Курсовая работа

РАЗРАБОТКА КОМПИЛЯТОРА ПОДМНОЖЕСТВА ПРОЦЕДУРНОГО ЯЗЫКА

**Описание программы**

**643.02068048.00001-01 13 01**

**Листов 18**

2023

АННОТАЦИЯ

Целью данной курсовой работы является разработка компилятора подмножества процедурного языка. В работе представлено подробное описание разработанной программы, а также основные этапы ее создания.

СОДЕРЖАНИЕ

[1. Общие сведения 4](#_Toc167103723)

[1.1. Обозначение и наименование программы 4](#_Toc167103724)

[1.2. Программное обеспечение, необходимое для функционирования программы 4](#_Toc167103725)

[1.3. Языки программирования, на которых написана программа 4](#_Toc167103726)

[2. Функциональное назначение 5](#_Toc167103727)

[3. Описание логической структуры 6](#_Toc167103728)

[3.1. Структура программы с описанием функций составных частей и связи между ними 6](#_Toc167103729)

[3.2. Лексический анализ 6](#_Toc167103730)

[3.3. Синтаксический анализ 7](#_Toc167103731)

[3.4. Генерация байт-кода 7](#_Toc167103732)

[4. Вызов и загрузка 8](#_Toc167103733)

[5. Входные данные 9](#_Toc167103734)

[5.1. Описание входного языка 9](#_Toc167103735)

[5.2. Описание токенов 9](#_Toc167103736)

[5.3 . Описание грамматики языка 10](#_Toc167103737)

[6. Выходные данные 17](#_Toc167103738)

[Перечень принятых сокращений 18](#_Toc167103739)

1. Общие сведения
   1. Обозначение и наименование программы

Кодовое название проекта: "Компилятор подмножества процедурного языка".

Название программы: "WLang".

* 1. Программное обеспечение, необходимое для функционирования программы

Для корректного функционирования разработанного компилятора необходимо наличие следующего программного обеспечения:

1. Java 1.8
2. Среда разработки – InteliJIDE или подобная
3. Виртуальная машина Java – JVM – необходима для запуска файлов расширения .class
   1. Языки программирования, на которых написана программа

Разработанный компилятор подмножества процедурного языка разработан на языке Java 1.8, используя сторонние библиотеки ASM, Antlr-4.

1. Функциональное назначение

Функциональное назначение разработанного компилятора подмножества процедурного языка заключается в автоматическом преобразовании исходного кода в подробное описание класса c последующей трансляцией в байт-код.

Компилятор выполняет следующие основные функции:

1. Лексический анализ.
2. Синтаксический анализ.
3. Генерация байт кода.
4. Описание логической структуры
   1. Структура программы с описанием функций составных частей и связи между ними

Программа компилятора подмножества процедурного языка состоит из следующих файлов:

|  |  |
| --- | --- |
| compiler\src\main\java\com\woody\compiler\Compiler.java | Запуск программы |
| antlr\src\main\antlr4\com.woody.antlr\Wlang.g4 | Грамматика для antlr4, на основании которой был сгенерирован WlangBaseVisitor |
| compiler\src\main\java\com\woody\bytecodegeneration | Модуль, содержащий вспомогательные функции, используемые при генерации байт-кода |
| WlangExamples\SumCalculator.w | Файл с программой на процедурном языке |
| compiler\src\main\java\com\woody\parsing\WlangTreeWalkErrorListener.java | Обработчик ошибок |
| compiler\src\main\java\com\woody\domain | Блок описания класса |
| compiler\src\main\java\com\woody\bytecodegeneration\BytecodeGenerator.java | Генерация кода |
| SumCalculator.class | Выходной файл в байт-коде |
| compiler\src\main\java\com\woody\antlr\WlangParser.java | Сгенерированный Antlr4 - Parser |
| compiler\src\main\java\com\woody\antlr\WlangLexer.java | Сгенерированный Antlr4 - Lexer |
| compiler\src\main\java\com\woody\antlr\WlangVisitor.java | Сгенерированный Antlr4 - Visitor |

* 1. Лексический анализ

В файле compiler\src\main\java\com\woody\antlr\WlangLexer.java реализуется лексический анализ. Он включает следующие шаги:

