MeMySelf

Naomi Macias Honti A01282098

FIRMA

Fecha de entrega

Contents

[Descripción del Proyecto 5](#_Toc56955092)

[Propósito y Alcance del Proyecto 5](#_Toc56955093)

[Análisis de Requerimientos y Casos de Uso Generales 5](#_Toc56955094)

[Descripción de los principales Test Cases 6](#_Toc56955095)

[Test Cases que no dan error 6](#_Toc56955096)

[Test Cases que dan error 10](#_Toc56955097)

[Descripción del proceso general seguido para el desarrollo del proyecto 13](#_Toc56955098)

[Bitácoras 13](#_Toc56955099)

[Reflexión 15](#_Toc56955100)

[Descripción del lenguaje 16](#_Toc56955101)

[Nombre del lenguaje 16](#_Toc56955102)

[Descripción genérica de las principales características del lenguaje 16](#_Toc56955103)

[Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución 16](#_Toc56955104)

[Error en Compilación 16](#_Toc56955105)

[Error en Ejecución 17](#_Toc56955106)

[Descripción del compilador 18](#_Toc56955107)

[Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto 18](#_Toc56955108)

[Descripción del Análisis de Léxico 18](#_Toc56955109)

[Tokens 18](#_Toc56955110)

[Palabras Reservadas 18](#_Toc56955111)

[Descripción del Análisis de Sintaxis 19](#_Toc56955112)

[Descripción de Generación de Código Intermedio y Análisis Semántico 21](#_Toc56955113)

[Tabla de consideraciones semánticas 23](#_Toc56955114)

[Descripción detallada del proceso de Administración de Memoria usado en la compilación 27](#_Toc56955115)

[Tabla de datos 27](#_Toc56955116)

[Cuádruplos 28](#_Toc56955117)

[Funciones especiales 29](#_Toc56955118)

[Descripción de la máquina virtual 31](#_Toc56955119)

[Equipo de cómputo, lenguaje y utilerías especiales usadas 31](#_Toc56955120)

[Descripción detallada del proceso de Administración de Memoria en ejecución 31](#_Toc56955121)

[Ejemplo de llamada al main 31](#_Toc56955122)

[Ejemplo de llamada a función 31](#_Toc56955123)

[Pruebas del funcionamiento del lenguaje 32](#_Toc56955124)

[Documentación del código del proyecto 37](#_Toc56955125)

[Run 37](#_Toc56955126)

[Función 37](#_Toc56955127)

[Código 37](#_Toc56955128)

[Read 37](#_Toc56955129)

[Función 37](#_Toc56955130)

[Código 37](#_Toc56955131)

[Parámetros 37](#_Toc56955132)

[Retorno 38](#_Toc56955133)

[CloseOutput 38](#_Toc56955134)

[Función 38](#_Toc56955135)

[Código 38](#_Toc56955136)

[Callf 38](#_Toc56955137)

[Función 38](#_Toc56955138)

[Código 38](#_Toc56955139)

[Parámetros 39](#_Toc56955140)

[OpenOutput 39](#_Toc56955141)

[Función 39](#_Toc56955142)

[Código 39](#_Toc56955143)

[Call 39](#_Toc56955144)

[Función 39](#_Toc56955145)

[Código 39](#_Toc56955146)

[Parámetros 42](#_Toc56955147)

[Retorno 42](#_Toc56955148)

[Op 42](#_Toc56955149)

[Función 42](#_Toc56955150)

[Código 42](#_Toc56955151)

[Parámetros 43](#_Toc56955152)

[Retorno 43](#_Toc56955153)

[BuscaVariable 43](#_Toc56955154)

[Función 43](#_Toc56955155)

[Código 43](#_Toc56955156)

[Parámetros 43](#_Toc56955157)

[Retorno 44](#_Toc56955158)

[BuscaTipo 44](#_Toc56955159)

[Función 44](#_Toc56955160)

[Código 44](#_Toc56955161)

[Parámetros 44](#_Toc56955162)

[Retorno 44](#_Toc56955163)

# Descripción del Proyecto

## Propósito y Alcance del Proyecto

El propósito de este proyecto es crear un compilador capaz de compilar y ejecutar el lenguaje definido como MeMySelf, especificado en este documento. El programa será capaz de correr funciones, manejar funciones locales y globales, utilizar estatutos de decisión, de repetición condicional y no condicional, de lectura y de escritura y mostrar un output gráfico.

## Análisis de Requerimientos y Casos de Uso Generales

1. El usuario debe de poder ejecutar el programa.
   1. El programa debe de seguir la estructura definida.
   2. El programa debe de sacar error ante cualquier inconsistencia.
2. El usuario debe de poder ingresar datos si la instrucción lo requiere.
   1. Los datos ingresados se deben de guardar en una variable del mismo tipo que el valor esperado.
   2. El programa debe de sacar un error en caso de que el valor ingresado sea incompatible con el tipo de variable.
3. El usuario debe de poder desplegar el valor de las variables en la pantalla.
   1. El valor por desplegar puede ser el resultado de una ecuación, un string o el valor de una variable.
   2. La variable por desplegar debe de estar definido e inicializado con algún valor antes de desplegarlo en la pantalla.
4. El usuario debe de poder declarar variables del tipo: int, float y char.
   1. El programa debe de también poder manejar variables del tipo booleano y strings.
   2. Las variables declaradas se identifican con nombres únicos.
5. El usuario debe de poder declarar funciones del tipo: void, int, float y char.
   1. El programa no debe de dar error si no hay funciones declaradas.
   2. Las funciones pueden nunca ser ejecutadas.
   3. Las funciones pueden no tener instrucciones.
6. El usuario debe de obligatoriamente declarar la función main con el código inicial a ejecutar.
7. El usuario debe de poder llamar a las funciones declaradas, excepto al main, el cual solo se ejecuta una vez.
   1. El programa debe de comprobar que la función llamada exista.
   2. El programa debe de poder manejar llamadas recursivas.
   3. El programa debe de comprobar que la cantidad de parámetros enviados con la llamada y el tipo de los parámetros enviados sean iguales a los definidos en la función.
8. El usuario debe de poder llamar a las funciones determinadas para dibujar en el output gráfico.
   1. El programa no desplegara el output grafico si el usuario no hace llamadas al output gráfico.
   2. El output grafico no debe cerrarse hasta que el usuario lo indique.
9. El usuario debe de poder crear instrucciones de decisión y loops.
10. El usuario debe de poder declarar variables locales para cada función.
    1. Las variables locales no se repiten entre los parámetros ni las variables globales.
    2. Las variables globales pueden ser utilizadas desde cualquier lugar.
    3. Las variables locales solo pueden ser utilizadas dentro de las funciones correspondientes.
11. El usuario debe de poder declarar parámetros para cada función.
12. Los parámetros deben de ser del tipo de variables aceptados.
13. Las funciones pueden ser declaradas sin parámetros.
14. El usuario debe de poder hacer operaciones con las variables numéricas.
15. El usuario debe de poder utilizar constantes numéricas, llamadas a funciones y variables en las expresiones.
16. Las llamadas a funciones en expresiones deben de regresar un valor numérico.
17. Las variables en expresiones deben de ser de tipo numérico.
18. El usuario debe de poder asignar valores a variables existentes.
19. El valor asignado a una variable debe de ser compatible con el tipo de la variable.

