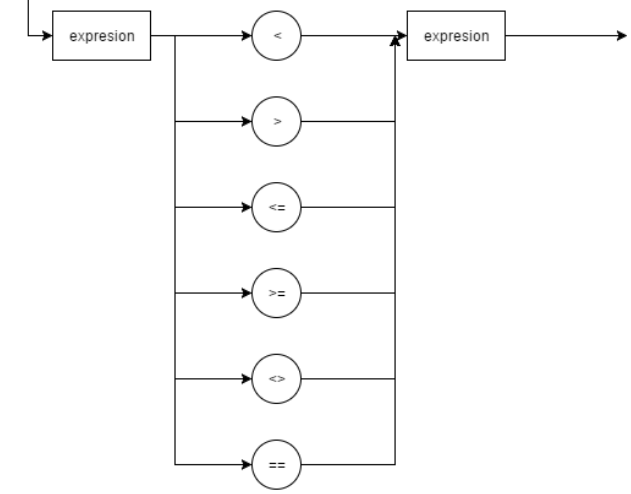
<var_cte>



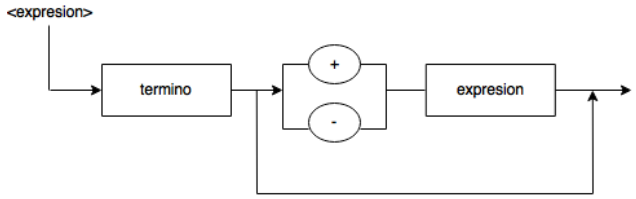
<comparador>

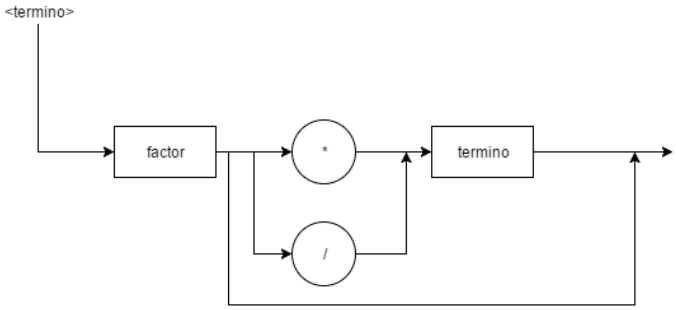


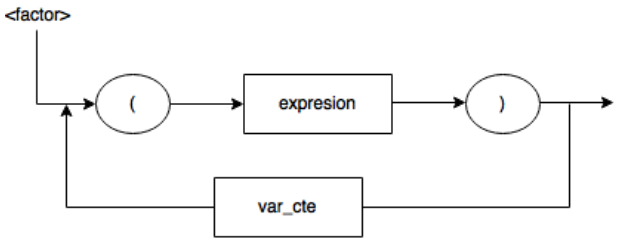
<exp>

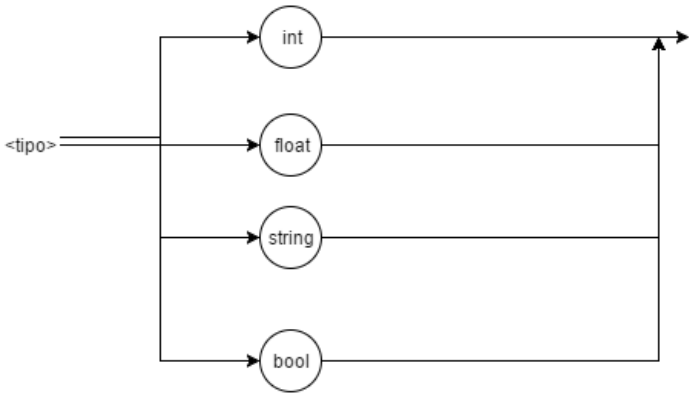


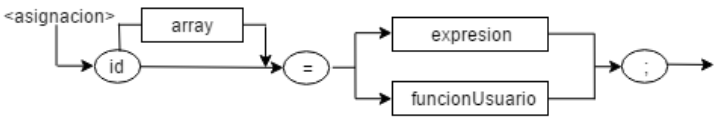
<expresion>

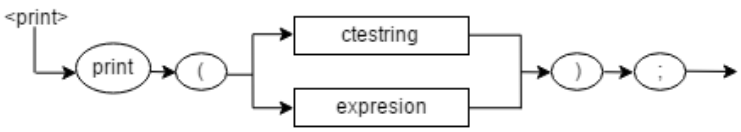


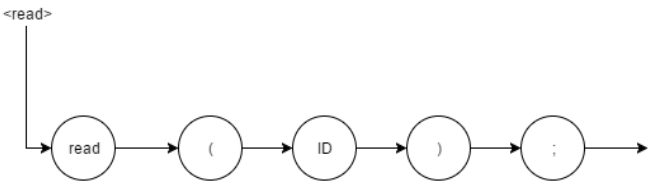

<termino>
factor
*
/
termino

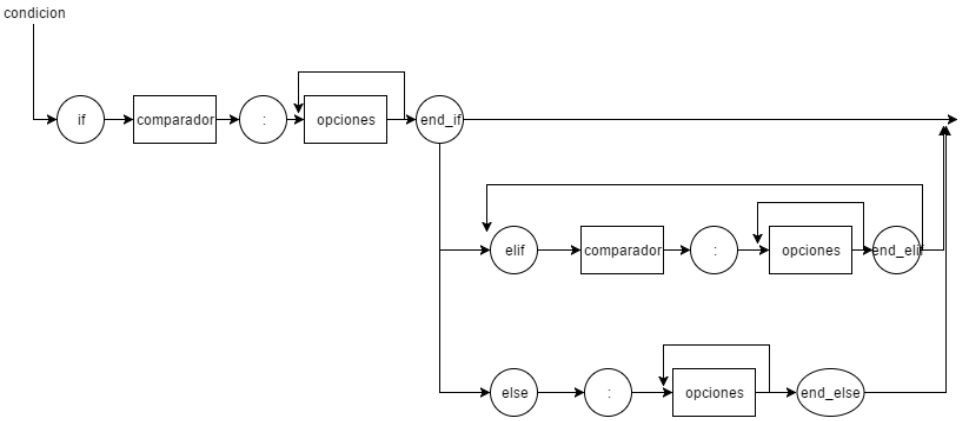

<factor>
(
expresion
)
var_cte

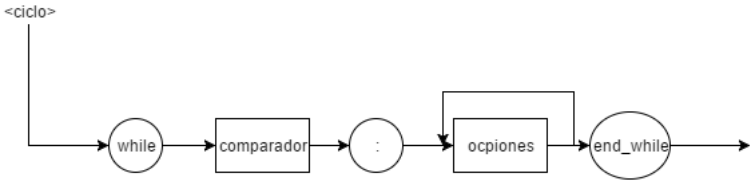

<tipo>
int
float
string
bool

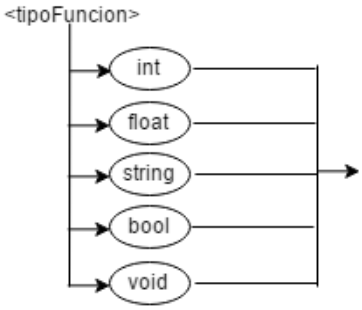

<asignacion>
array
id
=
expresion
funcionUsuario
;

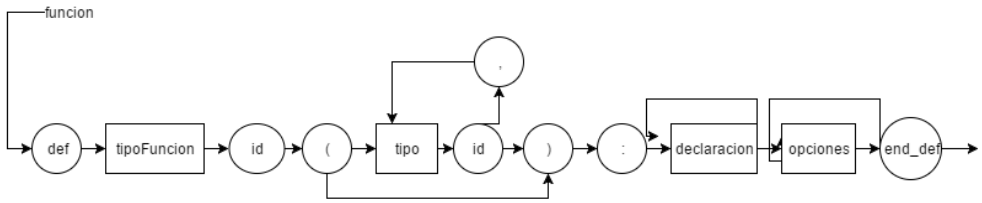

<print>
print
(
ctestring
expresion
)
;

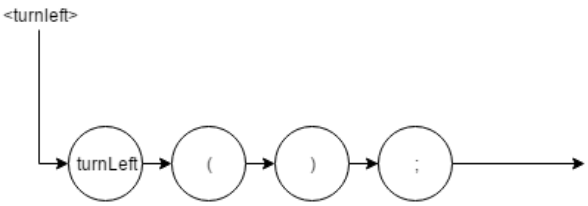

<read>
read
(
ID
)
;

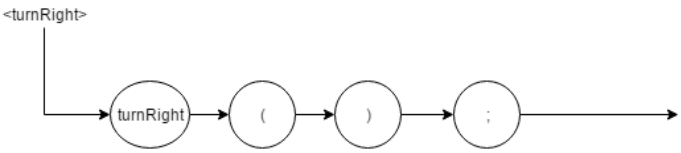

condicion
if
comparador
:
opciones
end_if
elif
comparador
:
opciones
end_elif
else
:
opciones
end_else

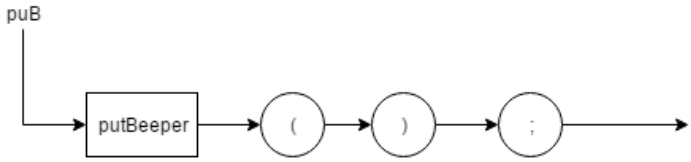

<ciclo>
while
comparador
:
ocpiones
end_while

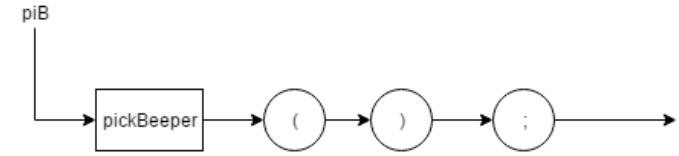

<tipoFuncion>
int
float
string
bool
void

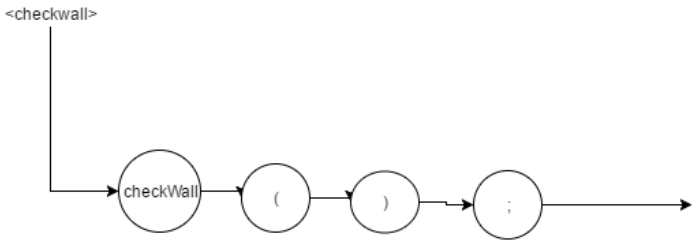

funcion
def
tipoFuncion
id
(
tipo
id
,
)
:
declaracion
opciones
end_def

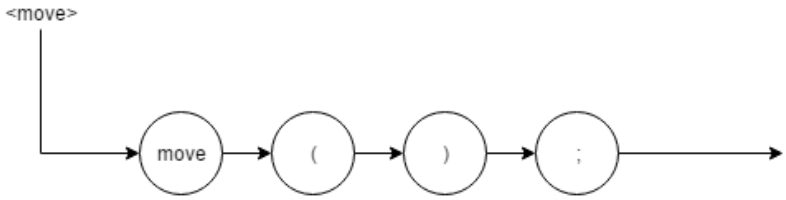

<turnleft>
turnLeft
(
)
;

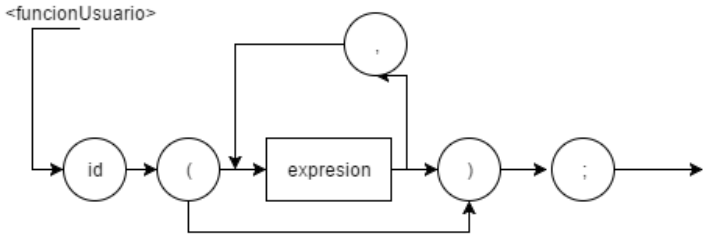

<turnRight>
turnRight
(
)
;

