PyTiger2C
Documentación del API

Yasser González Fernández
yglez@uh.cu

Ariel Hernández Amador
gnuaha7@uh.cu

Índice

1. Paquete pytiger2c

Paquete principal de PyTiger2C.

PyTiger2C es una implementación de un compilador del lenguaje de programación Tiger que genera código en lenguaje C y luego el código C resultante se compila para generar un ejecutable específico para una plataforma.

El código C generado será conforme al standard ISO/IEC 9899:1999, comúnmente conocido como C99, lo cual garantiza que pueda ser procesado por cualquier compilador de C que implemente dicho standard.

1.1. Módulos

- **ast**: Definición de los nodos del árbol de sintáxis abstracta.

  (Sección 2, página 18)

  - **accessnode**: Clase `AccessNode` del árbol de sintáxis abstracta.

    (Sección 3, página 22)

  - **aliastypedeclarationnode**: Clase `AliasTypeDeclarationNode` del árbol de sintáxis abstracta.

    (Sección 4, página 24)

  - **andoperatornode**: Clase `AndOperatorNode` del árbol de sintáxis abstracta.

    (Sección 5, página 28)

  - **arithmeticoperatornode**: Clase `ArithmeticOperatorNode` del árbol de sintáxis abstracta.

    (Sección 6, página 31)

  - **arrayaccessnode**: Clase `ArrayAccessNode` del árbol de sintáxis abstracta.

    (Sección 7, página 34)

  - **arraydeclarationnode**: Clase `ArrayDeclarationNode` del árbol de sintáxis abstracta.

    (Sección 8, página 38)

  - **arrayliteralexpressionnode**: Clase `ArrayLiteralExpressionNode` del árbol de sintáxis abstracta.

    (Sección 9, página 42)

  - **assignmentnode**: Clase `AssignmentNode` del árbol de sintáxis abstracta.

    (Sección 10, página 46)

  - **binarylogicaloperatornode**: Clase `BinaryLogicalOperatorNode` del árbol de sintáxis abstracta.

    (Sección 11, página 50)

  - **binaryoperatornode**: Clase `BinaryOperatorNode` del árbol de sintáxis abstracta.

(Sección 12, página 53)

- **breakstatementnode**: Clase `BreakStatementNode` del árbol de sintáxis abstracta.

  (Sección 13, página 55)

- **callabledeclarationnode**: Clase `CallableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 14, página 58)

- **declarationgroupnode**: Clase `DeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 15, página 61)

- **declarationnode**: Clase `DeclarationNode` del árbol de sintáxis abstracta.

  (Sección 16, página 64)

- **divideoperatornode**: Clase `DivideOperatorNode` del árbol de sintáxis abstracta.

  (Sección 17, página 66)

- **equalitylogicaloperatornode**: Clase `EqualityLogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 18, página 69)

- **equalsoperatornode**: Clase `EqualsOperatorNode` del árbol de sintáxis abstracta.

  (Sección 19, página 72)

- **expressionsequencenode**: Clase `ExpressionSequenceNode` del árbol de sintáxis abstracta.

  (Sección 20, página 74)

- **forstatementnode**: Clase `ForStatementNode` del árbol de sintáxis abstracta.

  (Sección 21, página 78)

- **functioncallnode**: Clase `FunctionCallNode` del árbol de sintáxis abstracta.

  (Sección 22, página 83)

- **functiondeclarationgroupnode**: Clase `FunctionDeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 23, página 87)

- **functiondeclarationnode**: Clase `FunctionDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 24, página 90)

- **greaterequalsthanoperatornode**: Clase `GreaterEqualsThanOperatorNode` del árbol de sintáxis abstracta.

  (Sección 25, página 94)

- **greaterthanoperatornode**: Clase `GreaterThanOperatorNode` del árbol de sintáxis abstracta.

(Sección 40, página 141)

- **oroperatornode**: Clase `OrOperatorNode` del árbol de sintáxis abstracta.

  (Sección 41, página 143)

- **plusoperatornode**: Clase `PlusOperatorNode` del árbol de sintáxis abstracta.

  (Sección 42, página 146)

- **proceduredeclarationnode**: Clase `ProcedureDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 43, página 149)

- **recordaccessnode**: Clase `RecordAccessNode` del árbol de sintáxis abstracta.

  (Sección 44, página 154)

- **recorddeclarationnode**: Clase `RecordDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 45, página 158)

- **recordliteralexpressionnode**: Clase `RecordLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 46, página 163)

- **relationallogicaloperatornode**: Clase `RelationalLogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 47, página 167)

- **staticvariabledeclarationnode**: Clase `StaticVariableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 48, página 170)

- **stringliteralexpressionnode**: Clase `StringLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 49, página 174)

- **timesoperatornode**: Clase `TimesOperatorNode` del árbol de sintáxis abstracta.

  (Sección 50, página 177)

- **typedeclarationgroupnode**: Clase `TypeDeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 51, página 180)

- **typedeclarationnode**: Clase `TypeDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 52, página 185)

- **unaryminusoperatornode**: Clase `UnaryMinusOperatorNode` del árbol de sintáxis abstracta.

  (Sección 53, página 187)

- **unaryoperatornode**: Clase `UnaryOperatorNode` del árbol de sintáxis abstracta.

(Sección 54, página 190)

- **valuedexpressionnode**: Clase `ValuedExpressionNode` del árbol de sintáxis abstracta.

  (Sección 55, página 192)

- **variableaccessnode**: Clase `VariableAccessNode` del árbol de sintáxis abstracta.

  (Sección 56, página 194)

- **variabledeclarationnode**: Clase `VariableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 57, página 198)

- **whilestatementnode**: Clase `WhileStatementNode` del árbol de sintáxis abstracta.

  (Sección 58, página 200)

- **code**: Clases utilizadas en la generación código C a partir de un programa Tiger.

  (Sección 59, página 204)

- **contrib**: Módulos desarrollados por terceros.

  (Sección 60, página 211)

- **dot**: Clases utilizadas en la generación de un archivo Graphviz DOT con el árbol de sintáxis abstracta creado a partir de un programa Tiger.

  (Sección 61, página 212)

- **errors**: Jerarquía de las excepciones lanzadas durante la ejecución de PyTiger2C.

  (Sección 62, página 214)

- **grammar**: Módulos relacionados con la definición de la gramática de Tiger utilizando PLY[1].

  (Sección 63, página 220)

- **scope**: Clases `Scope` y `RootScope` que representan ámbitos de ejecución en Tiger.

  (Sección 64, página 221)

- **types**: Módulos de la jerarquía de tipos de Tiger.

  (Sección 65, página 235)

  - **aliastype**: Clase de la jerarquía de tipos de Tiger representando el tipo alias.

    (Sección 66, página 236)

  - **arraytype**: Clase de la jerarquía de tipos de Tiger representando el tipo array.

    (Sección 67, página 238)

  - **basictype**: Clase de la jerarquía de tipos de Tiger representando los tipos básicos definidos en el lenguaje Tiger.

[1] http://www.dabeaz.com/ply/

(Sección 68, página 240)

- **functiontype**: Clase de la jerarquía de tipos de Tiger representando el tipo función.

  (Sección 69, página 242)

- **integertype**: Clase de la jerarquía de tipos de Tiger representando el tipo entero.

  (Sección 70, página 245)

- **niltype**: Clase de la jerarquía de tipos de Tiger representando el tipo `nil`.

  (Sección 71, página 247)

- **recordtype**: Clase de la jerarquía de tipos de Tiger representando el tipo record.

  (Sección 72, página 249)

- **stringtype**: Clase de la jerarquía de tipos de Tiger representando el tipo cadena de caracteres.

  (Sección 73, página 251)

- **tigertype**: Clase base de la jerarquía de tipos de Tiger.

  (Sección 74, página 253)

- **variabletype**: Clase de la jerarquía de tipos de Tiger representando la definición de una variable.

  (Sección 75, página 254)

1.2. Funciones

syntactic_analysis(*input_fd*)

Realiza análisis léxico-gráfico y sintáctico de un programa Tiger.

Argumentos

`input_fd:` Descriptor de fichero del programa Tiger al cual se le debe realizar el análisis sintáctico.

(type=`file`)

Valor de retorno

Como resultado del análsis sintáctico se obtiene el árbol de sintáxis abstracta correspondiente al programa Tiger recibido como argumento. El árbol se retorna a través del nodo de la raíz del árbol.

(type=`LanguageNode`)

Excepciones

`SyntacticError` Esta excepción se lanzará si se encuentra algún error de sintáxis durante el análisis del programa. La excepción contendrá información acerca del error, como por ejemplo, la línea y/o columna donde se encontró el error.

check_semantics(*ast*)

Realiza comprobación semántica de un programa Tiger representado por su árbol de sintáxis abstracta.

Argumentos

`ast:` Árbol de sintáxis asbtracta correspondiente a un programa Tiger.

(type=`LanguageNode`)

Excepciones

`SemanticError` Esta excepción se lanzará si se encuentra un error semántico en el árbol de sintáxis abstracta. La excepción contendrá información acerca del error.

generate_code(*ast*, *output_fd*)

Realiza la generación de código.

Argumentos

`ast:` Árbol de sintáxis asbtracta correspondiente a un programa Tiger.

(type=`LanguageNode`)

`output_fd:` Descriptor de fichero del archivo donde se debe escribir el código resultante de la traducción del programa Tiger descrito por el árbol de sintáxis abstracta.

(type=`file`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará si se produce algún error durante la generación de código. La excepción contendrá información acerca del error.

generate_dot(*ast*, *output_fd*)

Escribe un árbol de sintáxis abstracta correspondiente a un programa Tiger en un archivo con formato DOT de Graphviz.

Argumentos

`ast:` Árbol de sintáxis asbtracta correspondiente a un programa Tiger.

(type=`LanguageNode`)

`output_fd:` Descriptor de fichero del archivo donde se debe escribir el árbol de sintáxis abstracta en formato DOT de Graphviz.

(type=`file`)

tiger2dot(*tiger_filename*, *dot_filename*)

Genera un archivo en el formato DOT de Graphviz con el árbol de sintáxis abstracta correspondiente a un programa Tiger.

Se utiliza la función auxiliar `syntactic_analysis` para realizar el análisis léxico-gráfico y sintáctico durante el cual se reportará cualquier error en el programa Tiger. Luego, se utiliza la función auxiliar `generate_dot` para escribir el árbol de sintáxis abstracta en el archivo DOT.

Argumentos

`tiger_filename`: Ruta absoluta al archivo que contiene el código fuente del programa Tiger.

*(type=**str**)*

`dot_filename`: Ruta absoluta al archivo donde se generará el archivo DOT resultante. Si existe un archivo en la ruta especificada este será sobreescrito.

*(type=**str**)*

Excepciones

`PyTiger2CError` Además de las excepciones lanzadas por cada una de las funciones auxiliares, esta función puede lanzar esta excepción cuando se produce algún error al leer del archivo que contiene el programa Tiger que se quiere traducir o al escribir el árbol de sintáxis asbtracta resultante en el archivo DOT especificado.

tiger2c(*tiger_filename*, *c_filename*)

Traduce un programa Tiger a un programa C equivalente.

Se utiliza las funciones auxiliares `syntactic_analysis`, `check_semantics` y `generate_code` para llevar a cabo cada una de las fases de la compilación del programa: análisis léxico-gráfico y sintáctico, comprobación semántica y generación de código respectivamente. Cada una de estas funciones lanzará las excepciones `SyntacticError`, `SemanticError` y `CodeGenerationError` si se produce un error durante alguna de las fases. Consulte la documentación de cada función para conocer los detalles.

Argumentos

`tiger_filename`: Ruta absoluta al archivo que contiene el código fuente del programa Tiger.

*(type=**str**)*

`c_filename`: Ruta absoluta al archivo donde se generará el código C resultante. Si existe un archivo en la ruta especificada este será sobreescrito.

*(type=**str**)*

Excepciones

`PyTiger2CError` Además de las excepciones lanzadas por cada una de las funciones auxiliares, esta función puede lanzar esta excepción cuando se produce algún error al leer del archivo que contiene el programa Tiger que se quiere traducir o al escribir el código C resultante en el archivo especificado.

2. Paquete pytiger2c.ast

Definición de los nodos del árbol de sintáxis abstracta.

2.1. Módulos

- **accessnode**: Clase `AccessNode` del árbol de sintáxis abstracta.

  (Sección 3, página 22)

- **aliastypedeclarationnode**: Clase `AliasTypeDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 4, página 24)

- **andoperatornode**: Clase `AndOperatorNode` del árbol de sintáxis abstracta.

  (Sección 5, página 28)

- **arithmeticoperatornode**: Clase `ArithmeticOperatorNode` del árbol de sintáxis abstracta.

  (Sección 6, página 31)

- **arrayaccessnode**: Clase `ArrayAccessNode` del árbol de sintáxis abstracta.

  (Sección 7, página 34)

- **arraydeclarationnode**: Clase `ArrayDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 8, página 38)

- **arrayliteralexpressionnode**: Clase `ArrayLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 9, página 42)

- **assignmentnode**: Clase `AssignmentNode` del árbol de sintáxis abstracta.

  (Sección 10, página 46)

- **binarylogicaloperatornode**: Clase `BinaryLogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 11, página 50)

- **binaryoperatornode**: Clase `BinaryOperatorNode` del árbol de sintáxis abstracta.

  (Sección 12, página 53)

- **breakstatementnode**: Clase `BreakStatementNode` del árbol de sintáxis abstracta.

  (Sección 13, página 55)

- **callabledeclarationnode**: Clase `CallableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 14, página 58)

- **declarationgroupnode**: Clase `DeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 15, página 61)

- **declarationnode**: Clase `DeclarationNode` del árbol de sintáxis abstracta.

  (Sección 16, página 64)

- **divideoperatornode**: Clase `DivideOperatorNode` del árbol de sintáxis abstracta.

  (Sección 17, página 66)

- **equalitylogicaloperatornode**: Clase `EqualityLogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 18, página 69)

- **equalsoperatornode**: Clase `EqualsOperatorNode` del árbol de sintáxis abstracta.

  (Sección 19, página 72)

- **expressionsequencenode**: Clase `ExpressionSequenceNode` del árbol de sintáxis abstracta.

  (Sección 20, página 74)

- **forstatementnode**: Clase `ForStatementNode` del árbol de sintáxis abstracta.

  (Sección 21, página 78)

- **functioncallnode**: Clase `FunctionCallNode` del árbol de sintáxis abstracta.

  (Sección 22, página 83)

- **functiondeclarationgroupnode**: Clase `FunctionDeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 23, página 87)

- **functiondeclarationnode**: Clase `FunctionDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 24, página 90)

- **greaterequalsthanoperatornode**: Clase `GreaterEqualsThanOperatorNode` del árbol de sintáxis abstracta.

  (Sección 25, página 94)

- **greaterthanoperatornode**: Clase `GreaterThanOperatorNode` del árbol de sintáxis abstracta.

  (Sección 26, página 96)

- **ifthenelsestatementnode**: Clase `IfThenElseStatementNode` del árbol de sintáxis abstracta.

  (Sección 27, página 98)

- **ifthenstatementnode**: Clase `IfThenStatementNode` del árbol de sintáxis abstracta.

  (Sección 28, página 103)

- **inferredvariabledeclarationnode**: Clase `InferredVariableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 29, página 107)

- **integerliteralexpressionnode**: Clase `IntegerLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 30, página 111)

- **languagenode**: Clase base de la jerarquía de los nodos del árbol de sintáxis abstracta.

  (Sección 31, página 114)

- **lessequalsthanoperatornode**: Clase `LessEqualsThanOperatorNode` del árbol de sintáxis abstracta.

  (Sección 32, página 119)

- **lessthanoperatornode**: Clase `LessThanOperatorNode` del árbol de sintáxis abstracta.

  (Sección 33, página 121)

- **letnode**: Clase `LetNode` del árbol de sintáxis abstracta.

  (Sección 34, página 123)

- **logicaloperatornode**: Clase `LogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 35, página 129)

- **minusoperatornode**: Clase `MinusOperatorNode` del árbol de sintáxis abstracta.

  (Sección 36, página 131)

- **nilexpressionnode**: Clase `NilExpressionNode` del árbol de sintáxis abstracta.

  (Sección 37, página 134)

- **nonvaluedexpressionnode**: Clase `NonValuedExpressionNode` del árbol de sintáxis abstracta.

  (Sección 38, página 137)

- **notequalsoperatornode**: Clase `NotEqualsOperatorNode` del árbol de sintáxis abstracta.

  (Sección 39, página 139)

- **operatornode**: Clase `OperatorNode` del árbol de sintáxis abstracta.

  (Sección 40, página 141)

- **oroperatornode**: Clase `OrOperatorNode` del árbol de sintáxis abstracta.

  (Sección 41, página 143)

- **plusoperatornode**: Clase `PlusOperatorNode` del árbol de sintáxis abstracta.

  (Sección 42, página 146)

- **proceduredeclarationnode**: Clase `ProcedureDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 43, página 149)

- **recordaccessnode**: Clase `RecordAccessNode` del árbol de sintáxis abstracta.

  (Sección 44, página 154)

- **recorddeclarationnode**: Clase `RecordDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 45, página 158)

- **recordliteralexpressionnode**: Clase `RecordLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 46, página 163)

- **relationallogicaloperatornode**: Clase `RelationalLogicalOperatorNode` del árbol de sintáxis abstracta.

  (Sección 47, página 167)

- **staticvariabledeclarationnode**: Clase `StaticVariableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 48, página 170)

- **stringliteralexpressionnode**: Clase `StringLiteralExpressionNode` del árbol de sintáxis abstracta.

  (Sección 49, página 174)

- **timesoperatornode**: Clase `TimesOperatorNode` del árbol de sintáxis abstracta.

  (Sección 50, página 177)

- **typedeclarationgroupnode**: Clase `TypeDeclarationGroupNode` del árbol de sintáxis abstracta.

  (Sección 51, página 180)

- **typedeclarationnode**: Clase `TypeDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 52, página 185)

- **unaryminusoperatornode**: Clase `UnaryMinusOperatorNode` del árbol de sintáxis abstracta.

  (Sección 53, página 187)

- **unaryoperatornode**: Clase `UnaryOperatorNode` del árbol de sintáxis abstracta.

  (Sección 54, página 190)

- **valuedexpressionnode**: Clase `ValuedExpressionNode` del árbol de sintáxis abstracta.

  (Sección 55, página 192)

- **variableaccessnode**: Clase `VariableAccessNode` del árbol de sintáxis abstracta.

  (Sección 56, página 194)

- **variabledeclarationnode**: Clase `VariableDeclarationNode` del árbol de sintáxis abstracta.

  (Sección 57, página 198)

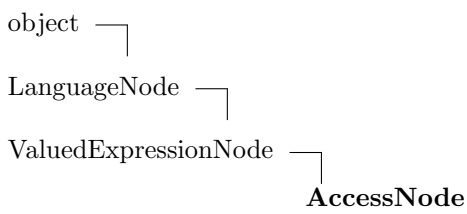
- **whilestatementnode**: Clase `WhileStatementNode` del árbol de sintáxis abstracta.

  (Sección 58, página 200)

3. Módulo pytiger2c.ast.accessnode

Clase `AccessNode` del árbol de sintáxis abstracta.

3.1. Clase AccessNode



Clase `AccessNode` del árbol de sintáxis abstracta.

Esta clase es la clase base para los nodos del árbol de sintáxis abstracta representando el acceso a una variable, un record o un array en el lenguaje Tiger. Para más información consulte la documentación de las clases `VariableAccessNode`, `RecordAccessNode` y `ArrayAccessNode`.

3.1.1. Métodos

read_only(*self*)

Método para obtener el valor de la propiedad `read_only`.

__init__(*self*)

Inicializa la clase `AccessNode`.

Overrides: object.__init__

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.1.2. Propiedades

Nombre	Descripción
Heredadas de object	

continúa en la página siguiente

Nombre	Descripción
__class__	

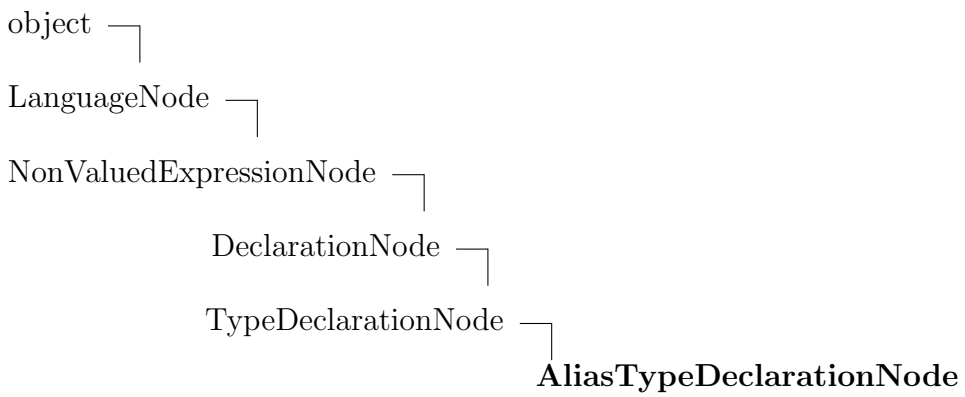
3.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

4. Módulo pytiger2c.ast.aliastypedeclarationnode

Clase `AliasTypeDeclarationNode` del árbol de sintáxis abstracta.

4.1. Clase AliasTypeDeclarationNode



Clase `AliasTypeDeclarationNode` del árbol de sintáxis abstracta.

Representa la declaración de un alias de un tipo del lenguaje Tiger. Un alias define un nuevo nombre en el ámbito local para definirse a un tipo definido anteriorment en el mismo ámbito o en un ámbito superior.

4.1.1. Métodos

alias_typename(*self*)

Método para obtener el valor de la propiedad `alias_typename`.

__init__(*self*, *name*, *alias_typename*)

Inicializa la clase `AliasTypeDeclarationNode`.

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `__init__` en la clase `TypeDeclarationNode`.

Argumentos

`alias_typename`: Nombre del tipo al que se le define el alias.
(type=`str`)

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Este método realiza la comprobación semántica de la definición de un alias. Primeramente, el método obtendrá del ámbito la instancia correspondiente al tipo que referencia el alias. Si este tipo no es un alias, actualizará la definición del alias en el ámbito padre del ámbito falso recibido como argumento para referenciar a la instancia del tipo real. Si este alias se define en función de otro se resolverá este en función de un tipo real y se procederá como se describió anteriormente.

