UNIVERSITY PROJECT

MJ COMPILER

Author:

Ana Zorić 2020/0009

University of Belgrade

Project description

This project represents code compiler for microjava programming language. It consists of follolwing phases:

1. Lexical analysis
2. Syntax analysis
3. Semantic analysis
4. Code generation

MJ programming language supports the following:

* Integer, bool and char standard data types
* Constants of those types
* Global, local and static variables
* One dimension arrays
* Classes with fields and methods
* Static initializers and static fields
* Class inheritance, polymorphysm and redefining methods
* Predeclared functions ord, chr and len, print and read
* Ord(char) returns ASCII code of char passed
* Chr(int) returns char with ASCII code passed
* Len(void[ ]) returns length of array passed
* Print prints standard types on standard output
* Read reads variables of standard types from standard input
* Namespaces

Lexic analysis – code

Code used for lexic analysis is located in mjlexer.lex file with Jflex tool. It generates Yyles.java class which does the analysis.

Syntax analysis – code

Code used for syntax analysis is located in mjparser.cup file. AST CUP generator is used to generate abstract syntax tree of classes from that file.Generated classes are located at src\rs\ac\bg\etf\pp1\ast subdirectory of project.

Semantic anlaysis – code

Visitor generated by AST CUP is used for semantic analysis. SemanticAnalyzer class directly inherits it and is used to traverse AST and semantically analyze code. Code is divided between more classes, whose methods SemanticAnalyzer calls during analysis :

* BaseAnalyzer – base class for all other analyzer classes
* ClassAnalyzer
* ConditionAnalyzer
* ConstAnalyzer
* DesignatorAnalyzer
* ExprAnalyzer
* FactorAnalyzer
* MethodAnalyzer
* NamespaceAnalyzer
* ProgramAnalyzer
* SemAnalyzerCommon – encapsulating helper methods and variables used in semantic analysis
* TermAnalyzer
* VarAndTypeAnalyzer

Code generation – code

Visitor generated by AST CUP is used for code generation. CodeGenerator class directly inherits it and is used to traverse AST and generate code. Code is divided between more classes, whose methods SemanticAnalyzer calls during analysis :

* BaseGenerator – base class for all other generator classes
* ClassGen
* CodeGenCommon – encapsulating helper methods and variables used in code generation
* ConditoinGen
* DesignatorGen
* DesignatorStmtGen
* FactorGen
* MethodGen
* StatementGen
* VisitorDes – used for special traversing of complex statements of array assignment