Instituto Tecnológico de Costa Rica
Escuela de Computación
Ingeniería en Computación
Sede Central

IC-5701 Compiladores e Intérpretes

Proyecto II: fase de análisis contextual de un compilador

Integrantes:

José Daniel Camacho
carné:2017043395
Andrey Torres Calderón
carné: 2017079723
Eduardo Jirón Alvarado
carné: 2017101878
Kristal Durán Araya
carné: 2017238958

Profesor: Ignacio Trejos Zelaya

I Semestre,
2019

Tabla contenidos

Introducción

El siguiente informe es una documentación externa del segundo proyecto del curso de compiladores e intérprete. En este documento se explican las soluciones a los principales problemas atacados, también enumera una lista de aspectos importantes como: los errores contextuales definidos, las pruebas positivas y negativas realizadas agregando los resultados obtenidos a cada una de ellas, y las tareas realizadas divididas según la elaboración por miembro del equipo de trabajo. Se indican además las conclusiones y experiencias al finalizar este proyecto.

Este proyecto tiene el objetivo de retar a los estudiantes para conseguir extender el analizador contextual que posee Triángulo agregando retos en cuanto a algunos ciclos, manejo de recursividad, privacidad de variables, paquetes, entre otros. Al final de este documento se encuentra un enlace para acceder al enunciado original y la última versión proporcionada por el profesor Ignacio Trejos.

Este proyecto se desarrolla como una segunda fase tomando como base el proyecto uno: análisis sintáctico, desarrollado por los mismo estudiantes: Jose Daniel Camacho, Kristal Durán Araya, Eduardo Jiron y Andrey Torres.

Desarrollo

Este apartado pretende mostrar una descripción general a la forma de abordar las soluciones de los principales problemas propuestos. Para ello se lista una serie de preguntas proporcionadas por el profesor Ignacio Trejos. También se muestra un listado de errores detectados, y un cuadro para mostrar las pruebas positivas y negativas que se realizaron.

1) Describir la manera en que se comprueban los tipos para todas las variantes de loop … end.

   Para la comprobación de variables utilizadas en el loop … end se realiza una validación tomando como base el tipo en que fue declarados los identificadores y las expresiones. La siguiente lista explica los datos validados según el ciclo.

   - **loop while do y loop until do**: En ambos casos es necesario considerar que no hay un identificador y que la expresión existente debe ser de tipo booleano.
   - **loop do while y loop do until**: En ambos casos es necesario considerar que no hay un identificador y que la expresión existente debe ser de tipo booleano.
   - **loop for do:** En este caso es necesario considerar que hay un identificador, y dos expresiones las cuales deben ser de tipo numérico.

- **loop for while do y loop for until do**: En ambos casos es necesario considerar que hay un identificador, y tres expresiones, donde las primeras dos deben ser de tipo numérico y la tercera expresión de tipo booleano.

Para determinar el tipo de las expresiones se utiliza se crea un nuevo objeto haciendo un parser de tipo TypeDenoter y una validación de este utilizando la sentencia “instanceof” y así determinar si es un IntTypeDenoter o BoolTypeDenoter.

2) Describir la solución dada al manejo de alcance, del tipo y de la protección de la variable de control del loop for … end.

El manejo del alcance de las variables e identificadores en los loop for … end se consigue utilizando el método openScope() ya implementado en el proyecto base proporcionado por el profesor Ignacio Trejos. El objetivo de esto es definir un nuevo nivel donde se incluye los identificadores y en caso de existir una tercer expresión se incluye esta y el comando permitiendo así que los valores definidos en este nivel conozcan del identificador.

3) Describir la solución creada para resolver las restricciones contextuales del comando choose Exp from Cases end.

Primero se comprueban todos los tipos, al hacer el visit de los int|char literal, ellos retornan su tipo, estos tipos se van comparando de abajo hacia arriba del árbol hasta llegar a la raíz asegurando que todos los tipos son iguales de lo contrario indica el error

Segundo, para comparar los rangos una vez verificados los tipos, se usa una lista dinámica a la cual se le añaden los elementos de cada rango del choose, siempre comparando que no existan elementos repetidos hasta terminar, en el momento que la lista contiene al menos 2 elementos repetidos significa que hay un overlap entre los rangos.

Lo anterior se hace a nivel del AST local del choose, por lo tanto no verifica tipos ni rangos de chooses anidados, lo cual está bien porque un choose anidado no debe interferir con su “padre”.