Durante este proceso de comprobación semántica se detectará si se forma un ciclo durante la definición de una secuencia de alias, reportándose este hecho como un error semántico. Igualmente se reportará un error si un alias se define en función de un tipo que no se encuentra definido anteriormente.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de TypeDeclarationNode (Sección 52.1)

type()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

4.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

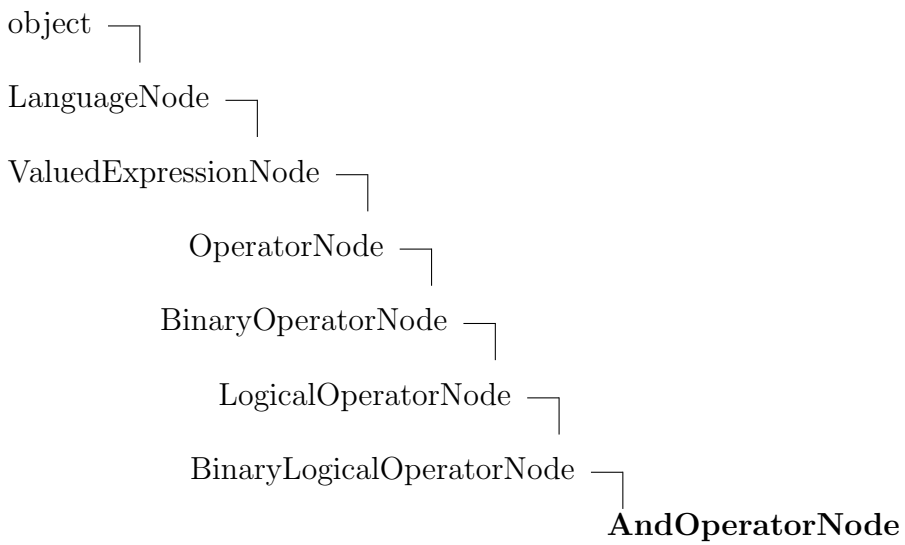
4.1.3. Variables de clase

Nombre	Descripción
Heredadas de TypeDeclarationNode (Sección 52.1) name	
Heredadas de LanguageNode (Sección 31.1) line_number, parent_node	

5. Módulo pytiger2c.ast.andoperatornode

Clase `AndOperatorNode` del árbol de sintáxis abstracta.

5.1. Clase AndOperatorNode



Clase `AndOperatorNode` del árbol de sintáxis abstracta.

Representa la operación lógica `AND`, representada con el operador `&` en Tiger, entre dos números enteros. Este operador retornará 1 en caso de que el resultado de evaluar la expresión sea verdadero, 0 en otro caso.

5.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `AndOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera el código C correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryLogicalOperatorNode (Sección 11.1)

check_semantics()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

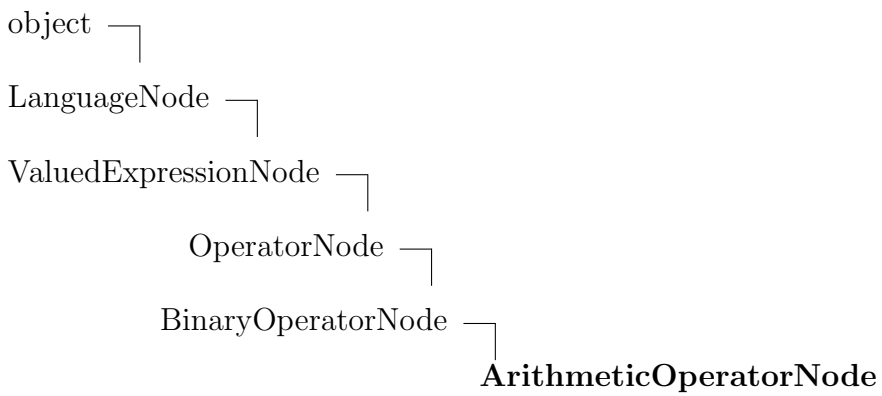
5.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

6. Módulo pytiger2c.ast.arithmeticoperatornode

Clase `ArithmeticOperatorNode` del árbol de sintáxis abstracta.

6.1. Clase ArithmeticOperatorNode



Clase `ArithmeticOperatorNode` del árbol de sintáxis abstracta.

6.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `ArithmeticOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

6.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

6.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	

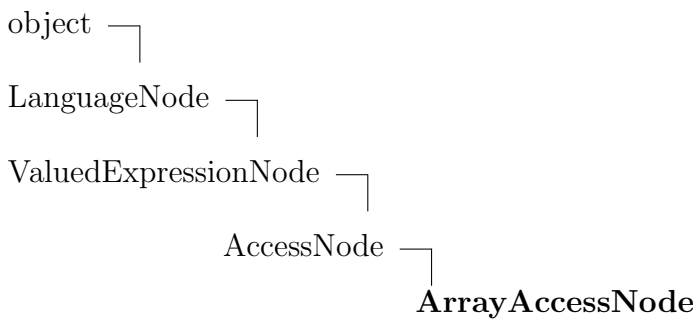
continúa en la página siguiente

Nombre	Descripción
line_number, parent_node	

7. Módulo pytiger2c.ast.arrayaccessnode

Clase `ArrayAccessNode` del árbol de sintáxis abstracta.

7.1. Clase ArrayAccessNode



Clase `ArrayAccessNode` del árbol de sintáxis abstracta.

Representa el acceso a un array del lenguaje Tiger. El acceso a un array del lenguaje Tiger permite obtener el valor del elemento que se encuentra en una posición determinada o asignarle un nuevo valor a este array en la misma posición. Esta estructura recibe la expresión que representa el acceso al array y la expresión correspondiente a la posición que se quiere acceder.

7.1.1. Métodos

array(*self*)

Método para obtener el valor de la propiedad array.

position(*self*)

Método para obtener el valor de la propiedad `position`.

__init__(*self*, *array*, *position*)

Inicializa la clase `ArrayAccessNode`.

Argumentos

`array`: Expresión correspondiente al array que se quiere acceder.

(type=`LanguageNode`)

`position`: Expresión correspondiente a la posición del array que se quiere acceder.

(type=`LanguageNode`)

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de acceso a array del lenguaje Tiger permite obtener el valor de un array en una posición determinada o asignarle un nuevo valor a este array en la misma posición. Esta estructura recibe la expresión que representa el acceso a la instancia de array y la expresión correspondiente a la posición que se quiere acceder.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se verifica que la expresión que se correspondiente al array retorne valor y que este sea del tipo array, luego se comprueba que la expresión de la posición retorne valor y que este sea de tipo entero.

En el proceso de comprobación semántica toman valor las propiedades `return_type` y `read_only`

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

En particular el nodo de acceso a un `array`, no genera ninguna instrucción de código `C` sino que toma valor la propiedad `code_name`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de AccessNode (Sección 3.1)

read_only()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

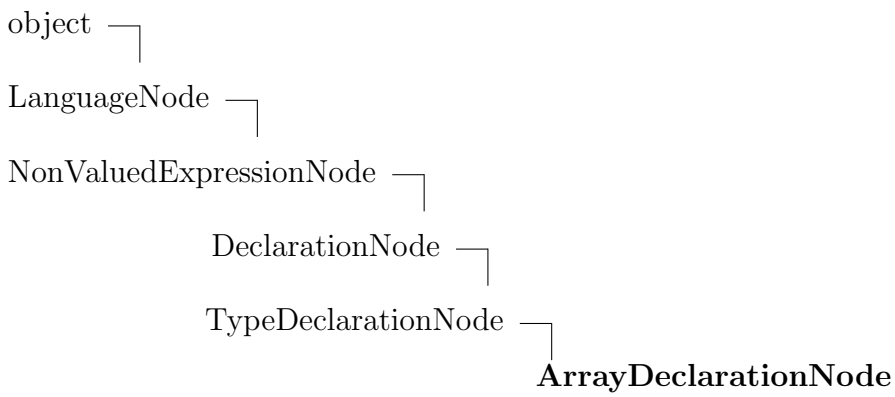
7.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

8. Módulo pytiger2c.ast.arraydeclarationnode

Clase `ArrayDeclarationNode` del árbol de sintáxis abstracta.

8.1. Clase ArrayDeclarationNode



Clase `ArrayDeclarationNode` del árbol de sintáxis abstracta.

Representa la estructura de declaración de un tipo array en el lenguaje Tiger. La estructura de declaración de array recibe un nombre que es el que representará a estos array concretos y el nombre del tipo que van a tener los valores.

8.1.1. Métodos

values_typename(*self*)

Método para obtener el valor de la propiedad `values_typename`.

__init__(*self*, *name*, *values_typename*)

Inicializa la clase `ArrayDeclarationNode`.

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `__init__` en la clase `TypeDeclarationNode`.

Argumentos

`values_typename`: Nombre del tipo que tendrán los valores del array.

*(type=**`str`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueba que el tipo de los valores del array se encuentre definido en el ámbito local.

Se reportarán errores semánticos si el tipo de los valores del array no se encuentra definido en el ámbito local, o en caso de que esté definido en el ámbito local, pero en otro grupo de declaraciones, en cuyo caso se considera una declaración de tipos mutuamente recursivos en distintos grupos de declaraciones de tipos.

Durante la comprobación semántica se define totalmente el valor de la propiedad `type`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de TypeDeclarationNode (Sección 52.1)

type()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

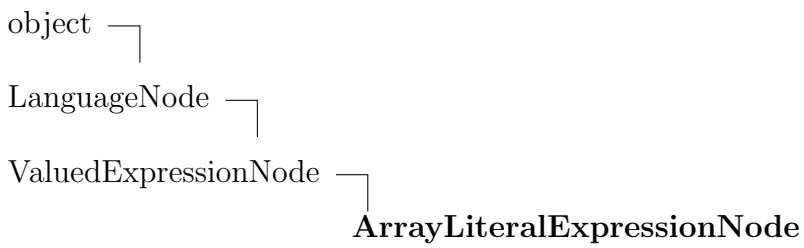
8.1.3. Variables de clase

Nombre	Descripción
Heredadas de TypeDeclarationNode (Sección 52.1) name	
Heredadas de LanguageNode (Sección 31.1) line_number, parent_node	

9. Módulo pytiger2c.ast.arrayliteralexpressionnode

Clase `ArrayLiteralExpressionNode` del árbol de sintáxis abstracta.

9.1. Clase ArrayLiteralExpressionNode



Clase `ArrayLiteralExpressionNode` del árbol de sintáxis abstracta.

Representa la creación de una instancia de un tipo array definido con anterioridad. La creación de una instancia de un tipo array recibe el nombre del tipo de array que se quiere crear, una expresión que corresponde a la cantidad de elementos que va a tener el array y por último una expresión que corresponde al valor con el que se inicializarán todos los miembros de este nuevo array.

9.1.1. Métodos

type_name(*self*)

Método para obtener el valor de la propiedad `type_name`.

count(*self*)

Método para obtener el valor de la propiedad `count`.

value(*self*)

Método para obtener el valor de la propiedad `value`.

__init__(*self*, *type_name*, *count*, *value*)

Inicializa la clase `ArrayLiteralExpressionNode`.

Argumentos

`type_name`: Nombre del tipo de array que se quiere crear.

*(type=**str**)*

`count`: Expresión correspondiente a la cantidad de elementos que se quiere crear.

*(type=**LanguageNode**)*

`value`: Expresión correspondiente al valor con el que se quiere inicializar los miembros del array

*(type=**LanguageNode**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La creación de una instancia de un tipo array recibe el nombre del tipo de array que se quiere crear, una expresión que corresponde a la cantidad de elementos que va a tener el array y por último una expresión que corresponde al valor con el que se inicializarán todos los miembros de este nuevo array.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueba que el tipo array que se quiere crear ha sido definido en el ámbito correspondiente, se comprueba que la expresión correspondiente a la cantidad de elementos del array tenga valor de retorno y que este sea entero, por último se comprueba que la expresión correspondiente al valor que se le asignará a cada miembro de este nuevo array retorne tipo y que este sea igual al correspondiente a los valores en la declaración del tipo de array.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

9.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

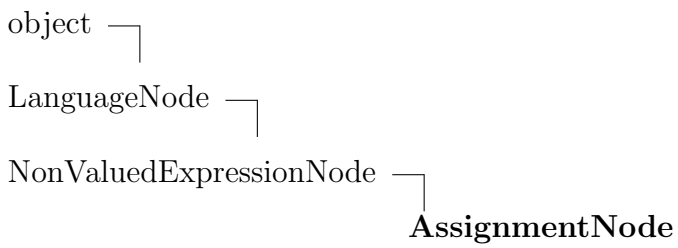
9.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

10. Módulo pytiger2c.ast.assignmentnode

Clase `AssignmentNode` del árbol de sintáxis abstracta.

10.1. Clase AssignmentNode



Clase `AssignmentNode` del árbol de sintáxis abstracta.

Representa la estructura de asignación del lenguaje Tiger. La estructura de asignación tiene una expresion (lvalue) de acceso a una variable, un elemento de un array o un campo de un record y un valor que se le asignará a este acceso.

10.1.1. Métodos

lvalue(*self*)

Método para obtener el valor de la propiedad `lvalue`.

expression(*self*)

Método para obtener el valor de la propiedad `expression`.

__init__(*self*, *lvalue*, *expression*)

Inicializa la clase `AssignmentNode`.

Argumentos

`lvalue:`	Expresión a la que se le quiere asignar la expresión. *(type=`AccessNode`.)*
`expresion:`	Expresión
`expression:`	*(type=`LanguageNode`)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de asignación tiene una expresion (lvalue) de acceso a una variable, un elemento de un array o un campo de un record y un valor que se le asignará a este acceso.

En la comprobación semántica de este nodo del árbol se comprueban semánticamente tanto el lvalue como la expresión, luego se comprueba que el lvalue como la expresión retornen valor y que el tipo de retorno de ambos sea el mismo. Además es necesario comprobar que el lvalue no es de solo lectura, pues en condiciones del lenguaje no es posible modificar el valor de una variable. Se reportarán errores si se encuentran errores durante la comprobación semántica del lvalue o de la expresión, si alguno de estos no retornan tipo, si los tipos no son iguales.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

10.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

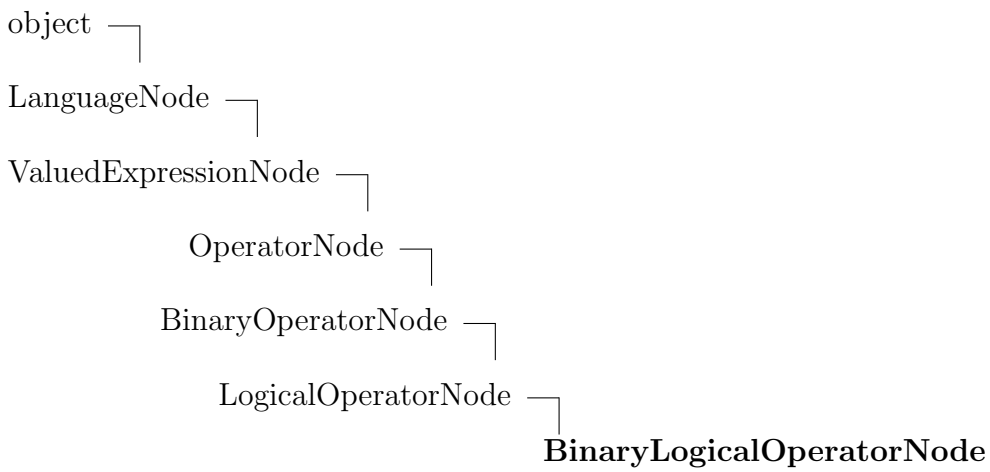
10.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

11. Módulo pytiger2c.ast.binarylogicaloperatornode

Clase `BinaryLogicalOperatorNode` del árbol de sintáxis abstracta.

11.1. Clase BinaryLogicalOperatorNode



Clase `BinaryLogicalOperatorNode` del árbol de sintáxis abstracta.

Esta clase implementa el método `check_semantics` para los operadores binarios con argumentos enteros del lenguaje Tiger. Estos operadores son los siguientes: el `OR` binario | y el `AND` binario `&`.

11.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `BinaryLogicalOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Los operadores cuyas clases del árbol de sintáxis abstracta derivan de esta deben recibir operandos enteros y siempre tendrán tipo de retorno entero (1 para el resultado verdadero, 0 para el falso).

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el tipo de retorno de ambas sea entero.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

generate_code(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.1.2. Propiedades

Nombre	Descripción
Heredadas de object	

continúa en la página siguiente

Nombre	Descripción
__class__	

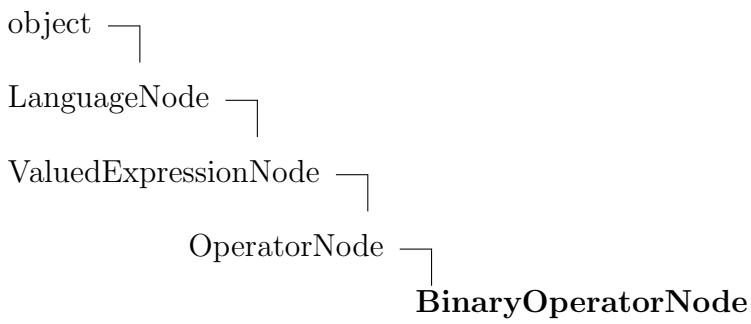
11.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

12. Módulo pytiger2c.ast.binaryoperatornode

Clase `BinaryOperatorNode` del árbol de sintáxis abstracta.

12.1. Clase BinaryOperatorNode



Clase `BinaryOperatorNode` del árbol de sintáxis abstracta.

Representa la clase base para los operadores que se realizan entre dos expresiones. De esta clase heredan los operadores aritméticos y lógicos.

12.1.1. Métodos

left(*self*)

Método para obtener el valor de la propiedad `left`.

right(*self*)

Método para obtener el valor de la propiedad `right`.

__init__(*self*, *left*, *right*)

Inicializa la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

(type=`LanguageNode`)

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

(type=`LanguageNode`)

Overrides: object.__init__

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

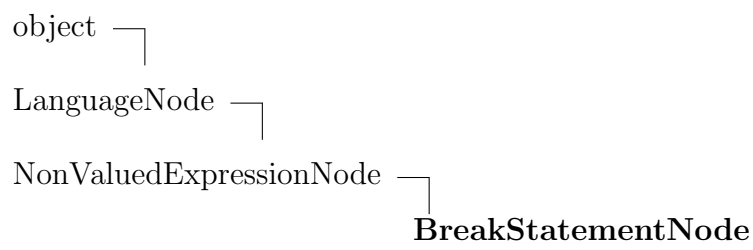
12.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

13. Módulo pytiger2c.ast.breakstatementnode

Clase `BreakStatementNode` del árbol de sintáxis abstracta.

13.1. Clase BreakStatementNode



Clase `BreakStatementNode` del árbol de sintáxis abstracta.

Representa la expresión `break` del lenguaje Tiger. La expresión `break` termina la evaluación de las instrucciones `while` y `for`.

13.1.1. Métodos

__**init**__(*self*)
Inicializa la clase `BreakStatementNode`.
Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La expresión `break` termina la evaluación de la instrucción `while` o `for` donde está contenida, dentro de la misma función o procedimiento. Cualquier otro uso de la expresión `break` es inválido.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se recorre el árbol hacia la raíz buscando una instrucción `while` o `for` que se debe encontrar antes de una declaración de función o procedimiento. Se reportarán errores semánticos si se llega a la raíz del árbol y no se encuentra una instrucción `while` o `for` o si se encuentra una declaración de función o procedimiento antes de encontrar una instrucción `while` o `for`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

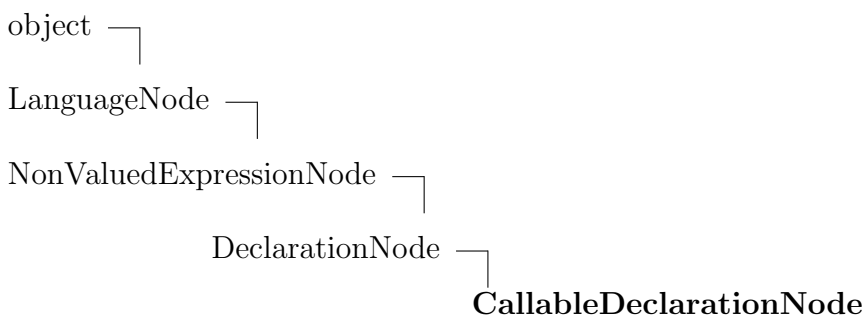
13.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

14. Módulo pytiger2c.ast.callabledeclarationnode

Clase `CallableDeclarationNode` del árbol de sintáxis abstracta.

14.1. Clase CallableDeclarationNode



Clase `CallableDeclarationNode` del árbol de sintáxis abstracta.

Clase base de los nodos `FunctionDeclarationNode` y `ProcedureDeclarationNode` del árbol de sintáxis abstracta. Esta clase tiene como objetivo factorizar los métodos y propiedades comunes de las clases representando declaraciones de procedimientos y funciones.

La comprobación semántica de los nodos del árbol de sintáxis abstracta descendientes de este nodo está dividida en dos partes: la comprobación semántica de la cabecera a través del método `check_header_semantics` y la comprobación semántica del cuerpo a través del método `check_semantics`. Para más información consulte la documentación de estos métodos.

14.1.1. Métodos

name(*self*)

Método para obtener el valor de la propiedad `name`.

parameters_names(*self*)

Método para obtener el valor de la propiedad `parameters_names`.

parameters_typenames(*self*)

Método para obtener el valor de la propiedad `parameters_typenames`.

body(*self*)

Método para obtener el valor de la propiedad `body`.

type(*self*)

Método para obtener el valor de la propiedad `type`.

__init__(*self*, *name*, *parameters_names*, *parameters_typenames*, *body*)

Inicializa la clase `CallableDeclarationNode`.

Argumentos

`name:`	Nombre del procedimiento o función cuya definición es representada por el nodo. *(type=`str`)*
`fields_names:`	Lista con los nombres de los parámetros de la función o procedimiento, por posición. *(type=`list`)*
`fields_typenames:`	Lista con los nombres de los tipos de los parámetros de la función o procedimiento, por posición. *(type=`list`)*
`body:`	Nodo del árbol de sintáxis abstracta correspondiente al cuerpo del procedimiento o función. *(type=`LanguageNode`)*

Overrides: object.__init__

check_header_semantics(*self*, *scope*, *errors*)

Este método realiza la comprobación semántica de la cabecera de la comunes a las funciones y procedimientos. Como resultado de esta comprobación se reportará cualquier error relacionados con el nombre de la función o procedimiento, los parámetros, los tipos de los parámetros y además tomará valor la propiedad `type` que contendrá el tipo de la función o procedimiento que define este nodo.

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

14.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

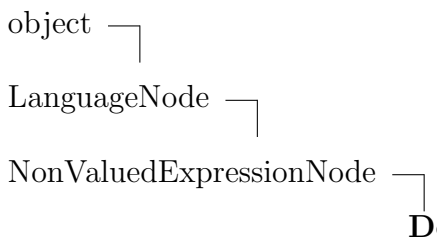
14.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

15. Módulo pytiger2c.ast.declarationgroupnode

Clase `DeclarationGroupNode` del árbol de sintáxis abstracta.

15.1. Clase DeclarationGroupNode



Clase `DeclarationGroupNode` del árbol de sintáxis abstracta.

Este nodo del árbol de sintáxis abstracta es la clase base de los nodos `TypeDeclarationGroupNode` y `FunctionDeclarationGroupNode`, los cuales representan un grupo de declaraciones consecutivas de tipos o funciones respectivamente. En nodos tienen el objetivo de garantizar durante la comprobación semántica que no se hagan declaraciones de variables entre declaraciones de tipos o funciones mutuamente recursivas ya que si esto sucede se producen situaciones ambiguas.

15.1.1. Métodos

declarations(*self*)

Método para obtener el valor de la propiedad `declarations`.

__init__(*self*)

Inicializa la clase `DeclarationGroupNode`.