* Определение токенов: задаются правила для распознавания различных токенов с помощью регулярных выражений – на основании файла грамматики Wlang.g4.
* Создание лексического анализатора: создается экземпляр лексического анализатора, который будет использоваться для обработки исходного кода – на основании файла грамматики Wlang.g4.
  1. Синтаксический анализ

В файле compiler\src\main\java\com\woody\antlr\WlangParser.java реализуется синтаксический анализ. В нем выполняются следующие шаги:

* Определение грамматики: задаются правила грамматики языка, определяющие структуру иерархии конструкций языка – на основании файла грамматики Wlang.g4.
* Создание описания класса: внутри функций парсера, проходясь по дереву токенов, создается описание класса (его название, функции, поля и методы).
  1. Генерация байт-кода

Генерация байт-кода происходит после успешного синтаксического анализа, подробное описание класса передается в модуль генерации байт-кода, где происходят подробные обращения к библиотеке ASM, конечным результатом генерации является файл расширения .class

1. Вызов и загрузка

В папке out\artifacts\the\_compiler\_jar находится собранный проект the\_compiler.jar.

Пример команды для запуска: java -jar the\_compiler.jar D:\SumCalculator.w

По завершении выполнения программы мы получаем скомпилированный класс, который можно запустить по средствам JVM.

Пример команды для запуска скомпилированного класса: java SumCalculator

1. Входные данные
   1. Описание входного языка

Входной язык программы, которую компилирует разработанный компилятор, имеет следующие особенности:

1. Операторные скобки.

2. Игнорируется индентация программы.

3. Входная программа представляет собой единый модуль, с поддержкой вызова функций.

Операторы:

1. Оператор присваивания.

2. Арифметика (\*, /, +, -,>, <, ==, <=,>=, !=).

3. Логические операторы (И, ИЛИ, НЕ).

4. Условный оператор (ЕСЛИ).

5. Операторы цикла (for).

6. Базовый вывод (строковый литерал, переменная).

7. Типы (логический, целочисленный 32 бита, с плавающей запятой 32 бита).

* 1. Описание токенов

|  |  |  |
| --- | --- | --- |
| Название токена | Регулярное выражение | Описание |
| VARIABLE | 'var' | Определение переменной |
| PRINT | 'print' | Системная функция вывода |
| EQUALS | '=' | Оператор присваивания |
| NUMBER | '-'?[0-9.]+ | Числовой тип (отрицательное, положительное, целое и с плывущей точкой) |
| BOOL | 'true' | 'false' | Логическое тип |
| STRING | '"'~('\r' | '\n' | '"')\*'"' | Строковый литерал |
| ID | [a-zA-Z0-9]+ | Идентификатор переменной |