## Descripción de los principales Test Cases

### Test Cases que no dan error

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test | Código | Consola Input | Consola Output |
| 1 | Program | Program MeMySelf;  main()  {  write("Hello World");  } |  | Hello World |
| 2 | Write and Read | Program MeMySelf;  var  int x;  char y;  main()  {  read(x,y);  write(x,y,"Hello World",4+1,x\*3);  } | 2  c | 2  c  Hello World  5  6 |
| 3.1 | Call | Program MeMySelf;  var  int x;  char y;  module void funcion();  {  read(x,y);  }  main()  {  funcion();  write(x,y,"Hello World");  } | 1  c | 1  c  Hello World |
| 3.2 | Call | Program MeMySelf;  var  int x;  char y;  module void funcion(int m);  {  read(x,y);  }  main()  {  x=3;  funcion(x);  write(x,y,"Hello World");  } | 1  c | 1  c  Hello World |
| 4 | Callr | Program MeMySelf;  var  int x;  char y;  module int funcion();  {  read(x,y);  return(3);  }  main()  {  x = funcion();  write(x,y,"Hello World");  } | 1  c | 3  c  Hello World |
| 5 | Asignación | Program MeMySelf;  var  float x;  char y;  module void funcion(float m);  {  read(x,y);  }  main()  {  x=3;  write(x);  funcion(x);  write(x,y,"Hello World");  } | 1  c | 3  1.0  c  Hello World |
| 6 | Expresiones | Program MeMySelf;  var  float x;  main()  {  x=3\*2-4/2;  write(x);  } |  | 4.0 |
| 7.1 | Decisiones | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 & x!=2) then {  write("True");  x=3;  } else {  write("False");  x=2;  }  write(x);  } |  | False  2 |
| 7.2 | Decisiones | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 | x!=2) then {  write("True");  x=3;  } else {  write("False");  x=2;  }  write(x);  } |  | True  3 |
| 7.3 | Decisiones | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 & x!=2) then {  write("True");  x=3;  }  write(x);  } |  | 1 |
| 8.1 | Repetición | Program MeMySelf;  var  int x;  main()  {  x=1;  from x = 0 to 3 do {  write(x);  }  } |  | 0  1  2  3 |
| 8.2 | Repetición | Program MeMySelf;  var  int x;  main()  {  x=1;  while(x<3) do {  write(x);  x=x+1;  }  } |  | 1  2 |
| 8.3 | Repetición | Program MeMySelf;  var  int x;  main()  {  x=1;  do {  write(x);  x=x+1;  }while(x<3)  } |  | 1  2 |
| 9 | Llamadas especiales | Program MeMySelf;  main()  {  Color("red");  Line(100);  Turn(90);  Color("blue");  Point();  Line(100);  Turn(90);  PenUp();  Line(100);  Turn(90);  PenDown();  Size(10);  Line(100);  Turn(90);  } |  |  |

### Test Cases que dan error

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Test | Código | Consola Input | Error en Consola |
| 2.1.1 | Write | Program MeMySelf;  var  int x;  char y;  main()  {  write(x,y,"Hello World");  } |  | ERROR: Variable no inicializada |
| 2.2.1 | Read | Program MeMySelf;  main()  {  read(x,y);  write(x,y,"Hello World");  } |  | ERROR: Variable no encontrada |
| 3.1.1 | Call | Program MeMySelf;  var  int x;  char y;  module void funcion();  {  read(x,y);  }  main()  {  x = funcion();  write(x,y,"Hello World");  } | c | ERROR: Llamada con return a funcion void |
| 3.2.1 | Call | Program MeMySelf;  var  int x;  char y;  module void funcion(int m);  {  read(x,y);  }  main()  {  x=3;  funcion(y);  write(x,y,"Hello World");  } |  | ERROR: Tipo de parametro esperado invalido |
| 3.2.2 | Call | Program MeMySelf;  var  int x;  char y;  module void funcion(int m);  {  read(x,y);  }  main()  {  x=3;  funcion();  write(x,y,"Hello World");  } |  | ERROR: Cantidad de parametros invalida |
| 4.1 | Callr | Program MeMySelf;  var  int x;  char y;  module int funcion();  {  read(x,y);  return(x);  }  main()  {  funcion();  write(x,y,"Hello World");  } |  | ERROR: Llamada a funcion con return en llamada void |
| 5.1 | Asignación | Program MeMySelf;  var  int x;  char y;  module float funcion();  {  read(x,y);  return(x);  }  main()  {  x = funcion();  write(x,y,"Hello World");  } |  | ERROR: Tipo de variable invalido en asignacion |
| 5.2 | Asignación | Program MeMySelf;  var  int x;  char y;  module void funcion(int m);  {  read(x,y);  }  main()  {  x='c';  write(x);  funcion(x);  write(x,y,"Hello World");  } |  | ERROR: Tipo de variable invalido en asignacion |

## Descripción del proceso general seguido para el desarrollo del proyecto

Se decidió utilizar el lenguaje Python para el compilador, agregando la librería Ply para el léxico y sintaxis, la librería Tortuga para el output gráfico y la librería Copy para simular las variables locales. Las variables del programa se guardaron en un objeto Json, dentro de su correspondiente función y junto con información adicional de cada función.

El proceso seguido fue de cuatro etapas. La primera etapa consistió en utilizar la librería Ply para poder definir el léxico y sintaxis del programa. Se definieron todos los tokens que utilizaría el programa, las palabras reservadas y la estructura del programa. En la segunda etapa se programó la semántica estática, creando el listado de duplas correspondientes y marcando los primeros errores de compilación. En la tercera etapa se programó la semántica dinámica, agregando la ejecución correspondiente a cada dupla, las verificaciones correspondientes y finalmente incluyendo el output gráfico.

### Bitácoras

#### Avance 1

8 de octubre 2020

##### Entrega

Archivo .py con el léxico y la sintaxis funcionando.

##### Diferencias entre la GramaticaInicial y GramaticaActual

Las funciones se inicializan con "module <tipo de retorno> nombre\_modulo" en lugar de "<tipo de retorno> module nombre\_modulo"

#### Avance 2

14 de octubre 2020

##### Entrega

Semántica básica

##### Avance

* Las variables son guardadas por globales y funciones, las variables de funciones no pueden repetirse entre las variables globales
* Los parámetros de funciones son guardados, los parámetros de funciones no pueden repetirse entre las variables globales y las variables de funciones no pueden repetirse entre los parámetros de dicha función
* Las funciones son guardadas con sus variables y parámetros correspondientes
* Las ecuaciones son transformadas en un arreglo de instrucciones en forma de cuádruplos
* Las asignaciones tienen todas sus instrucciones correspondientes

#### Avance 3

24 de octubre 2020

##### Entrega

Generación de código intermedio inicial

##### Avance

* Se agregaron los tokens: '>=', '!=' y '<='
* Las variables del sistema ahora tienen un espacio entre el nombre y su número. Esto ayuda a que el usuario no pueda por accidente crear una variable con el mismo nombre que una variable del sistema, ya que las variables del usuario son sin espacios.
* Se establecieron las estructuras que ira generando el programa en forma de cuádruplos.
* Todos los estatutos mandan su arreglo de instrucciones completo en el formato requerido.
* El arreglo de instrucciones del main es checado en busca de errores de semántica estática.
* El arreglo de instrucciones de cada función es checado en busca de errores de semántica estática.
* Se considero que el usuario puede no declarar variables.

##### Diferencias entre la GramaticaInicial y GramaticaActual

Se agrego la repetición condicional While Do.