Overrides: object.__init__

collect_definitions(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Este método define, recoge y devuelve las declaraciones de tipos o funciones (según sea el caso) hechas en el grupo de declaraciones representado por este nodo de sintáxis abstracta. Estos conjuntos de declaraciones se utilizan en la comprobación semántica de la estructura `let-in-end`, donde está contenido el grupo de declaraciones, para garantizar que no se hagan definiciones de tipos o funciones mutuamente recursivos iterrumpidos por declaraciones de variables.

Valor de retorno

Conjunto con los nombres de los tipos o funciones definidos en este grupo.

*(type=**set**)*

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

*(type=**str**)*

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

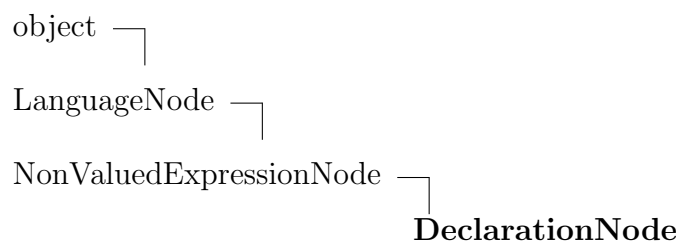
15.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

16. Módulo pytiger2c.ast.declarationnode

Clase `DeclarationNode` del árbol de sintáxis abstracta.

16.1. Clase DeclarationNode



Clase `DeclarationNode` del árbol de sintáxis abstracta.

16.1.1. Métodos

__**init**__(*self*)
Inicializa la clase `DeclarationNode`.
Overrides: object.__init__

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

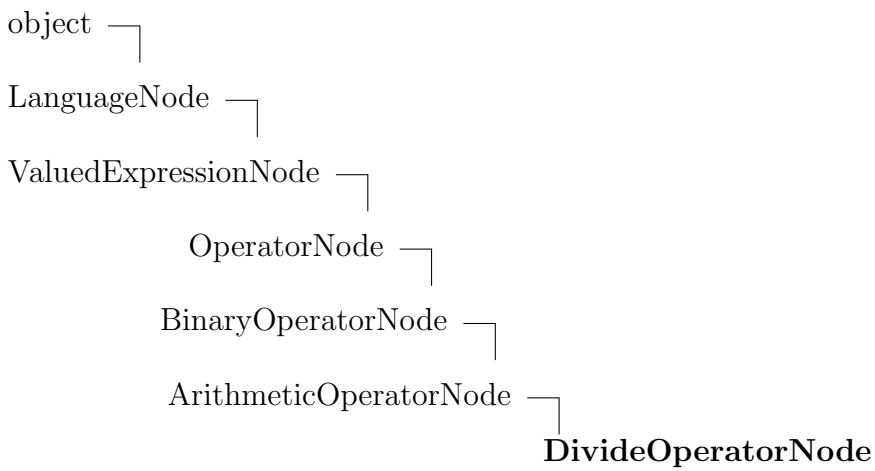
16.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

17. Módulo pytiger2c.ast.divideoperatornode

Clase `DivideOperatorNode` del árbol de sintáxis abstracta.

17.1. Clase DivideOperatorNode



Clase `DivideOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de división `/` entre dos números enteros del lenguaje Tiger.

17.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `DivideOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

El operador de división realiza la división entre los el valor de la expresión que se encuentra a la izquierda entre el valor de la derecha.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el valor de retorno de ambas sea entero.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

17.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

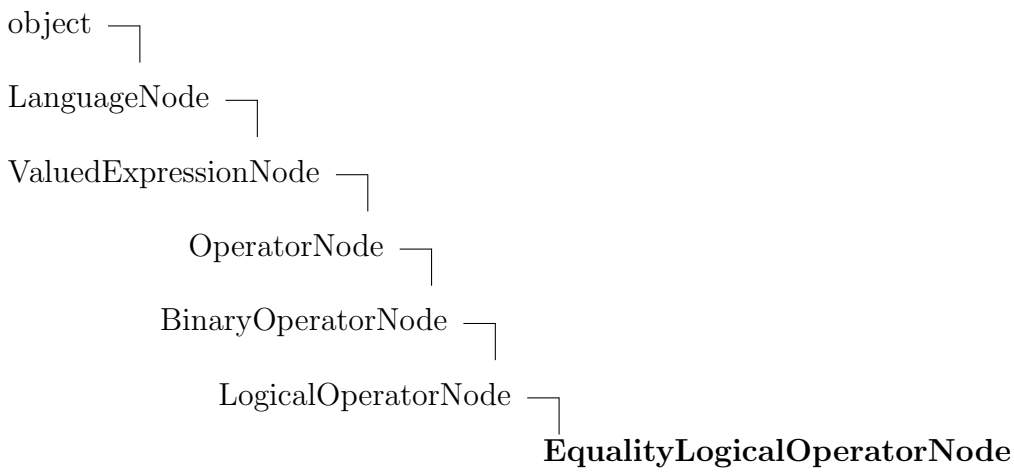
17.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

18. Módulo pytiger2c.ast.equalitylogicaloperatornode

Clase `EqualityLogicalOperatorNode` del árbol de sintáxis abstracta.

18.1. Clase EqualityLogicalOperatorNode



Clase `EqualityLogicalOperatorNode` del árbol de sintáxis abstracta.

Esta clase implementa el método `check_semantics` para los operadores logicos binarios de igualadd y desigualdad. Estos operadores son los siguientes: igual que `=`, no igual que <>.

18.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `EqualityLogicalOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Los operadores cuyas clases del árbol de sintáxis abstracta derivan de esta deben recibir en ambos operandos expresiones con valor de retorno y el tipo de estas debe ser el mismo. Siempre tienen tipo de retorno entero (1 para el resultado verdadero, 0 para el falso).

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el tipo de retorno de ambas sea el mismo.

Argumentos

`scope:` Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors:` Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código C correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Argumentos

`generator:` Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

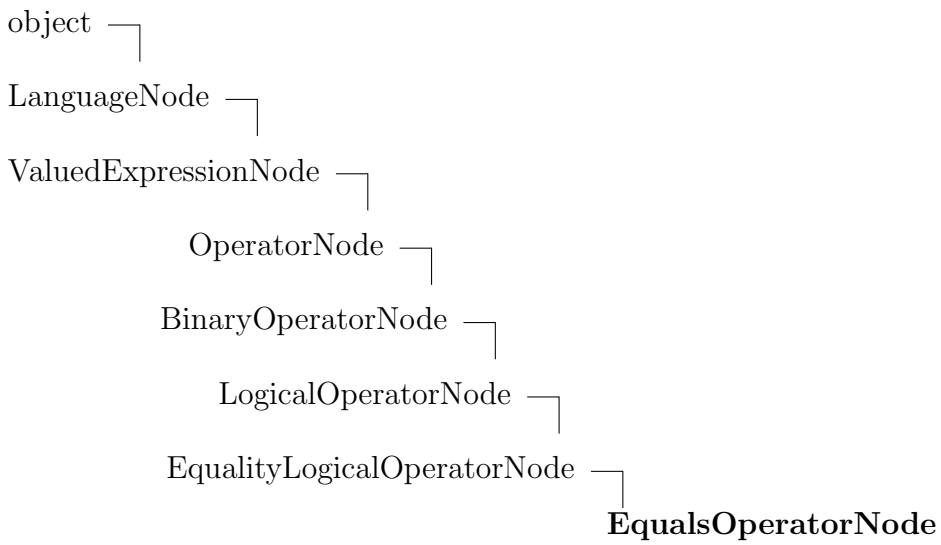
18.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

19. Módulo pytiger2c.ast.equalsoperatornode

Clase `EqualsOperatorNode` del árbol de sintáxis abstracta.

19.1. Clase EqualsOperatorNode



Clase `EqualsOperatorNode` del árbol de sintáxis abstracta.

Representa el operador `=` entre dos expresiones del lenguaje Tiger.

19.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `EqualsOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de EqualityLogicalOperatorNode (Sección 18.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

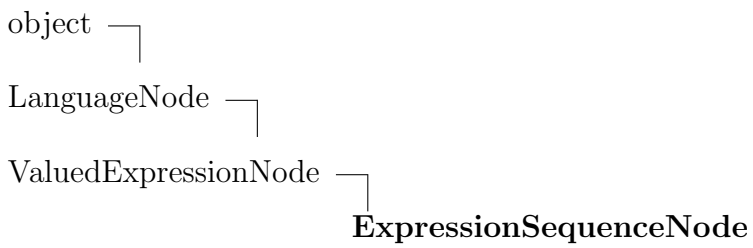
19.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

20. Módulo pytiger2c.ast.expressionsequencenode

Clase `ExpressionSequenceNode` del árbol de sintáxis abstracta.

20.1. Clase ExpressionSequenceNode



Clase `ExpressionSequenceNode` del árbol de sintáxis abstracta.

Este nodo representa una secuencia de expresiones del lenguaje Tiger separadas por el caracter punto y coma que se ejecutan en el orden en que estas aparecen. Esta secuencia de expresiones puede ser vacía. El valor de retorno de una secuencia de expresiones será el valor de la última expresión de la secuencia si esta existe y no tendrá valor de retorno en caso de que sea una secuencia de expresiones vacía.

20.1.1. Métodos

expressions(*self*)

Método para obtener el valor de la propiedad `expressions`.

__init__(*self*)

Inicializa la clase `ExpressionSequenceNode`.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxix abstracta se comprueban semánticamente cada una de las expresiones de la secuencia y posteriormente se asigna el tipo de retorno de la expresión que será el de la última expresión de la secuencia. Si el nodo representa una secuencia de expresiones vacía entonces no tendrá valor de retorno.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

Una secuencia de expresiones no tiene valor de retorno cuando es una secuencia vacía o cuando la última expresión de la secuencia no tiene valor de retorno. Debido a esto, este nodo debe redefinir el método `has_return_value` para cambiar la implementación provista por la clase `ValuedExpressionNode` que siempre retorna `True`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

20.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

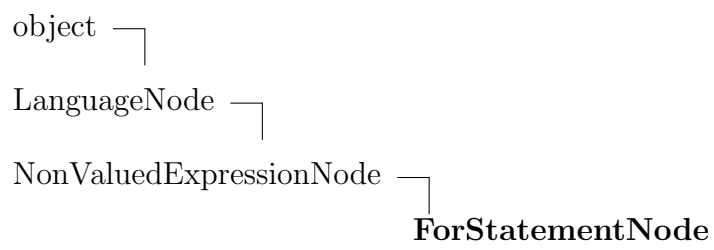
20.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

21. Módulo pytiger2c.ast.forstatementnode

Clase `ForStatementNode` del árbol de sintáxis abstracta.

21.1. Clase ForStatementNode



Clase `ForStatementNode` del árbol de sintáxis abstracta.

Representa la expresión `for` del lenguaje Tiger. La expresión `for` evalúa las expresiones correspondientes a los límites inferiores y superiores y por cada valor entero entre estos valores (incluidos los extremos) evalúa la expresión seguida del `do` con una variable entera con el nombre dado que toma el valor del entero correspondiente. La variable del índice de cada iteración sólo debe ser visible para la expresión seguida de `do` y es un error cambiar su valor.

21.1.1. Métodos

index_name(*self*)

Método para obtener el valor de la propiedad `index_name`.

lower_expression(*self*)

Método para obtener el valor de la propiedad `lower_expression`.

upper_expression(*self*)

Método para obtener el valor de la propiedad `upper_expression`.

expression(*self*)

Método para obtener el valor de la propiedad `expression`.

__init__(*self*, *index_name*, *lower_expression*, *upper_expression*, *expression*)

Inicializa la clase `ForStatementNode`.

Argumentos

`index_name`: Nombre de la variable de índice del ciclo.

*(type=**str**)*

`lower_expression`: Nodo del árbol de sintáxis abstrata correspondiente al límite inferior del rango para la variable de índice.

*(type=**LanguageNode**)*

`upper_expression`: Nodo del árbol de sintáxis abstrata correspondiente al límite superior del rango para la variable de índice.

*(type=**LanguageNode**)*

`expression`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión que se debe ejecutar para cada valor de la variable de índice.

*(type=**LanguageNode**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se crea un nuevo ámbito que contendrá la definicion de la variable de índice del ciclo como sólo lectura. Este ámbito tendrá como padre el ámbito donde fue definido el ciclo `for`.

Luegoo, se comprueban semánticamente las expresiones correspondientes a los extremos inferiores y superiores del intervalo. Estas expresiones deben tener valor de retorno entero. Además se comprueba semánticamente la expresión que se ejecutará en cada iteración y esta se deja libre de tener o no valor de retorno.

Argumentos

`scope:` Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors:` Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

21.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

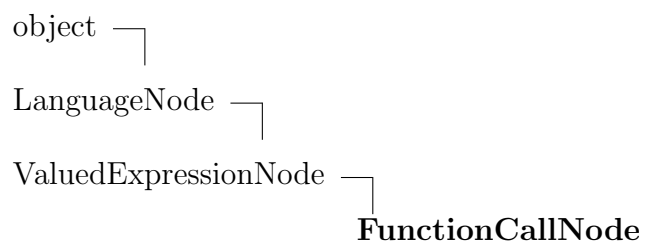
21.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

22. Módulo pytiger2c.ast.functioncallnode

Clase `FunctionCallNode` del árbol de sintáxis abstracta.

22.1. Clase FunctionCallNode



Clase `FunctionCallNode` del árbol de sintáxis abstracta.

Este nodo representa el llamado a una función o un procedimiento en el lenguaje Tiger. La función que se está llamando debe haber sido definida anteriormetne o ser una función de la biblioteca standard de Tiger. El tipo de retorno de un llamado a una función será el mismo tipo de retorno de las funciones y no tendrá valor de retorno en el caso de los procedimientos.

22.1.1. Métodos

name(*self*)

Método para obtener el valor de la propiedad `name`.

parameters(*self*)

Método para obtener el valor de la propiedad `parameters`.

__init__(*self*, *name*, *parameters*)

Inicializa la clase `FunctionCallNode`.

Argumentos

`name:` Cadena de caracteres correspondiente al nombre de la función que se está llamando.

(type=`str`)

`parameters:` Lista de expresiones que darán valor a cada uno de los parámetros de la función.

(type=`list`)

Overrides: object.__init__

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxix abstracta se comprueban semánticamente cada una de las expresiones de la lista de expresiones que dan valor a los parámetros de la función o procedimiento. Luego, se comprueba que la función o procedimiento que se está llamando exista y que el tipo de las expresiones de los parámetros coincida con los esperados por la definición de la función. Finalmente se asigna el tipo de retorno del nodo al tipo de retorno de la función o no tendrá valor de retorno en el caso de llamados a procedimientos.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

22.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

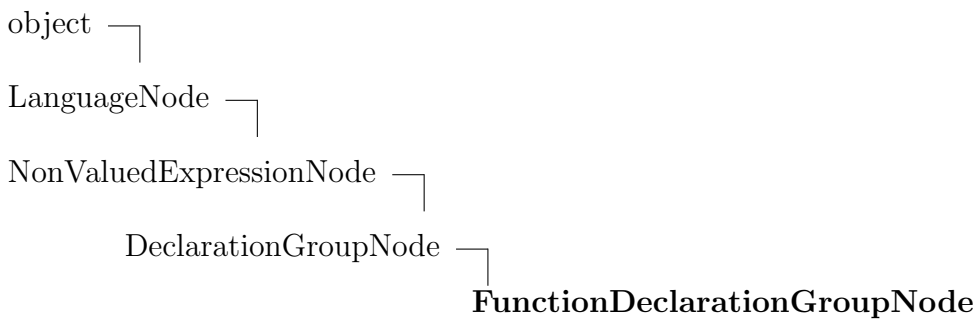
22.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

23. Módulo pytiger2c.ast.functiondeclarationgroupnode

Clase `FunctionDeclarationGroupNode` del árbol de sintáxis abstracta.

23.1. Clase FunctionDeclarationGroupNode



Clase `FunctionDeclarationGroupNode` del árbol de sintáxis abstracta.

23.1.1. Métodos

__init__(*self*)

Inicializa la clase `FunctionDeclarationGroupNode`.

Overrides: object.__init__

collect_definitions(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Realiza la comprobación semántica de las cabeceras de las funciones definidas en el grupo y además define en el ámbito dado las funciones del grupo.

Se reportarán errores si se produce alguno durante la comprobación semántica de las cabeceras de las funciones o si no puede ser definida una función.

Valor de retorno

Conjunto con los nombres de las funciones definidas en este grupo.

(type=`set`)

Overrides: pytiger2c.ast.declarationgroupnode.DeclarationGroupNode.collect_definitions

check_semantics(*self*, *scope*, *errors*, *used_types*=`None`)

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Un grupo de declaraciones de funciones del lenguaje Tiger se forma por declaraciones de funciones que aparecen uno a continuación de otros. Funciones mutuamente recursivas deben estar definidas en el mismo grupo de declaraciones de funciones. De modo que no es valido declarar funciones mutuamente recursivos con una declaración de variable o tipo entre estas.

En la comprobación de este nodo del árbol de sintáxis abstracta se comprueban semánticamente todas la declaraciones contenidas en este.

Se reportarán errores si se producen errores en la comprobación semántica de alguna de las declaraciones contenidas en este grupo.

Argumentos

`used_types`: Lista de los nombres de los tipos usados en el ámbito local.

(type=`list`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de DeclarationGroupNode (Sección 15.1)

declarations(), generate_dot()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

23.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

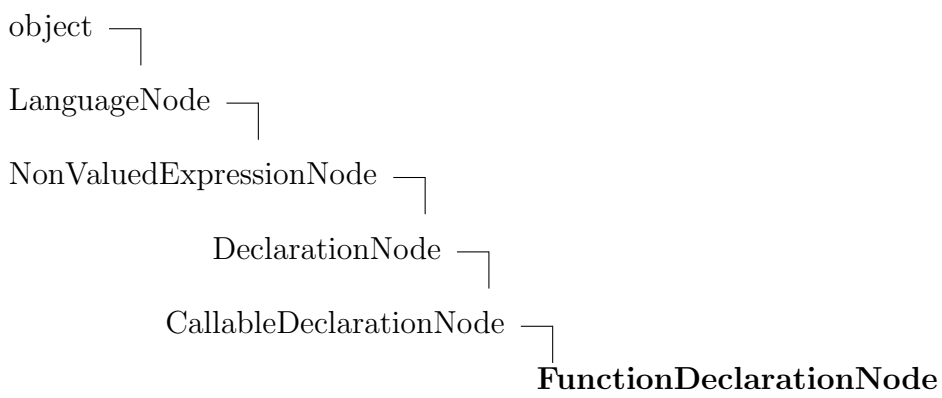
23.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

24. Módulo pytiger2c.ast.functiondeclarationnode

Clase `FunctionDeclarationNode` del árbol de sintáxis abstracta.

24.1. Clase FunctionDeclarationNode



Clase `FunctionDeclarationNode` del árbol de sintáxis abstracta.

Este nodo representa la declaración de una función en el lenguaje Tiger. Una función se diferencia de un procedimiento en que la primera siempre tiene un valor de retorno.

24.1.1. Métodos

return_typename(*self*)

Método para obtener el valor de la propiedad `return_typename`.

__init__(*self*, *name*, *parameters_names*, *parameters_typenames*, *body*, *return_typename*)

Inicializa la clase `FunctionDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `CallableDeclarationNode`.

Argumentos

`return_typename`: Cadena de caracteres correspondiente al nombre del tipo del valor de retorno de la función.

(type=`str`)

Overrides: object.__init__

check_header_semantics(*self*, *scope*, *errors*)

Este método realiza la comprobación semántica de la cabecera específica para la declaración de funciones. Para más información consulte la documentación método `check_header_semantics` en la clase `CallableDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Overrides: pytiger2c.ast.callabledeclarationnode.CallableDeclarationNode.check_header_semantics

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La comprobación semántica de este nodo del árbol de sintáxis abstracta está dividida en dos partes: la comprobación semántica de la cabecera a través del método `check_header_semantics` y la comprobación semántica del cuerpo a través del método `check_semantics`.

En este método se crea un nuevo ámbito que tendrá como padre el ámbito en el que se está definiendo la función y contendrá las definiciones de las variables correspondientes a los parámetros recibidos por el procedimiento. Luego, se comprueba semánticamente el cuerpo de la función, el cual debe tener valor de retorno.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de CallableDeclarationNode (Sección 14.1)

body(), generate_code(), name(), parameters_names(), parameters_typenames(), type()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

24.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

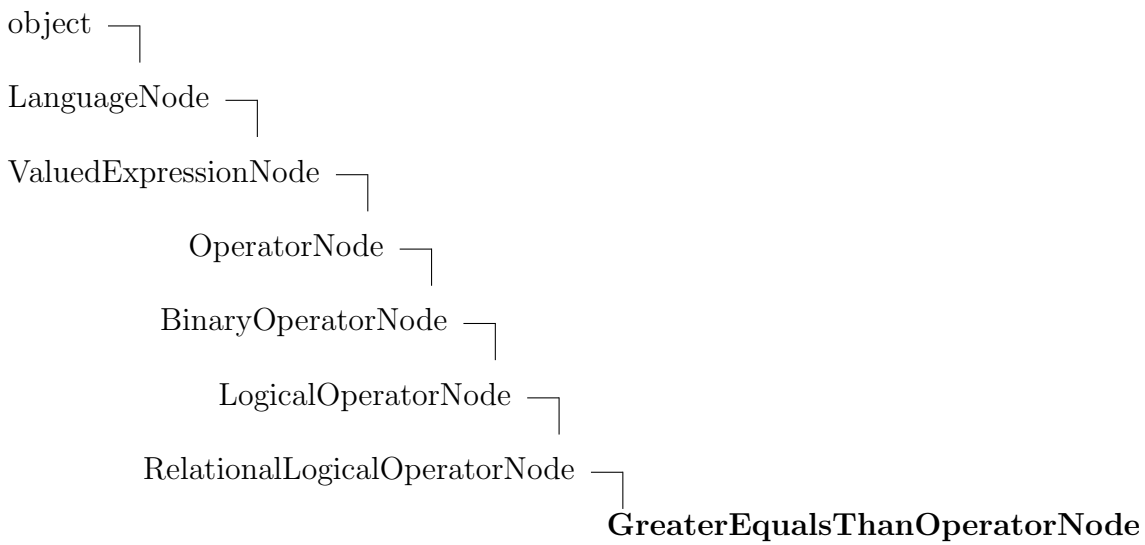
24.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

25. Módulo pytiger2c.ast.greaterequalsthanoperatornode

Clase `GreaterEqualsThanOperatorNode` del árbol de sintáxis abstracta.

25.1. Clase GreaterEqualsThanOperatorNode



Clase `GreaterEqualsThanOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de suma >= entre dos números enteros o dos cadenas de caracters del lenguaje Tiger.

25.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `GreaterEqualsThanOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de RelationalLogicalOperatorNode (Sección 47.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

25.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

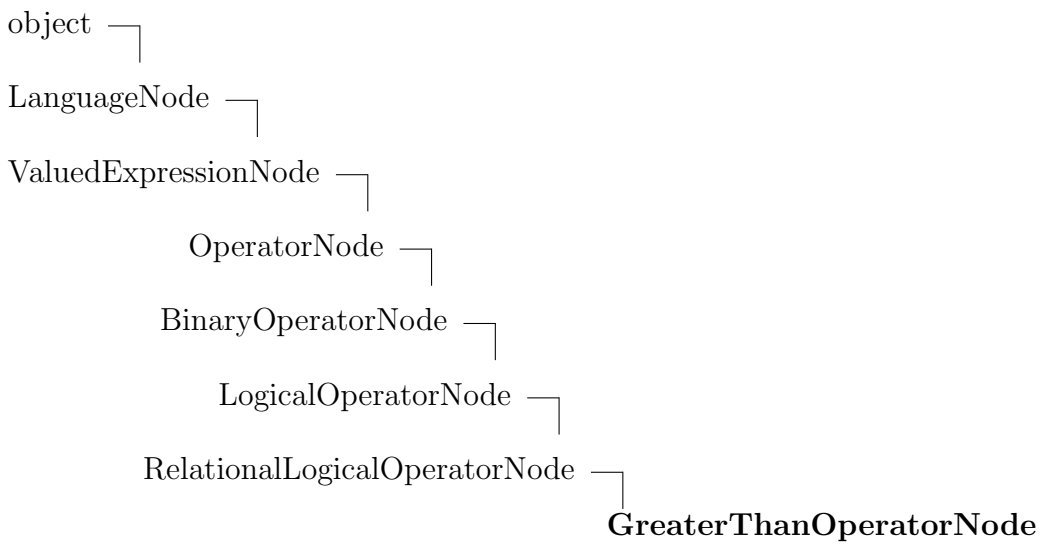
25.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

26. Módulo pytiger2c.ast.greaterthanoperatornode

Clase `GreaterThanOperatorNode` del árbol de sintáxis abstracta.

26.1. Clase GreaterThanOperatorNode



Clase `GreaterThanOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de suma > entre dos números enteros o dos cadenas de caracters del lenguaje Tiger.

26.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `GreaterThanOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de RelationalLogicalOperatorNode (Sección 47.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

26.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

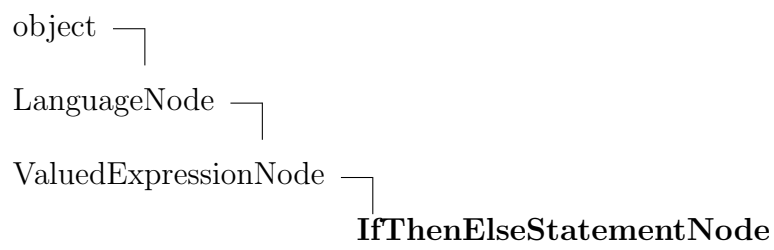
26.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

27. Módulo pytiger2c.ast.ifthenelsestatementnode

Clase `IfThenElseStatementNode` del árbol de sintáxis abstracta.

27.1. Clase IfThenElseStatementNode



Clase `IfThenElseStatementNode` del árbol de sintáxis abstracta.

Representa la expresión `if-then-else` del lenguaje Tiger. La expresión `if-then-else` permite la ejecución condicional fragmentos de código. Primeramente se evalúa la expresión seguida de la instrucción `if`, que debe retornar un número entero; si su valor es diferente de cero entonces se ejecutará la expresión seguida de la instrucción `then` y este será el valor de retorno de la expresión, en caso contrario se ejecutará la expresión seguida de la instrucción `else` y este será el valor de retorno de la expresión. Por tanto, la expresión seguida de la instrucción `then` y la expresión seguida de la instrucción `else` deben tener el mismo tipo de retorno o ambas no tener valor de retorno.

27.1.1. Métodos

condition(*self*)

Método para obtener el valor de la propiedad `condition`.

then_expression(*self*)

Método para obtener el valor de la propiedad `then_expression`.

else_expression(*self*)

Método para obtener el valor de la propiedad `else_expression`.

__init__(*self*, *condition*, *then_expression*, *else_expression*)

Inicializa la clase `IfThenElseStatementNode`.

Argumentos

`condition`: Expresión que se debe evaluar para decidir si se debe ejecutar la expresión seguida de la instrucción `then` o la expresión seguida de la instrucción `else`.

(type=`LanguageNode`)

`then_expression`: Expresión que se ejeccutará si el valor de retorno de la condición fue distinto de cero.

(type=`LanguageNode`)

`else_expression`: Expresión que se ejeccutará si el valor de retorno de la condición fue cero.

(type=`LanguageNode`)

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La expresión seguida de la instrucción `if`, cuyo nodo del árbol de sintáxis abstracta está almacenado en la propiedad `condition`, deberá tener valor de retorno entero. Las expresiones seguidas de las instrucciones `then` y `else` deben tener el mismo tipo de retorno o no tener valor de retorno ambas. El tipo del valor de retorno de la expresión `if-then-else` será el mismo que el de estas expresiones o no retornará valor si estas no lo hacen.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta primeramente se comprueba que la expresión seguida de la instrucción `if` esté correcta semánticamente, que tenga valor de retorno y que sea de tipo `IntegerType`. Luego, se comprueba que las expresiones seguidas de las instrucciones `then` y `else` estén correctas semánticamente y que el tipo de su valor de retorno sea el mismo o ambas no tengan valor de retorno. Para finalizar especifica el tipo de retorno que tendrá la expresión, el cual coincidirá con el tipo de las expresiones.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

La expresión `if-then-else` no tiene valor de retorno cuando las expresiones seguidas de las instrucciones `then` y `else` no tiene valor de retorno. Debido a esto, este nodo debe redefinir el método `has_return_value` para cambiar la implementación provista por la clase `ValuedExpressionNode` que siempre retorna `True`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

27.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

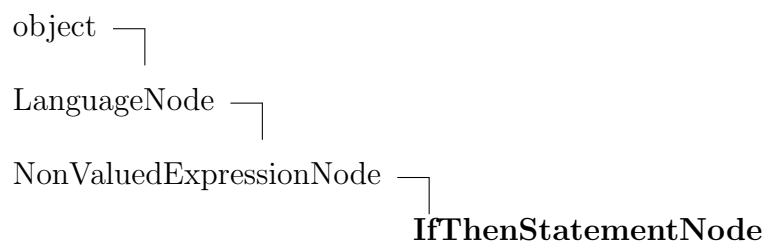
27.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

28. Módulo pytiger2c.ast.ifthenstatementnode

Clase `IfThenStatementNode` del árbol de sintáxis abstracta.

28.1. Clase IfThenStatementNode



Clase `IfThenStatementNode` del árbol de sintáxis abstracta.

Representa la expresión `if-then` del lenguaje Tiger. La expresión `if-then` permite la ejecución condicional de un fragmento de código. Primeramente se evalúa la expresión seguida de la instrucción `if`, que debe retornar un número entero; si su valor es diferente de cero entonces se ejecutará la expresión seguida de la instrucción `then` que no debe tener valor de retorno.

28.1.1. Métodos

condition(*self*)

Método para obtener el valor de la propiedad `condition`.

then_expression(*self*)

Método para obtener el valor de la propiedad `then_expression`.

__init__(*self*, *condition*, *then_expression*)

Inicializa la clase `IfThenStatementNode`.

Argumentos

`condition`: Expresión que se debe evaluar para decidir si se debe ejecutar la expresión seguida de la instrucción `then`.

*(type=**`LanguageNode`**)*

`then_expression`: Expresión que se ejeccutará si el valor de retorno de la condición fue distinto de cero. Esta expresión no debe tener valor de retorno.

*(type=**`LanguageNode`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La expresión seguida de la instrucción `if`, cuyo nodo del árbol de sintáxis abstracta está almacenado en la propiedad `condition`, deberá tener valor de retorno entero. La expresión seguida de la instrucción `then` no debe tener valor de retorno.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta primeramente se comprueba que la expresión seguida de la instrucción `if` esté correcta semánticamente, que tenga valor de retorno y que sea de tipo `IntegerType`. Luego se comprueba que la expresión seguida de la instrucción `then` esté correcta semánticamente y que no tenga valor de retorno.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

28.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

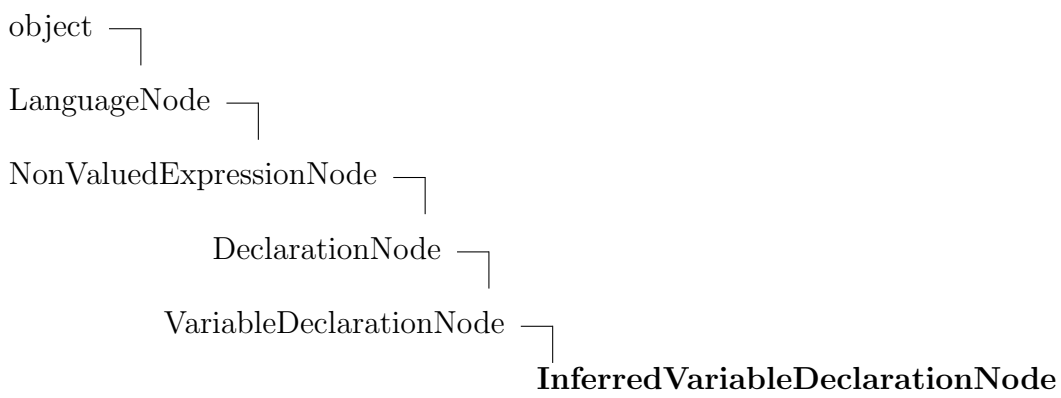
28.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

29. Módulo pytiger2c.ast.inferredvariabledeclarationnode

Clase `InferredVariableDeclarationNode` del árbol de sintáxis abstracta.

29.1. Clase InferredVariableDeclarationNode



Clase `InferredVariableDeclarationNode` del árbol de sintáxis abstracta.

Representa la estructura de declaración de variables sin expresar explícitamente el tipo de esta del lenguaje Tiger. Esta estructura recibe una expresión cuyo valor se le asignará a la variable y el tipo de la variable se infiere del tipo de esta expresión.

29.1.1. Métodos

__init__(*self*, *name*, *value*)

Inicializa la clase `InferredVariableDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`name`: Nombre de la variable.

`value`: `LanguageNode` correspondiente al valor que se asigna a la variable.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de declaración de variables sin especificar explícitamente el tipo de esta recibe una expresión cuyo valor se le asignará a la variable y el tipo de la variable se infiere del tipo de esta expresión.

En la comprobación semántica se comprueba semánticamente la expresión que se quiere asignar a la variable. Luego se comprueba que esta expresión retorne valor, que este valor no sea `nil` y que en su ámbito local el nombre que se quiere asignar a la variable no haya sido asignado a una función u otra variable. Se reportarán errores si se encuentran errores durante la comprobación semántica de la expresión, si esta no retorna valor o este es nil y por último si el nombre de la variable ya ha sido asignado a una función u otra variable en su ámbito local.

En el proceso de comprobación semántica la propiedad `type` toma valor y la variable es definida en su ámbito local.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de VariableDeclarationNode (Sección 57.1)

generate_code(), name(), type(), value()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

29.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

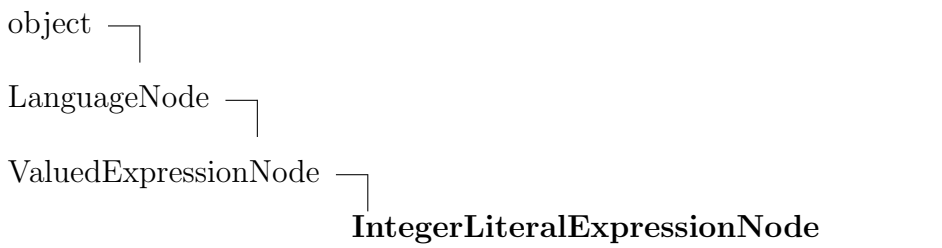
29.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

30. Módulo pytiger2c.ast.integerliteralexpressionnode

Clase `IntegerLiteralExpressionNode` del árbol de sintáxis abstracta.

30.1. Clase IntegerLiteralExpressionNode



Clase `IntegerLiteralExpressionNode` del árbol de sintáxis abstracta.

Representa un literal de un número entero en el lenguaje Tiger. El valor de retorno de esta expresión siempre será `IntegerType`.

30.1.1. Métodos

integer(*self*)

Método para obtener el valor de la propiedad `integer`.

__**init**__(*self*, *integer*)

Inicializa la clase `IntegerLiteralExpressionNode`.

Argumentos

`integer`: Valor del número entero literal.

*(type=**`int`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Este nodo del árbol de sintáxis abstracta no requiere comprobación semántica, solamente se da valor al tipo de retorno del nodo que siempre será `IntegerType`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

30.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

30.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

31. Módulo pytiger2c.ast.languagenode

Clase base de la jerarquía de los nodos del árbol de sintáxis abstracta.

31.1. Clase LanguageNode

object
└ **LanguageNode**

Clase base de la jerarquía de los nodos del árbol de sintáxis abstracta.

Todas las clases deben heredar de la clase base `LanguageNode` e implementar los métodos `check_semantics` y `generate_code` según corresponda a la estructura del lenguaje que representa.

31.1.1. Métodos

scope(*self*)

Método para obtener el valor de la propiedad `scope`.

Valor de retorno

Ámbito en el que se ejecuta el nodo. En el caso de que el nodo defina un ámbito, entonces esta función lo devuelve.

(type=`Scope`)

__init__(*self*)

Inicializa el nodo del árbol de sintáxis abstracta.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Comprueba que la estructura del lenguaje Tiger representada por el nodo sea correcta semánticamente.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

(type=`Scope`)

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

(type=`list`)

has_return_value(*self*)

Comprueba que la expresión representada por el nodo tiene un valor.

Este método es utilizado por las clases descendientes de `LanguageNode` que necesitan comprobar si la expresión representada por un nodo tiene o no valor de retorno. Los descendientes de `LanguageNode` deben implementar este método.

El objetivo de este método es poder considerar de forma transparente los casos especiales de expresiones que generalmente tienen un valor de retorno, por lo que derivan de `ValuedExpressionNode`, pero que en ocasiones no retornan un valor. Estos casos son los siguientes:

1. Una expresión `let`, representada por `LetNode` que no tenga expresiones entre `in` y `end`.
2. Una secuencia de expresiones vacía, representada por `ExpressionSequenceNode`.
3. Llamada a un procedimiento, representado por `FunctionCallNode` al igual que un llamado a función que sí retorna valor. Los llamados a funciones y procedimientos están representados por un mismo nodo del árbol de sintáxis abstracta ya que no es posible establecer la diferencia durante el análisis sintáctico.

La mayoría de los descendientes de `LanguageNode` utilizarán la implementación de este método provista por `ValuedExpressionNode` y `NonValuedExpressionNode`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

(type=`DotGenerator`)

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

generate_code(*self*, *generator*)

Genera el código C correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

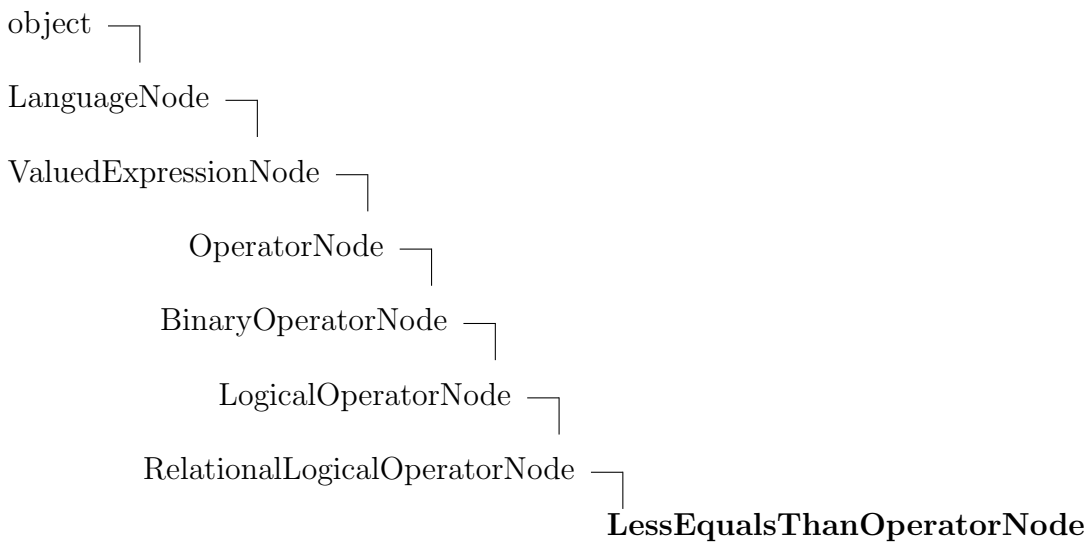
31.1.3. Variables de clase

Nombre	Descripción
line_number	**Value:** `property(_get_line_number, _set_line_number)`
parent_node	**Value:** `property(_get_parent_node, _set_parent_node)`

32. Módulo pytiger2c.ast.lessequalsthanoperatornode

Clase `LessEqualsThanOperatorNode` del árbol de sintáxis abstracta.

32.1. Clase LessEqualsThanOperatorNode



Clase `LessEqualsThanOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de suma <= entre dos números enteros o dos cadenas de caracters del lenguaje Tiger.

32.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `LessEqualsThanOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de RelationalLogicalOperatorNode (Sección 47.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

32.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

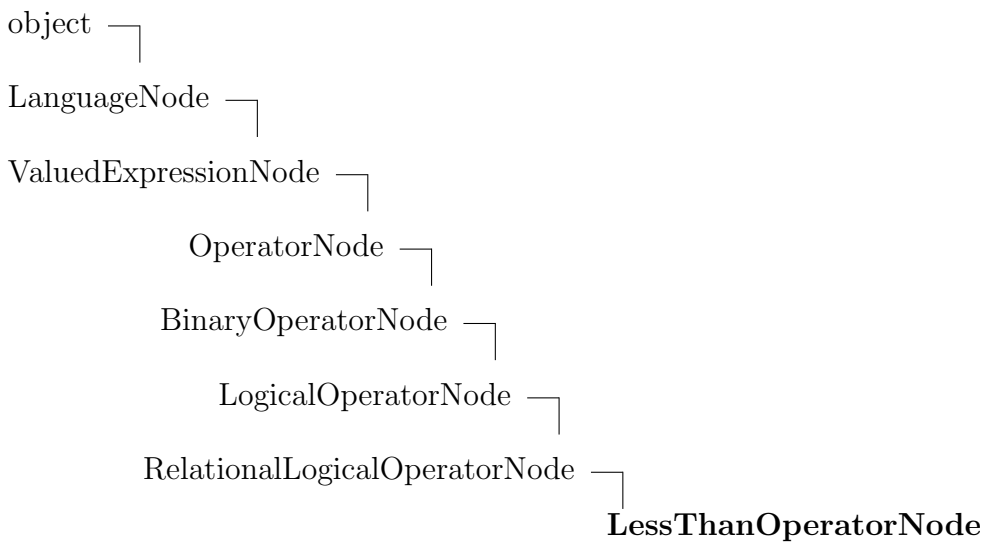
32.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

33. Módulo pytiger2c.ast.lessthanoperatornode

Clase `LessThanOperatorNode` del árbol de sintáxis abstracta.

33.1. Clase LessThanOperatorNode



Clase `LessThanOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de suma < entre dos números enteros o dos cadenas de caracters del lenguaje Tiger.

33.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `LessThanOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de RelationalLogicalOperatorNode (Sección 47.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

33.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

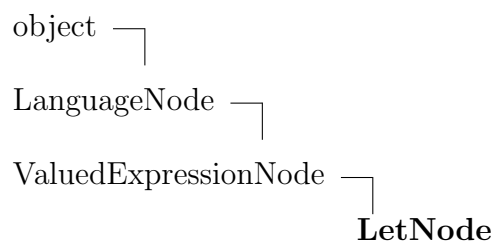
33.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

34. Módulo pytiger2c.ast.letnode

Clase `LetNode` del árbol de sintáxis abstracta.

34.1. Clase LetNode



Clase `LetNode` del árbol de sintáxis abstracta.

Representa la expresión `let-in-end` del lenguaje Tiger. La expresión `let-in-end` define tipos y funciones especificadas luego de la instrucción `let` y antes de la instrucción `in` de forma tal que esas declaraciones estén disponibles en el ámbito de ejecución de la secuencia de expresiones que se encuentran detrás de la instrucción `in` y antes de la instrucción `end`.

La expresión `let-in-end` retorna valor si la secuencia de expresiones retorna valor y su tipo de retorno es el mismo que la secuencia de expresiones.

34.1.1. Métodos

type_declaration_groups(*self*)

Método para obtener el valor de la propiedad `type_declaration_groups`.

function_declaration_groups(*self*)

Método para obtener el valor de la propiedad `function_declaration_groups`.

variable_declarations(*self*)

Método para obtener el valor de la propiedad `variable_declarations`.

expressions(*self*)

Método para obtener el valor de la propiedad `expressions`.

__init__(*self*, *type_declaration_groups*, *function_declaration_groups*, *variable_declarations*, *expressions*)

Inicializa la clase `LetNode`.

Argumentos

`type_declaration_groups:`	Lista de los grupos de las declaraciones de tipos, que forman parte de la lista de declaraciones de la estructura `let-in-end` representada por este nodo del árbol de sintáxis abstracta. *(type=`list`)*
`function_declaration_groups:`	Lista de los grupos de las declaraciones de funciones, que forman parte de la lista de declaraciones de la estructura `let-in-end` representada por este nodo del árbol de sintáxis abstracta. *(type=`list`)*
`variable_declarations:`	Lista de las declaraciones de variables que forman parte de la lista de declaraciones de la estructura `let-in-end` representada por este nodo del árbol de sintáxis abstracta. *(type=`list`)*
`expressions:`	Sequencia de expresiones que forman parte del cuerpo de la expresión `let-in-end` representada por este nodo del árbol de sintáxis abstracta. *(type=`ExpressionSequenceNode`)*

Overrides: object.__init__

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta primeramente se crea el nuevo ámbito que define la estructura y que tendrá como padre el ámbito donde se define la estructura `let-in-end`.

Se recorren todos los grupos de declaraciones de tipos, definiendo los tipos en el ámbito creado y se obtiene un conjunto de los tipos que se declaran en cada grupo a través del método `collect_definitions` y luego se comprueba semánticamente cada una de estas definiciones de tipos.

Se recorren todos los grupos de declaraciones de funciones, comprobando la cabecera de la declaración de la función, definiendo las funciones en el ámbito creado y se obtiene un conjunto de las funciones que se definen en cada grupo través del método `collect_definitions`.

Se comprueban semánticamente todas las declaraciones de variables.

Se comprueban semánticamente los cuerpos de las funciones declaradas en cada grupo de declaración de funciones.

Se comprueba semánticamente la secuencia de expresiones de la estructura `let-in-end` y se asigna el valor de retorno del nodo, en caso de que lo tenga.