* 1. . Описание грамматики языка

|  |  |  |
| --- | --- | --- |
| **Название нетерминала** | **Возможные продукции** | **Описание** |
| compilationUnit | classDeclaration | Описание класса |
| EOF | конец файла |
| classDeclaration | className | Название класса |
| { | Начало блока класса |
| classBody | Тело класса |
| } | Конец блока класса |
| className | qualifiedName | Имя класса в определенном формате |
| classBody | field\* | Переменные класса в неограниченном количестве |
| function\* | Функции класса в неограниченном количестве |
| field | type | Тип переменной |
| name | Название переменной |
| function | functionDeclaration | Описание функции |
| block | Тело функции |
| functionDeclaration | type | Тип результата функции |
| functionName | Название функции |
| parametersList | Список параметров |
| parametersList | parameter\* | Параметр |
| parameterWithDefaultValue | Параметр с начальным значением |
| ',' | Разделитель параметров |
| functionName | ID | Токен идентификатора переменной |
| parameter | type | Тип параметра |
| ID | Токен идентификатора переменной |
| parameterWithDefaultValue | type | Тип параметра |
| ID | Токен идентификатора переменной |
| '=' | Оператор присваивания для параметра |
| deafaultValue=expression | Начальное значение представляющее собой любое выражение |
| type | pirmitiveType | Примитивный тип |
| classType | Тип для хранения экземпляра класса |
| primitiveType | 'boolean' | Логический тип |
| 'string' | Строковый литералл |
| 'char' | Символьный тип |
| 'byte' | Целочисленный тип размером в 1 байт |
| 'short' | Целочисленный тип размером в 2 байта |
| 'int' | Целочисленный тип размером в 4 байта |
| 'long' | Целочисленный тип размером в 8 байт |
| 'float' | Вещественный тип размером в 4 байт |
| 'double' | Вещественный тип размером в 8 байт |
| 'void' | Тип null |
| classType | qualifiedName | Имя класса в определенном формате |
| block | '{' | Начало блока |
| statement\* | Выражения |
| '}' | Конец блока |
| statement | block | Блок кода |
| variableDeclaration | Определение переменной |
| assignment | Присвоение |
| printStatement | Вывод в консоль |
| forStatement | Цикл for |
| returnStatement | Оператор возвращение результата работы функции |
| isStatement | Условный оператор |
| expression | Выражение |
| variableDeclaration | VARIABLE | Токен определения переменной |
| name | Название переменной |
| EQUALS | Токен оператора присвоения |
| expression | Выражение |
| assignment | name | Название переменной |
| EQUALS | Токен оператора присвоения |
| expression | Выражение |
| printStatement | PRINT | Токен системной функции вывода в консоль |
| expression | Выражение |
| returnStatement | 'return' expression | Возвращение результата работы функции со значением |
| 'return' | Возвращение пустого результата работы функции |
| isStatement | 'if' | Начало условного оператора |
| ('(')? | Наличие или отсутствие левой скобки |
| expression | Выражение |
| (')')? | Наличие или отсутствие правой скобки |
| trueStatement=statement | Выражение если "истино" |
| ('else' falseStatement=statement)? | Наличие или отсутствие выражения если "ложь" |
| forStatement | 'for' | Начало цикла "for" |
| ('(')? | Наличие или отсутствие левой скобки |
| forCondition | Параметры цикла "for" |
| (')')? | Наличие или отсутствие правой скобки |
| statement | Тело цикла "for" |
| forCondition | iterator=variableReference | Переменная-итератор |
| 'from' | Обозначение начальной точки старта итератора |
| startExpr=expression | Начальная точка старта итератора |
| range='to' | Обозначение конечной точки итератора |
| endExpr=expression | Конечная точка итератора |
| name | ID | Токен идентификатора переменной |
| argumentList | argument? (',' a=argument)\* | Перечень аргументов без названия |
| namedArgument? (',' namedArgument)\* | Перечень аргументов с заданным названием |
| argument | expression | Выражение |
| namedArgument | name | Название аргумента |
| '->' | Обозначение на какое значение ссылается название аргумента |
| expression | Выражение |
| expression | variableReference | Ссылка на переменную |
| owner=expression '.' functionName '(' argumentList ')' | Вызов функции из выражения |
| functionName '(' argumentList ')' | Вызов функции |
| superCall='super' '('argumentList ')' | Вызов суперметода |
| newCall='new' className '('argumentList ')' | Вызов конструктора |
| value | Выражение значение |
| '('expression '\*' expression')' | Умножение со скобками |
| expression '\*' expression | Умножение без скобок |
| '(' expression '/' expression ')' | Деление со скобками |
| expression '/' expression | Деление без скобок |
| '(' expression '+' expression ')' | Сложение со скобками |
| expression '+' expression | Сложение без скобок |
| '(' expression '-' expression ')' | Вычитание со скобками |
| expression '-' expression | Вычитание без скобок |
| expression cmp='>' expression | Условное выражение(больше) |
| expression cmp='<' expression | Условное выражение(меньше) |
| expression cmp='==' expression | Условное выражение(равенство) |
| expression cmp='!=' expression | Условное выражение(неравенство) |
| expression cmp='>=' expression | Условное выражение(больше или равно) |
| expression cmp='<=' expression | Условное выражение(меньше или равно) |
| variableReference | ID | Токен идентификатора переменной |
| value | NUMBER | Токен числового типа |
| BOOL | Токен логического типа |
| STRING | Токен строкового литерала |
| qualifiedName | ID | Токен идентификатора переменной |
| ('.' ID)\* | Токен идентификатора переменной с точкой(для указание пакета) |

1. Выходные данные

Выходные данные компилятора состоят из сгенерированного байт-кода для JVM.

Если компиляция прошла успешно и не было обнаружено никаких ошибок, компилятор создает файл расширения .class. Этот файл представляет собой байт-код программы, который можно запустить с на JVM.

Перечень принятых сокращений

|  |  |
| --- | --- |
|  |  |
| JVM | Java Virtual Machine |
| FASM | Flat Assembler |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |