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](#_Toc136394736)

[1.1. Обозначение и наименование программы 4](#_Toc136394737)

[1.2. Программное обеспечение, необходимое для функционирования программы 4](#_Toc136394738)

[1.3. Языки программирования, на которых написана программа 4](#_Toc136394739)

[2. Функциональное назначение 5](#_Toc136394740)

[3. Описание логической структуры 6](#_Toc136394741)

[3.1. Структура программы с описанием функций составных частей и связи между ними 6](#_Toc136394742)

[3.2. Лексический анализ 6](#_Toc136394743)

[3.3. Синтаксический анализ 6](#_Toc136394744)

[3.4. Абстрактное синтаксическое дерево 7](#_Toc136394745)

[3.5. Таблица символов 7](#_Toc136394746)

[3.6. Семантический анализ 7](#_Toc136394747)

[3.7. Генерация целевого кода 7](#_Toc136394748)

[3.8. Оптимизация 7](#_Toc136394749)

[4. Вызов и загрузка 8](#_Toc136394750)

[5. Входные данные 9](#_Toc136394751)

[5.1. Описание входного языка 9](#_Toc136394752)

[5.2. Описание токенов 9](#_Toc136394753)

[5.3 . Описание грамматики языка 11](#_Toc136394754)

[6. Выходные данные 15](#_Toc136394755)

[Перечень принятых сокращений 16](#_Toc136394756)

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 – 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. Семантический анализ.
4. Оптимизация кода.
5. Генерация байт кода.
6. 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 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 |
| antlr\target\generated-sources\antlr4\com.woody.antlr\WlangParser.java | Сгенерированный Antlr4 - Parser |
| antlr\target\generated-sources\antlr4\com.woody.antlr\WlangLexer.java | Сгенерированный Antlr4 - Lexer |
| antlr\target\generated-sources\antlr4\com.woody.antlr\WlangVisitor.java | Сгенерированный Antlr4 - Visitor |

* 1. Лексический анализ

В файле antlr\target\generated-sources\antlr4\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 antlr\target\generated-sources\antlr4\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 узлов AST: внутри функций парсера создаются узлы абстрактного синтаксического дерева (AST) – на основании файла грамматики Wlang.g4.
* Создание синтаксического анализатора: создается экземпляр синтаксического анализатора, который использует определенную грамматику для разбора токенов и построения AST – на основании файла грамматики reviLang.g4.
  1. Абстрактное синтаксическое дерево

В файле reviLang.g4 реализованы токены и правила – на основании которых можно строить абстрактное синтаксическое дерево используя утилиту для Antlr4

* 1. Таблица символов

В файле AST.py реализована таблица символов. Свободные регистры используются каждый раз при выполнении операции с зарезервированными данными. После окончания операции регистры снова попадают в список свободных.

* 1. Семантический анализ

Семантический анализ в процессе компиляции выполняется после заполнения таблицы символов и направлен на проверку семантической правильности программы.

* 1. Генерация целевого кода

Генерация ассемблер-кода происходит после успешного семантического анализа и осуществляется рекурсивно по AST, конечным результатом генерации является файл revi.s

* 1. Оптимизация

Процесс оптимизации происходит на заключительном этапе непосредственно после применения правил. Оптимизация производит следующие действия:

* производит слияние идентичных глобальных переменных.
* удаляет неиспользуемые аргументы функций.
* добавляет атрибуты функциям в LLVM IR.
* разворачивает маленькие функции (с 5 или менее инструкциями) в вызывающую функцию.
* удаляет все неиспользуемые глобальные переменные и функции.
* производит различные глобальные оптимизации.
* выполняет межпроцедурную константную передачу.
* удаляет неиспользуемый код.
* упрощает граф потока управления.
* выполняет глобальную нумерацию значений.
* объединяет похожие инструкции для уменьшения количества инструкций.
* выполняет перемещение инварианты из циклов.
* выполняет разреженную константную передачу.
* заменяет составные скалярные значения на отдельные переменные.
* выполняет анализ псевдонимов на основе типов.
* выполняет простой анализ псевдонимов.

1. Вызов и загрузка

Для вызова и загрузки программы, исходный код должен быть размещен в папке программы в файле "test.revi". После этого нужно запустить файл "main.py".

По завершении выполнения программы мы получаем исполняемый (.exe) файл.

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. Комментарии любой длины.

4. 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 (while).

6. Базовый вывод (строковый литерал, переменная).