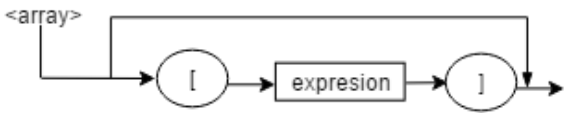

puB
putBeeper
(
)
;

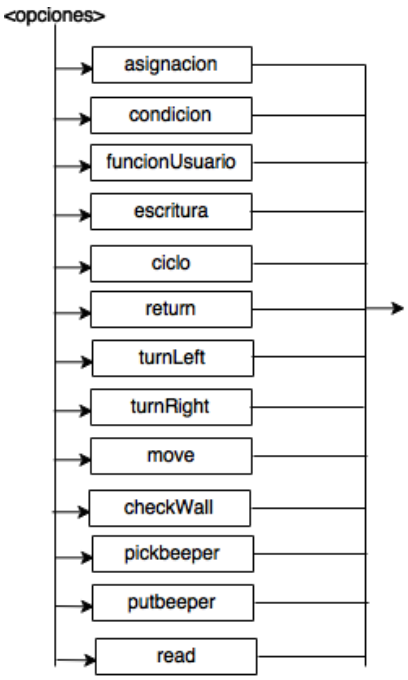

piB
pickBeeper
(
)
;


<checkwall>
checkWall
(
)
;


<move>
move
(
)
;


<funcionUsuario>
,
id
(
expresion
)
;


<array>
[
expresion
]



3.4 Descripción de Generación de Código Intermedio y Análisis Semántico.

3.4.1 Descripción semántica

3.4.1.1 Semántica

- No puede haber dos variables globales con el mismo nombre, y no puede haber ninguna local con el mismo nombre que una global, entre las variables de un mismo scope local no se pueden repetir los nombres, pero se pueden repetir entre los distintos scopes locales.
- No puede haber dos funciones con el mismo nombre.
- Una función de tipo void no puede regresar un valor.
- Una función del tipo que no sea void debe regresar un valor
- Una función que fue declarada con parámetros, cuando se mande llamar debe tener la misma cantidad y el mismo tipo de parámetros.
- Cuando se declara e indexa un arreglo, el índice debe ser siempre una variable entera, o bien una constante_int.
- Una variable que se utilice dentro del código debe haber sido previamente declarada ya sea en el scope global o en el scope local de donde se esté ejecutando.
- Las operaciones aritméticas deben siempre realizarse entre operadores del mismo tipo, en caso de la división, la división entre dos enteros siempre resulta en un flotante, por lo que el resultado de la división, debe operarse únicamente con constantes flotantes. Los operadores de comparación a pesar de que comparan variables de tipos iguales generan una variable de tipo “bool”.

3.4.1.2 Tabla de semántica

La siguiente tabla ejemplifica aquellas operaciones permitidas entre tipos de datos. Esta tabla fue utilizada como base para generar el cubo semántico, el cual es utilizado a lo largo del programa para verificar que las operaciones realizadas utilicen el tipo correcto de datos. La tabla usa notación para referirse a los tipos de dato utilizados y generados. Las primeras dos columnas representan los tipos de dato que hay, y el resto de las

columnas los operadores, y el contenido de las filas es el tipo de dato que se genera: donde i representa “int”, f representa “float”, s representa “string”, b representa “bool” y por último X es error.

op1	op2	+	-	*	/	<	>	<=	>=	<>	==	=	&	|
i	i	i	i	i	f	b	b	b	b	b	b	i	X	X
i	f	f	f	f	f	b	b	b	b	b	b	X	X	X
i	s	X	X	X	X	X	X	X	X	X	X	X	X	X
i	b	X	X	X	X	X	X	X	X	X	X	X	X	X
f	i	f	f	f	f	b	b	b	b	b	b	X	X	X
f	f	f	f	f	f	b	b	b	b	b	b	f	X	X
f	s	X	X	X	X	X	X	X	X	X	X	X	X	X
f	b	X	X	X	X	X	X	X	X	X	X	X	X	X
s	i	X	X	X	X	X	X	X	X	X	X	X	X	X
s	f	X	X	X	X	X	X	X	X	X	X	X	X	X
s	s	X	X	X	X	X	X	X	X	s	s	s	X	X
s	b	X	X	X	X	X	X	X	X	X	X	X	X	X
b	i	X	X	X	X	X	X	X	X	X	X	X	X	X
b	f	X	X	X	X	X	X	X	X	X	X	X	X	X
b	s	X	X	X	X	X	X	X	X	X	X	X	X	X
b	b	X	X	X	X	b	b	b	b	b	b	b	b	b

3.4.2 Direcciones Virtuales

Las direcciones virtuales fueron utilizadas para identificar a las variables durante la fase de compilación, las direcciones son divididas en cuatro categorías globales, locales, temporales yconstantes. Dentro de cada una de estas categorías se subdividen en cuatro categorías, divididas por tipo de dato: enteras, flotantes, strings y booleanas. Existen 1000 espacios disponibles para cada subtipo de las principales 4 categorías.

3.4.3 Funciones especiales

A+ es un lenguaje que contiene las muchas de las características de los lenguajes de programación como lo son operaciones aritméticas, condiciones, ciclos, definición de funciones, uso de arreglos unidimensionales y recursividad. Además de estas funciones incluye una serie de funciones orientadas a la parte gráfica de nuestro lenguaje, esta serie de funciones son un complemento para desarrollar unas bases sólidas de programación, ya que ayudan a entender los estatutos básicos de cualquier lenguaje de forma visual. Las funciones son las siguientes:

- move();

    - Dicha función mueve a el personaje de nuestro lenguaje 50 píxeles (o un cuadro de nuestro grid) hacia la posición donde esté orientado. En caso de estar enfrente del norte, avanza hacia arriba, en caso de estar enfrente del sur avanza hacia abajo. En caso de estar enfrente del este avanza a la izquierda. Y por último en caso de estar enfrente del oeste avanza hacia la derecha. Por default siempre avanza a la derecha.
- turnLeft();
    - Dicha función gira al personaje 90° a la izquierda, además cambia la dirección hacia donde se posiciona el frente del personaje.
- turnRight();
    - Dicha función gira al personaje 90° a la derecha, y cambia la dirección donde se posiciona el frente del personaje
- putBeeper();
    - Función que coloca una imagen en la posición donde esté el personaje al momento de llegar el comando.
- pickBeeper();
    - Función que recoge una imagen de la posición donde se encuentra el personaje al momento de que llegue el comando.
- checkWall();
    - Función que revisa si en la posición del frente del personaje se encuentra una pared, en caso de ser cierto regresa un valor verdadero, y falso, en caso de no haber pared.
- creación de laberintos en la cuadrícula
    - Para crear laberintos en la cuadrícula, es necesario hacer clic en un cuadro, y hacer clic en un segundo cuadro como se quieran unir los muros.

3.4.4 Tipos de datos

Los tipos de dato que se pueden utilizar en el lenguaje son los siguientes:

- int
    - Tipo de dato que acepta números positivos, mayores o igual a cero, que no tengan puntos decimales, utiliza un espacio de memoria de 4 bytes.
- float
    - Tipo de dato que solo acepta números positivos, puede tener un formato decimal o punto flotante, utiliza un espacio de memoria de 4 bytes.
- string
    - Tipo de dato que inicia y termina con comillas, puede incluir caracteres como números, letras , símbolos y espacios", este tipo tiene un espacio variable en memoria dependiendo de su longitud. Cada carácter ocupa un espacio de 1 byte.
- bool
    - Tipo de dato que puede tener solo dos valores true o false, es utilizado para comparaciones y ciclos ocupa un espacio de 4 bytes
- arreglos
    - Los arreglos son un tipo de estructura del lenguaje, se pueden construir arreglos de cualquier tipo de los datos ya mencionados, pero dentro de un arreglo no se pueden guardar diferentes tipos de datos.

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

Durante la compilación la memoria fue dividida en 4 secciones, global, temporal, local y constantes. Cada una de estas categorías a la vez se dividió en 4: int, float, string y bool. A continuación, se muestran los rangos de direcciones utilizados en memoria virtual.

Tipo	Rango de direcciones

Global int	5000 - 5999
Global float	6000 - 6999
Global string	7000 - 7999
Global bool	8000 - 8999
local int	10000 - 10999
local float	11000 - 11999
local string	12000 - 12999
local bool	13000 - 13999
temporal int	20000 - 20999
temporal float	21000 - 21999
temporal string	22000 - 22999
temporal bool	23000 - 23999
constante int	30000 - 30999
constante float	31000 - 31999
constante string	32000 - 32999
constante bool	33000 - 33999

Las direcciones fueron manejadas con una serie de contadores. Para variables globales y locales, al momento de ser declaradas se checaba su tipo de variable y en base a esto se le asignaba la siguiente dirección disponible. La dirección era almacenada junto con el resto de la información de la variable en la tabla de variables. Con respecto a las variables locales, al terminar la función, se mandaba llamar a un método llamado: destroy_Vars, el cual borrara estas variables del registro de la tabla de variables y reinicia los contadores de las variables locales.

Para las constantes, al momento de detectar una constante en la sintaxis, que no haya sido previamente declarada, se guardaba el valor de esta nueva constante, junto con el contador actual de constantes, se agregan estos registros a la lista y se incrementa el contador de constantes. Los temporales se manejaron al mismo tiempo que eran generados en los cuádruplos; al llegar un nuevo temporal se incrementa el contador de temporales de acuerdo al tipo de dato producido por los cuádruplos.

La tabla de variables se manejó con una clase “tablaVar” que contenía de atributos : nombre, scope, tipo de dato y dirección virtual. Al llegar una variable nueva se guardaban los datos en un objeto de tipo tablaVar y se agregaban a un diccionario. Se decidió utilizar un diccionario puesto que esta estructura resulta bastante eficiente en cuanto a memoria en Python, además de que contiene métodos que facilitan su manejo. Antes de agregarse se verificaba que no existiera previamente una variable con el mismo nombre.

La tabla de procedimientos se manejó con la clase TablaFunciones la cual contiene los siguientes atributos: nombre, tipo de dato a retornar, tipos de parámetros, direcciones de parámetros, inicio del cuádruplo, y dirección de la función. La mayoría de estos campos son necesarios durante la ejecución en máquina virtual para poder hacer los cuádruplos en el orden adecuado. Al recibir en sintaxis una función, se generaba un objeto con estos datos y se almacenaba en un diccionario. En caso de que la función no retornase valor, el atributo de dirección de la función quedaba vacío, al igual que los atributos de tipo y dirección de variables en caso de que no se recibieran parámetros. El atributo de tipo de parámetros es utilizado durante la llamada de una función para

verificar que la función que manda llamar a la almacenada en la tabla de Funciones tenga la cantidad y los tipos correctos de parámetros.

Los cuádruplos fueron utilizados con una clase llamada cuádruplos cuyos atributos, eran operador, operando 1, operando 2, y resultado. Este objeto se almacena en un arreglo de Cuádruplos, se decidió usar un arreglo como estructura ya que hay algunos cuádruplos con campos que necesitan llenarse posteriormente, como el caso del goto y goto. Los contadores eran almacenados en una pila de saltos. En caso de poder completar el cuádruplo se utilizaba el índice de ese número y uno de los métodos setter de la clase cuádruplo para completar el valor.

El cubo de datos fue manejado mediante un cubo de 4x4x13. Los valores del cubo fueron accedidos mientras se realizaban los cuádruplos, para verificar que los tipos de dato dados pudiesen hacer la operación. Este tipo de dato resultante era almacenado en una pila de tipos que sacaba sus valores junto con la pila de operando.