#### Avance 4

30 de octubre 2020

##### Entrega

Generación de código intermedio para estatutos no-lineales e inicio de funciones

##### Avance

* Se reestructuraron las variables virtuales. Se junto en una sola variable la información de todas las funcionas.
* Se creo una tabla de 3 dimensiones con los tipos de variables y tipos de operadores dando como resultado el tipo de valor del resultado.
* Se creo una función dedicada a determinar el tipo de valor, ya sea de un valor constante o de una variable.
* Se creo una función que regresa un apuntador a la variable buscada.
* Se creo una función run para iniciar la ejecución de las instrucciones.
* Se creo una función call para ejecutar las instrucciones de la función activa con variables locales por llamada a la función.
* Se creo una función por cada acción esperada a ejecutar, para todos los estatutos excepto las funciones especiales.
* Se movió el chequeo de la semántica estática para analizar todas las instrucciones en un solo lugar.
* La creación de funciones aparte de main se hizo opcional.
* Se arreglo un problema con las expresiones, el orden de evaluación se tuvo que corregir.
* Todas las constantes del código se guardan en una variable del sistema.
* El estatuto de repetición no condicional, FROM TO, ahora acepta expresiones para la inicialización de la variable y el valor limite.

### Reflexión

Fue un proceso interesante, ya que al inicio me encontraba realmente perdida con el lenguaje y la idea. Pensé que seria un logro demasiado difícil para mí. Incluso considere meterme a un equipo solo para poder contar con el apoyo de alguien, pero al ver que la dificultad aumentaba demasiado para mi gusto, decidí hacerlo por mi cuenta y buscar a alguien que me explicara. Cuando comencé con la primera etapa, estuve pegada a la documentación dudando de cada una de mis decisiones. La única persona que encontré que me podía ayudar, había llevado antes de la clase, pero sin permiso para utilizar librerías, asique la idea era que tenia que hacer por mi cuenta la primera parte y luego me explicaría como hacer la semántica. Recuerdo que para cuando terminé la primera parte me encontré de repente programando la semántica dinámica. Realmente no sabia lo que estaba haciendo, pero de alguna forma veía que estaba funcionando. Termine de programar la semántica dinámica casi el mismo día. Para cuando la persona que me iba a ayudar me pregunto en que necesitaba ayuda, ya había terminado el programa. Desde que lo termine me he sentido bastante satisfecha con el resultado, sobre todo por el miedo que le tenia al inicio. También me alegro de no haberme metido en equipo con alguien, mas que nada por el hecho de no tener que discutir una decisión con alguien más.

# Descripción del lenguaje

## Nombre del lenguaje

Vintage

## Descripción genérica de las principales características del lenguaje

Es un programa orientado a objetos muy parecido a lenguajes como C++ con algunas limitaciones. Puede hacer uso de variables globales y locales, de funciones void y con retorno, de loops como while do, do while y for in do, de decisión e instrucciones de lectura y escritura. Una característica única de este lenguaje es que permite hacer uso de llamadas a funciones especiales que permiten hacer uso de la interfaz grafica para dibujar en ella.

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

### Error en Compilación

1. Tipo de variable no encontrada: ocurre cuando no se reconoce el tipo de variable o, en caso de ser una variable, no se encuentra definida la variable con un tipo.
2. Variable no encontrada: ocurre cuando el nombre de la variable no se encuentra en ningún listado, ni entre las globales, ni entre las locales de la función activa en ese momento.
3. Tipo de valores inválidos: ocurre cuando en una expresión se utilizan variables diferentes del tipo int o float.
4. PROGRAM01: error en código del programa, consultar a un programador del lenguaje.
5. Tipo de variable invalido en asignación: ocurre cuando se intenta ingresar un valor incompatible en una variable. Por ejemplo, que se intente guardar un valor no numérico en una variable numérica.
6. Return en función void: ocurre cuando se encuentra un return dentro de una función declarada como void.
7. Tipo de return invalido: ocurre cuando el tipo de valor en un return no coincide con el tipo de función donde se encuentra el return.
8. Llamada a función invalida: ocurre cuando se hace una llamada a una función y dicha función no fue declarada en el programa.
9. Llamada con return a función void: ocurre cuando se hace una llamada a una función esperando un valor de regreso y dicha función es de tipo void.
10. Cantidad de parámetros invalida: ocurre cuando se hace una llamada a una función y la cantidad de parámetros enviados en la llamada es diferente a la cantidad de parámetros declarados en la función.
11. Tipo de parámetro esperado invalido en return: ocurre cuando se intenta hacer una llamada a una función y alguno de los tipos de los parámetros es diferente a su tipo de parámetro esperado según la definición de la función y el orden de los parámetros enviados.
12. Llamada a función con return en llamada void: ocurre cuando se hace una llamada a una función que no es void y no se utiliza el valor de retorno.
13. Llamada a main: ocurre cuando se hace una llamada a la función main.
14. Tipo de parámetro invalido: ocurre cuando se intenta hacer una llamada a una función y alguno de los tipos de los parámetros es diferente a su tipo de parámetro esperado según la definición de la función y el orden de los parámetros enviados.
15. PROGRAM02: error en código del programa, consultar a un programador del lenguaje.
16. Nombre de variable repetido, variable global declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre las variables globales.
17. Nombre de variable repetido, parámetro de función declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre los parámetros de la función.
18. Nombre de variable repetido, variable de función declarada: ocurre cuando el nombre de la variable que se intenta declarar, ya se encuentra declarada entre las variables de la función.
19. Nombre de función invalido: ocurre cuando se intenta declarar una función con el nombre de otra función previamente declarada.
20. Nombre de parámetro invalido: ocurre cuando el nombre del parámetro que se intenta declarar, ya se encuentra declarada entre las variables globales o entre los parámetros previamente declarados.

### Error en Ejecución

1. OP01: error en código del programa, consultar a un programador del lenguaje.
2. Variable no inicializada: ocurre cuando se quiere utilizar una variable en una expresión, llamada o escritura y la variable no cuenta con un valor en ese momento.
3. No llego a un return la función: ocurre cuando una función, de tipo no void, no regresa algún valor.
4. CALL01: error en código del programa, consultar a un programador del lenguaje.
5. CALLF01: error en código del programa, consultar a un programador del lenguaje.
6. Longitud de carácter mayor a 1: ocurre cuando en una instrucción read se recibe del usuario un string mayor a un carácter y el valor esperado es un char.
7. READ01: error en código del programa, consultar a un programador del lenguaje.

# Descripción del compilador

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

Equipo utilizado: Windows 7.

Lenguaje: Python.

Librerías: Ply, Copy y Tortuga.

Programas: Visual Studio Code para programar, compilar y ejecutar el código y Github para administrar las versiones y cambios.

## Descripción del Análisis de Léxico

### Tokens

1. AND = ‘&’
2. OR = ‘|’
3. LE = ‘<=’
4. LT = ‘<’
5. GE = ‘>=’
6. GT = ‘>’
7. EQ = ‘==’
8. NE = ‘!=’
9. EQUAL = ‘=’
10. LPAREN = ‘(‘
11. RPAREN = ‘)’
12. LPAREN2 = ‘{‘
13. LPAREN2 = ‘}’
14. PUNCOM = ‘;’
15. COMA = ‘,’
16. NINT = '\d+'
17. NFLOAT = '\d+\.\d+'
18. SCHAR = '\'.\''
19. SSTRING = '\".\*\"'
20. SVAR = '[a-zA-Z][a-zA-Z0-9]\*'

### Palabras Reservadas

1. PROGRAM = Program
2. MAIN = main
3. VAR = var
4. INT = int
5. FLOAT = float
6. CHAR = char
7. VOID = void
8. MODULE = module
9. RETURN = return
10. READ = read
11. WRITE = write
12. IF = if
13. THEN = then
14. ELSE = else
15. DO = do
16. WHILE = while
17. TRUE = true
18. FALSE = false
19. FROM = from
20. TO = to
21. TURN = Turn
22. LINE = Line
23. POINT = Point
24. PENUP = PenUp
25. PENDOWN = PenDown
26. COLOR = Color
27. SIZE = Size