En el procedimiento de comprobación semántica descrito anteriormente se realizan dos recorridos por ciertas partes del árbol de sintáxis abstracta descendiente de este nodo. Se recorren dos veces los nodos correspondientes a las declaraciones de tipos: una primera vez para obtener los nomrbes de los tipos que se definen en cada grupo y luego para comprobar semánticamente estas declaraciones de tipos. De manera semejante, se recorren dos veces los nodos del árbol de sintáxis abstracta correspondientes a las declaraciones de las funciones: una primera vez para obtener los nombres de las funciones que se definen en cada grupo y una segunda vez para comprobar semánticamente el cuerpo de las funciones.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

34.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

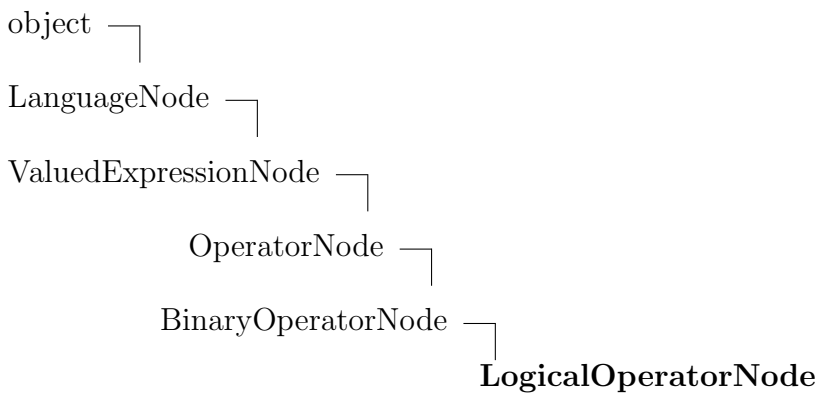
34.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

35. Módulo pytiger2c.ast.logicaloperatornode

Clase `LogicalOperatorNode` del árbol de sintáxis abstracta.

35.1. Clase LogicalOperatorNode



Clase `LogicalOperatorNode` del árbol de sintáxis abstracta.

35.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `LogicalOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

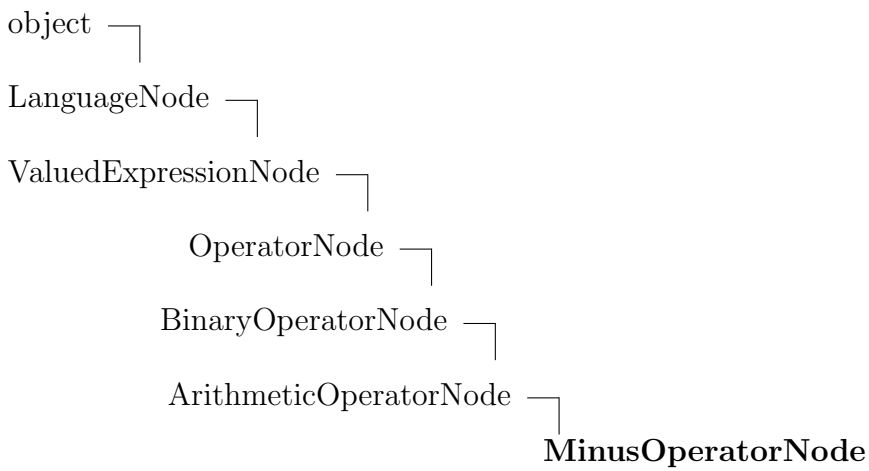
35.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1) line_number, parent_node	

36. Módulo pytiger2c.ast.minusoperatornode

Clase `MinusOperatorNode` del árbol de sintáxis abstracta.

36.1. Clase MinusOperatorNode



Clase `MinusOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de resta `-` entre dos números enteros del lenguaje Tiger.

36.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `MinusOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

El operador resta realiza la diferencia entre los el valor de la expresión que se encuentra a la izquierda con el valor de la derecha.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el valor de retorno de ambas sea entero.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

Heredados de ArithmeticOperatorNode (Sección 6.1)

generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

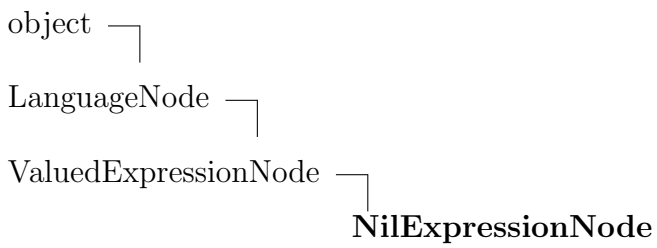
36.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

37. Módulo pytiger2c.ast.nilexpressionnode

Clase `NilExpressionNode` del árbol de sintáxis abstracta.

37.1. Clase NilExpressionNode



Clase `NilExpressionNode` del árbol de sintáxis abstracta.

Representa la palabra reservada `nil` del lenguaje Tiger. El valor de retorno de esta expresión siempre será `nil`.

37.1.1. Métodos

__**init**__(*self*)
Inicializa la clase `NilExpressionNode`.
Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Este nodo del árbol de sintáxis abstracta no requiere comprobación semántica, solamente se da valor al tipo de retorno del nodo que siempre será `NilType`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

37.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

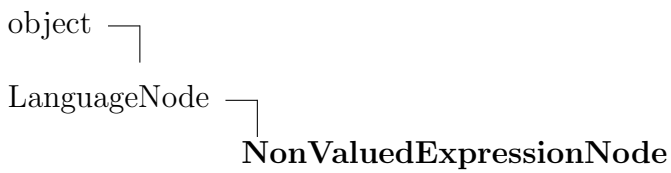
37.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

38. Módulo pytiger2c.ast.nonvaluedexpressionnode

Clase `NonValuedExpressionNode` del árbol de sintáxis abstracta.

38.1. Clase NonValuedExpressionNode



Clase `NonValuedExpressionNode` del árbol de sintáxis abstracta.

38.1.1. Métodos

__init__(*self*)

Inicializa la clase `NonValuedExpressionNode`.

Overrides: object.__init__

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

38.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

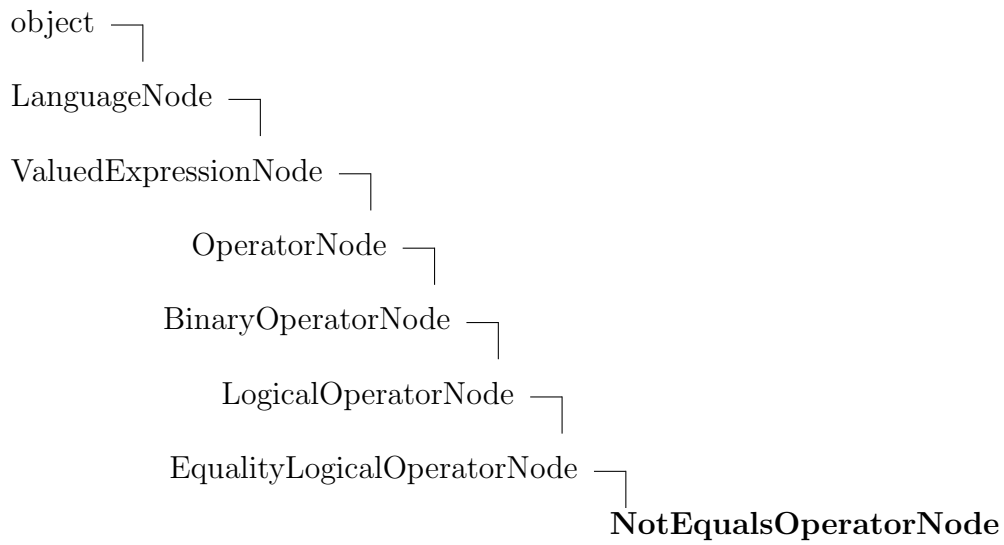
38.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

39. Módulo pytiger2c.ast.notequalsoperatornode

Clase `NotEqualsOperatorNode` del árbol de sintáxis abstracta.

39.1. Clase NotEqualsOperatorNode



Clase `NotEqualsOperatorNode` del árbol de sintáxis abstracta.

Representa el operador <> entre dos expresiones del lenguaje Tiger.

39.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `NotEqualsOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

Heredados de EqualityLogicalOperatorNode (Sección 18.1)

check_semantics(), generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

39.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

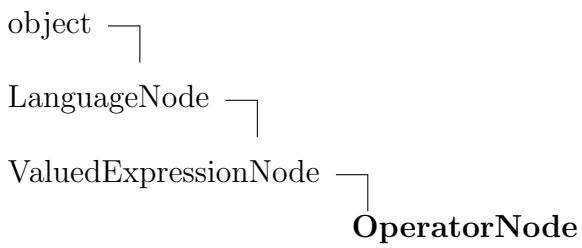
39.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

40. Módulo pytiger2c.ast.operatornode

Clase `OperatorNode` del árbol de sintáxis abstracta.

40.1. Clase OperatorNode



Clase `OperatorNode` del árbol de sintáxis abstracta.

40.1.1. Métodos

__**init**__(*self*)
Inicializa la clase `OperatorNode`.
Overrides: object.__init__

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

40.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

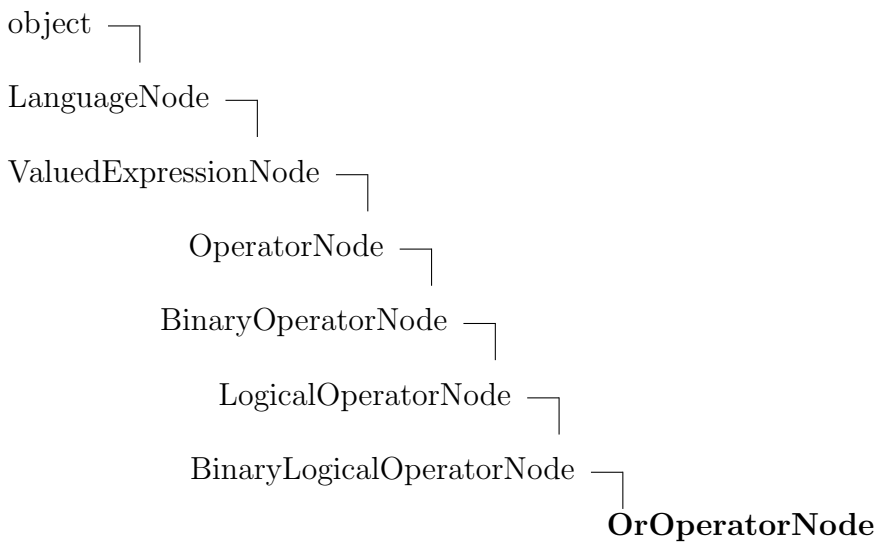
40.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

41. Módulo pytiger2c.ast.oroperatornode

Clase `OrOperatorNode` del árbol de sintáxis abstracta.

41.1. Clase OrOperatorNode



Clase `OrOperatorNode` del árbol de sintáxis abstracta.

Representa la operación lógica `OR`, representada con el operador | en Tiger, entre dos números enteros. Este operador retornará 1 en caso de que el resultado de evaluar la expresión sea verdadero, 0 en otro caso.

41.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `OrOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera el código C correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryLogicalOperatorNode (Sección 11.1)

check_semantics()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

41.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

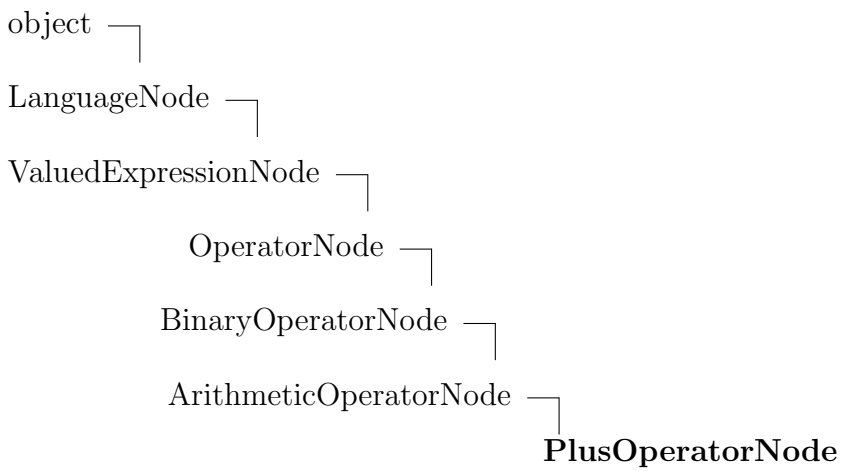
41.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

42. Módulo pytiger2c.ast.plusoperatornode

Clase `PlusOperatorNode` del árbol de sintáxis abstracta.

42.1. Clase PlusOperatorNode



Clase `PlusOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de suma `+` entre dos números enteros del lenguaje Tiger.

42.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `PlusOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

El operador suma realiza la suma entre los el valor de la expresión que se encuentra a la izquierda con el valor de la derecha.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el valor de retorno de ambas sea entero.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

Heredados de ArithmeticOperatorNode (Sección 6.1)

generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

42.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

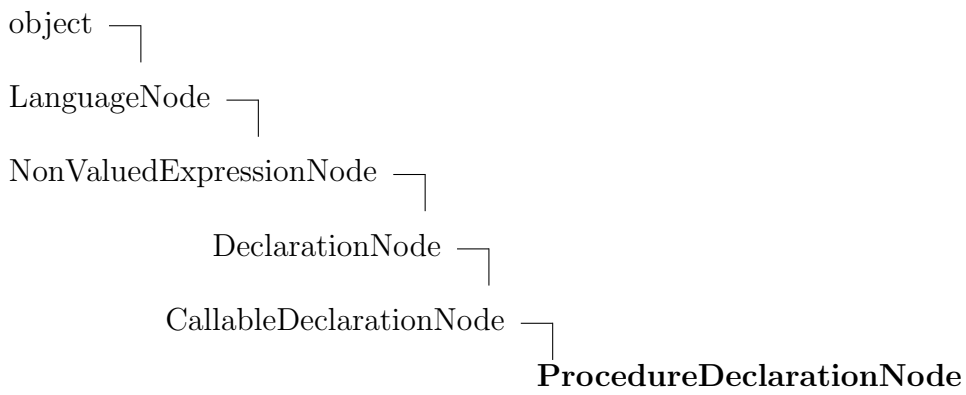
42.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

43. Módulo pytiger2c.ast.proceduredeclarationnode

Clase `ProcedureDeclarationNode` del árbol de sintáxis abstracta.

43.1. Clase ProcedureDeclarationNode



Clase `ProcedureDeclarationNode` del árbol de sintáxis abstracta.

Este nodo representa la declaración de un procedimiento en en lenguaje Tiger. Un procedimiento es una función que no tiene valor de retorno y que sólo se llamará por sus efectos colaterales.

43.1.1. Métodos

__init__(*self*, *name*, *parameters_names*, *parameters_typenames*, *body*)

Inicializa la clase `ProcedureDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `CallableDeclarationNode`.

Argumentos

`name:`	Nombre del procedimiento o función cuya definición es representada por el nodo.
`fields_names:`	Lista con los nombres de los parámetros de la función o procedimiento, por posición.
`fields_typenames:`	Lista con los nombres de los tipos de los parámetros de la función o procedimiento, por posición.
`body:`	Nodo del árbol de sintáxis abstracta correspondiente al cuerpo del procedimiento o función.

Overrides: object.__init__

check_header_semantics(*self*, *scope*, *errors*)

Este método realiza la comprobación semántica de la cabecera específica para la declaración de procedimientos. Para más información consulte la documentación método `check_header_semantics` en la clase `CallableDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Overrides: pytiger2c.ast.callabledeclarationnode.CallableDeclarationNode.check_header_semantics

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La comprobación semántica de este nodo del árbol de sintáxis abstracta está dividida en dos partes: la comprobación semántica de la cabecera a través del método `check_header_semantics` y la comprobación semántica del cuerpo a través del método `check_semantics`.

En este método se crea un nuevo ámbito que tendrá como padre el ámbito en el que se está definiendo la función y contendrá las definiciones de las variables correspondientes a los parámetros recibidos por el procedimiento. Luego, se comprueba semánticamente el cuerpo del procedimiento, el cual no debe tener valor de retorno.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de CallableDeclarationNode (Sección 14.1)

body(), generate_code(), name(), parameters_names(), parameters_typenames(), type()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

43.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

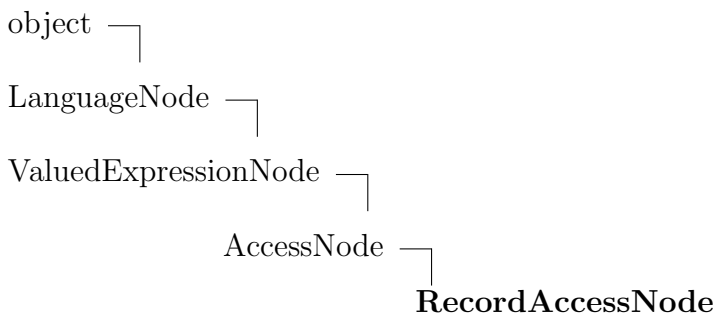
43.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

44. Módulo pytiger2c.ast.recordaccessnode

Clase `RecordAccessNode` del árbol de sintáxis abstracta.

44.1. Clase RecordAccessNode



Clase `RecordAccessNode` del árbol de sintáxis abstracta.

Representa la estructura de acceso a un campo de un record del lenguaje Tiger. La estructura de acceso a un campo de un record del lenguaje Tiger permite obtener el valor de un campo de un tipo record determinado o asignarle un nuevo valor a este record en este campo. Esta estructura recibe la expresión que representa el acceso al record y el nombre correspondiente al campo que se quiere acceder.

44.1.1. Métodos

field_name(*self*)

Método para obtener el valor de la propiedad `field_name`.

record(*self*)

Método para obtener el valor de la propiedad record.

__init__(*self*, *record*, *field_name*)

Inicializa la clase `RecordAccessNode`.

Argumentos

`record:` Expresión correspondiente al record que se quiere acceder.

*(type=**`LanguageNode`**)*

`field_name:` Nombre del campo del record al que se quiere acceder.

*(type=**`str`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de acceso a un campo de un record del lenguaje Tiger permite obtener el valor de un campo de un tipo record determinado o asignarle un nuevo valor a este record en este campo. Esta estructura recibe la expresión que representa el acceso al record y el nombre correspondiente al campo que se quiere acceder.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se verifica que la expresión que se correspondiente al record retorne valor y que este sea del tipo record, luego se comprueba que el tipo record tenga un campo con ese nombre.

En el proceso de comprobación semántica toma valor las propiedades `return_type` y `read_only`

Argumentos

`scope:` Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors:` Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

En particular el nodo de acceso a un `array`, no genera ninguna instrucción de código `C` sino que toma valor la propiedad `code_name`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de AccessNode (Sección 3.1)

read_only()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

44.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

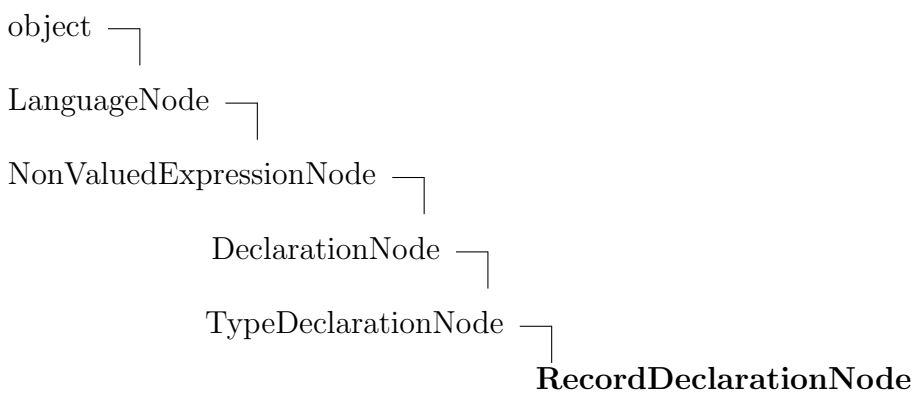
44.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

45. Módulo pytiger2c.ast.recorddeclarationnode

Clase `RecordDeclarationNode` del árbol de sintáxis abstracta.

45.1. Clase RecordDeclarationNode



Clase `RecordDeclarationNode` del árbol de sintáxis abstracta.

45.1.1. Métodos

fields_names(*self*)

Método para obtener el valor de la propiedad `fields_names`.

fields_typenames(*self*)

Método para obtener el valor de la propiedad `fields_typenames`.

__**init**__(*self*, *name*, *fields_names*, *fields_typenames*)

Inicializa la clase C{RecordDeclarationNode}.

@type fields_names: C{list}
@param fields_names: Lista con los nombres de los campos del record,
    por posición.

@type fields_typenames: C{list}
@param fields_typenames: Lista con los nombres de los tipos de los
    campos, por posición.

Para obtener información acerca del resto de los parámetros recibidos
por el método consulte la documentación del método C{__init__}
en la clase C{TypeDeclarationNode}.

Argumentos

`name`: Nombre que se le asignará a este nuevo tipo.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueba que los tipos de los campos del record se encuentren definidos en el ámbito local.

Se reportarán errores semánticos si alguno de los tipos de los campos no se encuentran definidos en el ámbito local, o en caso de que estén definidos en el ámbito local, pero en otro grupo de declaraciones, en cuyo caso se considera una declaración de tipos mutuamente recursivos en distintos grupos de declaraciones de tipos.

Durante la comprobación semántica se define totalmente el valor de la propiedad `type`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de TypeDeclarationNode (Sección 52.1)

type()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

45.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

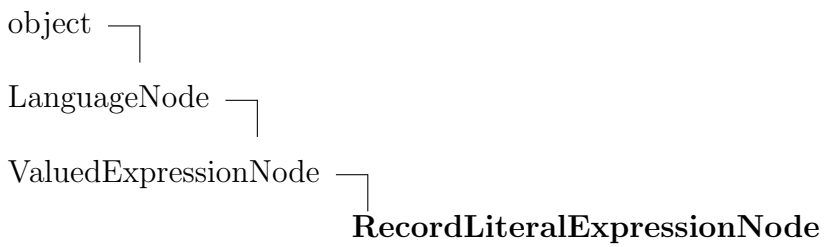
45.1.3. Variables de clase

Nombre	Descripción
Heredadas de TypeDeclarationNode (Sección 52.1) name	
Heredadas de LanguageNode (Sección 31.1) line_number, parent_node	

46. Módulo pytiger2c.ast.recordliteralexpressionnode

Clase `RecordLiteralExpressionNode` del árbol de sintáxis abstracta.

46.1. Clase RecordLiteralExpressionNode



Clase `RecordLiteralExpressionNode` del árbol de sintáxis abstracta.

Representa la creación de una instancia de un tipo record definido con anterioridad. La creación de una instancia de un tipo record recibe el nombre del tipo de record que se quiere crear, una lista con los nombres de los campos del record y otra lista con las expresiones correspondientes a los valores que se le quieren dar a cada campo del record.

46.1.1. Métodos

type_name(*self*)

Método para obtener el valor de la propiedad `type_name`.

fields_names(*self*)

Método para obtener el valor de la propiedad `fields_names`.

fields_values(*self*)

Método para obtener el valor de la propiedad `fields_values`.

__init__(*self*, *type_name*, *fields_names*, *fields_values*)

Inicializa la clase `RecordLiteralExpressionNode`.

Argumentos

`type_name`: Nombre del tipo record que se quiere crear.

*(type=**str**)*

`fields_names`: Lista con los nombres de los campos del record, en el mismo orden en que aparecen en el programa.

*(type=**list**)*

`fields_values`: Lista con las expresiones de los campos del record, en el mismo orden que aparecen en el programa.

*(type=**list**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La creación de una instancia de un tipo record recibe el nombre del tipo de record que se quiere crear, una lista con los nombres de los campos del record y otra lista con las expresiones correspondientes a los valores que se le quieren dar a cada campo del record.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueba que el tipo record que se quiere crear ha sido definido en el ámbito correspondiente, luego se comprueban que los campos.Cada campo debe tener exactamente el mismo nombre que el campo correspondiente en la declaración del tipo, en cuanto al tipo deben corresponder de igual manera, con la excepción de un tipo record en cuyo caso, es permitido también el tipo `nil`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

check_parameters(*self*, *errors*)

Comprueba semánticamente los campos dados. Cada campo debe tener exactamente el mismo nombre que el campo correspondiente en la declaración del tipo, en cuanto al tipo deben corresponder de igual manera, con la excepción de un tipo record en cuyo caso, es permitido también el tipo `nil`

Argumentos

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante esta comprobación.

(type=`list`)

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

46.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

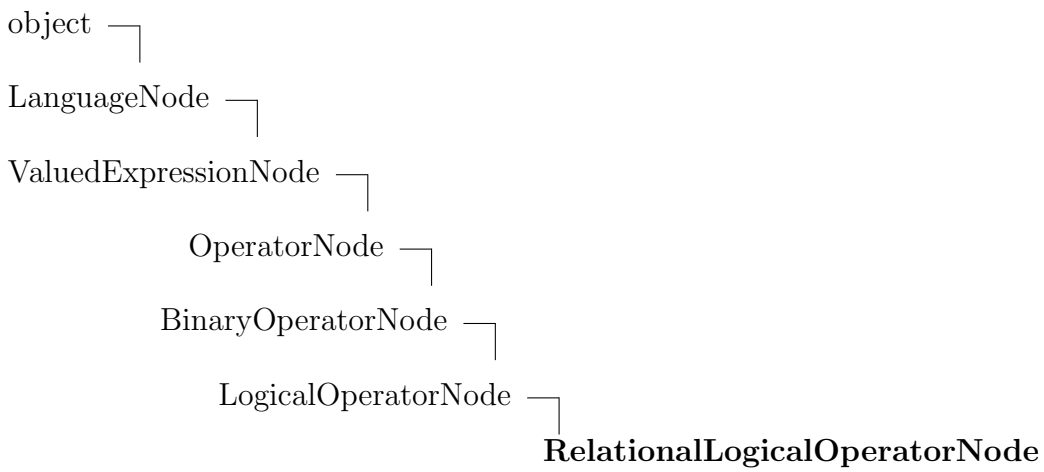
46.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

47. Módulo pytiger2c.ast.relationallogicaloperatornode

Clase `RelationalLogicalOperatorNode` del árbol de sintáxis abstracta.

47.1. Clase RelationalLogicalOperatorNode



Clase `RelationalLogicalOperatorNode` del árbol de sintáxis abstracta.

Esta clase implementa el método `check_semantics` para los operadores logicos binarios relacionales. Estos operadores son los siguientes: menor que <, menor igual que `<=`, mayor que {>} y mayor igual que `>=`.

47.1.1. Métodos

__init__(*self*, *left*, *right*)

Inicializa la clase `RelationalLogicalOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Los operadores cuyas clases del árbol de sintáxis abstracta derivan de esta deben recibir en ambos operandos números enteros o ambos cadenas de caracteres. Siempre tienen tipo de retorno entero (1 para el resultado verdadero, 0 para el falso).

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el tipo de retorno de ambas sea enteros o cadenas de caracteres.

Argumentos

`scope:` Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors:` Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código C correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Argumentos

`generator:` Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

47.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

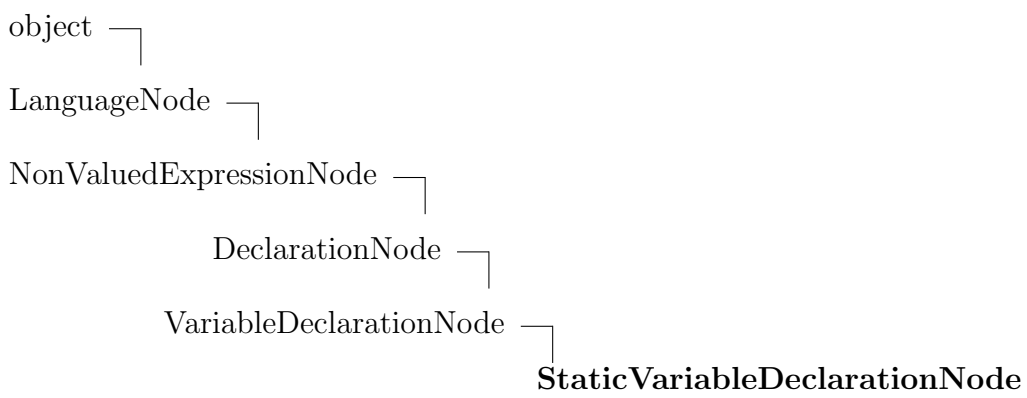
47.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

48. Módulo pytiger2c.ast.staticvariabledeclarationnode

Clase `StaticVariableDeclarationNode` del árbol de sintáxis abstracta.

48.1. Clase StaticVariableDeclarationNode



Clase `StaticVariableDeclarationNode` del árbol de sintáxis abstracta.

Representa la estructura de declaración de variables especificando explícitamente el tipo de esta del lenguaje Tiger. Esta estructura recibe una expresión cuyo valor se le asignará a la variable, además del tipo que tendrá la misma.

48.1.1. Métodos

type_name(*self*)

Método para obtener el valor de la propiedad `type_name`

__init__(*self*, *name*, *value*, *type_name*)

Inicializa la clase `StaticVariableDeclarationNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `BinaryOperatorNode`.

Argumentos

`type_name`: Nombre del tipo que se expresa explícitamente para esta variable

(type=`str`)

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de declaración de variables especificando explícitamente el tipo de esta recibe una expresión cuyo valor se le asignará a la variable , además del tipo que tendrá la misma.

En la comprobación semántica se comprueba semánticamente la expresión que se quiere asignar a la variable. Luego se comprueba que el tipo de la variable esté definido en el ámbito de esta, que la expresión que se le asigna retorne valor, que este valor sea del mismo tipo que el especificado o `nil` y que en su ámbito local el nombre que se quiere asignar a esta variable no haya sido asignado a una función u otra variable. Se reportarán errores si se encuentran errores durante la comprobación semántica de la expresión, si esta no retorna valor o este no es del mismo tipo que el de la variable o `nil` y por último si el nombre de la variable ya ha sido asignado a una función u otra variable en su ámbito local.

En el proceso de comprobación semántica la propiedad `type` toma valor y la variable es definida en su ámbito local.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de VariableDeclarationNode (Sección 57.1)

generate_code(), name(), type(), value()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

48.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

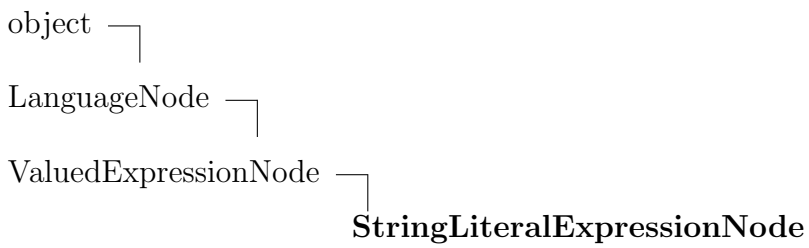
48.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

49. Módulo pytiger2c.ast.stringliteralexpressionnode

Clase `StringLiteralExpressionNode` del árbol de sintáxis abstracta.

49.1. Clase StringLiteralExpressionNode



Clase `StringLiteralExpressionNode` del árbol de sintáxis abstracta.

Representa un literal de cadena en el lenguaje Tiger. El valor de retorno de esta éxpresión siempre será `StringType`.

49.1.1. Métodos

string(*self*)

Método para obtener el valor de la propiedad `string`.

__**init**__(*self*, *string*)

Inicializa la clase `StringLiteralExpressionNode`.

Argumentos

`string`: Valor del literal de cadena.

*(type=**`str`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Este nodo del árbol de sintáxis abstracta no requiere comprobación semántica, solamente se da valor al tipo de retorno del nodo que siempre será `StringType`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

49.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

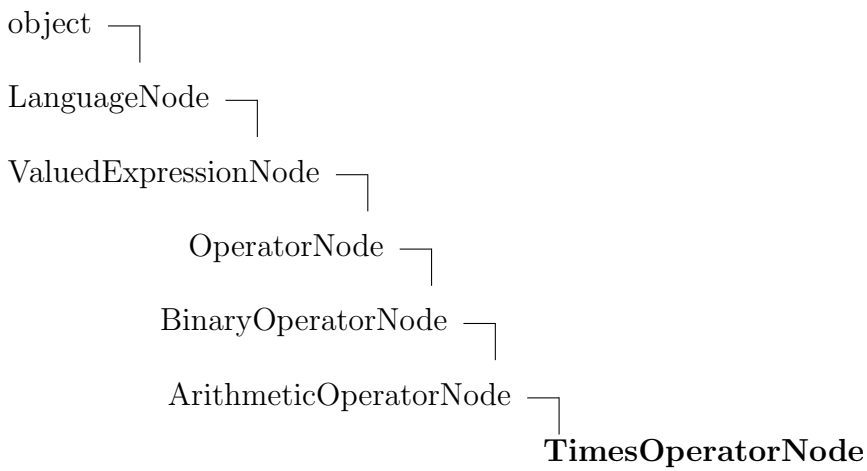
49.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

50. Módulo pytiger2c.ast.timesoperatornode

Clase `TimesOperatorNode` del árbol de sintáxis abstracta.

50.1. Clase TimesOperatorNode



Clase `MinusOperatorNode` del árbol de sintáxis abstracta.

Representa el operador de multiplicación `*` entre dos números enteros del lenguaje Tiger.

50.1.1. Métodos

__init__(*self*, *left*, *right*)

```
 Inicializa la clase C{TimesOperatorNode}.

Para obtener información acerca de los parámetros recibidos por este
método consulte la documentación del método C{__init__} en la clase
C{BinaryOperatorNode}.
```

Argumentos

`left`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la izquierda del operador.

`right`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión a la derecha del operador.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

El operador por realiza la multiplicación entre los el valor de la expresión que se encuentra a la izquierda por el valor de la derecha.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la expresión de la izquierda como la expresión de la derecha. Luego se comprueba que ambas retornen valor y que el valor de retorno de ambas sea entero.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

Heredados de ArithmeticOperatorNode (Sección 6.1)

generate_code()

Heredados de BinaryOperatorNode (Sección 12.1)

generate_dot(), left(), right()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

50.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

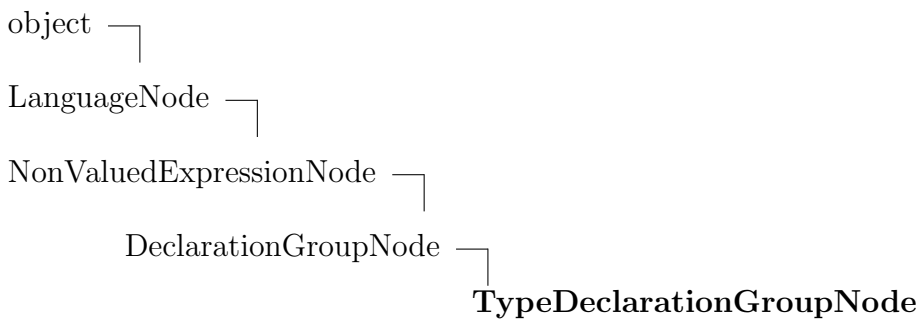
50.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

51. Módulo pytiger2c.ast.typedeclarationgroupnode

Clase `TypeDeclarationGroupNode` del árbol de sintáxis abstracta.

51.1. Clase TypeDeclarationGroupNode



Clase `TypeDeclarationGroupNode` del árbol de sintáxis abstracta.

Representa un grupo de declaraciones de tipos del lenguaje Tiger.

Un grupo de declaraciones de tipos del lenguaje Tiger se forma por declaraciones de tipos que aparecen uno a continuación de otros. Los tipos mutuamente recursivos deben estar definidos en un mismo grupo de declaraciones de tipos, de modo que no es válido declarar tipos mutuamente recursivos con una declaración de variable o función entre ellos ya que conduce a situaciones ambiguas.

51.1.1. Métodos

__**init**__(*self*)
Inicializa la clase `TypeDeclarationGroupNode`.
Overrides: object.__init__

collect_definitions(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Realiza la definición en el ámbito dado de los tipos definidos en este grupo de declaraciones. En el caso de los alias, se resuelve y se define el tipo concreto al que referencia.

Se reportarán errores si se referencia a un tipo que no se encuentra definido en el ámbito actual o si se declaran alias mutuamente referenciados, en cuyo caso se forma un ciclo de definiciones.

Valor de retorno

Conjunto con los nombres de los tipos definidos en este grupo.

(type=`set`)

Overrides:
pytiger2c.ast.declarationgroupnode.DeclarationGroupNode.collect_definitions

check_aliases_semantics(*self*, *scope*, *errors*)

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Realiza la comprobación semántica de los alias definidos en este grupo.

Se reportarán errores si se producen errores en la comprobación semántica de alguna de las declaraciones de alias contenidas en este grupo.

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca del resto de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

Un grupo de declaraciones de tipos del lenguaje Tiger se forma por declaraciones de tipos que aparecen uno a continuación de otro. Tipos definidos mutuamente recursivos deben estar definidos en el mismo grupo de declaraciones de tipos. Por tanto, no es valido declarar tipos mutuamente recursivos con una declaración de variable o función entre estos.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente todas la declaraciones contenidas en este.

Se reportarán errores si se producen errores en la comprobación semántica de alguna de las declaraciones contenidas en este grupo.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de DeclarationGroupNode (Sección 15.1)

declarations(), generate_dot()

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

51.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

51.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	

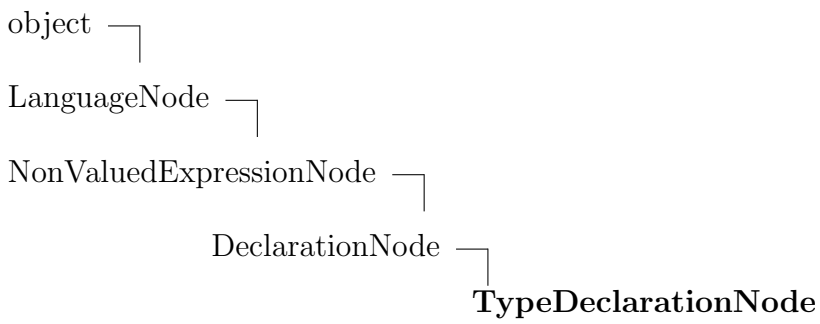
continúa en la página siguiente

Nombre	Descripción
line_number, parent_node	

52. Módulo pytiger2c.ast.typedeclarationnode

Clase `TypeDeclarationNode` del árbol de sintáxis abstracta.

52.1. Clase TypeDeclarationNode



Clase `TypeDeclarationNode` del árbol de sintáxis abstracta.

Representa las distintas declaraciones de tipos presentes en el lenguaje de Tige. De esta clase heredan las declaraciones de records, arrays y alias como tipos válidos de Tiger.

52.1.1. Métodos

type(*self*)

Método para obtener el valor de la propiedad `type`.

__init__(*self*, *name*)

Inicializa la clase `TypeDeclarationNode`.

Argumentos

`name`: Nombre que se le asignará a este nuevo tipo.

(type=`str`)

Overrides: object.__init__

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

52.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

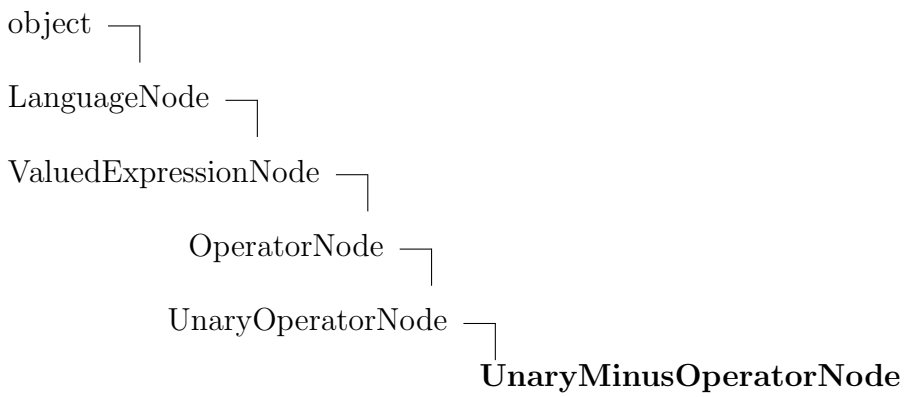
52.1.3. Variables de clase

Nombre	Descripción
name	**Value:** `property(_get_name, _set_name)`
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

53. Módulo pytiger2c.ast.unaryminusoperatornode

Clase `UnaryMinusOperatorNode` del árbol de sintáxis abstracta.

53.1. Clase UnaryMinusOperatorNode



Clase `UnaryMinusOperatorNode` del árbol de sintáxis abstracta.

Este nodo representa el operador menos unario `-` del lenguaje Tiger. Este operador se utiliza para cambiar el signo de expresiones que devuelvan valores enteros.

53.1.1. Métodos

__init__(*self*, *expression*)

Inicializa la clase `UnaryMinusOperatorNode`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `UnaryOperatorNode`.

Argumentos

`expression`: Nodo del árbol de sintáxis abstracta representando la expresión a la que se va a aplicar el operador unario.

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

El operador menos unario se aplica solamente a expresiones que devuelvan enteros y el tipo de retorno siempre será entero.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueba que la expresión a la que se le va aplicar el operador menos unario esté correcta semánticamente y que tenga valor de retorno entero. El tipo del valor de retorno de la expresión representada por este nodo siempre será `IntegerType`.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de UnaryOperatorNode (Sección 54.1)

expression(), generate_dot()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

53.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

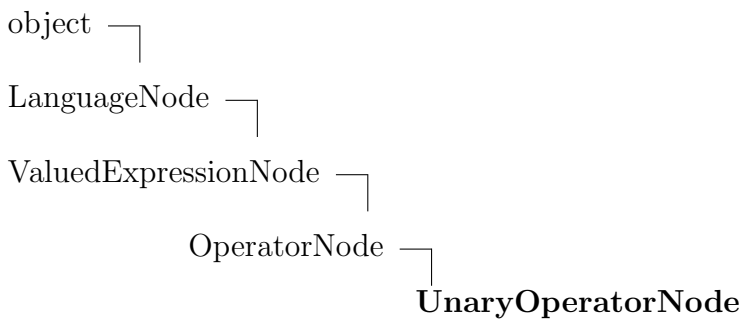
53.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

54. Módulo pytiger2c.ast.unaryoperatornode

Clase `UnaryOperatorNode` del árbol de sintáxis abstracta.

54.1. Clase UnaryOperatorNode



Clase `UnaryOperatorNode` del árbol de sintáxis abstracta.

Esta clase es la clase base de todos los nodos del árbol de sintáxis abstracta que representan operadores unarios de Tiger.

54.1.1. Métodos

expression(*self*)

Método para obtener el valor de la propiedad `expression`.

__**init**__(*self*, *expression*)

Inicializa la clase `UnaryOperatorNode`.

Argumentos

`expression`: Nodo del árbol de sintáxis abstracta representando la expresión a la que se va a aplicar el operador unario.

(type=`LanguageNode`)

Overrides: object.__init__

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

54.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

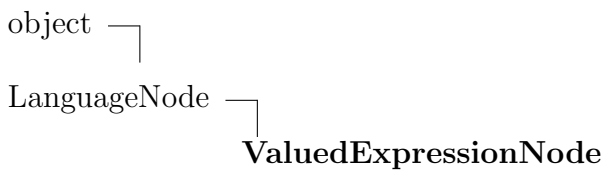
54.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

55. Módulo pytiger2c.ast.valuedexpressionnode

Clase `ValuedExpressionNode` del árbol de sintáxis abstracta.

55.1. Clase ValuedExpressionNode



Clase `ValuedExpressionNode` del árbol de sintáxis abstracta.

Esta clase encabeza la jerarquía de clases que representan estructuras del lenguaje Tiger que tienen un valor de retorno. Todas las clases que deriven de `ValuedExpressionNode` deben definir el valor de la propiedad `return_type`, según el tipo de retorno de la expresión, al terminar la ejecución del método `check_semantics`. El valor retornado por la propiedad `return_type` será una instancia de un descendiente de la clase `TigerType` ó `None` en el caso excepcional en que la expresión no tenga ningún valor de retorno.

55.1.1. Métodos

return_type(*self*)

Método para obtener el valor de la propiedad `return_type`.

code_name(*self*)

Método para obtener el valor de la propiedad `code_name`.

__init__(*self*)

Inicializa la clase `ValuedExpressionNode`.

Overrides: object.__init__

has_return_value(*self*)

Ver documentación del método `has_return_value` en `LanguageNode`.

Valor de retorno

Valor booleano indicando si la expresión representada por el nodo tiene valor de retorno.

(type=`bool`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.has_return_value

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_code(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

55.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

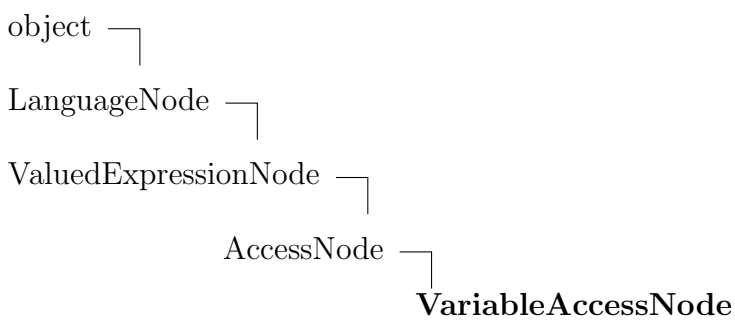
55.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

56. Módulo pytiger2c.ast.variableaccessnode

Clase `VariableAccessNode` del árbol de sintáxis abstracta.

56.1. Clase VariableAccessNode



Clase `VariableAccessNode` del árbol de sintáxis abstracta.

Representa la estructura de acceso a variable del lenguaje Tiger. La estructura de acceso a variable del leguaje Tiger permite obtener el valor de una variable y asignare un nuevo valor a esta. Esta estructura recibe el nombre de la variable que representa.

56.1.1. Métodos

name(*self*)

Método para obtener el valor de la propiedad `name`.

__**init**__(*self*, *name*)

Inicializa la clase `VariableAccessNode`.

Argumentos

`name`: Nombre de la variable a la que representa.

*(type=**`str`**)*

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La estructura de acceso a variable del leguaje Tiger permite obtener el valor de una variable y asignare un nuevo valor a esta. Esta estructura recibe el nombre de la variable que representa.

En la comprobación semántica de este nodo del árbol se verifica que la variable esté definida en el ámbito que ocurre su acceso. Se reportarán errores si la variable no está definida en el ámbito local.

En el proceso de comprobación semántica toma valor las propiedades `return_type` y `read_only`

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

En particular el nodo de acceso a una variable, no genera ninguna instrucción de código `C` sino que toma valor la propiedad `code_name`.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de AccessNode (Sección 3.1)

read_only()

Heredados de ValuedExpressionNode (Sección 55.1)

code_name(), has_return_value(), return_type()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

56.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

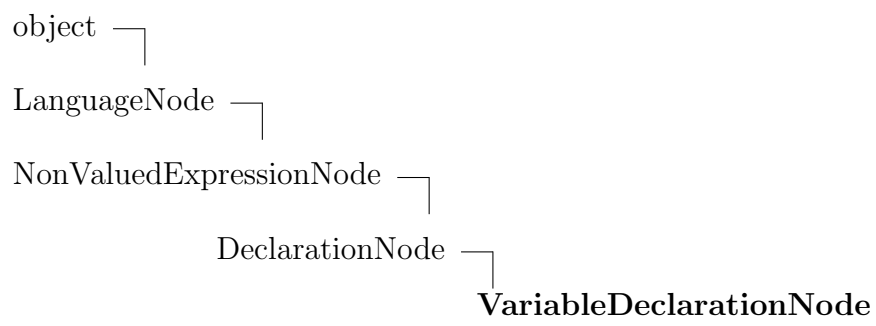
56.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

57. Módulo pytiger2c.ast.variabledeclarationnode

Clase `VariableDeclarationNode` del árbol de sintáxis abstracta.

57.1. Clase VariableDeclarationNode



Clase `VariableDeclarationNode` del árbol de sintáxis abstracta.

57.1.1. Métodos

name(*self*)

Método para obtener el valor de la propiedad `name`

value(*self*)

Método para obtener el valor de la propiedad `value`

type(*self*)

Método para obtener el valor de la propiedad `type`

__**init**__(*self*, *name*, *value*)

Inicializa la clase `VariableDeclarationNode`.

Argumentos

`name`: Nombre de la variable.

*(type=**`str`**)*

`value`: `LanguageNode` correspondiente al valor que se asigna a la variable.

*(type=**`LanguageNode`**)*

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

check_semantics(), generate_dot(), scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

57.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

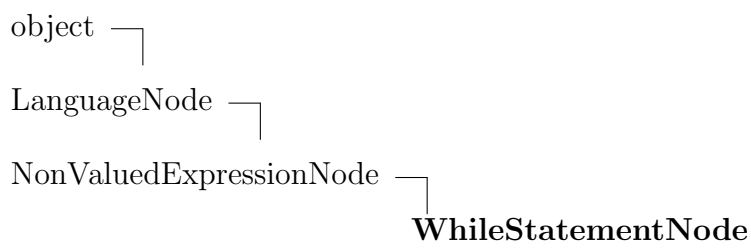
57.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

58. Módulo pytiger2c.ast.whilestatementnode

Clase `WhileStatementNode` del árbol de sintáxis abstracta.

58.1. Clase WhileStatementNode



Clase `WhileStatementNode` del árbol de sintáxis abstracta.

Representa la expresión `while-do` del lenguaje Tiger. La expresión `while-do` tiene una condición y una expresión, de forma que evalua la condición y si esta es distinta de cero, entonces la expresión es ejecutada.

58.1.1. Métodos

condition(*self*)

Método para obtener el valor de la propiedad `condition`.

expression(*self*)

Método para obtener el valor de la propiedad `expression`.

__init__(*self*, *condition*, *expression*)

Inicializa la clase `WhileStatementNode`.

Argumentos

`condition`: Nodo del árbol de sintáxis abstracta correspondiente a la condición que es evaluada, de forma que si su valor es distinto de cero, la expresión es ejecutada.

(type=`LanguageNode`)

`expression`: Nodo del árbol de sintáxis abstracta correspondiente a la expresión que es ejecutada, una vez que se verifica que la condición anterior es distinta de cero.

(type=`LanguageNode`)

Overrides: object.__init__

check_semantics(*self*, *scope*, *errors*)

Para obtener información acerca de los parámetros recibidos por el método consulte la documentación del método `check_semantics` en la clase `LanguageNode`.

La expresión `while-do` recibe una condición y una expresión, de forma que evalua la condición y si esta es distinta de cero, entonces la expresión es ejecutada.

En la comprobación semántica de este nodo del árbol de sintáxis abstracta se comprueban semánticamente tanto la condición como la expresión contenidas en este. Luego se comprueba que la condición retorne valor, que el mismo sea de tipo `IntegerType` y que la expresión no retorne valor. Se reportarán errores semánticos si se encuentran errores durante la comprobación semántica de la condición o la expresión, si la condición no retorna valor, si este valor de retorno no es te tipo `IntegerType` o si la expresión retorna algún valor.

Argumentos

`scope`: Ámbito en el que se ejecuta el nodo. Si un nodo define un ámbito nuevo entonces, creará una nueva instancia de `Scope` que tendrá como padre este ámbito. En ambos casos la propiedad `scope` será asignada al ámbito del nodo.

`errors`: Lista a la cual se deben añadir los mensajes de error de los errores semánticos encontrados durante la comprobación de la estructura del lenguaje representada por el nodo del árbol de sintáxis abstracta.

Overrides: pytiger2c.ast.languagenode.LanguageNode.check_semantics

generate_dot(*self*, *generator*)

Genera un grafo en formato Graphviz DOT correspondiente al árbol de sintáxis abstracta del programa Tiger del cual este nodo es raíz.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_dot` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código Graphviz DOT.

Valor de retorno

Identificador del nodo del grafo generado correspondiente a este todo del árbol de sintáxis abstracta. Este identificador podrá ser utilizado por otros nodos para añadir aristas al grafo que tengan este nodo como uno de sus extremos.

(type=`str`)

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_dot

generate_code(*self*, *generator*)

Genera el código correspondiente a la estructura del lenguaje Tiger representada por el nodo.

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `generate_code` de la clase `LanguageNode`.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

Excepciones

`CodeGenerationError` Esta excepción se lanzará cuando se produzca algún error durante la generación del código correspondiente al nodo. La excepción contendrá información acerca del error.

Overrides: pytiger2c.ast.languagenode.LanguageNode.generate_code

Heredados de NonValuedExpressionNode (Sección 38.1)

has_return_value()

Heredados de LanguageNode (Sección 31.1)

scope()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

58.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

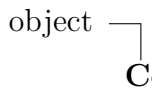
58.1.3. Variables de clase

Nombre	Descripción
Heredadas de LanguageNode (Sección 31.1)	
line_number, parent_node	

59. Módulo pytiger2c.code

Clases utilizadas en la generación código C a partir de un programa Tiger.

59.1. Clase CodeGenerator



Generador de código C.

59.1.1. Métodos

__**init**__(*self*, *stdlib_dir*=`None`)

Inicializa el generador de código C.

Argumentos

`stdlib_dir`: Ruta absoluta al directorio que contiene los archivos de código C con la implementación de la biblioteca standard de Tiger y las funciones y tipos auxiliares utilizados por PyTiger2C.

*(type=**`str`**)*

Overrides: object.__init__

define_struct(*self*, *code_name*, *field_names*, *field_code_types*)

Define un nuevo tipo correspondiente a una estructura del lenguaje C. Se definirá un nuevo tipo con el parametro `code_name` por nombre, y se tratarán de definir los campos de la estructuras con el nombre correspondiente en el parámetro `field_names`, en caso de que este coincida con una palabra reservada del lenguaje C, se tratará de a signarle uno similar.

Esta función se utiliza para generar el código de los ámbitos de ejecución, de los arrays y records del lenguaje Tiger que se definan en el programa.

Argumentos

`code_name:` Nombre fue definido para el tipo de la nueva estructura.

*(type=**str**)*

`field_names:` Nombres que se proponen para cada uno de los campos de la estructura.

*(type=**list**)*

`field_code_types:` Identificadores de código devueltos anteriormente por métodos de esta clase correspondientes a los tipos de cada uno de los campos de la estructura.

*(type=**list**)*

Valor de retorno

Lista con los identificadores de código que se pueden utilizar para acceder a cada uno de los campos de la estructura.

*(type=**list**)*

define_type_name(*self*, *name*)

Se tratará de definir un nuevo nombre para el tipo definido con el nombre dado en el parámetro `name` si este no coincide con una palabra reservada del lenguaje C o un tipo definido anteriormente, en este caso se tratará de definir un nombre lo más semejante posible.

Argumentos

`name:` Nombre del tipo al que se le quiere definir un nuevo tipo.

*(type=**str**)*

Valor de retorno

Nombre que fue definido para este tipo.

*(type=**str**)*

define_scope(*self*, *member_names*, *member_types*, *type_names*, *types*, *parent_code_type*=`None`)

Utilizado para generar las estructuras de código C que representan los ámbitos de ejecución de un programa Tiger. Este método llama a `define_struct` con nombres para las estructuras garantizando que son únicos utilizando un contador unido al nombre en lugar de los guiones bajos añadidos por el método `_disambiguate`.

Argumentos

`member_names`: Lista con los nombres de las variables y funciones que se definen en el ámbito de ejecución.

*(type=**list**)*

`member_types`: Lista con las instancias de herederos de `TigerType` correspondiente a cada uno de los nombres en la lista `member_names`. Este método da valor a las propiedades `code_name` o `code_type` de estas instancias, según corresponda.

*(type=**list**)*

`type_names`: Lista con los nombres de los tipos definidos en este ámbito

*(type=**list**)*

`types`: Lista con las instancias de `TigerTypes` correspondientes a los tipos definidos en este ámbito.

*(type=**list**)*

`parent_code_type`: Identificador en el código generado correspondiente a la estructura representando el ámbito de ejecución padre del ámbito que se quiere definir. Si el ámbito es raíz, especificar `None`.

*(type=**str**)*

Valor de retorno

Este método retorna una tupla con dos elementos. El primer es el nombre de una variable local del tipo de la nueva estructura definida. El segundo elemento es el identificador de código que se debe utilizar para referirse al tipo de la nueva estructura definida.

*(type=**tuple**)*

define_array(*self*, *code_name*, *field_code_type*)

Define un nuevo tipo array del lenguaje Tiger que corresponde a una estructura del lenguaje C. Con campos data y length, para más información consultar las anotaciones sobre el código generado.

Argumentos

`code_name:` Nombre fue definido para el tipo de la nueva estructura.

*(type=**str**)*

`field_type:` Nombre de la estructura del lenguaje C que representa el tipo correspondiente a los valores del `array`.

*(type=**str**)*

define_function(*self*, *func_name*, *func_type*, *scope_code_type*, *scope_code_name*)

Genera la cabecera de una función. Este método es llamado por `define_scope` para definir cada una de las funciones declaradas en un ámbito de ejecución. El cuerpo de una función definida utilizando este método se generará posteriormente cambiando la función actual del generador a esta utilizando el método `begin_function`.

Argumentos

`func_name:` Nombre dado a la función en el programa Tiger. El nombre de al función C resultante será este nombre con un prefijo que garantiza que la función resultante no coincida con alguna de la biblioteca standard. Además si este nombre coincide con alguna función definida anteriormente se añadirán caracteres underscore hasta que el nombre sea único.

*(type=**str**)*

`func_type:` Tipo correspondiente a la función que se define.

*(type=**FunctionType**)*

`scope_code_type:` Identificador de código del tipo de la estructura que representa el ámbito de ejecución donde está siendo definida la función.

*(type=**str**)*

`scope_code_name:` Identificador de código del nombre estructura que representa el ámbito de ejecución donde está siendo definida la función.

*(type=**str**)*

get_function_parameter(*self*, *n*)

Retorna un nombre único para el n-ésimo parámetro de una función. El nombre retornado será un prefijo seguido de un número y se utilizará en los nombres de los parámetros de todas las funciones definidas con este generador de código.

Argumentos

`n`: Indíce (comenzando en cero) del parámetro de la función.

*(type=**int**)*

Valor de retorno

Nombre para el n-ésimo parámetro de la función.

*(type=**str**)*

begin_function(*self*, *func_code_name*)

Cambia la función actual utilizada por el generador. Anteriormente se debe haber definido la cabecera de esta función utilizando el método `define_function`.

Argumentos

`func_code_name`: Identificador de código de la función.

*(type=**str**)*

Excepciones

`CodeGenerationError` Esta excepción se lanzará si se intenta cambiar la función actual del generador por una cuya cabecera no ha sido definida anteriormente.

end_function(*self*, *return_var*=`None`)

Termina la definición de la función actual. Luego de que este método se ejecute no será posible añadir código a la definición de dicha función.

Argumentos

`return_var`: Nombre de la variable local cuyo valor será el valor de retorno de la función. Si la función no tiene valor de retorno se debe especificar `None`.

*(type=**str**)*

define_local(*self*, *code_type*)

Añade la definición de una nueva variable local a la función actual.

Argumentos

`code_type`: Identificador de código C para el tipo de la variable.

*(type=**str**)*

Valor de retorno

Nombre asignado a la variable local. Se garantiza que este nombre no coincida con el de otra variable local de la función.

*(type=**str**)*

add_statement(*self*, *statement*)

Añade una instrucción al cuerpo de la función actual.

Argumentos

`statement`: Instrucción que se debe añadir al cuerpo de la función actual del generador de código.

*(type=**str**)*

close(*self*)

Termina la clase auxiliar utilizada en la generación de código. Luego de que se ejecute este método no es válido continuar generando código mediante esta clase, solamente es válido utilizar el método `write` para escribir el código generado hacia un descriptor de fichero.

Excepciones

`CodeGenerationError` Esta excepción se lanzará si se intenta cerrar el generador y existen funciones abiertas distintas de la función principal.

write(*self*, *output_fd*)

Escribe el código C generado hacia un descriptor de fichero.

Argumentos

`output_fd`: Descriptor de fichero donde se debe escribir el código C resultante de la traducción del programa Tiger.

*(type=**file**)*

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

59.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

60. Paquete pytiger2c.contrib

Módulos desarrollados por terceros.

La documentación de los módulos contenidos en este paquete no es nuestra responsabilidad por lo que no se incluye en la documentación del API generada automáticamente.

61. Módulo pytiger2c.dot

Clases utilizadas en la generación de un archivo Graphviz DOT con el árbol de sintáxis abstracta creado a partir de un programa Tiger.

61.1. Clase DotGenerator

Clase utilizada para la generación de grafos en formato Graphviz DOT.

61.1.1. Métodos

__init__(*self*)

Esta clase es utilizada en la generación de código Graphivz DOT a partir de un árbol de sintáxis abstracta de un programa Tiger.

Overrides: object.__init__

add_node(*self*, *label*)

Añade un nuevo nodo al grafo actualmente en creación.

Argumentos

`label`: Nombre del nodo que se quiere añadir.

*(type=**`str`**)*

Valor de retorno

Identificador del nuevo nodo añadido. Este identificador puede ser utilizado para crear nuevas aristas, utilizando el método `add_edge` de esta misma clase, que tengan este nodo como uno de los extremos.

*(type=**`str`**)*

add_edge(*self*, *from_node*, *to_node*)

Añade una arista no dirigida al grafo actualmente en creación.

Argumentos

`from_node`: Cadena de caracteres que identifica un nodo extremo de la arista.

*(type=**str**)*

`to_node`: Cadena de caracteres que identifica un nodo extremo de la arista.

*(type=**str**)*

write(*self*, *output_fd*)

Escribe el código Graphviz DOT en un descriptor de fichero.

Argumentos

`output_fd`: Descriptor de fichero donde se debe escribir el código Graphviz DOT resultante de la traducción del programa Tiger descrito por el árbol de sintáxis abstracta.

*(type=**file**)*

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

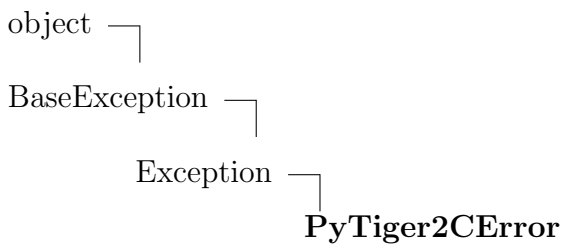
61.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

62. Módulo pytiger2c.errors

Jerarquía de las excepciones lanzadas durante la ejecución de PyTiger2C.

62.1. Clase PyTiger2CError



Clase base de todas las excepciones lanzadas en el paquete.

62.1.1. Métodos

__init__(*self*, *error*='Error', *message*='An error occurred during the execution of PyTiger2C')

Representa un error ocurrido durante la ejecución de PyTiger2C.

Argumentos

`error:` Tipo de error.

*(type=**str**)*

`message:` Descripción del error.

*(type=**str**)*

Overrides: object.__init__

__str__(*self*)

Retorna una cadena con el tipo de error ocurrido y una descripción del error.

Overrides: object.__str__

Heredados de Exception

__new__()

Heredados de BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

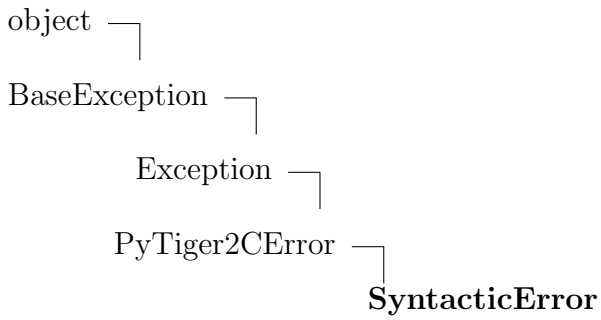
Heredados de object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

62.1.2. Propiedades

Nombre	Descripción
Heredadas de BaseException args, message	
Heredadas de object __class__	

62.2. Clase SyntacticError



Excepción lanzada durante el análisis léxico-gráfico y sintáctico.

62.2.1. Métodos

__init__(*self*, *message*='An error occurred during the syntactic analysis')

Representa un error de sintáxis en un programa Tiger.

Argumentos

`message`: Descripción del error.

(type=`str`)

Overrides: object.__init__

Heredados de PyTiger2CError (Sección 62.1)

__str__()

Heredados de Exception

__new__()

Heredados de BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

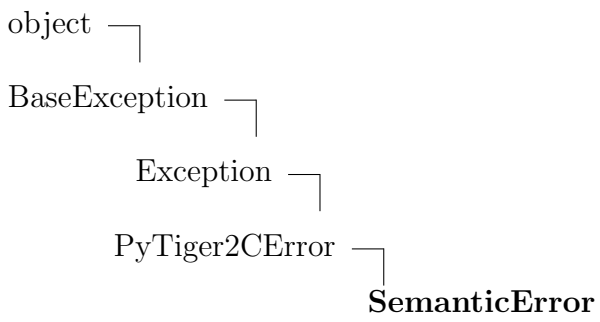
Heredados de object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

62.2.2. Propiedades

Nombre	Descripción
Heredadas de BaseException args, message	
Heredadas de object __class__	

62.3. Clase SemanticError



Excepción lanzada durante el análisis semántico.

62.3.1. Métodos

__init__(*self*, *messages*=`None`)

Representa un error semántico en un programa Tiger.

Argumentos

`message:` Lista de los mensajes de error ocurridos durante el análisis semántico..

`messages:` *(type=`list`)*

Overrides: object.__init__

__str__(*self*)

Retorna una cadena con el tipo de error ocurrido y una descripción del error.

Overrides: object.__str__

Heredados de Exception

__new__()

Heredados de BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

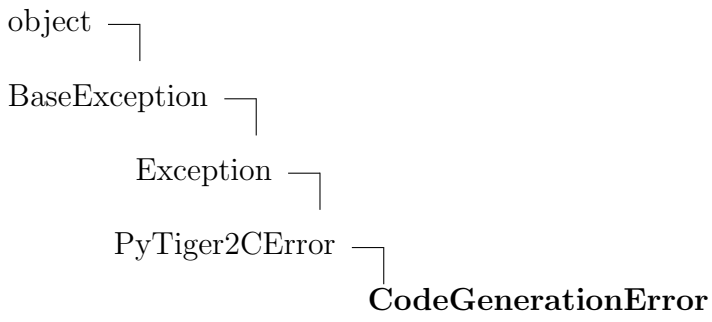
Heredados de object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

62.3.2. Propiedades

Nombre	Descripción
Heredadas de BaseException args, message	
Heredadas de object __class__	

62.4. Clase CodeGenerationError



Excepción lanzadas durante la generación de código.

62.4.1. Métodos

__init__(*self*, *message*='An error occurred during the code generation')

Representa un error ocurrido durante la generación de código.

Argumentos

`message`: Descripción del error.

*(type=**`str`**)*

Overrides: object.__init__

Heredados de PyTiger2CError (Sección 62.1)

__str__()

Heredados de Exception

__new__()

Heredados de BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Heredados de object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

62.4.2. Propiedades

Nombre	Descripción
Heredadas de BaseException args, message	
Heredadas de object __class__	

63. Paquete pytiger2c.grammar

Módulos relacionados con la definición de la gramática de Tiger utilizando PLY[2].

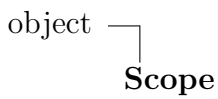
PLY utiliza los *docstrings* para especificar las reglas de la gramática que especifican las acciones a realizar cuando se reduce una producción. Por esta razón, resulta difícil incluir los módulos contenidos en este paquete en la documentación del API generada automáticamente.

[2] http://www.dabeaz.com/ply/

64. Módulo pytiger2c.scope

Clases `Scope` y `RootScope` que representan ámbitos de ejecución en Tiger.

64.1. Clase Scope



Clase `Scope` que representa un ámbito de ejecución de Tiger.

Esta clase gestiona los tipos, variables y funciones disponibles en un ámbito de ejecución en Tiger. Además mantiene una referencia a un ámbito padre donde se encuentra contenido este ámbito.

64.1.1. Métodos

parent(*self*)

Método para obtener el valor de la propiedad `parent`.

depth(*self*)

Método para obtener el valor de la propiedad `depth`.

code_name(*self*)

Método para obtener el valor de la propiedad `code_name`.

code_type(*self*)

Método para obtener el valor de la propiedad `code_type`.

__**init**__(*self*, *parent*)

Inicializa la clase `Scope`.

Argumentos

`parent`: Ámbito en el que se define este nuevo ámbito.

(type=`Scope`)

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera una estructura del lenguaje C que contiene las definiciones de las variables incluídas en este ámbito de ejecución.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

get_variable_code(*self*, *name*)

Genera el código necesario para acceder a una variable definida en este ámbito o en alguno superior.

Una variable definida en un ámbito superior puede ser accedida desde cualquier ámbito inferior por lo que la variable en cuestión puede estar definida en el ámbito actual, en su padre o en algún ancestro.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable.

(type=`str`)

Valor de retorno

Cadena de caracteres correspondiente al código C necesario para acceder a la variable.

(type=`str`)

define_type(*self*, *name*, *tiger_type*)

Añade una definición de tipos al ámbito actual.

Un tipo puede definirse con el mismo nombre que alguno contenido en algún ámbito superior y este nuevo tipo oculta al existente en el ámbito superior. En caso de que el nuevo tipo tenga el mismo nombre que alguno definido en el mismo ámbito, ocurre un error.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre del tipo que se declara.

*(type=**str**)*

`type`: Instancia de `TigerType` correspondiente a la declaración de tipo.

*(type=**TigerType**)*

Excepciones

`ValueError` Se lanza una excepción `ValueError` si el tipo que se intenta declarar fue definido anteriormente en este ámbito.

get_type_definition(*self*, *name*)

Retorna la definición de tipos correspondiente al nombre dado. Si el tipo no se encuentra en el ámbito actual se buscará en los ancestros hasta llegar al ámbito raíz que lanzará una excepción si una definición de tipo para el nombre dado no es encontrado finalmente.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre del tipo que se quiere obtener.

*(type=**str**)*

Valor de retorno

Instancia de `TigerType` correspondiente a la definición de tipo buscada.

*(type=**TigerType**)*

Excepciones

`KeyError` Se lanza una excepción `KeyError` si el tipo no está definido en este ámbito o en alguno superior.

define_function(*self*, *name*, *function_type*)

Añade una definición de funciones al ámbito actual.

Una función puede definirse con el mismo nombre que alguna contenido en algún ámbito superior y esta nueva función oculta a la existente en el ámbito superior. En caso de que la nueva función tenga el mismo nombre que alguna definida en el mismo ámbito, ocurre un error.

Argumentos

`name:` Cadena de caracteres correspondiente al nombre de la función que se declara.

*(type=**`str`**)*

`function_type:` Instancia de `FunctionType` correspondiente a la definición de función.

*(type=**`FunctionType`**)*

Excepciones

`ValueError` Se lanza una excepción `ValueError` si la función que se intenta definir se definió anteriormente en este ámbito.

get_function_definition(*self*, *name*)

Retorna la definición de la función correspondiente al nombre dado.

Argumentos

`name:` Cadena de caracteres correspondiente al nombre de la función.

*(type=**`str`**)*

Valor de retorno

Instancia de `FunctionType` correspondiente a la definición de la función.

*(type=**`FunctionType`**)*

Excepciones

`KeyError` Se lanza un `KeyError` si la función no está definida en este scope o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una función.

define_variable(*self*, *name*, *tiger_type*)

Añade una definición de variable al ámbito actual.

Una variable puede definirse con el mismo nombre que alguna contenida en algún ámbito superior y esta nueva variable oculta a la existente en el ámbito superior. En caso de que la nueva variable tenga el mismo nombre que alguna definida en el mismo ámbito, ocurre un error.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable que se declara.

*(type=**`str`**)*

`tiger_type`: Instancia de `VariableType` correspondiente a la definición de la variable que se quiere definir.

*(type=**`VariableType`**)*

Excepciones

`ValueError` Se lanza una excepción `ValueError` si la variable que se intenta definir se definió anteriormente en este ámbito.

get_variable_definition(*self*, *name*)

Retorna la definición de una variable de este ámbito o algún ámbito ancestro.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable.

*(type=**`str`**)*

Valor de retorno

Instancia de `VariableType` correspondiente a la declaración de la variable.

*(type=**`VariableType`**)*

Excepciones

`KeyError` Se lanza una excepción `KeyError` si la variable no está definida en este ámbito o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una variable.

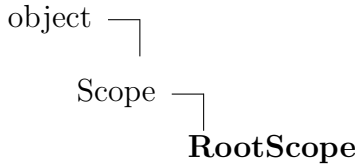
Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

64.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

64.2. Clase RootScope



Clase `RootScope` que representa el ámbito raíz de un programa Tiger.

Esta clase gestiona los tipos, variables y funciones disponibles en un ámbito de ejecución detereminado en Tiger.

En esta clase se encuentran las definiciones de las funciones de la librería estándar y tipos básicos de Tiger. Además, esta clase es la encargada de detener la búsqueda de tipos, funciones y variables por los ámbitos padres lanzando una excepción `KeyError` si no se encuentra una definición en este ámbito.

64.2.1. Métodos

__**init**__(*self*)

Inicializa el ámbito raíz de un programa de Tiger.

Inicializa las declaraciones de las funciones de la biblioteca standard y los tipos básicos.

Argumentos

`parent`: Ámbito en el que se define este nuevo ámbito.

Overrides: object.__init__

get_type_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `get_type_definition` en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre del tipo que se quiere obtener.

Valor de retorno

Instancia de `TigerType` correspondiente a la definición de tipo buscada.

(type=`TigerType`)

Excepciones

`KeyError` Se lanza una excepción `KeyError` si el tipo no está definido en este ámbito o en alguno superior.

Overrides: pytiger2c.scope.Scope.get_type_definition

get_function_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `get_function_definition` en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la función.

Valor de retorno

Instancia de `FunctionType` correspondiente a la definición de la función.

(type=`FunctionType`)

Excepciones

`KeyError` Se lanza un `KeyError` si la función no está definida en este scope o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una función.

Overrides: pytiger2c.scope.Scope.get_function_definition

get_variable_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `get_variable_definition` en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable.

Valor de retorno

Instancia de `VariableType` correspondiente a la declaración de la variable.

(type=`VariableType`)

Excepciones

`KeyError` Se lanza una excepción `KeyError` si la variable no está definida en este ámbito o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una variable.

Overrides: pytiger2c.scope.Scope.get_variable_definition

Heredados de Scope (Sección 64.1)

code_name(), code_type(), define_function(), define_type(), define_variable(), depth(), generate_code(), get_variable_code(), parent()

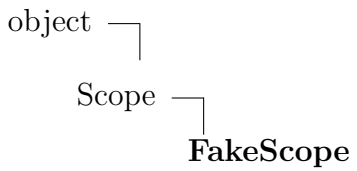
Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

64.2.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

64.3. Clase FakeScope



Clase `FakeScope` que representa un ámbito falso de un programa de Tiger.

Este ámbito recibe el calificativo de falso porque no guardará definiciones de tipos, ni variables, ni funciones; cualquier consulta con el objetivo de definir u obtener un miembro del ámbito la redigirá al ámbito padre. El objetivo de este ámbito es detectar la declaración de funciones o tipos mutuamente recursivos en grupos de declaraciones diferentes. Si la situación anterior se detecta se lanzará una excepción indicando que el tipo no está definido en lugar de continuar la búsqueda a través del ámbito padre.

64.3.1. Métodos

relationships(*self*)

Método para obtener el valor de la propiedad `relationships`.

__init__(*self*, *parent*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método `__init__` en la clase `Scope`.

Argumentos

`parent`: Ámbito en el que se define este nuevo ámbito.

Overrides: object.__init__

generate_code(*self*, *generator*)

Genera una estructura del lenguaje C que contiene las definiciones de las variables incluídas en este ámbito de ejecución.

Argumentos

`generator`: Clase auxiliar utilizada en la generación del código C correspondiente a un programa Tiger.

(type=`CodeGenerator`)

Overrides: pytiger2c.scope.Scope.generate_code

define_type(*self*, *name*, *tiger_type*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre del tipo que se declara.

`type`: Instancia de `TigerType` correspondiente a la declaración de tipo.

Excepciones

`ValueError` Se lanza una excepción `ValueError` si el tipo que se intenta declarar fue definido anteriormente en este ámbito.

Overrides: pytiger2c.scope.Scope.define_type

define_function(*self*, *name*, *function_type*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la función que se declara.

`function_type`: Instancia de `FunctionType` correspondiente a la definición de función.

Excepciones

`ValueError` Se lanza una excepción `ValueError` si la función que se intenta definir se definió anteriormente en este ámbito.

Overrides: pytiger2c.scope.Scope.define_function

define_variable(*self*, *name*, *tiger_type*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable que se declara.

`tiger_type`: Instancia de `VariableType` correspondiente a la definición de la variable que se quiere definir.

Excepciones

`ValueError` Se lanza una excepción `ValueError` si la variable que se intenta definir se definió anteriormente en este ámbito.

Overrides: pytiger2c.scope.Scope.define_variable

get_variable_code(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable.

Valor de retorno

Cadena de caracteres correspondiente al código C necesario para acceder a la variable.

(type=`str`)

Overrides: pytiger2c.scope.Scope.get_variable_code

get_variable_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la variable.

Valor de retorno

Instancia de `VariableType` correspondiente a la declaración de la variable.

(type=`VariableType`)

Excepciones

`KeyError` Se lanza una excepción `KeyError` si la variable no está definida en este ámbito o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una variable.

Overrides: pytiger2c.scope.Scope.get_variable_definition

get_type_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

En este método se implementa la comprobación de la definición de tipos mutuamente recursivos, consulte la documentación del método `check_mutual_recursion` para más información.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre del tipo que se quiere obtener.

Valor de retorno

Instancia de `TigerType` correspondiente a la definición de tipo buscada.

(type=`TigerType`)

Excepciones

`KeyError` Se lanza una excepción `KeyError` si el tipo no está definido en este ámbito o en alguno superior.

Overrides: pytiger2c.scope.Scope.get_type_definition

get_function_definition(*self*, *name*)

Para obtener información acerca de los parámetros recibidos por este método consulte la documentación del método con el mismo nombre en la clase `Scope`.

En este método se implementa la comprobación de la definición de funciones o procedimientos mutuamente recursivos, consulte la documentación del método `check_mutual_recursion` para más información.

Argumentos

`name`: Cadena de caracteres correspondiente al nombre de la función.

Valor de retorno

Instancia de `FunctionType` correspondiente a la definición de la función.

(type=`FunctionType`)

Excepciones

`KeyError` Se lanza un `KeyError` si la función no está definida en este scope o en alguno superior.

`ValueError` Se lanza una expceción `ValueError` si existe un miembro en algún ámbito con el nombre dado pero no es una función.

Overrides: pytiger2c.scope.Scope.get_function_definition

check_mutual_recursion(*self*, *name*)

Este método es utilizado por los métodos `get_function_definition` y `get_type_definition` para comprobar que el tipo, función o procedimiento identificado por el nombre dado `name` no tenga una definición mutualmente recursiva en función de un tipo, función o procedimiento de otro grupo de definiciones.

Si el miembro `name` se encuentra definido en un grupo de definiciones hermano del grupo actual se comprobará que no exista una definición mutuamente recursiva. En caso de que todavía no se tenga suficiente información para afirmar o negar que exista una definición mutuamente recursiva se actualizará el diccionario de relaciones para que la definición mutuamente recursiva, si existe, sea detectada cuando se compruebe la definición del otro tipo, función o procedimiento.

Argumentos

`name`: Nombre del tipo, función o procedimiento para el cual se debe realizar la comprobación.

(type=`str`)

Excepciones

`KeyError` Se lanza una excepción `KeyError` si el miembro del scope identificado por el nombre `name` tiene una definición mutuamente recursiva con otro miembro de un grupo de definiciones diferente.

Heredados de Scope (Sección 64.1)

code_name(), code_type(), depth(), parent()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

64.3.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

64.3.3. Variables de clase

continúa en la página siguiente

Nombre	Descripción

Nombre	Descripción
current_member	**Value:** `property(_get_current_member, _set_current_member)`
current_siblings	**Value:** `property(_get_current_siblings, _set_current_siblings)`
max_recursion_depth	**Value:** `property(_get_max_recursion_depth, _set_max_recursion_depth)`

65. Paquete pytiger2c.types

Módulos de la jerarquía de tipos de Tiger.

65.1. Módulos

- **aliastype**: Clase de la jerarquía de tipos de Tiger representando el tipo alias.

  (Sección 66, página 236)

- **arraytype**: Clase de la jerarquía de tipos de Tiger representando el tipo array.

  (Sección 67, página 238)

- **basictype**: Clase de la jerarquía de tipos de Tiger representando los tipos básicos definidos en el lenguaje Tiger.

  (Sección 68, página 240)

- **functiontype**: Clase de la jerarquía de tipos de Tiger representando el tipo función.

  (Sección 69, página 242)

- **integertype**: Clase de la jerarquía de tipos de Tiger representando el tipo entero.

  (Sección 70, página 245)

- **niltype**: Clase de la jerarquía de tipos de Tiger representando el tipo `nil`.

  (Sección 71, página 247)

- **recordtype**: Clase de la jerarquía de tipos de Tiger representando el tipo record.

  (Sección 72, página 249)

- **stringtype**: Clase de la jerarquía de tipos de Tiger representando el tipo cadena de caracteres.

  (Sección 73, página 251)

- **tigertype**: Clase base de la jerarquía de tipos de Tiger.

  (Sección 74, página 253)

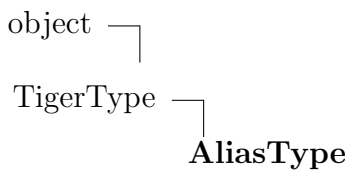
- **variabletype**: Clase de la jerarquía de tipos de Tiger representando la definición de una variable.

  (Sección 75, página 254)

66. Módulo pytiger2c.types.aliastype

Clase de la jerarquía de tipos de Tiger representando el tipo alias.

66.1. Clase AliasType



Clase de la jerarquía de tipos de Tiger representando el tipo alias, un alias es representado por una instancia de esta clase durante la resolución del tipo concreto al que hace referencia en chequeo semántico del nodo donde fue definido.

66.1.1. Métodos

alias_typename(*self*)

Método para obtener el valor de la propiedad `alias_typename`.

__init__(*self*, *alias_typename*)

Inicializa la clase representando el tipo alias.

Argumentos

`alias_typename`: Nombre del tipo al que hace referencia este alias.
(type=`str`)

Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

66.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

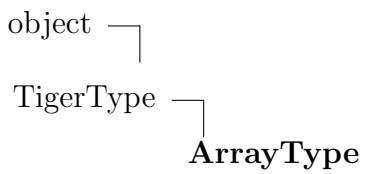
66.1.3. Variables de clase

Nombre	Descripción
Heredadas de TigerType (Sección 74.1)	
code_type	

67. Módulo pytiger2c.types.arraytype

Clase de la jerarquía de tipos de Tiger representando el tipo array.

67.1. Clase ArrayType



Clase de la jerarquía de tipos de Tiger representando el tipo array.

67.1.1. Métodos

fields_typenames(*self*)

Método para obtener el valor de la propiedad `fields_typenames`.

__**init**__(*self*, *values_typename*)

Inicializa la clase representando el tipo array.

Argumentos

`values_typename`: Nombre del tipo que tendrán los valores del array.

(type=`str`)

Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

67.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

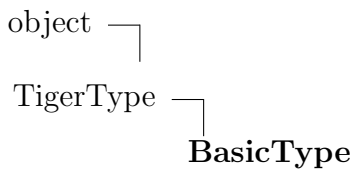
67.1.3. Variables de clase

Nombre	Descripción
code_name	**Value:** `property(_get_code_name, _set_code_name)`
fields_types	**Value:** `property(_get_fields_types, _set_fields_types)`
Heredadas de TigerType (Sección 74.1) code_type	

68. Módulo pytiger2c.types.basictype

Clase de la jerarquía de tipos de Tiger representando los tipos básicos definidos en el lenguaje Tiger.

68.1. Clase BasicType



Clase de la jerarquía de tipos de Tiger representando los tipos básicos definidos en el lenguaje Tiger.

Esta clase representa los tipos definidos en la librería estándar de Tiger. Estos tipos son `nil`, `int` y `string`.

68.1.1. Métodos

__init__(*self*)

Inicializa la clase representando los tipos básicos del Lenguaje Tiger.

Overrides: object.__init__

__eq__(*self*, *other*)

Implementación por defecto de la comparación entre los tipos básicos del lenguaje Tiger. Dos tipos básicos serán iguales si ambos son instancia de la misma clase.

Argumentos

`other`: Otro tipo de Tiger con el que efectuar la comparación.

(type=`TigerType`)

Valor de retorno

Retorna `True` si los ambos son iguales.

(type=`bool`)

__**ne**__(*self*, *other*)

Esta comparación se define como la negación del resultado obtenido por __eq__. Ver documentación del método __eq__ para más información.

Argumentos

`other`: Otro tipo de Tiger con el que efectuar la comparación.

(type=`TigerType`)

Valor de retorno

Retorna `True` si los ambos no son iguales.

(type=`bool`)

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

68.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

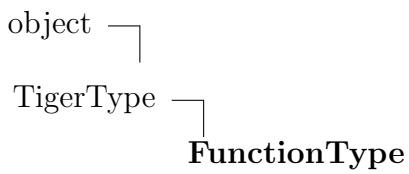
68.1.3. Variables de clase

Nombre	Descripción
Heredadas de TigerType (Sección 74.1)	
code_type	

69. Módulo pytiger2c.types.functiontype

Clase de la jerarquía de tipos de Tiger representando el tipo función.

69.1. Clase FunctionType



Clase de la jerarquía de tipos de Tiger representando el tipo función.

69.1.1. Métodos

parameters_typenames(*self*)

Método para obtener el valor de la propiedad `parameters_types`.

__init__(*self*, *return_type*, *parameters_types*, *parameters_typename*)

Inicializa la clase representando el tipo función.

Argumentos

`return_type`: Instance de `TigerType` correspondiente al tipo del valor de retorno de la función. Si la función no tiene valor de retorno el valor de este argumento debe ser especificado como `None`.

(type=`TigerType`)

`parameters_types`: Lista de las instancias de `TigerType` correspondientes a los tipos de los parámetros recibidos por la función. Las posiciones de los elementos de la lista deben corresponder con las posiciones de los parámetros de la función. Si la función no recibe parámetros el valor de este argumento debe ser especificado como una lista vacía.

(type=`list`)

`parameters_typenames`: Lista de los nombres de los tipos de los parámetros de la función. Las posiciones de los elementos de la lista deben corresponder con las posiciones de los parámetros de la función. Si la función no recibe parámetros el valor de este argumento debe ser especificado como una lista vacía.

(type=`list`)

Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

69.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

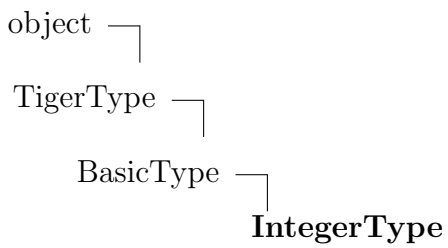
69.1.3. Variables de clase

Nombre	Descripción
return_type	**Value:** `property(_get_return_type, _set_return_type)`
parameters_types	**Value:** `property(_get_parameters_types, _set_parameters_types)`
code_name	**Value:** `property(_get_code_name, _set_code_name)`
scope_depth	**Value:** `property(_get_scope_depth, _set_scope_depth)`
Heredadas de TigerType (Sección 74.1) code_type	

70. Módulo pytiger2c.types.integertype

Clase de la jerarquía de tipos de Tiger representando el tipo entero.

70.1. Clase IntegerType



Clase de la jerarquía de tipos de Tiger representando el tipo entero

70.1.1. Métodos

__**init**__(*self*)
Inicializa la clase representando el tipo array.
Overrides: object.__init__

Heredados de BasicType (Sección 68.1)

__eq__(), __ne__()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

70.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

70.1.3. Variables de clase

Nombre	Descripción
Heredadas de TigerType (Sección 74.1)	

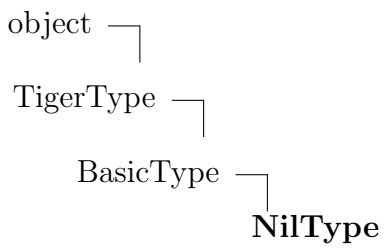
continúa en la página siguiente

Nombre	Descripción
code_type	

71. Módulo pytiger2c.types.niltype

Clase de la jerarquía de tipos de Tiger representando el tipo `nil`.

71.1. Clase NilType



Clase de la jerarquía de tipos de Tiger representando el tipo `nil`.

71.1.1. Métodos

__**init**__(*self*)
Inicializa la clase representando el tipo `nil`.
Overrides: object.__init__

Heredados de BasicType (Sección 68.1)

__eq__(), __ne__()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

71.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

71.1.3. Variables de clase

Nombre	Descripción
Heredadas de TigerType (Sección 74.1)	

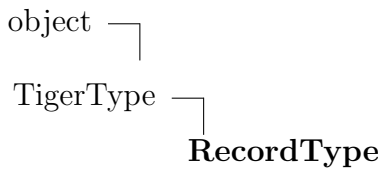
continúa en la página siguiente

Nombre	Descripción
code_type	

72. Módulo pytiger2c.types.recordtype

Clase de la jerarquía de tipos de Tiger representando el tipo record.

72.1. Clase RecordType



Clase de la jerarquía de tipos de Tiger representando el tipo record.

72.1.1. Métodos

fields_typenames(*self*)

Método para obtener el valor de la propiedad `fields_typenames`.

fields_names(*self*)

Método para obtener el valor de la propiedad `fields_names`.

__init__(*self*, *fields_names*, *fields_typenames*)

Inicializa la clase representando el tipo record.

Argumentos

`fields_names`: Lista con los nombres de los campos del record, por posición.

(type=`list`)

`fields_typenames`: Lista con los nombres de los tipos de los campos, por posición.

(type=`list`)

Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

72.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

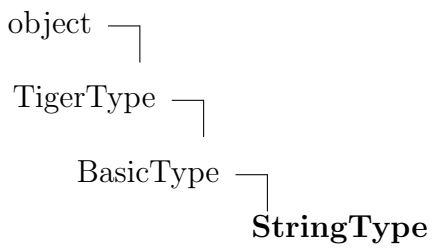
72.1.3. Variables de clase

Nombre	Descripción
code_name	**Value:** `property(_get_code_name, _set_code_name)`
fields_types	**Value:** `property(_get_fields_types, _set_fields_types)`
field_code_names	**Value:** `property(_get_field_code_names, _set_field_code_names)`
Heredadas de TigerType (Sección 74.1) code_type	

73. Módulo pytiger2c.types.stringtype

Clase de la jerarquía de tipos de Tiger representando el tipo cadena de caracteres.

73.1. Clase StringType



Clase de la jerarquía de tipos de Tiger representando el tipo cadena de caracteres.

73.1.1. Métodos

__**init**__(*self*)
Inicializa la clase representando el tipo cadena de caracteres.
Overrides: object.__init__

Heredados de BasicType (Sección 68.1)

__eq__(), __ne__()

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

73.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

73.1.3. Variables de clase

Nombre	Descripción
Heredadas de TigerType (Sección 74.1)	

continúa en la página siguiente

Nombre	Descripción
code_type	

74. Módulo pytiger2c.types.tigertype

Clase base de la jerarquía de tipos de Tiger.

74.1. Clase TigerType

object
└ **TigerType**

Clase base de la jerarquía de tipos de Tiger.

Todas las clases representando tipos válidos del lenguaje Tiger deben heredar de la clase base `TigerType`.

74.1.1. Métodos

__**init**__(*self*)
Inicializa la clase base de la jerarquía.
Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

74.1.2. Propiedades

Nombre	Descripción
Heredadas de object	
__class__	

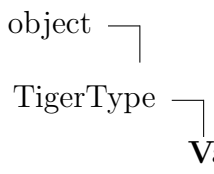
74.1.3. Variables de clase

Nombre	Descripción
code_type	**Value:** `property(_get_code_type, _set_code_type)`

75. Módulo pytiger2c.types.variabletype

Clase de la jerarquía de tipos de Tiger representando la definición de una variable.

75.1. Clase VariableType



Clase de la jerarquía de tipos de Tiger representando la definición de una variable.

75.1.1. Métodos

read_only(*self*)

Método para obtener el valor de la propiedad `read_only`.

__**init**__(*self*, *tiger_type*, *read_only*=`False`)

Inicializa la clase representando la definición de una variable.

Argumentos

`tiger_type`: Instancia de `TigerType` correspondiente al tipo de la variable que se quiere definir.

(type=`TigerType`)

`read_only`: Indica si la variable que se define debe ser tratada como una variable de sólo lectura. El valor por defecto de este argumento es `False`.

(type=`bool`)

Overrides: object.__init__

Heredados de object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

75.1.2. Propiedades

Nombre	Descripción
Heredadas de object __class__	

75.1.3. Variables de clase

Nombre	Descripción
type	**Value:** `property(_get_type, _set_type)`
code_name	**Value:** `property(_get_code_name, _set_code_name)`
Heredadas de TigerType (Sección 74.1) code_type	