|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_

**Расчётно-пояснительная записка**

**к курсовому проекту на тему**

**«?»**

**по курсу «Проектирование компиляторов»**

Студент \_\_\_ИУ7-21М\_\_\_\_ \_\_\_\_Карпухин А.С. \_\_\_\_

(Группа) (И.О.Фамилия)

Студент \_\_\_ИУ7-22М\_\_\_\_ \_\_\_\_Магазинов Н.А.\_\_\_\_

(Группа) (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_Ступников А.А.\_\_\_\_

(И.О.Фамилия)

*2021 г*

**Оглавление**

[Введение 3](#_Toc74070649)

[**1.** **Аналитический раздел** 4](#_Toc74070650)

[**1.1.** **Лексический анализ** 4](#_Toc74070651)

[**1.2.** **Синтаксический анализ** 4](#_Toc74070652)

[**1.2.1.Нисходящий разбор** 5](#_Toc74070653)

[**1.2.2.Восходящий разбор** 5](#_Toc74070654)

[**1.3.** **Средства построения лексического и синтаксического анализаторов** 5](#_Toc74070655)

[**1.4.** **Грамматика языка Scala** 6](#_Toc74070656)

[**1.5.** **Таблицы символов** 7](#_Toc74070657)

[2. Конструкторский раздел 8](#_Toc74070658)

[3. Технологический раздел 9](#_Toc74070659)

[Заключение 10](#_Toc74070660)

[Список использованных источников 11](#_Toc74070661)

Введение

Цель курсовой работы – разработать программное обеспечение для компилирования кода из языка Scala в язык ?

Для достижения поставленной цели необходимо решить следующие задачи:

* реализовать лексический анализатор;
* реализовать синтаксический анализатор;
* реализовать семантический анализ, по результатам которого получить:
  + таблицы локальных и глобальных символов (классов, объектов, типов, функций, переменных);
  + граф иерархии наследования;
  + граф вызовов.

1. **Аналитический раздел**
   1. **Лексический анализ**

Лексический анализ образует первый этап процесса компиляции. На этом этапе символы, составляющие исходную программу, считываются и группируются в отдельные лексические элементы – лексемы. Лексический анализатор заменяет в программе идентификаторы и константы лексемами, что делает представление программы удобнее для дальнейшей обработки. Так же лексический анализатор устраняет ненужные пробелы и комментарии.

Лексический анализ может быть представлен и как самостоятельная фаза трансляции, и как составная часть фазы синтаксического анализа. В первом случае лексический анализатор реализуется в виде отдельного модуля, который принимает последовательность символов, составляющих текст компилируемой программы, и выдаёт список обнаруженных лексем. Во втором случае лексический анализатор фактически является подпрограммой, вызываемой синтаксическим анализатором для получения очередной лексемы.

В процессе лексического анализа так же выявляются лексические ошибки – ошибки, связанные с наличием недопустимых символов, некорректной записью идентификаторов, строк и т.д.

Результатом работы лексического анализа является цепочка лексем.

* 1. **Синтаксический анализ**

Второй фазой процесса компиляции является фаза синтаксического анализа. Входными данными синтаксического анализатора является результат работы лексического анализатора – цепочка лексем. Синтаксический анализ – разбор, в котором исследуется цепочка лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в синтаксисе языка. Результатом работы синтаксического анализатора является дерево разбора, которое представляет синтаксическую структуру исходной программы.

* + 1. **Нисходящий разбор**

В нисходящем анализе дерево вывода цепочки строится от корня к листьям, т.е. нисходящие анализаторы строят вывод, начиная от аксиомы грамматики и заканчивая цепочкой терминальных символов. В общем виде нисходящий анализ представлен в анализе методом рекурсивного спуска, который может использовать откаты, т.е. производить повторный просмотр считанных символов.

* + 1. **Восходящий разбор**

Восходящий анализатор предназначен для построения дерева разбора, начиная с листьев и двигаясь вверх к корню дерева разбора. Можно представить себе этот процесс как "свертку" исходной строки к аксиоме грамматики. Каждый шаг свертки заключается в сопоставлении некоторой подстроки и правой части какого-то правила грамматики, и замене этой подстроки на нетерминал, являющийся левой частью правила. Если на каждом шаге подстрока выбирается правильно, то в результате получается правый вывод строки.

* 1. **Средства построения лексического и синтаксического анализаторов**

Существует множество различных стандартных средств для построения синтаксических анализаторов: Lex и Yacc, Coco/R, ANTLR и др.