4) Describir el procesamiento de la declaración de variable inicializada (var Id ::= Exp).

Lo primero que ocurre cuando se llega a una variable inicializada es que se toma la expresión y se evalúa para asignar estáticamente el tipo de la variable y se le nombra como una variable para después poder ser reasignada, luego se introduce el nombre de la variable en la tabla de símbolos y se busca por si acaso existen otros elementos de la tabla con el mismo nombre para notificar el error.

5) Descripción de su validación de la unicidad de los nombres de parámetros en las declaraciones de funciones o procedimientos.

6) Describir la manera en que se procesa la declaración compuesta private. Interesa que la primera declaración introduzca identificadores que son conocidos privadamente (localmente) por la segunda declaración; se "exporta" solamente lo introducido por la segunda declaración.

Primero se utiliza una función en la tabla de símbolos para abrir un scope privado, esto quiere decir que llegará el momento en el que esas declaraciones que se están introduciendo en este scope van a ser eliminadas de la tabla, y se empiezan a agregar las declaraciones, cuando se terminan de agregar se cierra el scope privado y se procesa el resto del private, una vez se termina de procesar el resto entonces se realiza un clear private scope lo que hace esto es eliminar todos los nodos marcados como privados en la tabla de símbolos y se continúa procesando el resto del árbol.

7) Describir la manera en que se procesa la declaración compuesta par. Interesa la no-repetición de los identificadores introducidos por cada declaración colateral. Los identificadores declarados en una misma declaración compuesta par se "exportan" para ser conocidos en el resto del bloque donde aparece el par.

   En par primero se visita el árbol de manera que cada declaración abre su propio scope, se agregan las declaraciones a la tabla y se trabajan y cuando se termina de procesar se cierra el scope para que todas las declaraciones tengan el mismo contexto y una vez asegurado de que ninguna se use entre sí entonces se visitan todas las declaraciones como se haría normalmente y se agregan a la tabla de símbolos para ser usadas posteriormente.

8) Describir el procesamiento de la declaración compuesta recursive. Interesa la no-repetición de los identificadores (de función o procedimiento) declarados y que estos identificadores sean conocidos en los cuerpos de las funciones o procedimientos declarados en una misma declaración compuesta recursive. Nuevas rutinas de análisis contextual, así como cualquier modificación a las existentes.

   Utilizando el objeto nulo que se pasa en los visits se implementó un tipo de bandera que indica cuando se está procesando una declaración recursive, así se procede a primero realizar los inserts de todos los identificadores de proc / func a la tabla, en caso de que alguno sea repetido dará error por la implementación de la tabla para verificar identificadores repetidos, posteriormente se procede a procesar la lista de parámetros dentro de un scope nuevo junto con el cuerpo de cada procedimiento y función, sin embargo para este último paso no se logró procesar las llamadas a funciones / procedimientos dentro del cuerpo de los mismos .

Lista de errores contextuales detectados

- "Boolean expression expected here"
- "\"%\" is not a binary operator"
- "incompatible argument types for \"%\""
- "wrong argument type for \"%\""
- "\"%\" is not a function identifier"
- "incompatible limbs in if-expression"
- "\"%\" is not a unary operator
- "identifier \"%\" already declared"

- "wrong expression type, must be an integer type"
- "body of function \"%\" has wrong type"
- "the expression type must be char or integer"
- "the expression type and the literals must be of the same type"
- "the ranges cannot overlap"
- "the range expressions cannot be the same"
- "the range case must have at least one case"
- "all the cases literals must match all charLiteral or all intLiteral"
- "the range case must be charLiteral..charLiteral or intLiteral..intLiteral"
- "incompatible array-aggregate element"
- "duplicate field \"%\" in record"
- "duplicated formal parameter \"%\""
- "const actual parameter not expected here"
- "wrong type for const actual parameter"
- "func actual parameter not expected here"
- "wrong signature for function \"%\""
- "wrong type for function \"%\""
- "\"%\" is not a procedure identifier"
- "proc actual parameter not expected here"
- "wrong signature for procedure \"%\""
- "actual parameter is not a variable"
- "var actual parameter not expected here"
- "wrong type for var actual parameter"
- "too few actual parameters"
- "too many actual parameters"
- "incorrect number of actual parameters"
- "arrays must not be empty"
- "\"%\" is not a type identifier"
- "record expected here"
- "no field \"%\" in this record type"
- "\"%\" is not a const or var identifier"
- "array expected here"
- "Integer expression expected here"

Plan de pruebas para validar el compilador. Debe separar las pruebas para la fase de análisis contextual.
Análisis de la ‘cobertura’ del plan de pruebas (interesa que valide tanto el funcionamiento "normal" como la capacidad de detectar errores contextuales). En las pruebas del analizador contextual es importante comprobar que los componentes de análisis léxico y sintáctico siguen funcionando bien. Si hacen correcciones a ellos, deben documentar los cambios en un apéndice de este Proyecto #2.

Pruebas positivas

Objetivo	Diseño	Resultado esperado	Resultado observado
Loop_for_while_do_end Todos los tipos de expresiones son correctos	loop for id from 5 to 6 while 9<2 do pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_for_while_do_end La expresión 3 tiene alcance de identificador	loop for id from 5 to 6 while id<2 do pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_for_while_do_end Validar que el comando tenga alcance del identificador	loop for id from 5 to 6 while 9<2 do let const i ~ id in pass end end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_while_do Validar estructura correcta	loop while 1=1 do pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_until_do Validar estructura correcta	loop until 1<0 do pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_do_while Validar estructura correcta	loop do pass while 3=3 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_do_until Validar estructura correcta	loop do pass until 3=3 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Loop_for_do Validar estructura correcta	loop for i from 3 to 9 do pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was

			successful.
Loop_for_do Comando tiene alcance de identificador	loop for id from 2 to 9 do let const i ~ id in pass end end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.
loop_for_until_do Verificar estructura Expresión 3 tiene alcance de identificador	loop for i from 5 to 6 until i < 5 do pass;pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
loop_for_until_do Comando tiene alcance de identificador	loop for id from 9 to 9 until id<2 do let const i ~ id in pass end end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
choose Rangos y choose anidado validación de overlaps correcta	let var a:Integer in choose a from when 1..2|4..5|6 then choose 2 from when 1..8 then pass when 10 then pass end when 7..12 then pass else!<--defaultcase pass end end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.

packages
declaración secuencial de packages y uso de variables declaradas dentro de packages con packageId$IdentificadorDeVariable
	package myP ~
	var a: Integer;
	var b: Integer
	
	end;
	package mypackage ~
	var c: Integer
	end;
	let
	var d: Integer
	in
	putint(c);
	putint (myP $ a)
	end
		Compilación satisfactoria	Syntactic Analysis ...
			Contextual Analysis
			Compilation was successful.

If Validar estructura correcta	if 4>3 then puteol() else pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Par Valida estructura	let par var x ::= 1 | const y ~ 1 | var z ::= 9 | var p : Integer end in x := 2; p := 2 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Array Validación de estructura	let var x:array 10 of Integer in pass end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Private Validación de estructura	let private func x() : Boolean ~ 1>3 in var p : Integer end in p:=1 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis Compilation was successful.
Private anidado	let private private var x ::= 1 in var y ::= x - 1 end in	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.

	var b ::= y + 1 end ; var p : Integer in p := b - 1 end		
let validación de estructura	let var a:Integer in a := 2 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.
var validación de estructura	let var a:Integer in a := 2 end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.
record validación de estructura	let type fecha~record d:Integer, m:Integer, y:Integer end in let var a:fecha in pass end end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.
proc validación de estructura	let proc hola(mensaje:Char)~ put(mensaje) end; var msj:Char in msj := 'h'; hola(msj) end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.
func validación de estructura	let func suma(a:Integer, b:Integer):Integer~ a + b; var a:Integer in a := suma(12,12) end	Compilación satisfactoria	Syntactic Analysis ... Contextual Analysis ... Compilation was successful.

Pruebas Negativas

Objetivo	Diseño	Resultado esperado	Resultado observado
Loop_for_while_do_end Validar expresión 1	loop for id from 5<4 to 6 while 9<3 do pass end	Resultado esperado: Expresión 1 tipo erróneo, debe ser integer	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be an integer type 1..0 Compilation was unsuccessful.
Loop_for_while_do_end Validar expresión 2	loop for id from 5 to 6<0 while 9<3 do pass end	Resultado esperado: Expresión 2 tipo erróneo, debe ser integer	Syntactic Analysis ... Contextual Analysis ERROR: Integer expression expected here 1..0 Compilation was unsuccessful.
Loop_for_while_do_end Validar expresión 3	loop for id from 5 to 6 while 9 do pass end	Resultado esperado: Expresión 3 tipo erróneo, debe ser boolean	Syntactic Analysis ... Contextual Analysis ... ERROR: Boolean expression expected here 1..1 Compilation was unsuccessful.
Loop_for_while_do_end Validar existencia de identificador (error sintáctico)	loop for 7 from 5 to 6 while 9<2 do pass end	Requiere identificador	Syntactic Analysis ... ERROR: identifier expected here 1..1 Compilation was unsuccessful.
Loop_for_while_do_end Validar que no se utilice el identificar como variable para asignarle otro valor en el comando	loop for id from 5 to 6 while 9<2 do id:=3 end	Identificador no se le puede asignar valor	Syntactic Analysis ... Contextual Analysis ... ERROR: LHS of assignment is not a variable 1..1 Compilation was unsuccessful.
Loop_for_while_do_end Validar que el paso por parámetro del id no puede hacerse	loop for id from 5 to 6 while 9<2 do put(id) end	Id no puede ser pasado por parámetro	Syntactic Analysis ... Contextual Analysis ERROR: wrong type for const actual parameter 3..3 Compilation was

			unsuccessful.
Loop_while_do Validar tipo expresión	loop while 1 do pass end	Expresión 1 debe ser tipo boolean	Syntactic Analysis ... Contextual Analysis ERROR: Boolean expression expected here 1..1 Compilation was unsuccessful.
Loop_until_do Validar tipo expresión	loop until 1 do pass end	Expresión 1 debe ser tipo boolean	Syntactic Analysis ... Contextual Analysis ERROR: Boolean expression expected here 1..1 Compilation was unsuccessful.
Loop_do_while Validar tipo expresión	loop do pass while 3 end	Expresión 1 debe ser tipo boolean	Syntactic Analysis ... Contextual Analysis ... ERROR: Boolean expression expected here 1..1 Compilation was unsuccessful.
Loop_do_until Validar tipo expresión	loop do pass until 3 end	Expresión 1 debe ser tipo boolean	Syntactic Analysis ... Contextual Analysis ... ERROR: Boolean expression expected here 1..1 Compilation was unsuccessful.
loop_for_do Validar existencia de identificador (Error sintáctico)	loop for 9 from 7 to 9 do pass end	Requiere identificador	Syntactic Analysis ... ERROR: identifier expected here 1..1 Compilation was unsuccessful.
loop_for_do Validar expresión 1	loop for id from 5=8 to 6 do pass;pass end	Expresión 1 debe ser tipo integer	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be

			an integer type 1..0 Compilation was unsuccessful.
loop_for_do Validar expresión 2	loop for id from 8 to 9=10 do pass;pass end	Expresión 2 debe ser tipo integer	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be an integer type 1..0 Compilation was unsuccessful.
loop_for_do Expresión 1 no tiene el alcance de identificador	loop for id from 8 to id do pass;pass end	El tipo de expresión uno no es integer	Syntactic Analysis ... Contextual Analysis ERROR: "do" is not declared 1..1 ERROR: wrong expression type, must be an integer type 1..1 Compilation was unsuccessful.
loop_for_do Expresión 2 no tiene el alcance de identificador	loop for id from id to id 9 do pass;pass end	El tipo de expresión uno no es integer	Syntactic Analysis ... Contextual Analysis ERROR: "to" is not declared 1..1 ERROR: wrong expression type, must be an integer type 1..1 Compilation was unsuccessful.
loop_for_until_do Validar expresión 1	loop for i from 5=8 to 6 until i < 5 do pass;pass end	Expresión 1 debe ser tipo integer	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be an integer type 1..0 ERROR: wrong argument type for "<" 1..1 ERROR: wrong argument type for "<" 1..1 Compilation was unsuccessful.

loop_for_until_do Validar expresión 1	loop for i from 5 to 6=8 until i < 5 do pass;pass end	Expresión debe ser de tipo integer	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be an integer type 1..0 Compilation was unsuccessful.
loop_for_until_do Validar existencia identificador (error sintáctico)	loop for 9 from 5 to 6 until i < 5 do pass;pass end	Requiere identificador	Syntactic Analysis ... ERROR: identifier expected here 1..1 Compilation was unsuccessful.
loop_for_until_do Variables de la expresión 3	loop for i from 5 to 6 until id < 5 do pass;pass end	La variable id no existe	Syntactic Analysis ... Contextual Analysis ERROR: "<" is not declared 1..1 ERROR: "<" is not declared 1..1 Compilation was unsuccessful.
loop_for_until_do Validar expresión 3	loop for i from 5 to 6 until 5 do pass;pass end	La expresión 3 debe ser tipo booleano	Syntactic Analysis ... Contextual Analysis ERROR: wrong expression type, must be an boolean type 1..1 Compilation was unsuccessful.
choose validación de tipos	let var a:Char in choose a from when 1..2|4..5|6 then pass else!<--defaultcase pass end end	El tipo de la variable choose es diferente al de los rangos	Syntactic Analysis ... Contextual Analysis ... ERROR: the expression type and the literals must be of the same type 4..4 Compilation was unsuccessful.
choose overlap de rangos	let var a:Integer	No puede haber overlap de rangos	Syntactic Analysis ... Contextual Analysis ...

	in choose a from when 1..2|4..5|6 then pass when 4 then pass else!<--defaultcase pass end end		ERROR: the ranges cannot overlap 6..6 Compilation was unsuccessful.

packages
identificadores iguales de package
	package myP ~
	var a: Integer;
	var b: Integer
	end;
	package myP ~
	var c: Integer
	end;
	let
	var d: Integer
	in
	putint(c);
	putint (myP $ a)
	end
		No puede haber identificadores de paquetes iguales	Syntactic Analysis ...
			Contextual Analysis ...
			ERROR: packageIdentifier "myP" already declared 6..8
			Compilation was unsuccessful.

packages
referencia de una variable de otro paquete
	package myP ~
	var a: Integer;
	var b: Integer
	end;
	package mypackage ~
	var c: Integer
	end;
	let
	var d: Integer
	in
	putint(c);
	putint (mypackage $ a)
	end
		No se puede referenciar a una variable de otro paquete	Syntactic Analysis ...
			Contextual Analysis ...
			ERROR: variable a doesnt belong to packageIdentifier "mypackage" 12..12
			Compilation was unsuccessful.

If Validar expresión	if 4 then puteol() else pass end	Expresión debe ser de tipo booleano	Syntactic Analysis ... Contextual Analysis ERROR: Boolean

			expression expected here 1..1 Compilation was unsuccessful.
Par ERROR La declaración 1 no conoce a la declaración 2	let par var x ::= 1 | const y ~ x | var z ::= 9 | var p : Integer end in x := 2; p := 2 end	ERROR variables x no esta definida	Syntactic Analysis ... Contextual Analysis ... ERROR: "x" is not declared 2..2 ERROR: "x" is not declared 2..2 ERROR: "x" is not declared 2..2 ERROR: "x" is not declared 2..2 Compilation was unsuccessful.
Par ERROR Los identificadores deben ser únicos	let par var x ::= 1 | const y ~ 1 | var x ::= 9 | var p : Integer end in x := 2; p := 2 end	El identificador x ya fue declarado	Syntactic Analysis ... Contextual Analysis ... ERROR: identifier "x" already declared 2..2 ERROR: identifier "x" already declared 2..2 Compilation was unsuccessful.
Private ERROR El identificador no es visible	let private func x() : Boolean ~ 1>3 in var p : Integer end in x() end	El identificador x no está declarado	Syntactic Analysis ... Contextual Analysis ... ERROR: "x" is not declared 1..1 Compilation was unsuccessful.
let ERROR Declaración ya existe	let var a:Integer; var a::= 0 in pass end	El identificador a ya está declarado	Syntactic Analysis ... Contextual Analysis ... ERROR: identifier "a" already declared 1..1 Compilation was unsuccessful.
var ERROR Tipos no concuerdan	let var a:Integer in a := '2' end	Los tipos no son compatibles	Syntactic Analysis ... Contextual Analysis ... ERROR: assignment incompatibilty 3..3

			Compilation was unsuccessful.

record
ERROR
No existe el identificador en record
	let type fecha~record
	  d:Integer,
	  m:Integer,
	  y:Integer
	  end
	in
	  let var a:fecha
	  in
	  a.q := '2'
	  end
	end
		El identificador q no existe en el record fecha	Syntactic Analysis ...
			Contextual Analysis ...
			ERROR: no field "q" in this record type 9..9
			Compilation was unsuccessful.

proc
ERROR
Tipo de parámetro malo
	let proc
	hola(mensaje:Char)~
	  put(mensaje)
	  end;
	var msj:Integer
	in
	  msj := 1;
	  hola(msj)
	end
		El tipo del parámetro no concuerda con lo pasado	Syntactic Analysis ...
			Contextual Analysis ...
			ERROR: wrong type for const actual parameter 7..7
			Compilation was unsuccessful.

func
ERROR
Tipo de retorno no concuerda con la declaración
	let func
	suma(a:Integer,
	b:Integer):Integer~
	  'a';
	var a:Integer
	in
	  a := suma(12,12)
	end
		El tipo de retorno no concuerda con la declaración	Syntactic Analysis ...
			Contextual Analysis ...
			ERROR: body of function "suma" has wrong type 2..2
			Compilation was unsuccessful.