En este ejemplo se pueden observar las estructuras más importantes creadas durante compilación:

```
int a,b,res,itera;
def int fibo(int num1, int num2, int max):
        int cont,num3;
        cont = 1;
        num3 = 0;
        w hile (cont <= max):
                num1 = num2;
                num2 = num3;
                num3 = num1 + num2;
                cont = cont + 1;
        end_w hile
        return num3;
end_def
main:

a = 1;
b = 1;
itera = 10;
res = fibo(a,b,itera);
print(res);
```

```
variable número 0
itera
int
5000
-----------------
variable número 1
res
int
5001
-----------------
variable número 2
b
int
5002
-----------------
variable número 3
a
int
5003
-----------------
variable número 4
fibo
int
5004
-----------------


función número 0
fibo
int
[0, 0, 0]
[10000, 10001, 10002]
1
5004
-----------------


Cuádruplo num 0
16 14 nul nul
Cuádruplo núm 1
6 30000 nul 10004
Cuádruplo núm 2
6 30001 nul 10003
Cuádruplo núm 3
11 10004 10000 23000
Cuádruplo núm 4
17 23000 nul 12
Cuádruplo núm 5
6 10001 nul 10002
Cuádruplo núm 6
6 10003 nul 10001
Cuádruplo núm 7
0 10002 10001 20000
Cuádruplo núm 8
6 20000 nul 10003
Cuádruplo núm 9
0 10004 30000 20001
Cuádruplo núm 10
6 20001 nul 10004
Cuádruplo núm 11
16 3 nul nul
Cuádruplo núm 12
```

```
23 10003 nul nul
Cuádruplo núm 13
22 nul nul nul
Cuádruplo núm 14
6 30000 nul 5003
Cuádruplo núm 15
6 30000 nul 5002
Cuádruplo núm 16
6 30002 nul 5000
Cuádruplo núm 17
18 fibo nul nul
Cuádruplo núm 18
20 5000 nul 10000
Cuádruplo núm 19
20 5002 nul 10001
Cuádruplo núm 20
20 5003 nul 10002
Cuádruplo núm 21
19 fibo nul nul
Cuádruplo núm 22
6 5004 nul 20000
Cuádruplo núm 23
6 20000 nul 5001
Cuádruplo núm 24
13 5001 nul nul
Cuádruplo núm 25
15 nul nul nul
```

4. Descripción de la máquina virtual

4.1 Equipo de cómputo, lenguaje y utilerías especiales usadas

La máquina virtual se realizó en Python, no utilizamos ninguna librería especial

4.2 Descripción detallada del proceso de Administración de Memoria en ejecución

Nuestra memoria en ejecución consiste en una serie de diccionarios (5) :

diccionarioMemGlobal	Diccionario para variables globales
diccionarioMemTemporalGl	Diccionario para variables temporales globales
diccionarioMemConstante	Diccionario para constantes
diccionarioMemLocal	Arreglo de diccionarios para variables locales
diccionarioMemTemporalLl	Arreglo de diccionarios para variables temporales locales

Decidimos utilizar esta arquitectura ya que los diccionarios facilitaban las operaciones de memoria, se utilizaba la dirección virtual como key del diccionario y dentro de la dirección se almacena el resultado. La información de la tabla de constantes pasa directamente al diccionarioMemConstante donde se almacenan esos valores y sirven para dar valores a los otros diccionarios. Cuando se accede a memoria local, se agrega un nuevo registro a la lista de diccionarios, con un diccionario nuevo, se trabaja un contador, para que siempre se trabajen las memorias con índice menor.

Cada vez que llega un era se agrega ese nuevo registro y cuando llega un cuádruplo de ret, borra esa memoria activa y regresa el contador al último contador.
Al hacer esto se puede asegurar que siempre existan únicamente dos memorias activas la global y la local en ejecución. Gracias a que se utilizan las direcciones virtuales como llave se puede identificar a qué diccionario se va a asignar la memoria.

La máquina virtual es un switch que recibe cuádruplo por cuádruplo y dependiendo al código de operación de cada cuádruplo realiza una acción correspondiente. Los métodos que trabajan con memoria tienen su base en

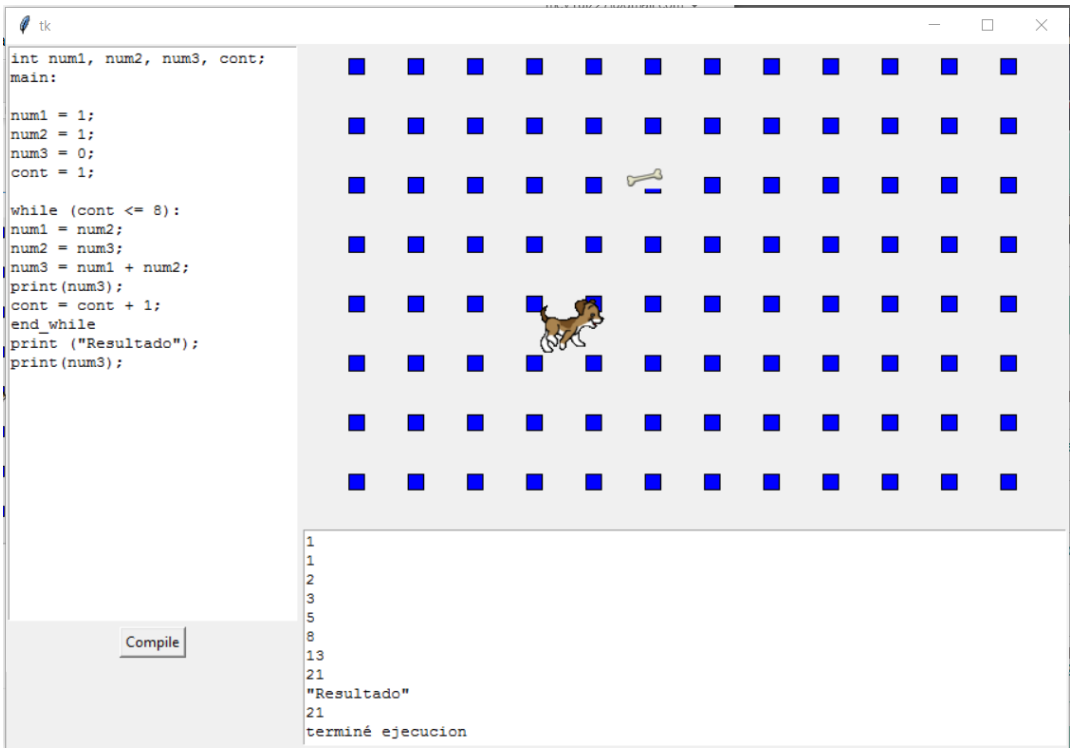
dos funciones que administran toda la memoria en máquina virtual: getValor(dirVirtual) y setValor(dirVirtual,valor). El primero de ellos regresa un valor dado una dirección de memoria, y el segundo almacena un valor usando la dirección virtual como llave.

Se asignaron las siguientes direcciones

Globales	Variables globales int	vgi	5000
	Variables globales float	vgf	6000
	Variables globales string	vgs	7000
	Variables globales bool	vgb	8000
Locales	Variables locales int	vli	10000
	Variables locales float	vlf	11000
	Variables locales string	vlb	12000
	Variables locales bool	vls	13000
Temporales	tgi	20000	
	Temporales float	tgf	21000
	Temporales string	tgs	22000
	Temporales bool	tgb	23000
Constantes	Constantes int	ctei	30000
	Constantes float	ctef	31000
	Constantes string	ctes	32000
	Constantes bool	cteb	33000

5. Pruebas del funcionamiento del lenguaje

A continuación, se incluyen unas pruebas realizadas sobre el lenguaje A+ con sus respectivos cuádruplos:
Prueba de Fibonacci Iterativo

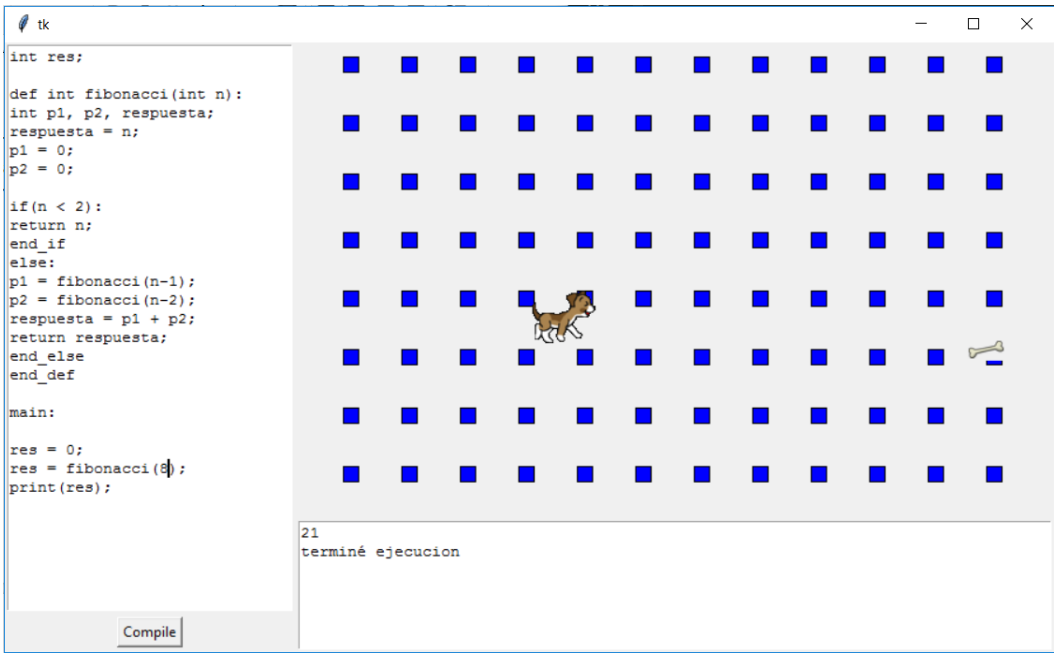


variable número 0
cont
int
5000
---
variable numero 1
num3
int
5001
---
variable numero 2
num2
int
5002
---
variable numero 3
num1
int
5003

Cuádruplo núm 0
16 1 nul nul
Cuádruplo núm 1
6 30000 nul 5003
Cuádruplo núm 2
6 30000 nul 5002
Cuádruplo núm 3
6 30001 nul 5001
Cuádruplo núm 4
6 30000 nul 5000
Cuádruplo núm 5
11 5000 30002 23000
Cuádruplo núm 6
17 23000 nul 15
Cuádruplo núm 7
6 5002 nul 5003

Cuádruplo núm 8
6 5001 nul 5002
Cuadruplo num 9
0 5003 5002 20000
Cuadruplo num 10
6 20000 nul 5001
Cuadruplo num 11
13 5001 nul nul
Cuadruplo num 12
0 5000 30000 20001
Cuadruplo num 13
6 20001 nul 5000
Cuadruplo num 14
16 5 nul nul
Cuadruplo num 15
13 32000 nul nul
Cuadruplo num 16
13 5001 nul nul
Cuadruplo num 17
15 nul nul nul

