Procesadores de lenguajes. Práctica 8.

EL ÁRBOL DE SINTAXIS ABSTRACTA DE TINTO

Índice

1.1 Estructura del código

- La etapa de análisis del compilador de Tinto tiene como objetivo leer un fichero fuente y generar una estructura de datos que almacene la información contenida en dicho fichero fuente.
- Esta estructura se conoce como “árbol de sintaxis abstracta” o “abstract sintax tree”.
- Para desarrollar esta estructura se utiliza el paquete tinto.ast que incluye otros subpaquetes.
- Paquetes y subpaquetes:
  - tinto.ast:Paquete raíz de todas las clases de la estructura de datos. Contiene las clases que desarrollan el sistema de tipos, utilizados por muchas clases de los distintos subpaquetes.
  - tinto.ast.expresión:Paquete con clases dedicadas a describir las diferentes expresiones aritméticas y lógicas.
  - tinto.ast.statement: Paquete que describe las diferentes sentencias (IF, WHILE, RETURN, etc.)
  - tinto.ast.struct:paquete que describe los componentes de alto nivel (bibliotecas, funciones, etc.)

1.2 tinto.ast

- tinto.ast.Access: define los tipos de acceso “public” y “private”
- tinto.ast.Type: define las constantes para describir los tipos de datos (void, int, char, boolean)
- tinto.ast.TypeSystem: desarrolla los métodos asociados a los tipos:
  - boolean isInteger(int type)
  - boolean isBoolean(int type)
  - boolean isChar(int type)
  - boolean isVoid(int type)
  - boolean isNumeric(int type) verifica que un tipo sea numérico
  - boolean isOrderable(int type1, int type2) verifica que dos tipos se puedan ordenar con > o <
  - boolean isRegularWord(int type) verifica que un tipo se represente en un registro de propósito general 32 bits.
  - int getSize(int type) obtiene el tamaño del tipo de datos (en bytes)

1.3 tinto.ast.struct

- tinto.ast.struct.LibraryDeclaration: clase que contiene información recopilada de un archivo fuente.
  - name: nombre de la biblioteca
  - imported: vector con los diferentes nombres de las bibliotecas importadas
  - function: vector que almacena la lista de las descripciones de las funciones definidas en la biblioteca.
- tinto.ast.struct.Function: una función de una biblioteca contiene los siguientes campos:
  - libname: nombre biblioteca a la que pertenece
  - name: nombre de la función
  - access: tipo de acceso (public o private)
  - type:tipo de dato que retorna
  - argument: lista de argumentos de llamada a la función, Cada argumento es objeto de Variable
  - localVar: lista de variables locales definidas en el cuerpo del método. Cada una es Variable

1.3 tinto.ast.struct

- tinto.ast.struct.Function: una función de una biblioteca contiene los siguientes campos:
  - Body: bloque de instrucciones del método. Objeto de la clase tinto.ast.statement.BlockStatement
  - Métodos dedicados a acceder y a editar estos campos
  - int [] getArgumentType(): obtiene la lista de los tipos de los argumentos. Para detectar que no existan dos funciones con el mismo nombre y los mismos tipos de argumentos.
  - boolean match(String, int[]) verifica si a la función le corresponde ese nombre y esos tipos de argumentos
- tinto.ast.struct.Variable:
  - name: nombre de la variable
  - type: tipo de datos de la variable

1.3 tinto.ast.struct

- tinto.ast.struct.SymbolTable:
  - Esta clase desarrolla la tabla de símbolos del compilador.
  - La tabla de símbolos contiene el conjunto de bibliotecas analizadas por el compilador y se encarga de identificar la biblioteca y la función que se está analizando en cada momento.
  - La tabla de símbolos construye una pila de ámbitos de declaración de variables.
  - Cuando una instrucción de la función activa abre un nuevo ámbito de declaración, se apila un nuevo ámbito en esta clase (por ejemplo, un bloque de instrucciones de un while).
  - Los ámbitos se representan por medio de tablas hash que almacena los nombres de las variables y las referencias a los objetos Variable asociados.

1.3 tinto.ast.struct

- tinto.ast.struct.SymbolTable:Lós métodos son:
  - void addLibrary(LibraryDeclaration lib) añade una biblioteca a la tabla de símbolos
  - LibraryDeclaration getLibrary(String libname) Busca en la tabla de símbolos una biblioteca
  - LibraryDeclaration getActiveLibrary() Obtiene la biblioteca activa
  - void setActiveLibrary(String libname) asigna la biblioteca activa
  - LibraryDeclaration [] getLibraries() obtiene las bibliotecas en un array
  - Function getActiveFunction() obtiene la función activa que se está analizando
  - void setActiveFunction(String name, int [] type) Asigna la función activa
  - void createScope() crea un nuevo ámbito (tabla hash) y lo almacena en la pila
  - void deleteScope() elimina el ámbito almacenado en la cima de la pila
  - void addLocalVariable(Variable var) añade una declaración de variable en la función activa y la almacena en el ámbito de la cima de la pila

1.3 tinto.ast.struct

- tinto.ast.struct.SymbolTable:Lós métodos son:
  - void addArgument(Variable var) Añade un argumento a la función activa y lo almacena en el ámbito de la cima de la pila
  - Variable getVariable(String name) Busca una variable con el nombre indicado en todos los ámbitos de la pila
  - Variable getVariableInScope(String name) Busca una variable con el nombre indicado en el ámbito de la cima de la pila

1.4 tinto.ast.statement

- tinto.ast.statement.Statement:
  - Clase abstracta que sirve de superclase de cualquier clase que defina una instrucción
  - La clase sólo contiene el método abstracto returns(), que debe verificar si la instrucción alcanza siempre un return o no
  - Alcanzan siempre return:
    - Instrucción return
    - Bloque de instrucciones que terminen en return
    - Instrucción if-else
- tinto.ast.statement.IfStatement: clase que describe instrucción if.
  - condition: Condición de la instrucción if. Referencia a un objeto Expression. De tipo Boolean
  - thenInst: instrucción a ejecutar cuando la condición sea cierta. Referencia objeto Statement.
  - elseInst: instrucción a ejecutar cuando la condición sea falsa. Referencia objeto Statement. Sin else nulo.
  - Returns() devuelve cierto si las instrucciones thenInst y elseInst alcanzan un return

1.4 tinto.ast.statement

- tinto.ast.statement.WhileStatement: clase que describe instrucción while
  - condition: condición de control del bucle. Referencia a un objeto Expression de tipo boolean.
  - instruction: describe el cuerpo del bucle. Referencia a un objeto Statement
  - El método returns devuelve siempre falso
- tinto.ast.statement.ReturnStatement: Describe una instrucción return
  - expression: expresión asociada a la instrucción return. Referencia a objeto Expression. El tipo de expresión debe coincidir con el tipo devuelto por la función. Si la instrucción es vacía devueleve nulo.
  - Returns() devuelve siempre cierto

1.4 tinto.ast.statement

- tinto.ast.statement.AssignStatement: Describe una instrucción de asignación
  - lefthand: Contiene la referencia a la variable que aparece en la parte izquierda de la asignación. Es un objeton Variable
  - expression: representa la parte derecha de la asignación, la expresión que permite calcular el valor a asignar. Referencia a un objeto Expression. El tipo de la expresión debe coincidir con el tipo de la variable.
  - Returns() devuelve siempre falso.
- tinto.ast.statement.CallStatement: instrucción de llamada a una función.
  - exp: contiene la referencia al objeto Expression que describe la expresión de llamada a la función.
- tinto.ast.statement.BlockStatement: bloque de instrucciones. Normalmente en if y while
  - list: lista de instrucciones del bloque, lista de Statement
  - returns: marcador que indica si se alcanza un return. Se actualiza con cada nueva instrucción
  - Returns() devuelve el valor del marcador

1.5 tinto.ast.expression

- tinto.ast.expresión:
  - Contiene las clases que permite describir los diferentes tipos de expresiones
  - Son la parte derecha de las asignaciones, como argumentos en las llamadas a funciones y como condiciones en las instrucciones if y while
- tinto.ast.expresión.Expression: clase abstracta que se utiliza como superclase del resto.
  - type: contiene el tipo de dato de la expresión. Este valor se almacena como un entero y puede tomar los valores definidos en la interfaz tinto.ast.Type
- tinto.ast.expresión.OperatorExpression: clase abstracta que encapsula todas las expresiones que representan una operación sobre otras expresiones. Son subclases BinaryExpression y UnaryExpression.

1.5 tinto.ast.expression

- tinto.ast.expression.LiteralExpression: Clase abstracta que encapsula a todas las expresiones que representan un valor literal. Subclases BooleanLiteralExpression, CharLiteralExpression y IntegerLiteralExpression.
- tinto.ast.expression.ReferenceExpression: Clase abstracta que encapsula a todas las expresiones que representan una referencia (a una variable o función)
- tinto.ast.expression.BinaryExpression: describe expresiones binarias, expresiones formadas por un operador y dos operandos. Se definen constantes para identificar los diferentes operadores binarios: operadores lógicos (AND y OR), operadores aritméticos (PLUS, MINUS, PROD, DIV y MOD) y operadores de comparación (EQ, NEQ, GT, GE, LT y LE).
  - op: código del operador binario
  - left: operando izquierdo de la expresión. Referencia a un objeto Expression.
  - right: operando derecho de la expresión. Referencia a un objeto Expression.

1.5 tinto.ast.expression

- tinto.ast.expression.UnaryExpression: describe expresiones unarias, formadas por un operador y un único operando. Se definen constantes para identificar los diferentes operadores unarios: el operador lógico de negación (NOT) y los operadores aritméticos unarios (PLUS y MINUS). También incluye la constante NONE, para indicar que no se aplica ningún operador unario al operando.
  - op: código del operador binario. Valor definido en las constantes de la clase
  - exp: operando de la expresión. Referencia al objeto Expression.
- tinto.ast.expression.BooleanLiteralExpression: describe un literal de tipo boolean, representa las constantes True y False.
  - value: almacena el valor del literal. El tipo de datos de la expresión es boolean
  - Clase con dos constructores que permiten construir el literal analizando el lexema (cadenas “true” y “false), o directamente con el valor.

1.5 tinto.ast.expression

- tinto.ast.expression.CharLiteralExpression: representa literales de tipo carácter
  - value: almacena el valor del literal. El tipo de datos de la expresión es char
  - La clase contiene dos constructores: uno basado directamente en el valor del literal y otro basado en el lexema. Para analizar el lexema se utiliza getCharFromString(String), que analiza las diferentes formas de representar un literal: caracteres editables (‘a’), caracteres no editables descritos con la barra invertida (‘\n’), caracteres descritos en formato octal (‘\071’) y caracteres descritos en formato Unicode (‘\u007A’)
- tinto.ast.expression.IntegerLiteralExpression: literales de tipo entero.
  - value: almacena el valor del literal. El tipo de datos de la expresión es int
  - Los constructores permiten crear los objetos a partir del lexema o directamente a partir del valor.
- tinto.ast.expression.VariableExpression: Expresión formada por la referencia a una variable local o a un argumento del método.
  - var: almacena la referencia al objeto Variable que describe la variable. Tipo de datos de la expresión es igual al tipo de datos de la variable.

1.5 tinto.ast.expression

- tinto.ast.expression.CallExpression: Llamada a una función. El tipo de datos de la expresión se corresponde al tipo de datos que devuelve la función
  - method: referencia al objeto Method que describe la función a la que se llama
  - library: referencia al objeto Library en el que está definida la función a la que se llama
  - call: descripción de argumentos de la llamada. Objeto de la clase CallParameters.

- tinto.ast.expresión.CallParameters:
  - No es subclase de Expression
  - Describe los argumentos de llamada a una función, la lista de expresiones utilizadas en la definición de la llamada.
  - parameter: lista de expresiones