Laboratorio 1 Construcción de Compiladores

Reglas de tipos:

Ø ⊢true꞉Bool

Ø ⊢false꞉Bool

Ø ⊢[0-9]+ ꞉Int

Ø ⊢'"' ('\\' [btnfr"'\\] | ~[\r\n\\"])* '"'꞉String

Ø ⊢void꞉ void

Ø ⊢ “ClassName” : ClassType

Ø ⊢self꞉SelfType

Ø ⊢ "Main" : ClassType (Todo programa en YAPL debe contener una clase Main)

Γ ⊢ "main" : MainMethodType (La clase Main debe contener un método main sin parámetros formales)

Γ ⊢ "Int" : DataType

Γ ⊢ "String" : DataType

Γ ⊢ "Bool" : DataType

Reglas de concordancia de tipos:

- En asignaciones, llamadas a métodos y retornos de métodos, los tipos deben coincidir o ser compatibles por herencia.
- No se permiten conversiones explícitas entre tipos no relacionados.

Reglas para constructores:

- Toda clase debe definir un constructor sin parámetros que inicialice los atributos a sus valores por defecto.
- Es posible definir constructores adicionales que reciban parámetros para inicializar atributos.
- Dentro de un constructor, se permite el acceso directo a los atributos de la clase, incluso antes de que sean declarados.

Diseño de tabla de símbolos:

Agregar símbolo (AgregarSimbolo): Este método permitirá agregar nuevos símbolos a la tabla. Recibirá como parámetros el nombre del símbolo, su tipo y su ámbito, y los almacenará en la tabla.

Buscar símbolo (BuscarSimbolo): Este método buscará un símbolo en la tabla por su nombre y devolverá la información asociada a él, si se encuentra. En caso contrario, devolverá un indicador de error.

Imprimir tabla (ImprimirTabla): Este método imprimirá en pantalla la tabla de símbolos de forma legible.símbolos.

Sistema de tipos y operaciones:

Declaraciones de Clases:

$\Gamma \vdash$ "Main" : ClassType

Γ, "Main" : ClassType $\vdash$ "main" : MainMethodType

Tipos de Datos:

$\Gamma \vdash$ "Int" : DataType

$\Gamma \vdash$ "String" : DataType

$\Gamma \vdash$ "Bool" : DataType

Operaciones Aritméticas y de Comparación:

$\Gamma \vdash a : Int$

$\Gamma \vdash b : Int$

$\Gamma \vdash a + b : Int$

$\Gamma \vdash a - b : Int$

$\Gamma \vdash a * b : Int$

$\Gamma \vdash a / b : Int$ (siempre que $b \neq 0$)

$\Gamma \vdash a : Bool$

$\Gamma \vdash b : Bool$

$\Gamma \vdash a \wedge b : Bool$

$\Gamma \vdash a \vee b : Bool$

$\Gamma \vdash \neg a : Bool$

$\Gamma \vdash a == b : Bool$ (si a y b son del mismo tipo)

$\Gamma \vdash a \neq b : Bool$ (si a y b son del mismo tipo)

Asignaciones:

$\Gamma \vdash a : T$

$\Gamma, x : T \vdash x = a : T$

Llamadas a funciones:

Dado un tipo de retorno T y una lista de argumentos de tipo T1, T2, ..., Tn:

$\Gamma \vdash f : T1 \times T2 \times ... \times Tn \rightarrow T$

$\Gamma \vdash a1 : T1, a2 : T2, ..., an : Tn$

$\Gamma \vdash f(a1, a2, ..., an) : T$

Casteo:

$\Gamma \vdash a : Bool$

$\Gamma \vdash b : Int$

$\Gamma \vdash$ Bool to Int (a) : Int (False es 0, True es 1)

$\Gamma \vdash$ Int to Bool (b) : Bool (0 es False, cualquier valor positivo es True)

Ámbito:

Un atributo dentro de una clase debe ser declarado antes de su uso. Un método dentro de una clase puede ser llamado de forma recursiva. Existen dos ámbitos dentro de una clase: el ámbito Global y el ámbito Local. Todos los atributos y métodos dentro de una clase tienen acceso público. Los identificadores de un ámbito local ocultan la definición de identificadores en el ámbito global. Ningún identificador puede ser definido más de una vez dentro de un mismo ámbito.

- Atributos y métodos tienen acceso público.
- Identificadores locales ocultan identificadores globales.
- No redefinición de identificadores en un ámbito.
- Métodos heredados deben mantener su firma original.

Herencia:

Si B hereda de A y B sobrescribe un método de A, este método debe poseer la misma firma con la que fue declarado en A. No es posible la herencia múltiple de clases y la herencia recursiva.

Valores Default:

Los objetos creados a partir de la clase Int, poseen valor default 0. Los objetos creados a partir de la clase String, poseen valor default "" (cadena vacía). Los objetos creados a partir de la clase Bool, poseen valor default false.

- Int: 0
- String: ""
- Bool: false

Estructuras de Control:

El tipo de dato estático de la <expr> utilizada en una estructura de control if o while debe ser de tipo Bool. El tipo de dato del condicional if es el tipo de dato del bloque que sea un supertipo de ambas ramas del condicional. El tipo de dato de la estructura while es Object.

- Condicionales if requieren expresión Bool.
- Tipo de if es el supertipo de ambas ramas.
- Tipo de while es Object.

Clases Especiales:

Existe una clase especial IO que define funciones de entrada y salida de valores tipo Int y Bool. La tabla de símbolos debe de contener de forma predefinida la definición de las clases IO, Int, String y Bool junto a sus métodos ya definidos.

- Existencia de clase IO predefinida.
- La clase Object define el método toString().
- Todas las clases heredan de Object.

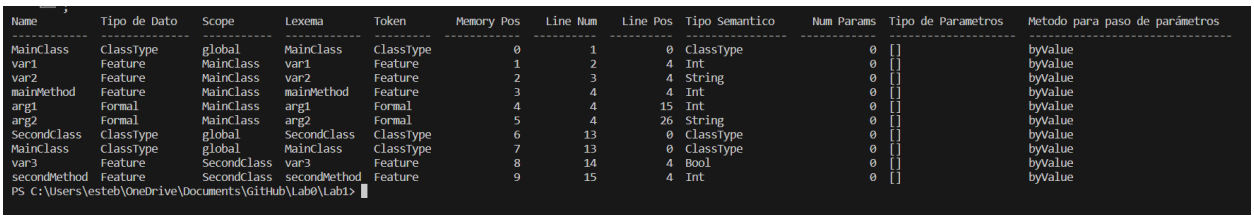
Manejo de excepciones:

- Detección y manejo de excepciones durante la ejecución.
- Mecanismo de captura y manejo de excepciones.

Tabla de Símbolos implementada en Código

Correr dentro del programa

Python LAB1.py input.txt



Name	Tipo de Dato	Scope	Lexema	Token	Memory Pos	Line Num	Line Pos	Tipo Semantico	Num Params	Tipo de Parametros	Metodo para paso de parámetros
MainClass	ClassType	global	MainClass	ClassType	0	1	0	ClassType	0	[]	byValue
var1	Feature	MainClass	var1	Feature	1	2	4	Int	0	[]	byValue
var2	Feature	MainClass	var2	Feature	2	3	4	String	0	[]	byValue
mainMethod	Feature	MainClass	mainMethod	Feature	3	4	4	Int	0	[]	byValue
arg1	Formal	MainClass	arg1	Formal	4	4	15	Int	0	[]	byValue
arg2	Formal	MainClass	arg2	Formal	5	4	26	String	0	[]	byValue
SecondClass	ClassType	global	SecondClass	ClassType	6	13	0	ClassType	0	[]	byValue
MainClass	ClassType	global	MainClass	ClassType	7	13	0	ClassType	0	[]	byValue
var3	Feature	SecondClass	var3	Feature	8	14	4	Bool	0	[]	byValue
secondMethod	Feature	SecondClass	secondMethod	Feature	9	15	4	Int	0	[]	byValue

PS C:\Users\esteb\OneDrive\Documents\GitHub\Lab0\Lab1>