Prueba Fibonacci Recursivo



variable número 0
res
int
5000
---
variable número 1
fibonacci
int
5001
---

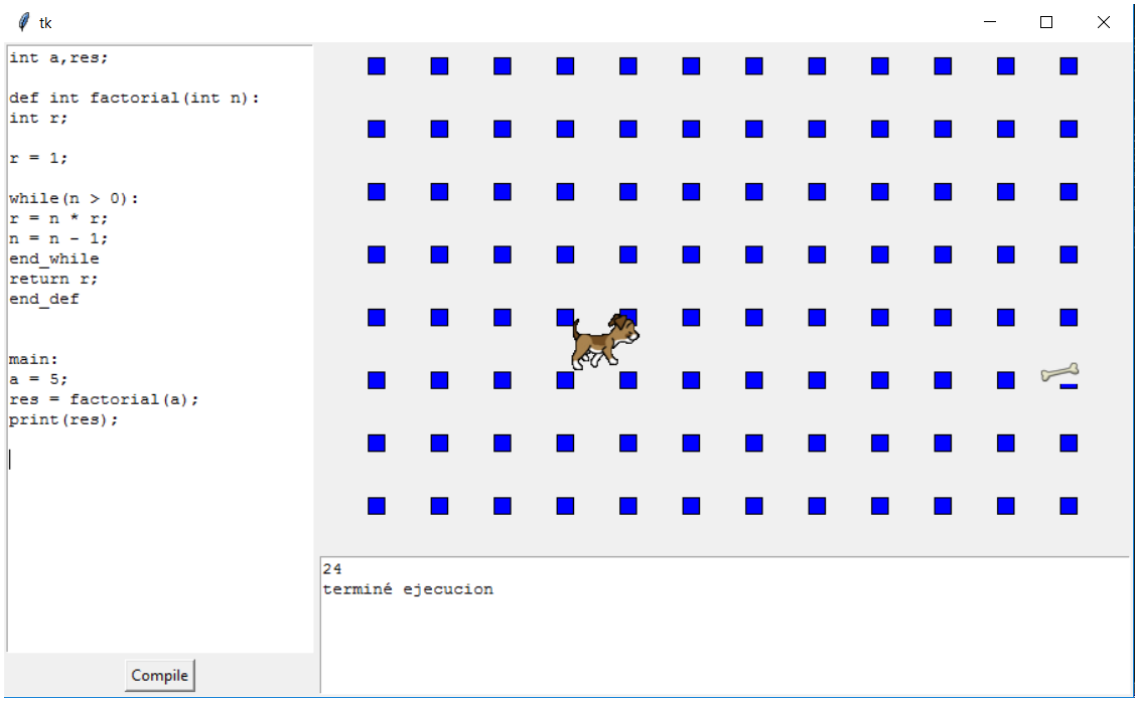
función número 0
fibonacci
int

[10000]
1

Cuádruplo num 0
16 24 nul nul
Cuádruplo num 1
6 10000 nul 10001
Cuádruplo num 2
6 30000 nul 10003
Cuádruplo num 3
6 30000 nul 10002
Cuádruplo num 4
4 10000 30001 23000
Cuádruplo num 5
17 23000 nul 8
Cuádruplo num 6
23 10000 nul nul
Cuádruplo num 7
16 23 nul nul
Cuádruplo num 8
18 fibonacci nul nul
Cuádruplo num 9
1 10000 30002 20000
Cuádruplo num 10
20 20000 nul 10000
Cuádruplo num 11
19 fibonacci nul nul
Cuádruplo num 12
6 5001 nul 20001
Cuádruplo num 13
6 20001 nul 10003
Cuádruplo num 14
18 fibonacci nul nul
Cuádruplo num 15
1 10000 30001 20002
Cuádruplo num 16
20 20002 nul 10000
Cuádruplo num 17
19 fibonacci nul nul
Cuádruplo num 18
6 5001 nul 20003
Cuádruplo num 19
6 20003 nul 10002
Cuádruplo num 20
0 10003 10002 20004
Cuádruplo num 21
6 20004 nul 10001
Cuádruplo num 22
23 10001 nul nul
Cuádruplo num 23
22 nul nul nul
Cuádruplo num 24
6 30000 nul 5000
Cuádruplo num 25
18 fibonacci nul nul
Cuádruplo num 26
20 30003 nul 10000
Cuádruplo num 27
19 fibonacci nul nul
Cuádruplo num 28
6 5001 nul 20000
Cuádruplo num 29
6 20000 nul 5000

Cuádruplo num 30
13 5000 nul nul
Cuádruplo num 31
15 nul nul nul

Prueba Factorial Iterativo



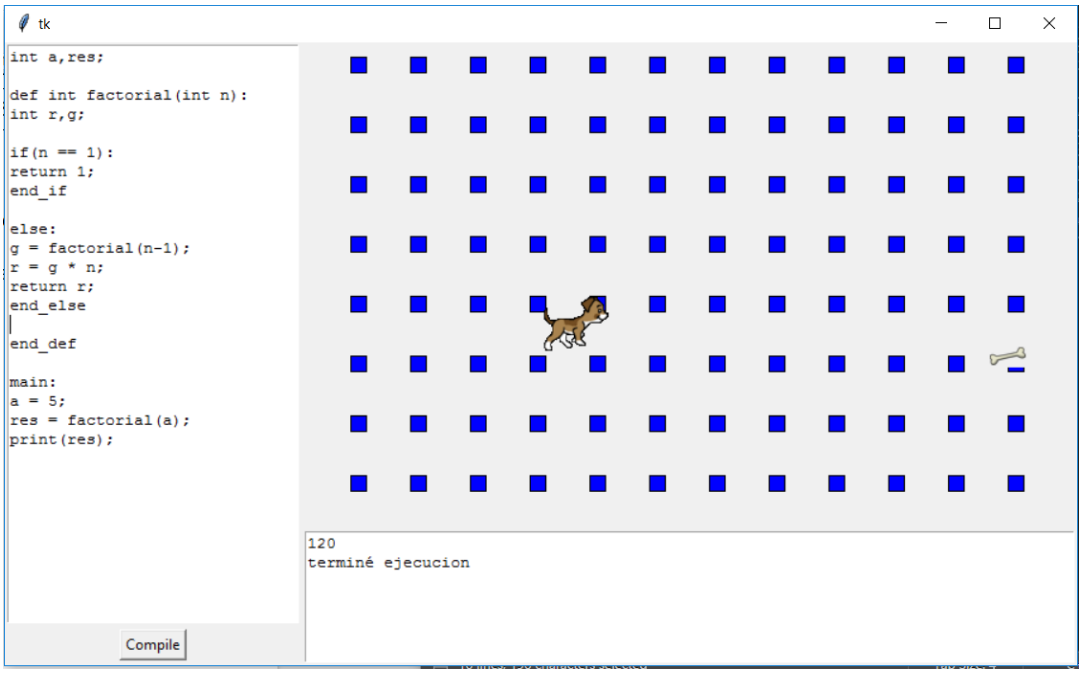
variable número 0
res
int
5000
---
variable numero 1
a
int
5001
---
variable número 2
factorial
int
5002
---

funcion numero0
factorial
int
[10000]
1

Cuadruplo num 0
16 11 nul nul
Cuadruplo num 1
6 30000 nul 10001
Cuadruplo num 2

5 10000 30001 23000
Cuadruplo num 3
17 23000 nul 9
Cuadruplo num 4
2 10000 10001 20000
Cuadruplo num 5
6 20000 nul 10001
Cuadruplo num 6
1 10000 30000 20001
Cuadruplo num 7
6 20001 nul 10000
Cuadruplo num 8
16 2 nul nul
Cuadruplo num 9
23 10001 nul nul
Cuadruplo num 10
22 nul nul nul
Cuadruplo num 11
6 30002 nul 5001
Cuadruplo num 12
18 factorial nul nul
Cuadruplo num 13
20 5001 nul 10000
Cuadruplo num 14
19 factorial nul nul
Cuadruplo num 15
6 5002 nul 20000
Cuadruplo num 16
6 20000 nul 5000
Cuadruplo num 17
13 5000 nul nul
Cuadruplo num 18
15 nul nul nul

Prueba Factorial Recursivo

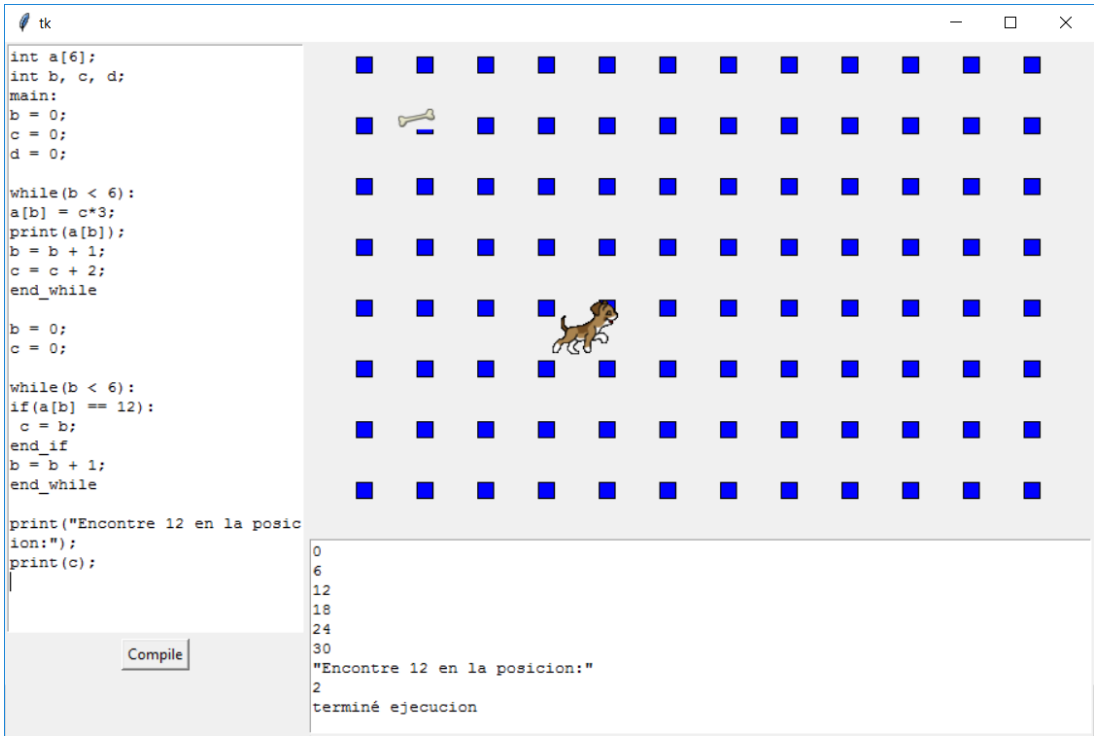


variable numero 0
res
int
5000

```
----------------
variable numero 1
a
int
5001
----------------
variable numero 2
factorial
int
5002
----------------


funcion numero0
factorial
int
[10000]
1
----------------

Cuadruplo num 0
16      15      nul     nul
Cuadruplo num 1
8       10000   30000   23000
Cuadruplo num 2
17      23000   nul     5
Cuadruplo num 3
23      30000   nul     nul
Cuadruplo num 4
16      14      nul     nul
Cuadruplo num 5
18      factorial nul   nul
Cuadruplo num 6
1       10000   30000   20000
Cuadruplo num 7
20      20000   nul     10000
Cuadruplo num 8
19      factorial nul   nul
Cuadruplo num 9
6       5002    nul     20001
Cuadruplo num 10
6       20001   nul     10001
Cuadruplo num 11
2       10001   10000   20002
Cuadruplo num 12
6       20002   nul     10002
Cuadruplo num 13
23      10002   nul     nul
Cuadruplo num 14
22      nul     nul     nul
Cuadruplo num 15
6       30001   nul     5001
Cuadruplo num 16
18      factorial nul   nul
Cuadruplo num 17
20      5001    nul     10000
Cuadruplo num 18
19      factorial nul   nul
Cuadruplo num 19
6       5002    nul     20000
Cuadruplo num 20
6       20000   nul     5000
Cuadruplo num 21
13      5000    nul     nul
```