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

* 1. Описание токенов

|  |  |  |
| --- | --- | --- |
| Название токена | Регулярное выражение | Описание |
| while | 'while' | Циклическая структура |
| String | '"' (ESC | ~["\\])\* '"' | Описание строки |
| ESC | '\\' . | Однострочный комментарий |
| ID | [a-zA-Z]+ DIGIT\* | Описание имени экземпляра |
| TIMES | '\*' | Операция умножения |
| REM | '%' | Операция деления с остатком |
| DIV | '/' | Деление без остатка |
| PLUS | '+' | Операция сложения |
| MINUS | '-' | Операция вычитания |
| LPAREN | '(' | Левосторонняя круглая скобка |
| RPAREN | ')' | Правосторонняя круглая скобка |
| POW | '^' | Операция возведения в степень |
| EQ | '=' | Операция присвоения |
| EQ\_EQ | '==' | Операция сравнение (равенство) |
| NOT\_EQ | '!=' | Операция сравнение (неравенство) |
| LBRACE | '{' | Левосторонняя фигурная скобка |
| RBRACE | '}' | Правосторонняя фигурная скобка |
| END\_STATE | ';' | Состояние завершения выражения |
| GT | '>' | Левосторонняя угловая скобка |
| LT | '<' | Правосторонняя угловая скобка |
| INT | DIGIT+ | Описание содержания типа данных |
| DECIMAL | INT '.' INT | Описание содержания типа данных |
| DIGIT | DIGIT: [0-9] | Описание цифры |
| SIGN | (MINUS)? | Описание допускающего отрицательного значения |
| WS | [ \r\n\t] + -> skip | Описание пропуска пробелов |
| COMMENT | '/\*' .\*? '\*/' -> skip | Многострочные комментарии |
| LINE\_COMMENT | '//' ~[\r\n]\* -> skip | Однострочные комментарии |

* 1. . Описание грамматики языка

|  |  |  |
| --- | --- | --- |
| **Название нетерминала** | **Возможные продукции** | **Описание** |
| program | function\* | Начало программы с запуска функции |
| function | functionHead | Название функции и указание типа данных |
| functionBody | Тело функции |
| functionHead | type | Тип данных функции |
| arguments | Агрументы функции |
| type | 'int' | Описание типа данных int |
| 'double' | Описание типа данных double |
| arguments | arg | Аргументы функции |
| arg | type | Содержание аргументов функции |
| functionBody | statement | Содержание тела функции |
| statement | assignmentStatement | Условие присвоения |
| returnStatement | Возврат значения из фукнции |
| ifStatement | Оператор ветвления |
| whileStatement | Оператор циклического прохождения |
| printStatement | Оператор вывода информации |
| moveValueVariable | Описание переприсвоения |
| atom | INT | Определение типа данных int |
| DECIMAL | Определение типа данных double |
| ID | Описание имени экземпляра |
| relop | EQ\_EQ | Оператор сравнение (равенство) |
| GT | Левосторонняя угловая скобка |
| LT | Правосторонняя угловая скобка |
| NOT\_EQ | Оператор сравнения(неравенство) |
| functionCall | params | Вывоз функции и передача параметров |
| params | param | Определение параментра функции |
| param | expression | Содержание параметра фунцкии |
| equation | expression | Выражение |
| relop | Переопределение/работа с выражениями |
| expression | convert\_type | Операция конвертации данных |
| functionCall | Операция вызова функции |
| expression op=POW expression | Операция возведения в степень |
| expression op=(TIMES | DIV| REM) expression | Операция умножения, делений |
| expression op=(PLUS | MINUS) expression | Простые арифметические операции |
| expression | Вызов выражения |
| (PLUS | MINUS)\* atom | Простые арифметические операции с токеном atom |
| STRING | Строковые литералы |
| convert\_type | type | Конвертация в указанный тип данных |
| expression | Дочернее выражения к конвертации |
| assignmentStatement | type | Указание типа данных |
| expression | Выражение к присвоению данных |
| returnStatement | 'return' | Ключевое слово инструкции к выполнению возврата |
| expression | Выражение для возврата |
| whileStatement | WHILE LPAREN equation RPAREN LBRACE | Выражение сравнения данных для работы цикла |
| (statement|expression)\* | Содержание тела циклической операции |
| moveValueVariable | expression | Переприсвоение значения выражения |
| printStatement | 'print' | Ключевое слово инструкции к выводу данных |
| (STRING|atom) | Содержание выведенных данных |
| ifStatement | 'if' | Ключевое слово инструкции выполнения ветвления |
| equation | Операция сравнения данных |
| statement | Содержания тела инструкции |

1. Выходные данные

Выходные данные компилятора состоят из сгенерированного целевого кода ассемблер .

Если компиляция прошла успешно и не было обнаружено никаких ошибок, компилятор создает целевой код в формате ASM. Этот код представляет собой низкоуровневое промежуточное представление исходной программы. Он будет сохранен в файле "revi.s".

Перечень принятых сокращений

|  |  |
| --- | --- |
|  |  |
| AST | Абстрактное синтаксическое дерево |
| 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 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |