Ausarbeitung

Ausarbeitung

Compilerbau

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang Informatik
erstellte Ausarbeitung
für das Modul
Formale Sprachen und Compilerbau

von
Alexander Weidemann
Bijan Riesenberg
Johanna Kraken
Matr.-Nr. 7206374

Betreuer: Prof. Dr. Robert Rettinger

Dortmund, September 25, 2020

Contents

1 Introduction

2 Festlegung Token

$TOKEN :$

$\{$

$< INT : "int" >$

$| < DOUBLE : "double" >$

$| < FLOAT : "float" >$

$| < CHAR : "char" >$

$| < BOOLEAN : "boolean" >$

$| < STRING : "string" >$

$| < TYPE : "type" >$

$| < VOID : "void" >$

$| < NULL : "null" >$

$| < SMARTSWITCH : "@" >$

$| < RETURN : "return" >$

$| < ENUM_SEPERATOR : "," >$

$| < EXPRESSION_TERMINATOR : ";" >$

$| < IF : "if" >$

$| < ELSE : "else" >$

$| < WHILE : "while" >$

$| < FOREACH : "foreach" >$

$| < IN : "in" >$

$| < LETTER : ["A" - "Z", "_{"}, "a" - "z"] >$

$| < DIGIT : ["0" - "9"] >$

$| < DATE :< TWO_DIGIT > " - " < TWO_DIGIT > " - " < TWO_DIGIT >< TWO_DIGIT >>$

$| < TWO_DIGIT : (["0" - "9"])2 >$

$| < AMPM : "am" | "pm" >$

$| < TIME :< TWO_DIGIT > " : " < TWO_DIGIT >>$

$| < PUNCT : "punct" >$

$| < GRAPH : "graph" > //Letters, numbersandpunctuation$

$| < LOWER : "lower" > //LowercaseLetters$

$| < ALPHA : "alpha" > //Letters$

$| < ALNUM : "alnum" > //Alphanumerics$

$| < PRINT : "print" > //Letters, numbers, punctuationandwhitespace$

$| < CNTRL : "cntrl" > //Controlcharacters$

$| < SPACE : "space" > //Spacecharacters$

$| < BLANK : "blank" > //Spaceandtab$

$| < DIGITS : "digit" > //Digits$

$| < INTEGER_LITERAL :< DECIMAL_LITERAL >>$

$| < \#DECIMAL_LITERAL : (" + "|" - ")?["1" - "9"](["0" - "9"])* >$

$| < FLOATING_POINT_LITERAL : ("+"|"-")?(["0"-"9"])+"."(["0"-"9"])* >$

$| < CHARACTER_LITERAL : "\backslash'"(["\backslash'", "\backslash\backslash", "", "\Sigma]|"\backslash\backslash"(["n", "t", "b", "r", "f", "\backslash\backslash", "\backslash'", "$

$"7"](["0" - "7"])?|["0" - "3"]["0" - "7"]["0" - "7"]))"\backslash'" >$

$| < STRING_LITERAL : "\backslash""(["\backslash"", "\backslash\backslash", "", "\Sigma]|"\backslash\backslash"(["n", "t", "b", "r", "f", "\backslash\backslash", "\backslash'", "\backslash""]|[$

$"7"](["0"-"7"])?|["0"-"3"]["0"-"7"]["0"-"7"]|(["", "\Sigma]|\Sigma))) * "\backslash"" > | < TYPE_LITERAL : "$

3 Grammatik

G = (N, T, R, S)

N = { Start, Element, Block, Expression, AssingmentExpression, VariableDefinitionExpression, , VariableDefinitionExpression, CompareExpression, , AdditiveExpression, MultiplicativeExpression, PrefixExpression, UnaryExpression, ValueExpression, FunctionCallExpression, Identifier, FunctionReturnExpression, FunctionBodyDefinition, FunctionHeaderDefinition, FunctionDefinition, ForeachLoopDefinition, WhileLoopDefinition, IfDefinition, SmartSwitchSelektor, SmartSwitchConditionDefinition, SmartSwitchCaseDefinition, SmartSwitchDefinition, IsDatatype, ToDatatype, LengthDatatype, BasicDatatype, Datatype, ReturnDatatype } (Funktionennamen in NewAwk.jjt)

T = { int, double, float, char, boolean, string, type, void, null, @, return, ',', ';', if, else, while, foreach, in, LETTER, DIGIT, am, pm, ':', toBoolean, toCharacter, toDouble, toInteger, toString, isBoolean, isCharacter, isDouble, isInterger, isString, (,), +, -, *, /, =, >, <, |, &, !, %, punct, graph, lower, alpha, alnum, print, cntrl, space, blank, digit, EOF }

S = { Start }

R = { Start -> FunctionDefinition Element EOF Element -> Block | Expression ; Block -> WhileLoopDefinition | ForeachLoopDefinition | IfDefinition Expression -> AssingmentExpression AssingmentExpression -> VariableDefinitionExpression |

VariableDefinitionExpression = LogicalExpression VariableDefinitionExpression -> ??? LogicalExpression -> CompareExpression | CompareExpression && LogicalExpression | CompareExpression || LogicalExpression CompareExpression -> AdditiveExpression | AdditiveExpression == CompareExpression | AdditiveExpression != CompareExpression | AdditiveExpression <= CompareExpression | AdditiveExpression >= CompareExpression | AdditiveExpression < CompareExpression | AdditiveExpression > CompareExpression AdditiveExpression -> MultiplicativeExpression | MultiplicativeExpression + AdditiveExpression | MultiplicativeExpression - AdditiveExpression MultiplicativeExpression -> PrefixExpression | PrefixExpression * MultiplicativeExpression | PrefixExpression / MultiplicativeExpression PrefixExpression -> UnaryExpression | ! UnaryExpression | UnaryExpression SmartSwitchDefinition | ! UnaryExpression SmartSwitchDefinition UnaryExpression -> (LogicalExpression) | FunctionCallExpression | IsDatatype | ToDatatype | LengthDatatype | ValueExpression ValueExpression -> ??? | Identifier FunctionCallExpression -> Identifier (LogicalExpression) | Identifier (LogicalExpression , LogicalExpression) | ??? Identifier -> ??? FunctionReturnExpression -> return LogicalExpression ; FunctionBodyDefinition -> Element | Element FunctionReturnExpression | ??? FunctionHeaderDefinition -> ReturnDatatype Identifier (Datatype Identifier) | ReturnDatatype Identifier (Datatype Identifier, Datatype Identifier) | ??? FunctionDefinition -> FunctionHeaderDefinition FunctionBodyDefinition ForeachLoopDefinition -> foreach (Identifier in UnaryExpression) FunctionBodyDefinition WhileLoopDefinition -> while (LogicalExpression) FunctionBodyDefinition IfDefinition -> if (LogicalExpression) FunctionBodyDefinition | if (LogicalExpression) FunctionBodyDefinition else FunctionBodyDefinition | if (LogicalExpression) FunctionBodyDefinition else if (LogicalExpression) FunctionBodyDefinition | ??? SmartSwitchSelektor -> punct | graph | lower | alpha | alnum | print | cntrl | space | blank | digit SmartSwitchConditionDefinition -> ! : SmartSwitchSelektor : | : SmartSwitchSelektor : SmartSwitchCaseDefinition -> SmartSwitchConditionDefinition FunctionBodyDefinition SmartSwitchDefinition-> @ SmartSwitchCaseDefinition IsDatatype -> isBoolean (UnaryExpression) | isCharacter (UnaryExpression) | isDouble (UnaryExpression) | isInteger (UnaryExpression) | isString (UnaryExpression) ToDatatype -> toBoolean (UnaryExpression) | toCharacter (UnaryExpression) | toDouble (UnaryExpression) | toInteger (UnaryExpression) | toString (UnaryExpression) LengthDatatype -> length (

UnaryExpression) BasicDatatype -> int | double | float | char | boolean | string | type | ??? Datatype -> BasicDatatype [] | BasicDatatype [][] | ??? ReturnDatatype -> Datatype | void }

4 Semantische Regeln

5 Aufruf des Compilers