Cuadruplo num 22
15 nul nul nul

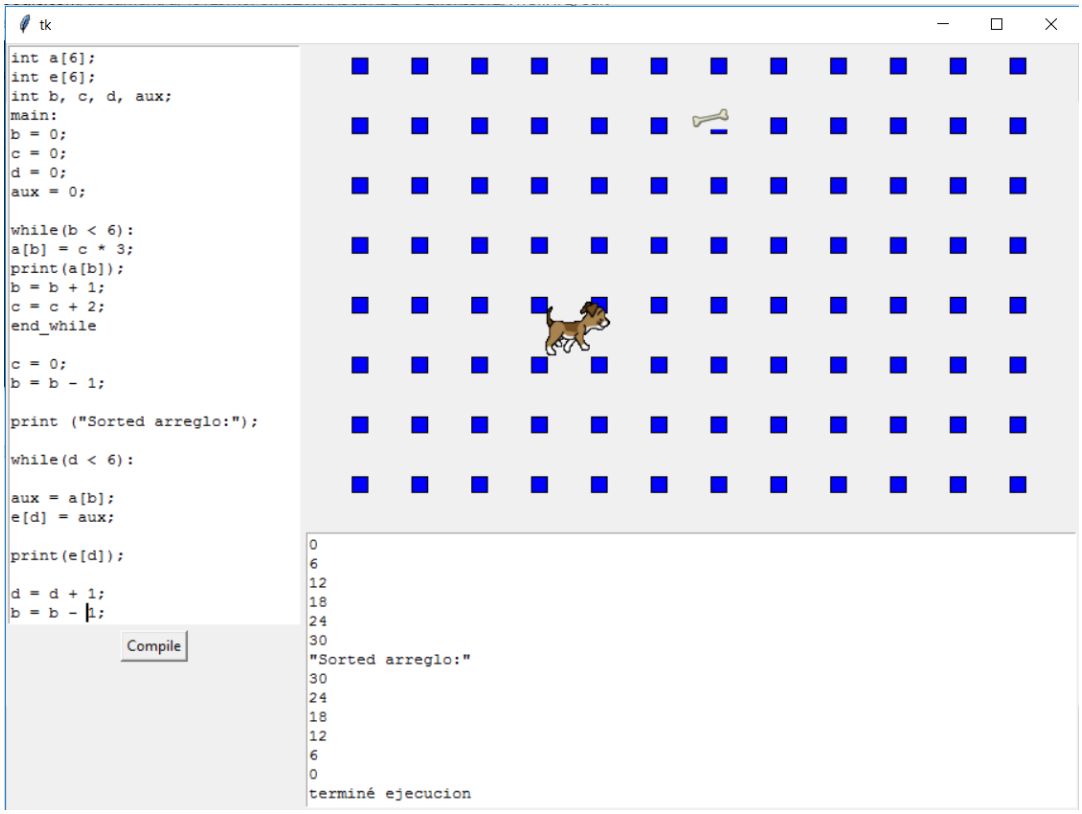
Find en arreglos



variable numero 0
a
int
5000
---
variable numero 1
d
int
5006
---
variable numero 2
c
int
5007
---
variable numero 3
b
int
5008

Cuadruplo num 0
16 1 nul nul
Cuadruplo num 1
6 30000 nul 5008
Cuadruplo num 2
6 30000 nul 5007
Cuadruplo num 3
6 30000 nul 5006
Cuadruplo num 4
4 5008 30001 23000
Cuadruplo num 5
17 23000 nul 18

Cuadruplo num 6
21 5008 0 5
Cuadruplo num 7
30 5008 5000 -20000
Cuadruplo num 8
2 5007 30002 20001
Cuadruplo num 9
6 20001 nul -20000
Cuadruplo num 10
21 5008 0 5
Cuadruplo num 11
30 5008 5000 -20002
Cuadruplo num 12
13 -20002 nul nul
Cuadruplo num 13
0 5008 30003 20003
Cuadruplo num 14
6 20003 nul 5008
Cuadruplo num 15
0 5007 30004 20004
Cuadruplo num 16
6 20004 nul 5007
Cuadruplo num 17
16 4 nul nul
Cuadruplo num 18
6 30000 nul 5008
Cuadruplo num 19
6 30000 nul 5007
Cuadruplo num 20
4 5008 30001 23001
Cuadruplo num 21
17 23001 nul 31
Cuadruplo num 22
21 5008 0 5
Cuadruplo num 23
30 5008 5000 -20005
Cuadruplo num 24
8 -20005 30005 23002
Cuadruplo num 25
17 23002 nul 28
Cuadruplo num 26
6 5008 nul 5007
Cuadruplo num 27
16 28 nul nul
Cuadruplo num 28
0 5008 30003 20006
Cuadruplo num 29
6 20006 nul 5008
Cuadruplo num 30
16 20 nul nul
Cuadruplo num 31
13 32000 nul nul
Cuadruplo num 32
13 5007 nul nul
Cuadruplo num 33
15 nul nul nul



variable numero 0
a
int
5000
---
variable numero 1
e
int
5006
---
variable numero 2
aux
int
5012
---
variable numero 3
d
int
5013
---
variable numero 4
c
int
5014
---
variable numero 5
b
int
5015

Cuadruplo num 0
16 1 nul nul
Cuadruplo num 1
6 30000 nul 5015
Cuadruplo num 2
6 30000 nul 5014
Cuadruplo num 3
6 30000 nul 5013
Cuadruplo num 4
6 30000 nul 5012
Cuadruplo num 5
4 5015 30001 23000
Cuadruplo num 6
17 23000 nul 19
Cuadruplo num 7
21 5015 0 5
Cuadruplo num 8
30 5015 5000 -20000
Cuadruplo num 9
2 5014 30002 20001
Cuadruplo num 10
6 20001 nul -20000
Cuadruplo num 11
21 5015 0 5
Cuadruplo num 12
30 5015 5000 -20002
Cuadruplo num 13
13 -20002 nul nul
Cuadruplo num 14
0 5015 30003 20003
Cuadruplo num 15
6 20003 nul 5015
Cuadruplo num 16
0 5014 30004 20004
Cuadruplo num 17
6 20004 nul 5014
Cuadruplo num 18
16 5 nul nul
Cuadruplo num 19
6 30000 nul 5014
Cuadruplo num 20
1 5015 30003 20005
Cuadruplo num 21
6 20005 nul 5015
Cuadruplo num 22
13 32000 nul nul
Cuadruplo num 23
4 5013 30001 23001
Cuadruplo num 24
17 23001 nul 39
Cuadruplo num 25
21 5015 0 5
Cuadruplo num 26
30 5015 5000 -20006
Cuadruplo num 27
6 -20006 nul 5012
Cuadruplo num 28
21 5013 0 5
Cuadruplo num 29
30 5013 5006 -20007
Cuadruplo num 30
6 5012 nul -20007
Cuadruplo num 31
21 5013 0 5
Cuadruplo num 32
30 5013 5006 -20008

```
Cuadruplo num 33
13      -20008  nul     nul
Cuadruplo num 34
0       5013    30003   20009
Cuadruplo num 35
6       20009   nul     5013
Cuadruplo num 36
1       5015    30003   20010
Cuadruplo num 37
6       20010   nul     5015
Cuadruplo num 38
16      23      nul     nul
Cuadruplo num 39
15      nul     nul     nul
```

6. Listados perfectamente documentados del proyecto:

6.1 Comentarios de cada módulo:

```
#función sintáctica donde se declaran variables
#se recibe el id y el tipo de variable y se guarda en la tabla de variables
def p_declaracion(p)

#función auxiliar para generar el cuádruplo dentro de la función término
#saca de la fila * o / en caso de ser el tope
#verifica los tipos de datos
#genera un cuádruplo
def p_cuaFactor(p)

#función que recibe un ID
#verifica que ese ID exista previamente en la tabla de variables
#en caso de no existir genera excepción
#si existe guarda la dirección de esa variable en la tabla de variables
def p_matchID(p)

#función auxiliar que recibe una expresión para indexar un arreglo
#crea cuádruplo de verifica
#y crea cuádruplo con dirección base + offset
def p_arregloAux(p)

#función sintáctica que se realiza al terminar todas las condiciones, saca de la pila de saltos auxiliar los valores para llenar los goto
def p_condicion(p):

#genera el cuádruplo de gotof después de llegar una condición
def p_cuacondicion(p)

#guarda la función en la tabla de funciones con los atributos como nombre tipo, parámetros, entre otros.
def p_agregaFunc(p)

#función que borra las variables locales de la tabla de variables
#reseta los valores de los contadores
def p_destroyVars(p)

#función que verifica que los parámetros de una función sean consistentes a como se declaró en la tabla de funciones
def p_function_2(p):
#función que dado un valor de dirección virtual regresa su valor en ejecución
#accede al diccionario de variables en el scope dependiendo de la dirección virtual
getValor(op1)

#función que almacena un valor en una dirección en ejecución, dado una dirección virtual
setValor(op1,res)
```

```
#función que expande el registro de memoria
era(funcion)

#función que da el valor de una dirección global a un parámetro local
param(op1, res)

#función que verifica que el valor indexado este en el rango permitido
ver(op1,op2,res)

#función que guarda el valor actual en la dirección global de la función
return(op1)
```