Conclusión

Discusión y análisis de los resultados obtenidos

Para este proyecto se solicita que los estudiantes modifican la estructura de análisis contextual del compilador Triángulo, de este se deben añadir validaciones y manejo de alcances para: loop (while do, until do, do while, do until, for do, for until, for while), choose, var, private, par, recursive, package. De las indicaciones por cambiar se concluyen satisfactorias, cinco de seis, faltando por concluir el cambio de la sentencia recursive, esta misma se modifica sin embargo no se finaliza satisfactoriamente.

Conclusiones a partir de esto

- Se finaliza satisfactoriamente las modificaciones para las sentencias de loop … end
- Se finaliza satisfactoriamente las modificaciones para las sentencias de choose y cases
- Se finaliza satisfactoriamente las modificaciones para las sentencias de var
- Se finaliza satisfactoriamente las modificaciones para las sentencias de private
- Se finaliza satisfactoriamente las modificaciones para las sentencias de par
- Se finaliza satisfactoriamente las modificaciones para las sentencias de package
- No se finaliza satisfactoriamente las modificaciones para las sentencias de recursive
- Se realizan todas las pruebas negativas y positivas para todas las sentencias

Una reflexión sobre la experiencia de modificar fragmentos de un compilador/ambiente escrito por terceras personas.

Al concluir con las indicaciones solicitadas por el profesor nosotros como estudiantes nos sentimos satisfechos al superar los diferentes retos presentes, tanto el tener que comprender el código ya diseñado por otras personas como el poder incluir en este las restricciones necesarias para que al incluir nuevas sentencias estas se ejecuten y se logre realizar un análisis contextual satisfactoria. Los conocimientos adquiridos al pasar de la teoría a la prácticas son muchos y muy enriquecedores.

Anexos

Tareas realizadas por cada miembro del grupo de trabajo

Tarea realizada	Miembro del grupo
Validaciones de loop do while, loop do until	Kristal Durán
Validaciones y manejo de alcances de loop for do, loop for until	Eduardo Jirón
Validaciones y manejo de alcances de loop for until	Kristal Durán
Validaciones y manejo de alcances de cases y choose	José Daniel Camacho
Validaciones y manejo de alcances de par	Andrey Torres
Validaciones y manejo de alcances de private	Andrey Torres y Eduardo Jirón
Validaciones y manejo de alcances de recursive	Jose Daniel Camacho y Kristal Durán
Validaciones y manejo de alcances de proc y func	Andrey Torres
Validaciones y manejo de alcances de var	Andrey Torres
Validaciones y manejo de alcances de package	José Daniel Camacho
Validaciones y manejo de alcances de v-name	
Validaciones y manejo de alcances de array	Eduardo Jirón
Validaciones y manejo de alcances de record	Eduardo Jirón
Diseño de pruebas para loop … end	Kristal Durán
Diseño de pruebas para package	José Daniel Camacho
Diseño de pruebas para choose	José Daniel Camacho
Diseño de pruebas para par	Andrey Torres
Diseño de pruebas para variable inicializada	Andrey Torres
Diseño de pruebas para private	Andrey Torres y Eduardo Jirón

Cómo debe compilarse su programa

Abrir netbeans en un sistema operativo windows 7 o superior y seleccionar la opción de abrir proyecto, busca el directorio donde tiene guardado el proyecto y lo abre. Luego selecciona el botón de clean and build para compilar el proyecto.

Cómo debe ejecutarse su programa

Luego de compilado el proyecto se deberá ir a la carpeta dist se habrá creado un archivo llamado IDE-Triangle.jar, se da doble click en el archivo para ejecutar el programa.

Links:

- Enunciado: https://drive.google.com/file/d/1RALYRBYm7N1-FraYh2Cvh3yBMP4jJVpu/view?usp=sharing