## Descripción del Análisis de Sintaxis

program : programInicio decvar decfuntemp programMain

decfuntemp : decfunCodigo decfuntemp

| empty

programMain : programMainIni LPAREN2 estatutos RPAREN2

programMainIni : MAIN LPAREN RPAREN

programInicio : PROGRAM SVAR PUNCOM

decvar : VAR decvar2

| empty

decvar2 : tipo SVAR nomvar decvar2

| empty

nomvar : COMA SVAR nomvar

| PUNCOM

tipo : INT

| FLOAT

| CHAR

decfunCodigo : decfunCodigoIni LPAREN funpara RPAREN PUNCOM decvar LPAREN2 estatutos RPAREN2

decfunCodigoIni : MODULE funtipo SVAR

funtipo : tipo

| VOID

funpara : tipo SVAR funpara2

| empty

funpara2 : COMA tipo SVAR funpara2

| empty

estatutos : asignacion estatutos

| llamada estatutos

| lectura estatutos

| escritura estatutos

| desicion estatutos

| repeticion estatutos

| funespecial estatutos

| retorno

| empty

asignacion : SVAR EQUAL asitipos PUNCOM

asitipos : expr

| SCHAR

expr : exprCode

exprCode : expr1 exprCodeT

exprCodeT : '+' expr1 exprCodeT

| '-' expr1 exprCodeT

| empty

expr1 : expr2 expr1T

expr1T : '\*' expr2 expr1T

| '/' expr2 expr1T

| empty

expr2 : LPAREN exprCode RPAREN

| term

term : NINT

| NFLOAT

| SVAR posfun

posfun : LPAREN parametros RPAREN

| empty

parametros : asitipos parametros2

| empty

parametros2 : COMA asitipos parametros2

| empty

llamada : SVAR LPAREN parametros RPAREN PUNCOM

retorno : RETURN LPAREN asitipos RPAREN PUNCOM

lectura : READ LPAREN SVAR lectura2 RPAREN PUNCOM

lectura2 : COMA SVAR lectura2

| empty

escritura : WRITE LPAREN escritura2 escritura3 RPAREN PUNCOM

escritura2 : SSTRING

| asitipos

escritura3 : COMA escritura2 escritura3

| empty

desicion : IF LPAREN expresion RPAREN THEN LPAREN2 estatutos RPAREN2 desicion2

desicion2 : ELSE LPAREN2 estatutos RPAREN2

| empty

repeticion : condicional

| nocondicional

condicional : DO LPAREN2 estatutos RPAREN2 WHILE LPAREN expresion RPAREN

| WHILE LPAREN expresion RPAREN DO LPAREN2 estatutos RPAREN2

expresion : comp1 expresion2

expresion2 : comp2 comp1 expresion2

| empty

comp2 : AND

| OR

comp1 : asitipos comp3 asitipos

| TRUE

| FALSE

comp3 : EQ

| GT

| GE

| LT

| LE

| NE

nocondicional : FROM SVAR EQUAL expr TO expr DO LPAREN2 estatutos RPAREN2

funespecial : LINE LPAREN expr RPAREN PUNCOM

| TURN LPAREN expr RPAREN PUNCOM

| SIZE LPAREN expr RPAREN PUNCOM

| POINT LPAREN RPAREN PUNCOM

| PENUP LPAREN RPAREN PUNCOM

| PENDOWN LPAREN RPAREN PUNCOM

| COLOR LPAREN SSTRING RPAREN PUNCOM

## Descripción de Generación de Código Intermedio y Análisis Semántico

1. [program] = Ultima instrucción de compilación. Repasa cada función declarada y analiza los cuádruplos generados de la función. Busca errores de semántica dinámica. Al finalizar llama la función Run para ejecutar el código.
2. [decfuntemp] = Sin código.
3. [programMain] = Guarda el arreglo de duplas recibido de *estatutos* en “run” dentro de “main”.
4. [programMainIni] = Coloca como función activa la función “main”.
5. [programInicio] = Coloca como función activa la función “main”.
6. [decvar] = Sin código.
7. [decvar2] = En caso de no ser *empty*, crea un arreglo con los valores recibidos en *SVAR* y *nomvar*. Repasa el arreglo, comprueba que el nombre de la variable no se encuentre ya declarado y agrega la variable en la tabla de variables de la función activa con el tipo de variable recibido de *tipo*.
8. [nomvar] = En caso de no ser *PUNCOM*, va creando un arreglo con los valores recibidos en *SVAR*.
9. [tipo] = Regresa el valor encontrado.
10. [decfunCodigo] = Guarda el arreglo de duplas recibido de estatutos en “run” dentro de la función activa.
11. [decfunCodigoIni] = Coloca como función activa el nombre de función recibido en *SVAR*. Crea la tabla de información de la función, indicando el nombre, tipo de función (recibido de *funtipo*) y el valor inicial de sus variables temporales.
12. [funtipo] = Regresa el valor encontrado.
13. [funpara] = En caso de no ser *empty*, crea un arreglo de duplas con el nombre de la variable y el tipo de variable, luego recorre el arreglo comprobando que los nombres no hayan sido declarados anteriormente y, en caso de no ser así, agregando la variable a la tabla de la función activa junto con su tipo.
14. [funpara2] = En caso de no ser *empty*, regresa un arreglo de duplas con el nombre de la variable y el tipo de variable.
15. [estatutos] = En caso de no ser *empty*, regresa un arreglo con los arreglos recibidos de los diferentes estatutos.
16. [asignacion] = Envía los cuádruplos necesarios para llegar al resultado, si hay alguno, y agrega un cuádruplo asignando el valor a la variable recibida en *SVAR*.
17. [asitipos] = Envía la lista de cuádruplos recibidos en *expr* o una variable temporal con el valor recibido en *SCHAR*.
18. [expr] = Recibe un arreglo de vector polaco y lo convierte en un listado de cuádruplos.
19. [exprCode] = Envía un arreglo de vector polaco juntando los arreglos que recibe de *expr1* y *exprCodeT*.
20. [exprCodeT] = En caso de no ser empty, envía un arreglo de vector polaco juntando los arreglos que recibe de *expr1* y *exprCodeT* y agrega el valor de signo recibido.
21. [expr1] = Envía un arreglo de vector polaco juntando los arreglos que recibe de *expr2* y *expr1T*.
22. [expr1T] = En caso de no ser empty, envía un arreglo de vector polaco juntando los arreglos que recibe de *expr2* y *expr1T* y agrega el valor de signo recibido.
23. [expr2] = Envía un arreglo vector polaco recibido de *exprCode* o el valor recibido de *term* en un arreglo.
24. [term] = Regresa el valor encontrado, si recibe un valor de *posfun*, regresa la función con su nombre y parámetros.
25. [posfun] = En caso de no ser *empty*, envía un arreglo con los parámetros recibidos de *parametros*.
26. [parametros] = En caso de no ser *empty*, regresa un arreglo juntando lo recibido de *asitipos* y *parametros2*.
27. [parametros2] = En caso de no ser *empty*, regresa un arreglo juntando lo recibido de *asitipos* y *parametros2*.
28. [llamada] = Envía los cuádruplos necesarios para llegar a los resultados, si hay algunos, y agrega un cuádruplo llamando a la función recibida en *SVAR* con los parámetros recibidos de *parametros*.
29. [retorno] = Envía los cuádruplos necesarios para llegar al resultado, si hay alguno, y agrega un cuádruplo de return junto con el ultimo valor de *asitipos*.
30. [lectura] = Agrega un cuádruplo de read por cada valor recibido de *lectura2*.
31. [lectura2] = En caso de no ser *empty*, egresa el arreglo de los valores recibidos en *SVAR* y *lectura2*.
32. [escritura] = Agrega un cuádruplo de *write* por cada valor recibido de *escritura2* y *escritura3*.
33. [escritura2] = Regresa la variable recibida *SVAR* o una variable temporal con el valor recibido de *SSTRING*.
34. [escritura3] = En caso de no ser *empty*, envía un arreglo juntando lo recibido de *escritura2* y *escritura3*.
35. [desicion] = Envía un arreglo juntando el arreglo recibido de *expresion*, un cuádruplo creado de gotof, el arreglo recibido de *estatutos* y, en caso de que se reciba algo de *desicion2*, un cuádruplo creado de goto y el arreglo recibido de *desicion2*.
36. [desicion2] = En caso de no ser *empty*, regresa el arreglo recibido de *estatutos*.
37. [repeticion] = Regresa el valor encontrado.
38. [condicional] = Envía un arreglo juntando los arreglos recibidos de *expresion* y *estatutos* y dos cuádruplos creados de goto y gotof, cada uno en su correspondiente lugar.
39. [expresion] = Regresa un arreglo juntando el arreglo recibido de *comp1* con el arreglo de cuádruplos creados para llegar al resultado siguiendo el arreglo de duplas recibido de *expresion2*.
40. [expresion2] = Envía un arreglo con la dupla creada con los valores encontrados en *comp2* y *comp1* y los valores recibidos de *expresion2*.
41. [comp1] = Envía un arreglo juntando los arreglos recibidos de *asitipos* y un cuádruplo creado de comparación.
42. [comp2] = Regresa el valor encontrado.
43. [comp3] = Regresa el valor encontrado.
44. [nocondicional] = Envía un arreglo juntando los recibidos de *expr* y *estatutos* y los cuádruplos creados de asignación, suma, gotof y goto necesarios en sus lugares correspondientes.
45. [funespecial] = Envía un arreglo juntando el recibido de *expr*, si hay alguno, y el cuádruplo creado con la llamada a la función especial.