6.2 Módulos Principales:

```
#función para declarar variables
def p_declaracion(p):
  '''
  declaracion : tipo ID declaracion_aux PUNTO_Y_COMA
  '''
  global bscope
  global iContadorDiccionarioVar
  global dV
  global tipoDeclaracion
  global vgi, vli, vgf, vlf, vgs, vls
  #vars locales
  obj = ""
  var = 0

  #si no es la primera variable a guardar checa si no se repite el nombre
  if(iContadorDiccionarioVar != 0):
    for x in range(0,iContadorDiccionarioVar):
      #compara los nombres
      if(p[2] == dV[x].getNombre()):
        raise errorSemantico("Variable ya definida: " + p[2])


  #checa el tipo de variable y su scope y guarda esa dirección de memoria  en var
  if(tipoDeclaracion == "int" and bscope == 0):
    var = vgi
    vgi += 1
  elif(tipoDeclaracion == "int" and bscope == 1):
    var = vli
    vli+=1
  elif(tipoDeclaracion == "float" and bscope == 0):
    var = vgf
    vgf+=1
  elif(tipoDeclaracion == "float" and bscope == 1):
    var = vgf
    vlf+=1
  elif(tipoDeclaracion == "string" and bscope == 0):
    var = vgs
    vgs+=1
  elif(tipoDeclaracion == "string" and bscope == 1):
    var = vls
    vls+=1

  #crea el objeto tablaVar con : nombre, tipo, scope, dirección
  if(bscope == 0):
    obj = tablaVar(p[2],tipoDeclaracion,'global',var,1)
  else:
    obj = tablaVar(p[2],tipoDeclaracion,'local',var,1)

  #en caso de agregarla la guarda en el diccionario
  if(iContadorDiccionarioVar == 0):
          dV = {iContadorDiccionarioVar : obj}
  else:
          dV[iContadorDiccionarioVar] = obj
```

```
  #incrementa contador
  iContadorDiccionarioVar = iContadorDiccionarioVar + 1



#funcion para cuádruplo de factor
def p_cuaFactor(p):
  '''
  cuaFactor : empty
  '''
  #generacion de cuadruplos
  global POper, PilaO, PTipo
  global tgi, tgf
  global resultado
  global dicOperadores
  global iContadorTemporal, iContadorCuadruplos
  var = 0
  #entra si ya entro una multiplicación o división a la pila o suma o resta
  if(len(POper) > 0):
        #pregunta si el tope es multiplicación o división en caso de serlo prosigue
    if(POper[-1] == "*" or POper[-1] == "/"):
      op2 = PTipo.pop()
      op1 = PTipo.pop()


      if(POper[-1] == "*"):
            op = dicOperadores["*"]
      else:
            op = dicOperadores["/"]


      tipo = cubo[op1][op2][op]


      if(tipo == -1):
            raise errorSemantico("uso incorrecto de tipos ")
      else:
            if(tipo == 0):
                  var = tgi
                  tgi+=1
            elif(tipo == 1):
                  var = tgf
                  tgf+=2
            #saca el operador y ambos operando
            operador = POper.pop()
            operadorAux = dicOperadores[operador]
            operando2 = PilaO.pop()
            operando1 = PilaO.pop()
            iContadorTemporal += 1
            PTipo.append(tipo)
            #al arreglo de resultados mete el número de temporal
            resultado.append(var)
            #genera un nuevo cuádruplo
            arregloCuadruplos.append(cuadruplo(operadorAux,operando1,operando2,resultado[iContadorCuadruplos]))
            #mete el temporal
            PilaO.append(var)
            iContadorCuadruplos += 1

#match de id checa que exista y guarda el tipo en la pila de tipo
def p_matchID(p):
  '''
  matchID : ID
  '''
  global dV, dicTipos
  global iContadorDiccionarioVar
  global PilaO, PTipo
```

```
  global nombreIDArr
  varAux = 0
  tipo = ""
  auxTipo = -2
  #Checa si variable está o no declarada
  for x in range(0,iContadorDiccionarioVar):
    if(p[1] != dV[x].getNombre()):
      varAux += 1
    else:
      #cubo semántico tipo de dato correcto
      nombreIDArr = p[1]
      tipo = dV[x].getTipo()
      auxTipo = dicTipos[tipo]
      PTipo.append(auxTipo)
      #Meter a pila operadores paso 1 del algoritmo
      dir = dV[x].getDireccion()
      PilaO.append(dir)
  #No está declarada
  if(varAux == iContadorDiccionarioVar):
    raise errorSemantico("Variable no declarada: " + p[1])


#función que una expresión para indexar el arreglo
def p_arregloAux(p):
        '''
        arregloAux : CORCHETE_IZQ  validaDimensiones  expresion CORCHETE_DER
        '''
        global PilaO,  PTipo,  resultado, dicOperadores
        global arregloCuadruplos
        global nombreIDArrAux
        global iContadorDiccionarioVar,iContadorCuadruplos
        global tgi
        tam = 0
        direccion = 0
        #obtiene el tipo de dato de la expresión del arreglo
        tipo = PTipo.pop();
        #si no es int no se puede
        if(tipo != 0):
                raise errorSemantico("El índice del arreglo solo debe contener expresiones enteras")
        #saca el operando1 que contiene la expresión
        operando1 = PilaO.pop()
        #busca la variable que se llame igual
        for x in range(0,iContadorDiccionarioVar):
                if(nombreIDArrAux == dV[x].getNombre()):
                        #saca el tamaño y la dirección
                        tam = dV[x].getSize()
                        direccion = dV[x].getDireccion()
        #cuádruplo verifica
        #para no perder la cuenta
        resultado.append("nul")
        #cuádruplo verifica simplificado
        #como solo recibe un número
        #ver exp 0 tam-1
        operador = dicOperadores["Ver"]
        arregloCuadruplos.append(cuadruplo(operador,operando1,0,tam-1))
        #suma al contador de cuádruplos
        iContadorCuadruplos+=1
        #segundo cuádruplo donde suma direccion base
        op = dicOperadores["SumVer"]
        #es int la operacion por que es una dirección + una expresion "int"
        PTipo.append(0)
        dirAux = -1*tgi
        #agregas el temporal que es una dirección de memoria temporal int
        PilaO.append(dirAux)
        #para no perder la cuenta
        resultado.append(dirAux)
```

```
            #genera cuádruplo dirección base más offset
            arregloCuadruplos.append(cuadruplo(op,operando1,direccion,dirAux))
            #suma contadores
            tgi+=1
            iContadorCuadruplos+=1



#función para aceptar condiciones s
def p_condicion(p):
  '''
  condicion : imprimeIf PARENTESIS_IZQ comparadores PARENTESIS_DER DOS_PUNTOS cuacondicion1 estatuto_2
imprimeEndIf condicion_3 condicion_4
  '''
  global PSaltosAux
  global iContadorCuadruplos
  global arregloCuadruplos
  #llenas en donde se regresa cuando acabe el else
  #el elif ocupa la pila auxiliar
  while(len(PSaltosAux)>0):
            res = PSaltosAux.pop()
            #rellena el cuádruplo
            arregloCuadruplos[res].setOperando1(iContadorCuadruplos)



#genera el cuadruplo de condición
def p_cuacondicion1(p):
  '''
  cuacondicion1 : empty
  '''
  global operador, operando1, operando2, resultado
  global iContadorCuadruplos
  global PSaltos, PilaO, dicOperadores
  global arregloCuadruplos
  #genera de operador gotof
  operador = dicOperadores["GotoF"]
  #el operando 1 es el temporal o última variable localizada en pila o
  operando1 = PilaO.pop()
  #agrega la posición actual a la pila de saltos
  PSaltos.append(iContadorCuadruplos)
  #el resultado le asigna -2 para estandarizar que está vacío
  resultado.append(-2)
  #genera el cuádruplo
  arregloCuadruplos.append(cuadruplo(operador,operando1,"nul",resultado[iContadorCuadruplos]))
  #suma uno al contador
  iContadorCuadruplos += 1

#cuando acaba el if genera el primer goto porque no sabes si va a venir o no elif o else
def p_imprimeEndIf(p):
  '''
  imprimeEndIf : END_IF
  '''
  global PSaltos,dicOperadores,PSaltosAux
  global arregloCuadruplos
  global iContadorCuadruplos
  global operador, resultado
  global bIf

  #saca el tope de Psaltos , que es el apuntador al "goto"
  res = PSaltos.pop()
  #al cuádruplo ubicado en la posición res le mete contador temporal + 1 porque apunta a la siguiente dirección
  arregloCuadruplos[res].setResultado(iContadorCuadruplos+1)
  PSaltosAux.append(iContadorCuadruplos)
  operador = dicOperadores["Goto"]
```

```
  #almacena el resultado en el arreglo de resultados para no perder la cuenta
  resultado.append("nul")
  #genera el cuádruplo
  arregloCuadruplos.append(cuadruplo(operador,-2,"nul","nul"))
  #sigue la cuenta del contador y resetea la variable boleana
  iContadorCuadruplos+=1
  bIf = 0



#función auxiliar que agrega la función a la tabla de funciones
def p_agregaFunc(p):
        '''
        agregaFunc : empty
        '''
        global bscope
        global listaParamFuncion
        global iContadorDiccionarioFuncion, iContadorDiccionarioVar,iContadorCuadruplos
        global dF,dV
        global tipoDeclaracionFuncion
        global dVM
        global nombreFuncion

        #checa que no exista una función que se llame igual
        for x in range(0,iContadorDiccionarioFuncion):
                varFunc = dF[x]
                if(nombreFuncion == varFunc.getNombre()):
                        raise errorSemantico("Función previamente definida: " + p[1])

        #agrega los parametros de la función
        listaAux = []
        listaDirecciones = []
        listaAux.extend(listaParamFuncion)
        listaDirecciones.extend(listaDireccionesFuncion)
        varAux = tablaFunciones(nombreFuncion,tipoDeclaracionFuncion,listaAux,listaDirecciones, iContadorCuadruplos,
dVM)

        #la agrega al diccionario
        if(iContadorDiccionarioFuncion == 0):
                dF = {iContadorDiccionarioFuncion : varAux}
        else:
                dF[iContadorDiccionarioFuncion] = varAux

        #incrementa el contador
        iContadorDiccionarioFuncion = iContadorDiccionarioFuncion + 1
        #si no es void crea una variable global con el mismo nombre



#destruye variables de lista paramfuncion y del Dic.de vars
def p_destroyVars(p):
  '''
  destroyVars : empty
  '''
  global arregloCuadruplos
  global dV,dicOperadores
  global iContadorInicioLocal, iContadorDiccionarioVar, iContadorTemporal, iContadorCuadruplos
  global resultado
  global vli,vlf,vls,vlb,tgi,tgf,tgs,tgb, dVM
  global bRetorna
  global listaParamFuncion,  listaDireccionesFuncion


  iAux = 0
  iAux = iContadorInicioLocal
```

```
  #borra todas las declaradas desde inicio local-->actual si son locales
  for x in range(iContadorInicioLocal , iContadorDiccionarioVar):
    if(dV[x].getScope() == "local"):
      del dV[x]
  del listaParamFuncion[:]
  del listaDireccionesFuncion[:]

  #para que se resetee bien el diccionario en caso de que haya una var/funcion
  if(bRetorna == 1):
    iContadorDiccionarioVar = iAux + 1
  else:
    iContadorDiccionarioVar = iAux

  #resetea temporal
  iContadorTemporal = 1
  #resetea memoria
  vli= 10000
  vlf = 11000
  vlb = 12000
  vls = 13000
  tgi = 20000
  tgf = 21000
  tgs = 22000
  tgb = 23000
  #genera cuadruplo ret
  resultado.append("nul")
  operador = dicOperadores["Ret"]
  arregloCuadruplos.append(cuadruplo(operador,"nul","nul","nul"))
  iContadorCuadruplos+=1
  #elimina  bRetorna
  bRetorna = 0
  dVM = -2



#declara parámetros
def p_function_2(p):
  '''
  function_2 : tipo ID function_3
  '''
  global bscope
  global iContadorDiccionarioVar
  global dV
  global tipoDeclaracion
  global vli, vlf, vls
  global listaParamFuncion
  #vars locales
  obj = ""
  var = 0

  #si no es la primera variable a guardar checa si no se repite el nombre
  if(iContadorDiccionarioVar != 0):
    for x in range(0,iContadorDiccionarioVar):
      #compara los nombres
      if(p[2] == dV[x].getNombre()):
        raise errorSemantico("Variable ya definida: " + p[2])



  #checa el tipo de variable y su scope y guarda esa dirección de memoria  en var
  if(tipoDeclaracion == "int"):
    var = vli
    vli+=1
  elif(tipoDeclaracion == "float"):
```

```
    var = vgf
    vlf+=1
  elif(tipoDeclaracion == "string"):
    var = vls
    vls+=1

  #se genera el objeto y se agrega a la listaParam
  obj = tablaVar(p[2],tipoDeclaracion,'local',var,1)
  aux = dicTipos[tipoDeclaracion]
  listaParamFuncion.append(aux)
  listaDireccionesFuncion.append(var)

  #en caso de agregarla la guarda en el diccionario
  if(iContadorDiccionarioVar == 0):
        dV = {iContadorDiccionarioVar : obj}
  else:
        dV[iContadorDiccionarioVar] = obj

  #incrementa contador
  iContadorDiccionarioVar = iContadorDiccionarioVar + 1



#cuádruplo gosub + parche guadalupano
def p_go_sub(p):
        '''
        go_sub : empty
        '''
        global arregloCuadruplos, dF,resultado, dicTipos, dicOperadores
        global iContadorCuadruplos, iContadorDiccionarioFuncion, iContadorTemporal
        global funcionActiva
        global PTipo
        global tgi,tgf,tgs,tgb
        tipo = ""
        tipoDic = -2
        var = 0
        dirVM = 0
        #cuádruplo gosub
        resultado.append("nul")
        operador = dicOperadores["Gosub"]
        arregloCuadruplos.append(cuadruplo(operador,funcionActiva,"nul","nul"))
        iContadorCuadruplos+=1
        #parche guadalupano
        #checa el tipo
        for x in range(0,iContadorDiccionarioFuncion):
                if(dF[x].getNombre() == funcionActiva):
                        tipo = dF[x].getTipo();
                        dirVM = dF[x].getDirs()
        #si no es void DEBE haber parche
        if(tipo != "void"):
                #agrega el tipo de función a la pila de tipos
                tipoDic = dicTipos[tipo]
                PTipo.append(tipoDic)
                #direccion del temporal
                if(tipoDic == 0):
                        var = tgi
                        tgi+=1
                elif(tipoDic == 1):
                        var = tgf
                        tgf+=1
                elif(tipoDic == 2):
                        var = tgs
                        tgs+=1
                else:
                        var = tgb
```

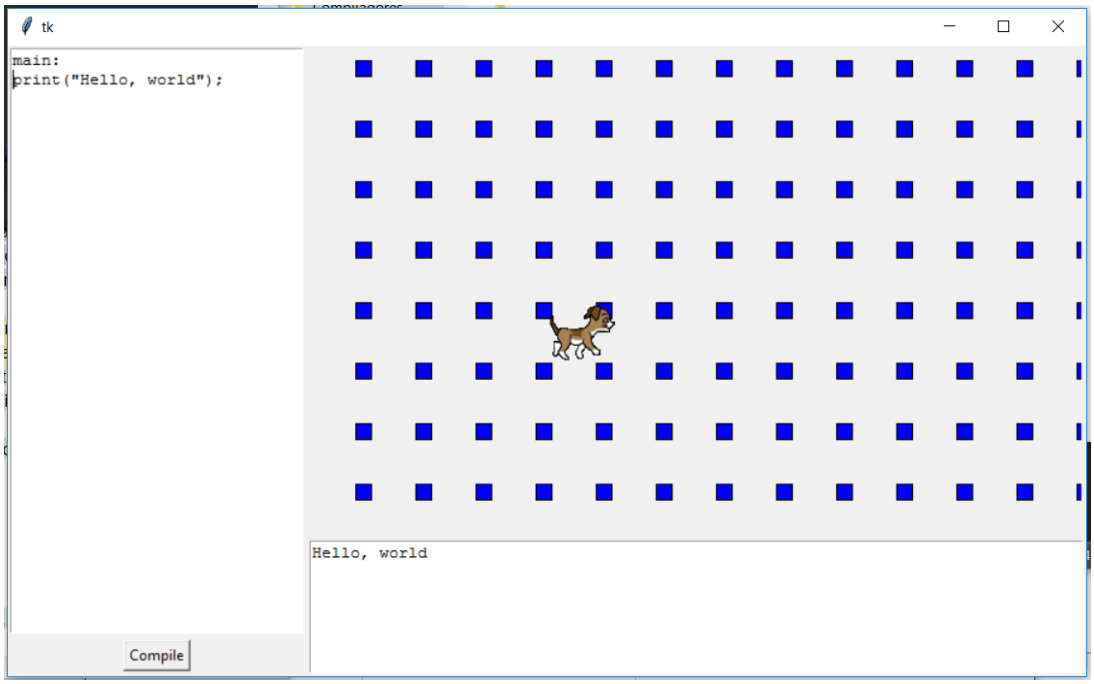
```
                    tgb+=1
                #agrega la función a la pila de operadores
                PilaO.append(var)
                resultado.append(var)
                #genera parche guadalupano
                operador = dicOperadores["="]
                arregloCuadruplos.append(cuadruplo(operador,dirVM,"nul",resultado[iContadorCuadruplos]))
                iContadorTemporal += 1
                iContadorCuadruplos += 1



dicOperadores = {"+" : 0, "-" : 1, "*" : 2, "/" : 3, "<" : 4, ">": 5, "=" : 6,"<>" : 7, "==" : 8, "&": 9, "|": 10, "<=": 11, ">=": 12, "print" : 13
, "read": 14, "end": 15, "Goto": 16, "GotoF": 17, "Era":18, "Gosub":19, "Param":20, "Ver":21, "Ret":22, "Return":23, "move":24 ,
"checkwall":25, "turnRight":26, "turnLeft":27, "pickBeeper":28, "putBeeper":29, "SumVer":30}

dicTipos = {"int" : 0, "float" : 1, "string" : 2, "bool" : 3, "error" : -1}



#función de memoria que retorna el valor almacenado en una direccion
def getValor(memoriaVirtual):
        global diccionarioMemGlobal, diccionarioMemLocal, diccionarioMemConstante
        global diccionarioMemTemporalGl, diccionarioMemTemporalLl
        global bFuncion, FuncionActiva
        #global
        if(memoriaVirtual > 4999 and memoriaVirtual < 9000):
                try:
                        return diccionarioMemGlobal[memoriaVirtual]
                except:
                        raise errorEjecucion("Valor no asignado")
        #local
        elif(memoriaVirtual > 9999 and memoriaVirtual < 14000):
                try:
                        return diccionarioMemLocal[FuncionActiva-1][memoriaVirtual]
                except:
                        raise errorEjecucion("Valor no asignado")
        #temporal
        elif((memoriaVirtual > 19999 and memoriaVirtual < 24000) or (memoriaVirtual > -24000 and memoriaVirtual < -
19999)):
                if(bFuncion == 0):
                        try:
                                return diccionarioMemTemporalGl[memoriaVirtual]
                        except:
                                raise errorEjecucion("Valor no asignado")
                else:
                        try:
                                return diccionarioMemTemporalLl[FuncionActiva-1][memoriaVirtual]
                        except:
                                raise errorEjecucion("Valor no asignado")
        #constantes
        elif(memoriaVirtual > 29999 and memoriaVirtual < 34000):
                try:
                        return diccionarioMemConstante[memoriaVirtual]
                except:
                        raise errorEjecucion("Valor no asignado")

#función de memoria que asigna un valor a una direccion
def setValor(memoriaVirtual, valor):
        global diccionarioMemGlobal, diccionarioMemLocal, diccionarioMemConstante
        global diccionarioMemTemporalGl, diccionarioMemTemporalLl
        global bFuncion, FuncionActiva
        #global
        if(memoriaVirtual > 4999 and memoriaVirtual < 9000):
                diccionarioMemGlobal[memoriaVirtual] = valor
```

```
		#local
		elif(memoriaVirtual > 9999 and memoriaVirtual < 14000):
			diccionarioMemLocal[FuncionActiva-1][memoriaVirtual] = valor
		#temporal
		elif((memoriaVirtual > 19999 and memoriaVirtual < 24000) or (memoriaVirtual > -24000 and memoriaVirtual < -
19999)):
			if(bFuncion == 0):
				diccionarioMemTemporalGl[memoriaVirtual]  = valor
			else:
				diccionarioMemTemporalLl[FuncionActiva-1][memoriaVirtual] = valor
		#cte
		elif(memoriaVirtual > 29999 and memoriaVirtual < 34000):
			diccionarioMemConstante[memoriaVirtual] = valor


#expande registro de memoria
def Era(op1):
		global bFuncion
		global FuncionActiva, iPosArray
		global diccionarioMemTemporalLl,  diccionarioMemLocal
		global arregloFunciones,numParametros,iContadorParametros
		#entra a una funcion
		bFuncion = True
		#crea memoria local y temporal
		diccionarioMemLocal.append({})
		diccionarioMemTemporalLl.append({})
		iContadorParametros = 0
		#obtiene indice de función
		#es para no tener que buscarlo siempre,optimiza
		for x in range(0,len(arregloFunciones)):
			if(op1 == arregloFunciones[x].getNombre()):
				iPosArray = x
				numParametros  = len(arregloFunciones[x].getDirecciones())

#va a función
def Gosub(op1):
		global InstruccionActual, iSaveInstruccionActual, iPosArray,FuncionActiva
		#agrega la instrucción actual a la pila
		iSaveInstruccionActual.append(InstruccionActual + 1)
		#nueva instrucción actual
		InstruccionActual = arregloFunciones[iPosArray].getStart() - 1

#parámetros recibidos
def Param(op1,result):
		global FuncionActiva
		global iContadorParametros,numParametros
		#obtiene valor de op1
		iContadorParametros += 1
		valor = getValor(op1)
		#accede a siguiente memoria
		FuncionActiva+=1
		#guarda el valor en la memoria recién creada
		setValor(result,valor)
		if(iContadorParametros != numParametros):
			FuncionActiva -= 1



#verifica que el índice de un arreglo esté en el rango
def Ver(op1,op2,result):
		global i
		var = True
		#verifica que esté en el rango
		valor1 = getValor(op1)
		if(valor1 >= op2):
			if(valor1 <= result):
```

```
                        var = True
                else:
                        var = False
        else:
                var = False
        #levanta una excepción
        if(var == False):
                raise errorEjecucion("Índice no válido: " + str(valor1))
                i = False

#guarda la dirección base en el temporal
def SumVer(op1,op2,result):
        valor1 = getValor(op1)
        newKey = valor1+op2
        setValor(result,newKey)

#termina función
def Ret():
        global InstruccionActual, iSaveInstruccionActual
        global diccionarioMemLocal, diccionarioMemTemporalLl
        global FuncionActiva,bFuncion
        #saca la último contador a la instrucción local
        sav = iSaveInstruccionActual.pop()
        InstruccionActual = sav - 1
        #elimina el era
        diccionarioMemLocal.pop(FuncionActiva-1)
        diccionarioMemTemporalLl.pop(FuncionActiva-1)
        #termina función
        bFuncion = False
        #resetear contadores
        FuncionActiva -= 1
        iPosArray = -2

#regresa un valor
def Return(op1):
        global InstruccionActual
        #almacena el valor en la dirección de la función
        dvm = arregloFunciones[iPosArray].getDirs()
        valor = getValor(op1)
        setValor(dvm,valor)
```

Manual de Usuario

¡Bienvenido al lenguaje A+!



Nuestro lenguaje fue hecho pensando en personas que deseen entrar al mundo de la programación. Cuenta con una sintaxis fácil de entender y puedes realizar varios cambios al personaje como hacerlo girar, moverse y poner marcas por donde pase, para más información sobre esto pasa a la sección “Comandos Especiales”

Instalación

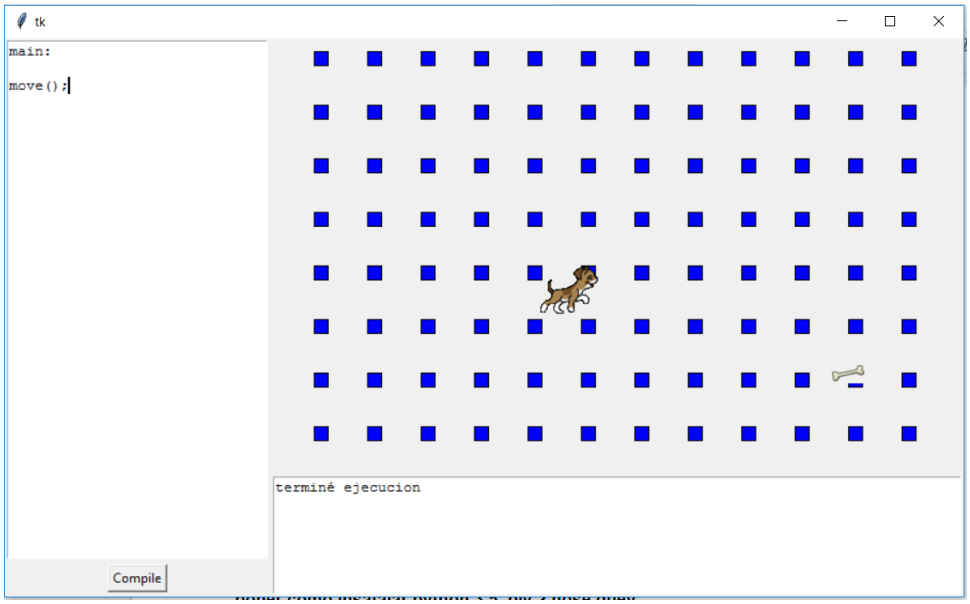
A+ utiliza Python 3, este se puede descargar para Windows, Mac y Linux en la siguiente página: https://www.python.org/downloads/
Los archivos necesarios para utilizar PLY (Python Lex-Yacc) ya vienen incluidos en el proyecto. Para más información puede visitar: http://www.dabeaz.com/ply/

Iniciar A+

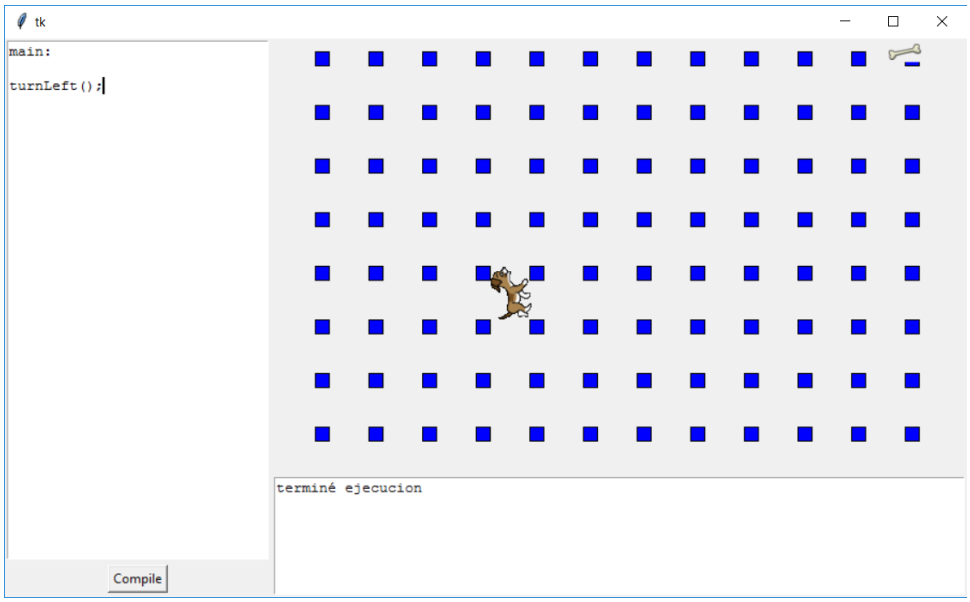
Para iniciar nuestra aplicación es necesario entrar a la aplicación “Símbolo del sistema”, en inglés, Command Prompt. Ahi hacer cd “directorio” donde directorio es el directorio donde guardó la aplicación. A continuación, se da enter e ingresa la siguiente instrucción “python interface.py”. Enseguida la interfaz aparecerá.

Comandos Especiales

move()
Mueve al personaje una casilla al frente de donde está observando. En este ejemplo dio unos pasos a la derecha.

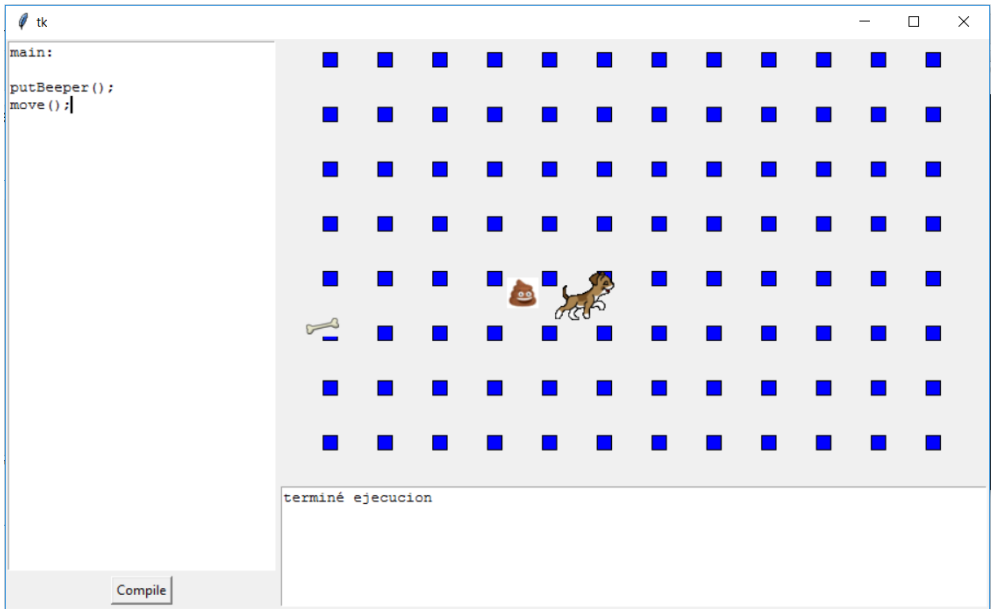


turnLeft() y turnRight()
TurnLeft() da un giro de 90° a la izquierda, mientras que turnRight() da un giro de 90° a la derecha.

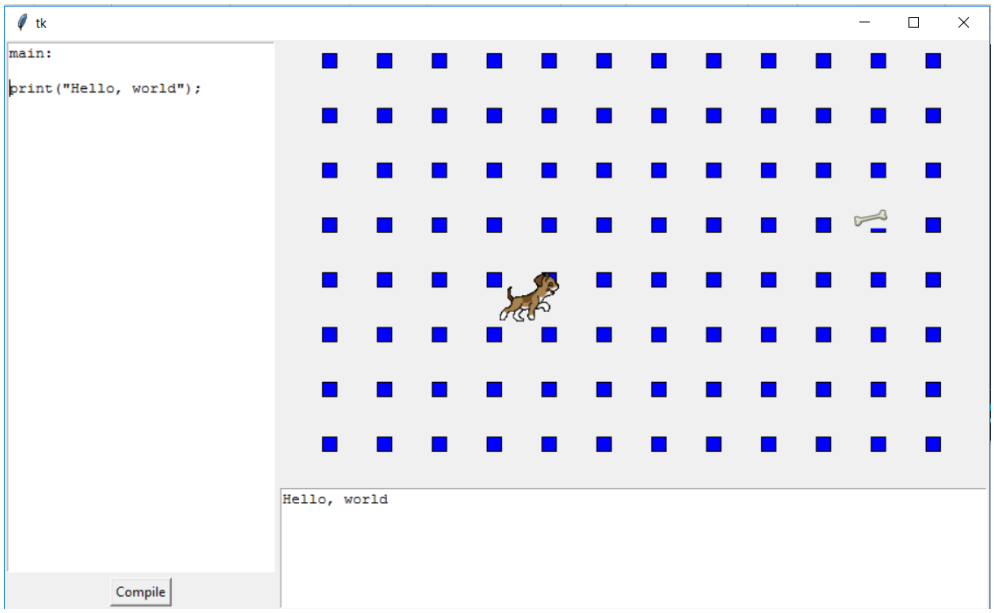


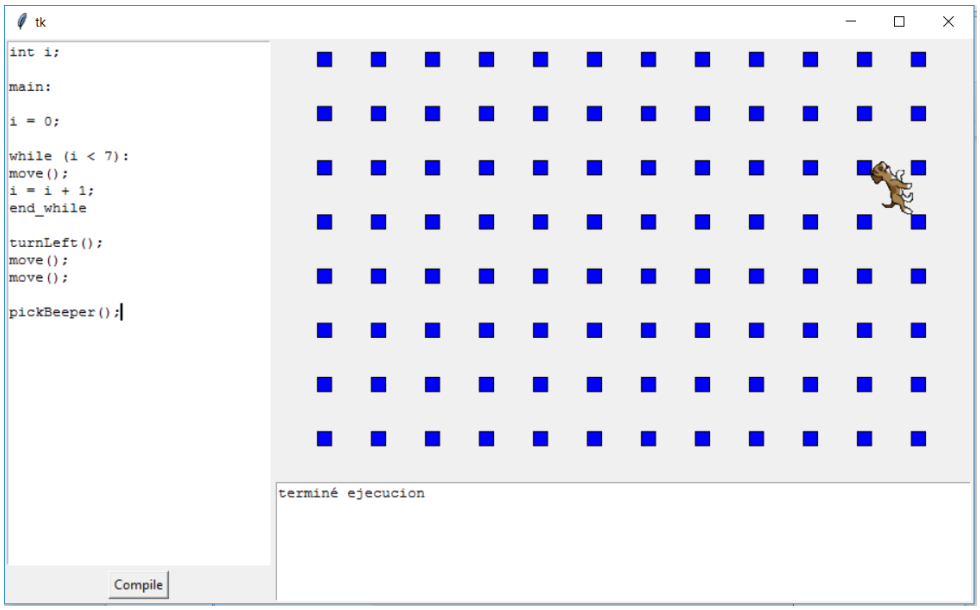
putBeeper()

PutBeeper() hace que el personaje deje una marca en donde quiera que esté posicionado.



pickBeeper()
Mueve al personaje utilizando código para llegar a la casilla donde se posiciona el hueso y tómalo usando pickBeeper().





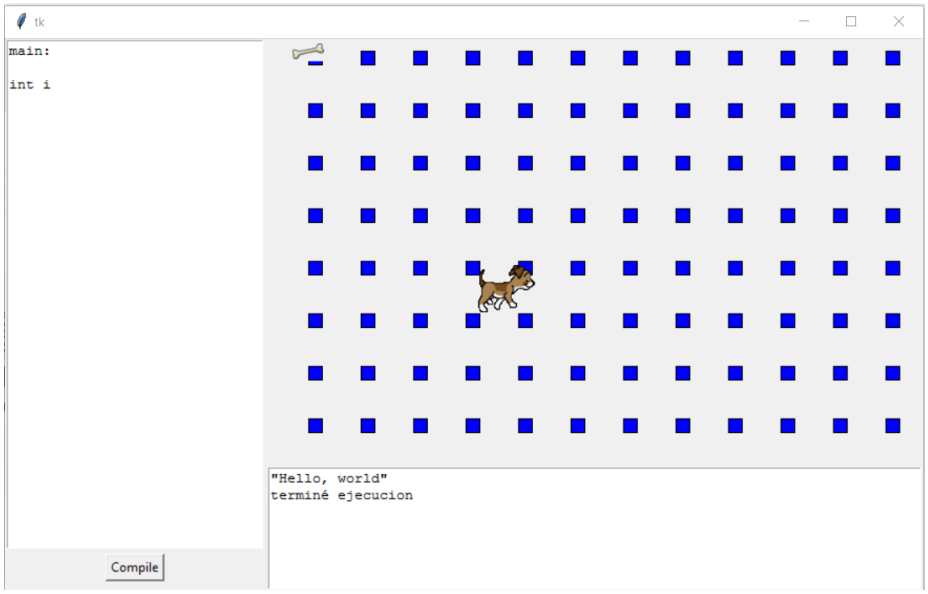
Estructura de Código

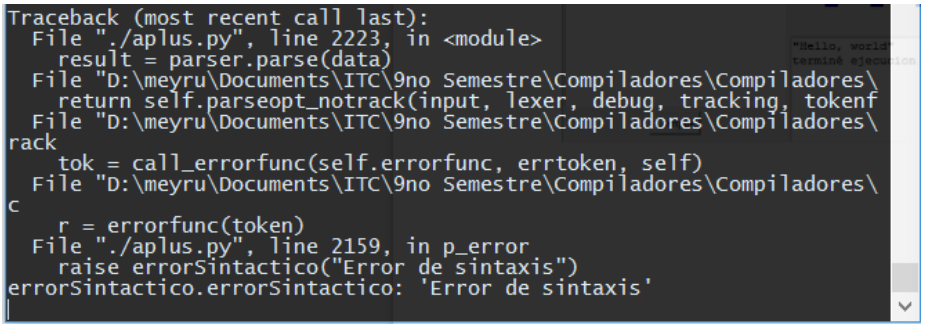
Los programas en A+ tienen una estructura muy similar a Python. Algunas diferencias son:

- Las variables y funciones se deben inicializar antes de describir el “main”
- Las funciones def, while e if deben terminar con end_*nombre de la función*, por ejemplo def terminaría con end_def.
- Cada instrucción termina con un punto y coma.

Errores

Los errores del programa se muestran en la consola donde se corre el programa.




Traceback (most recent call last):
File "./aplus.py", line 2223, in <module>
result = parser.parse(data)
File "D:\meyru\Documents\ITC\9no Semestre\Compiladores\Compiladores\
return self.parseopt_notrack(input, lexer, debug, tracking, tokenf
File "D:\meyru\Documents\ITC\9no Semestre\Compiladores\Compiladores\
rack
tok = call_errorfunc(self.errorfunc, errtoken, self)
File "D:\meyru\Documents\ITC\9no Semestre\Compiladores\Compiladores\
c
r = errorfunc(token)
File "./aplus.py", line 2159, in p_error
raise errorSintactico("Error de sintaxis")
errorSintactico.errorSintactico: 'Error de sintaxis'