### Tabla de consideraciones semánticas

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo1 | Tipo2 | Operador | Resultado |
| Int | Int | + | Int |
| Int | Int | - | Int |
| Int | Int | \* | Int |
| Int | Int | / | Float |
| Int | Int | == | Bool |
| Int | Int | != | Bool |
| Int | Int | <= | Bool |
| Int | Int | < | Bool |
| Int | Int | >= | Bool |
| Int | Int | > | Bool |
| Int | Int | & | Error |
| Int | Int | | | Error |
| Int | Float | + | Float |
| Int | Float | - | Float |
| Int | Float | \* | Float |
| Int | Float | / | Float |
| Int | Float | == | Bool |
| Int | Float | != | Bool |
| Int | Float | <= | Bool |
| Int | Float | < | Bool |
| Int | Float | >= | Bool |
| Int | Float | > | Bool |
| Int | Float | & | Error |
| Int | Float | | | Error |
| Int | Char | + | Error |
| Int | Char | - | Error |
| Int | Char | \* | Error |
| Int | Char | / | Error |
| Int | Char | == | Bool |
| Int | Char | != | Bool |
| Int | Char | <= | Error |
| Int | Char | < | Error |
| Int | Char | >= | Error |
| Int | Char | > | Error |
| Int | Char | & | Error |
| Int | Char | | | Error |
| Int | Bool | + | Error |
| Int | Bool | - | Error |
| Int | Bool | \* | Error |
| Int | Bool | / | Error |
| Int | Bool | == | Bool |
| Int | Bool | != | Bool |
| Int | Bool | <= | Error |
| Int | Bool | < | Error |
| Int | Bool | >= | Error |
| Int | Bool | > | Error |
| Int | Bool | & | Error |
| Int | Bool | | | Error |
| Float | Int | + | Float |
| Float | Int | - | Float |
| Float | Int | \* | Float |
| Float | Int | / | Float |
| Float | Int | == | Bool |
| Float | Int | != | Bool |
| Float | Int | <= | Bool |
| Float | Int | < | Bool |
| Float | Int | >= | Bool |
| Float | Int | > | Bool |
| Float | Int | & | Error |
| Float | Int | | | Error |
| Float | Float | + | Float |
| Float | Float | - | Float |
| Float | Float | \* | Float |
| Float | Float | / | Float |
| Float | Float | == | Bool |
| Float | Float | != | Bool |
| Float | Float | <= | Bool |
| Float | Float | < | Bool |
| Float | Float | >= | Bool |
| Float | Float | > | Bool |
| Float | Float | & | Error |
| Float | Float | | | Error |
| Float | Char | + | Error |
| Float | Char | - | Error |
| Float | Char | \* | Error |
| Float | Char | / | Error |
| Float | Char | == | Bool |
| Float | Char | != | Bool |
| Float | Char | <= | Error |
| Float | Char | < | Error |
| Float | Char | >= | Error |
| Float | Char | > | Error |
| Float | Char | & | Error |
| Float | Char | | | Error |
| Float | Bool | + | Error |
| Float | Bool | - | Error |
| Float | Bool | \* | Error |
| Float | Bool | / | Error |
| Float | Bool | == | Bool |
| Float | Bool | != | Bool |
| Float | Bool | <= | Error |
| Float | Bool | < | Error |
| Float | Bool | >= | Error |
| Float | Bool | > | Error |
| Float | Bool | & | Error |
| Float | Bool | | | Error |
| Float | Int | + | Int |
| Char | Int | - | Int |
| Char | Int | \* | Int |
| Char | Int | / | Float |
| Char | Int | == | Bool |
| Char | Int | != | Bool |
| Char | Int | <= | Bool |
| Char | Int | < | Bool |
| Char | Int | >= | Bool |
| Char | Int | > | Bool |
| Char | Int | & | Error |
| Char | Int | | | Error |
| Char | Float | + | Error |
| Char | Float | - | Error |
| Char | Float | \* | Error |
| Char | Float | / | Error |
| Char | Float | == | Bool |
| Char | Float | != | Bool |
| Char | Float | <= | Error |
| Char | Float | < | Error |
| Char | Float | >= | Error |
| Char | Float | > | Error |
| Char | Float | & | Error |
| Char | Float | | | Error |
| Char | Char | + | Error |
| Char | Char | - | Error |
| Char | Char | \* | Error |
| Char | Char | / | Error |
| Char | Char | == | Bool |
| Char | Char | != | Bool |
| Char | Char | <= | Error |
| Char | Char | < | Error |
| Char | Char | >= | Error |
| Char | Char | > | Error |
| Char | Char | & | Error |
| Char | Char | | | Error |
| Char | Bool | + | Error |
| Char | Bool | - | Error |
| Char | Bool | \* | Error |
| Char | Bool | / | Error |
| Char | Bool | == | Bool |
| Char | Bool | != | Bool |
| Char | Bool | <= | Error |
| Char | Bool | < | Error |
| Char | Bool | >= | Error |
| Char | Bool | > | Error |
| Char | Bool | & | Error |
| Char | Bool | | | Error |
| Bool | Int | + | Error |
| Bool | Int | - | Error |
| Bool | Int | \* | Error |
| Bool | Int | / | Error |
| Bool | Int | == | Bool |
| Bool | Int | != | Bool |
| Bool | Int | <= | Error |
| Bool | Int | < | Error |
| Bool | Int | >= | Error |
| Bool | Int | > | Error |
| Bool | Int | & | Error |
| Bool | Int | | | Error |
| Bool | Float | + | Error |
| Bool | Float | - | Error |
| Bool | Float | \* | Error |
| Bool | Float | / | Error |
| Bool | Float | == | Bool |
| Bool | Float | != | Bool |
| Bool | Float | <= | Error |
| Bool | Float | < | Error |
| Bool | Float | >= | Error |
| Bool | Float | > | Error |
| Bool | Float | & | Error |
| Bool | Float | | | Error |
| Bool | Char | + | Error |
| Bool | Char | - | Error |
| Bool | Char | \* | Error |
| Bool | Char | / | Error |
| Bool | Char | == | Bool |
| Bool | Char | != | Bool |
| Bool | Char | <= | Error |
| Bool | Char | < | Error |
| Bool | Char | >= | Error |
| Bool | Char | > | Error |
| Bool | Char | & | Error |
| Bool | Char | | | Error |
| Bool | Bool | + | Error |
| Bool | Bool | - | Error |
| Bool | Bool | \* | Error |
| Bool | Bool | / | Error |
| Bool | Bool | == | Bool |
| Bool | Bool | != | Bool |
| Bool | Bool | <= | Error |
| Bool | Bool | < | Error |
| Bool | Bool | >= | Error |
| Bool | Bool | > | Error |
| Bool | Bool | & | Bool |
| Bool | Bool | | | Bool |

## Descripción detallada del proceso de Administración de Memoria usado en la compilación

### Tabla de datos

# Estructura para variables temporales dentro de var

# {

# "nombre\_de\_funcion ##": {

#    "type": "tipo de variable",

#    "value": "valor de la variable"

#    },

# "contador sys": ##

# }

# Estructura

# {

# "nombre\_de\_funcion" : {

#    "type": "tipo de funcion"

#    "var": {

#         "nombre\_de\_variable":{...}

#    },

#    "param": {

#         "nombre\_de\_parametro":{...}

#    },

#    "run": [(...)]

#    }

# "active sys": "nombre de funcion"

# }

# ejemplo de llamada

# variables["nombre\_de\_funcion"]["var"]["nombre\_de\_variable"]["type"]

# inicializado con la funcion main

variables = {"main":{"type":"void","param":{},"run":[],"var":{"contador sys": 0}}}

Toda la memoria se guarda en la variable global llamada “variables”. En el primer nivel se guardan el nombre de las funciones, una variable llamada “active sys” utilizada para poner el nombre de la función que se esta analizando en el momento (la función activa) y una variable llamada “output avtive” utilizada para señalar si el usuario utiliza el output gráfico. Dentro de cada función se guardan cuatro datos: “type” que señala el tipo de función; “var” que contiene todas las variables declaradas dentro de la función, así como sus variables temporales y una variable especial llamada “contador sys” que contiene el numero de variables temporales definidas en el momento; “param” que contiene todos los parámetros definidos con la función; “run” que contiene el arreglo de cuádruplos con las instrucciones de la función. Para crear una nueva variable temporal se combina el nombre de la función y el siguiente número del contador. Cada variable y parámetro es guardada con los valores de “type” que señala el tipo de la variable y “value” que contiene el valor de la variable.

### Cuádruplos

# Estructura

# (op, opdo1, opdo2, result)

# (=, var, null, result)

# (callr, func, param, result)

# (callf, func, param, null)

# (call, func, param, null)

# (goto, null, null, pos)

# (gotof, value, null, pos)

# (return, null, null, result)

# (write, null, null, result)

# (read, null, null, result)

(op, opdo1, opdo2, result): se utiliza el signo enviado en *op* para evaluar *opdo1* y *opdo2* y guarda el resultado en *result*.

(=, var, null, result): guarda el valor de *var* en *result*.

(callr, func, param, result): llama a la función *func* con los parámetros *param* y guarda el valor de retorno esperado en *result*.

(call, func, param, null): llama a la función *func* con los parámetros *param*.

(callf, func, param, null): llama a la función especial *func* con los parámetros *param*.

(goto, null, null, pos): le suma al contador actual el valor de *pos*.

(gotof, value, null, pos): comprueba si el valor en *value* es falso y, en caso de ser asi, le suma al contador actual el valor de *pos*.

(return, null, null, result): regresa el valor de *result*.

(write, null, null, result): escribe en consola el valor de *result*.

(read, null, null, result): recibe del usuario un valor que guarda en *result*.

### Funciones especiales

# Funciones especiales

# PenUp() => turtle.penup()

# PenDown() => turtle.pendown()

# Point() => turle.dot(10, 0, 0, 0)

# Turn(nomber) => turtle.left(nomber)

# Line(nomber) => turtle.forward(nomber)

# Color(string) => turtle.pencolor(string)

# Size(nomber) => turtle.pensize(nomber)

PenUp: deja de pintar el camino que sigue la tortuga.

PenDown: dibuja el camino que sigue la tortuga.

Point: coloca un punto en la posición actual de la tortuga.

Turn(nomber): gira el valor de *nomber* en grados a la tortuga hacia la izquierda.

Line(nomber): hace avanzar a la tortuga el valor de *nomber* hacia delante.

Color(string): define el color de las líneas como el valor de *string*.

Size(nomber): define el tamaño de las líneas como el valor de *nomber*.

# Descripción de la máquina virtual

## Equipo de cómputo, lenguaje y utilerías especiales usadas

Se utilizo las mismas especificadas en el compilador.

## Descripción detallada del proceso de Administración de Memoria en ejecución

Se utilizo la estructura señalada en la Tabla de datos. Para manejar la diferencia entre variables globales y locales, cuando se corre la función main, se utiliza un apuntador a las variables de main dentro de “variables”, cuando se llama a una función, sus parámetros y variables se les hace un “deepcopy” que, en lugar de mandar un apuntador a las variables originales, las copia en otra parte de la memoria y manda el apuntador a las copias.

### Ejemplo de llamada al main

call("main",variables["main"]["param"],variables["main"]["var"])

### Ejemplo de llamada a función

call(funcion,TempParametros,copy.deepcopy(variables[funcion]["var"]))

En este caso, los parámetros se definen antes como:

TempParam = copy.deepcopy(variables[funcion]["param"])

y se les asigna el valor correspondiente antes de ser enviados en la llamada

# Pruebas del funcionamiento del lenguaje

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Código | Cuádruplos | Input | Output |
| 1 | Program MeMySelf;  main()  {  write("Hello World");  } | \*\*\* main  1 : write | None | None | main 1 |  | Hello World |
| 2 | Program MeMySelf;  var  int x;  char y;  main()  {  read(x,y);  write(x,y,"Hello World",4+1,x\*3);  } | \*\*\* main  1 : read | None | None | x  2 : read | None | None | y  3 : write | None | None | x  4 : write | None | None | y  5 : write | None | None | main 1  6 : + | main 2 | main 3 | main 4  7 : write | None | None | main 4  8 : \* | x | main 5 | main 6  9 : write | None | None | main 6 | 2  c | 2  c  Hello World  5  6 |
| 3.1 | Program MeMySelf;  var  int x;  char y;  module void funcion();  {  read(x,y);  }  main()  {  funcion();  write(x,y,"Hello World");  } | \*\*\* main  1 : call | funcion | [] | None  2 : write | None | None | x  3 : write | None | None | y  4 : write | None | None | main 1  \*\*\* funcion  1 : read | None | None | x  2 : read | None | None | y | 1  c | 1  c  Hello World |
| 3.2 | Program MeMySelf;  var  int x;  char y;  module void funcion(int m);  {  read(x,y);  }  main()  {  x=3;  funcion(x);  write(x,y,"Hello World");  } | \*\*\* main  1 : = | main 1 | None | x  2 : call | funcion | ['x'] | None  3 : write | None | None | x  4 : write | None | None | y  5 : write | None | None | main 2  \*\*\* funcion  1 : read | None | None | x  2 : read | None | None | y | 1  c | 1  c  Hello World |
| 4 | Program MeMySelf;  var  int x;  char y;  module int funcion();  {  read(x,y);  return(3);  }  main()  {  x = funcion();  write(x,y,"Hello World");  } | \*\*\* main  1 : callr | funcion | [] | main 1  2 : = | main 1 | None | x  3 : write | None | None | x  4 : write | None | None | y  5 : write | None | None | main 2  \*\*\* funcion  1 : read | None | None | x  2 : read | None | None | y  3 : return | None | None | funcion 1 | 1  c | 3  c  Hello World |
| 5 | Program MeMySelf;  var  float x;  char y;  module void funcion(float m);  {  read(x,y);  }  main()  {  x=3;  write(x);  funcion(x);  write(x,y,"Hello World");  } | \*\*\* main  1 : = | main 1 | None | x  2 : write | None | None | x  3 : call | funcion | ['x'] | None  4 : write | None | None | x  5 : write | None | None | y  6 : write | None | None | main 2  \*\*\* funcion  1 : read | None | None | x  2 : read | None | None | y | 1  c | 3  1.0  c  Hello World |
| 6 | Program MeMySelf;  var  float x;  main()  {  x=3\*2-4/2;  write(x);  } | \*\*\* main  1 : \* | main 1 | main 2 | main 5  2 : / | main 3 | main 4 | main 6  3 : - | main 5 | main 6 | main 7  4 : = | main 7 | None | x  5 : write | None | None | x |  | 4.0 |
| 7.1 | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 & x!=2) then {  write("True");  x=3;  } else {  write("False");  x=2;  }  write(x);  } | \*\*\* main  1 : = | main 1 | None | x  2 : == | x | main 2 | main 3  3 : != | x | main 4 | main 5  4 : & | main 3 | main 5 | main 6  5 : gotof | main 6 | None | 4  6 : write | None | None | main 7  7 : = | main 8 | None | x  8 : goto | None | None | 3  9 : write | None | None | main 9  10 : = | main 10 | None | x  11 : write | None | None | x |  | False  2 |
| 7.2 | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 | x!=2) then {  write("True");  x=3;  } else {  write("False");  x=2;  }  write(x);  } | \*\*\* main  1 : = | main 1 | None | x  2 : == | x | main 2 | main 3  3 : != | x | main 4 | main 5  4 : | | main 3 | main 5 | main 6  5 : gotof | main 6 | None | 4  6 : write | None | None | main 7  7 : = | main 8 | None | x  8 : goto | None | None | 3  9 : write | None | None | main 9  10 : = | main 10 | None | x  11 : write | None | None | x |  | True  3 |
| 7.3 | Program MeMySelf;  var  int x;  main()  {  x=1;  if(x==2 & x!=2) then {  write("True");  x=3;  }  write(x);  } | \*\*\* main  1 : = | main 1 | None | x  2 : == | x | main 2 | main 3  3 : != | x | main 4 | main 5  4 : & | main 3 | main 5 | main 6  5 : gotof | main 6 | None | 3  6 : write | None | None | main 7  7 : = | main 8 | None | x  8 : write | None | None | x |  | 1 |
| 8.1 | Program MeMySelf;  var  int x;  main()  {  x=1;  from x = 0 to 3 do {  write(x);  }  } | \*\*\* main  1 : = | main 1 | None | x  2 : = | main 2 | None | x  3 : <= | x | main 3 | main 4  4 : gotof | main 4 | None | 5  5 : write | None | None | x  6 : + | x | main 6 | main 5  7 : = | main 5 | None | x  8 : goto | None | None | -5 |  | 0  1  2  3 |
| 8.2 | Program MeMySelf;  var  int x;  main()  {  x=1;  while(x<3) do {  write(x);  x=x+1;  }  } | \*\*\* main  1 : = | main 1 | None | x  2 : < | x | main 2 | main 3  3 : gotof | main 3 | None | 5  4 : write | None | None | x  5 : + | x | main 4 | main 5  6 : = | main 5 | None | x  7 : goto | None | None | -5 |  | 1  2 |
| 8.3 | Program MeMySelf;  var  int x;  main()  {  x=1;  do {  write(x);  x=x+1;  }while(x<3)  } | \*\*\* main  1 : = | main 1 | None | x  2 : write | None | None | x  3 : + | x | main 2 | main 3  4 : = | main 3 | None | x  5 : < | x | main 4 | main 5  6 : gotof | main 5 | None | 2  7 : goto | None | None | -5 |  | 1  2 |
| 9 | Program MeMySelf;  main()  {  Color("red");  Line(100);  Turn(90);  Color("blue");  Point();  Line(100);  Turn(90);  PenUp();  Line(100);  Turn(90);  PenDown();  Size(10);  Line(100);  Turn(90);  } | \*\*\* main  1 : callf | Color | ['main 1'] | None  2 : callf | Line | ['main 2'] | None  3 : callf | Turn | ['main 3'] | None  4 : callf | Color | ['main 4'] | None  5 : callf | Point | [] | None  6 : callf | Line | ['main 5'] | None  7 : callf | Turn | ['main 6'] | None  8 : callf | PenUp | [] | None  9 : callf | Line | ['main 7'] | None  10 : callf | Turn | ['main 8'] | None  11 : callf | PenDown | [] | None  12 : callf | Size | ['main 9'] | None  13 : callf | Line | ['main 10'] | None  14 : callf | Turn | ['main 11'] | None |  |  |

# Documentación del código del proyecto

## Run

### Función

Comienza la ejecución de las instrucciones empezando por la función inicial main.

### Código

def run():

     # Declara a main como función activa

     variables["active sys"] = "main"

     # Declara False al output grafico

     variables["output active"] = False

     # Comienza la ejecución con un apuntador a las variables y parámetros globales de main

     call("main",variables["main"]["param"],variables["main"]["var"])

     # Manda a llamar a la última instrucción del output grafico

     closeOutput()

## Read

### Función

Se encarga de pedirle un dato al usuario y se asegura de que el dato ingresado sea del *tipo* esperado.

### Código

def read(tipo):

     if tipo == "int":

          return int(input())

     elif tipo == "float":

          return float(input())

     elif tipo == "char":

          inputTemp = str(input())

          if len(inputTemp) != 1:

               print("ERROR: Longitud de caracter mayor a 1")

               raise CalcError("Input invalido")

          return inputTemp

     print("ERROR: READ01")

     raise CalcError("Error en sistema")

### Parámetros

*tipo*: tipo de dato esperado a recibir por el usuario.

### Retorno

Regresa el dato recibido por el usuario.

## CloseOutput

### Función

En caso de que el output grafico haya sido utilizado, corre la ultima instrucción de cierre: *exitonclick*, evita que el output grafico se cierre automáticamente en cuanto termine de ejecutar las instrucciones.

### Código

def closeOutput():

     # Comprueba que el output grafico haya sido abierto

     if variables["output active"] == True:

          turtle.exitonclick()

## Callf

### Función

Ejecuta la función necesaria para manipular el output grafico.

### Código

def callf(func,param):

     # Llama a abrir el output grafico

     openOutput()

     if func == "PenUp":

          turtle.penup()

     elif func == "PenDown":

          turtle.pendown()

     elif func == "Point":

          turtle.dot(10,0,0,0)

     elif func == "Turn":

          turtle.left(param[0])

     elif func == "Line":

          turtle.forward(param[0])

     elif func == "Color":

          turtle.pencolor(param[0])

     elif func == "Size":

          turtle.pensize(param[0])

     else:

          print("ERROR: CALLF01")

          raise CalcError("Error en sistema")

### Parámetros

*func*: contiene el nombre de la función que se quiere ejecutar.

*param*: contiene los valores de los parámetros necesarios para la función de ser necesario.

## OpenOutput

### Función

Ejecuta la primera instrucción necesaria para abrir el output grafico. Utiliza un booleano global para asegurarse de solo ejecutar la instrucción una vez.

### Código

def openOutput():

     # Comprueba que el output grafico no haya sido abierto ya

     if variables["output active"] == False:

          variables["output active"] = True

          turtle.shape("turtle")

## Call

### Función

Utiliza un contador para ejecutar en orden cada instruccion de la funcion *function*, por cada instruccion encontrada manda a llamar la correspondiente instruccion o funcion necesaria.

### Código

def call(function,param,var):

     contador = 0

     while contador < len(variables[function]["run"]):

          a = variables[function]["run"][contador][0]

          b = variables[function]["run"][contador][1]

          c = variables[function]["run"][contador][2]

          d = variables[function]["run"][contador][3]

          if a == "read":

               # Recibe el apuntador a la variable

               vard = buscaVariable(param,var,d)

               vard["value"] = read(vard["type"])

          elif (a == '\*' or a == '/' or a == '-' or a == '+' or a == '==' or a == '>' or a == '<' or a == '<=' or a == '>=' or a == '!=' or a == '&' or a == '|'):

               # Recibe el apuntador a la variable

               varb = buscaVariable(param,var,b)

               # Comprueba que la variable se encuentre inicializada

               if varb.get("value") == None:

                    print("ERROR: Variable no inicializada")

                    raise CalcError("Expresion invalida")

               varc = buscaVariable(param,var,c)

               if varc.get("value") == None:

                    print("ERROR: Variable no inicializada")

                    raise CalcError("Expresion invalida")

               vard = buscaVariable(param,var,d)

               # Recibe el resultado de la operacion

               vard["value"] = op(a,varb["value"],varc["value"])

          elif a == '=':

               # Recibe el apuntador a la variable

               varb = buscaVariable(param,var,b)

               # Comprueba que la variable se encuentre inicializada

               if varb.get("value") == None:

                    print("ERROR: Variable no inicializada")

                    raise CalcError("Expresion invalida")

               vard = buscaVariable(param,var,d)

               # Ejecuta la asignacion

               vard["value"] = varb["value"]

          elif a == "callr" or a == "call":

               # Activa la nueva funcion a ejecutar

               variables["active sys"] = b

               # Hace una copia local de los parametros

               tparam = copy.deepcopy(variables[b]["param"])

               tcont = 0

               # Inicializa cada parametro con el valor recibido

               for x in tparam:

                    varc = buscaVariable(param,var,c[tcont])

                    if varc.get("value") == None:

                         print("ERROR: Variable no inicializada")

                         raise CalcError("Expresion invalida")

                    tparam[x]["value"] = varc["value"]

                    tcont = tcont + 1

               if a == "callr":

                    # Ejecuta la funcion con parametros locales, ya inicializados, y envia una copia local de las variables

                    # Guarda el retorno de la funcion en una variable temporal

                    temp = call(b,tparam,copy.deepcopy(variables[b]["var"]))

                    # Activa la funcion anterior

                    variables["active sys"] = function

                    # Si la funcion regresa un valor, lo guarda en la variable asignada

                    if temp == "Sys None":

                         print("ERROR: No llego a un return la funcion")

                         raise CalcError("Estatuto faltante")

                    else:

                         vard = buscaVariable(param,var,d)

                         vard["value"] = temp

               else:

                    # Ejecuta la funcion con parametros locales, ya inicializados, y envia una copia local de las variables

                    call(b,tparam,copy.deepcopy(variables[b]["var"]))

                    # Activa la funcion anterior

                    variables["active sys"] = function

          elif a == "gotof":

               varb = buscaVariable(param,var,b)

               # Si el valor de la variable es falso, actualiza el contador

               if varb["value"] == False:

                    contador = contador + d - 1

          elif a == "goto":

               # Actualiza el contador

               contador = contador + d - 1

          elif a == "return":

               vard = buscaVariable(param,var,d)

               if vard.get("value") == None:

                    print("ERROR: Variable no inicializada")

                    raise CalcError("Expresion invalida")

               # Regresa el valor de la variable

               return vard["value"]

          elif a == "write":

               vard = buscaVariable(param,var,d)

               if vard.get("value") == None:

                    print("ERROR: Variable no inicializada")

                    raise CalcError("Expresion invalida")

               # Imprime el valor de la variable

               print(vard["value"])

          elif a == "callf":

               tempValues = []

               # Crea un listado con los valores de los parametros recibidos

               for value in c:

                    varc = buscaVariable(param,var,value)

                    tempValues.append(varc["value"])

               # Llama a ejecutar la funcion especial

               callf(b,tempValues)

          else:

               print("ERROR: CALL01")

               raise CalcError("Error en sistema")

          contador = contador + 1

     return "Sys None"

### Parámetros

*function*: nombre de la función a ejecutar.

*param*: parámetros de la función que se ejecuta.

*var*: variables de la función que se ejecuta.

### Retorno

Valor de retorno de la función o el string “Sys None” en caso de no llegar a un return.

## Op

### Función

Regresa el resultado de la operación enviada.

### Código

def op(op,opdo1,opdo2):

     if op == '\*':

          return opdo1 \* opdo2

     elif op == '/':

          return opdo1 / opdo2

     elif op == '-':

          return opdo1 - opdo2

     elif op == '+':

          return opdo1 + opdo2

     elif op == '==':

          return opdo1 == opdo2

     elif op == '<':

          return opdo1 < opdo2

     elif op == '>':

          return opdo1 > opdo2

     elif op == '<=':

          return opdo1 <= opdo2

     elif op == '>=':

          return opdo1 >= opdo2

     elif op == '!=':

          return opdo1 != opdo2

     elif op == '&':

          return opdo1 and opdo2

     elif op == '|':

          return opdo1 or opdo2

     else:

          print("ERROR: OP01")

          raise CalcError("Error en sistema")

### Parámetros

*op*: signo de la ecuación a ejecutar.

*opdo1*: primer numero de la ecuación.

*opdo2*: segundo numero de la ecuación.

### Retorno

Resultado de la ecuación.

## BuscaVariable

### Función

Busca la variable especificada en *var* entre las variables globales y locales.

### Código

def buscaVariable(temparam,tempvar,var):

     # Busca la variable en entre las variables globales

     if variables["main"]["var"].get(var) != None:

          return variables["main"]["var"][var]

     elif variables["active sys"] != "main":

          # Si la funcion activa no es main, busca entre los parametros y variables locales recibidos

          if tempvar.get(var) != None:

               return tempvar[var]

          elif temparam.get(var) != None:

               return temparam[var]

     print("ERROR: Variable no encontrada")

     raise CalcError("Variable invalida")

### Parámetros

*temparam*: apuntador a los parámetros de la función activa.

*tempvar*: apuntador a las variables de la función activa.

*var*: nombre de la variable a buscar.

### Retorno

Apuntador a la variable encontrada.

## BuscaTipo

### Función

Busca el tipo del dato recibido, ya sea una constante o una variable.

### Código

def buscaTipo(tipo):

     if type(tipo) == type(1):

          return "int"

     elif type(tipo) == type(1.0):

          return "float"

     elif tipo[0] == "'":

          return "char"

     elif tipo[0] == '"':

          return "str"

     elif (tipo == "true" or tipo == "false"):

          return "bool"

     elif variables["main"]["var"].get(tipo) != None:

          return variables["main"]["var"][tipo]["type"]

     elif variables["active sys"] != "main":

          if variables[variables["active sys"]]["var"].get(tipo) != None:

               return variables[variables["active sys"]]["var"][tipo]["type"]

          elif variables[variables["active sys"]]["param"].get(tipo) != None:

               return variables[variables["active sys"]]["param"][tipo]["type"]

     print("ERROR: Tipo de variable no encontrada")

     raise CalcError("Variable invalida")

### Parámetros

*tipo*: constante o variable a la cual buscar el tipo de dato.

### Retorno

Tipo de dato encontrado.