Генератор Yacc (Yet Another Compiler Compiler) - стандартный генератор синтаксических анализаторов в Unix-системах. Yacc генерирует парсер на основе аналитической грамматики, описанной в нотации BNF (форма Бэкуса-Наура) или контекстно-свободной грамматики. Как правило, Yacc используется в связке с Lex –генератором лексических анализаторов.

Генератор lex является стандартным генератором лексических анализаторов в операционных системах Unix.

ANTLR – это генератор синтаксических анализаторов для чтения, обработки или трансляции как структурированных текстовых, так и бинарных файлов. На основе заданной грамматики языка ANTLR генерирует код нисходящего синтаксического анализатора, который может строить абстрактное синтаксического дерево и производить его обход.

* + 1. **Выбор методов лексического и синтаксического анализа**

В данной работе для реализации целевого программного обеспечения ввиду больших объемов грамматики выбранного языка было принято решение использовать генератор лексических и синтаксических анализаторов ANTLR. Результат генерации включает в себя четыре файла на выбранном целевом языке:

* Файл с кодом лексического анализатора;
* Файл с кодом нисходящего синтаксического анализатора;
* Файл с кодом базовой реализации паттерна «Visitor»;
* Файл с кодом базовой реализации паттерна «Listener».

Упомянутые реализации паттернов поведения используются для обхода полученного в результате синтаксического анализа дерева разбора исходного кода компилируемой программы.

* 1. **Формат грамматики целевого языка**

В качестве компилируемого языка был выбран языка Scala. Грамматика языка описана в подходящем для ANTLR формате g4. Структура описания грамматики имеет вид, представленный в листинге 1.1.

Листинг 1.1 – Общий вид грамматики

grammar Name;

rule1

...

ruleN

Правила грамматики начинаются с ":" и заканчиваются ";". Несколько правил разделяются знаком "|". Пример правил языка Scala приведен в листинге 1.2.

Листинг 1.2 – Пример грамматики языка Scala

|  |
| --- |
| expr  : (bindings | 'implicit'? Id | '\_') '=>' expr  | expr1  ;  expr1  : 'if' '(' expr ')' NL\* expr ('else' expr)?  | 'while' '(' expr ')' NL\* expr  | 'try' expr ('catch' expr)? ('finally' expr)?  | 'do' expr 'while' '(' expr ')'  | 'for' ('(' enumerators ')' | '{' enumerators '}') 'yield'? expr  | 'throw' expr  | 'return' expr?  | ((simpleExpr | simpleExpr1 '\_'?) '.')? Id '=' expr  | simpleExpr1 argumentExprs '=' expr  | postfixExpr ascription?  | postfixExpr 'match' '{' caseClauses '}'  ; |

* 1. **Методы обхода дерева разбора**

По умолчанию ANTLR реализует два способа обхода дерева разбора, генерируемого в результате синтаксического анализа. Оба метода реализованы с использованием стандартных поведенческих шаблонов проектирования – «Visitor» («Посетитель») и «Listener» («Слушатель»).

* + 1. **Паттерн «Visitor»**

Паттерн «Посетитель» предполагает определение для набора объектов, принадлежащих различным классам, методов обработки каждого уникального типа в наборе.

В качестве реализации создается базовый класс Visitor с методами Visit() для каждого подкласса родительского Element, описывающего произвольный объект в наборе. В иерархию Element, в свою очередь, добавляется метод Accept(visitor), вызывающий соответствующую реализацию метода Visit() для данного объекта. Для каждой операции, выполняемой на объектах иерархии Element, создается производный от Visitor класс.

* + 1. **Паттерн «Listener»**

Реализация паттерна «Слушатель» в ANTLR далека от классической, использующей механизм подписчиков и издателей. В данном случае реализация основывается на базовом классе Listener, в котором для каждого класса объекта в наборе создается два метода – Enter и Exit, вызываемые при входе в узел и выходе из узла дерева разбора соответственно. По результатам обхода поддерева текущего узла в методе Exit выполняются некоторые действия.

* + 1. **Выбор метода обхода**

Для обхода дерева разбора исходного кода был выбран метод, основанный на реализации паттерна «Visitor» ввиду его простоты и эффективности при разборе древовидных структур. В данном случае эта реализация также удобна тем, каждый метод класса Visitor обязан возвращать некоторый результат. Это может быть удобно при выводе типов выражений подобно атрибутивным грамматикам.

* 1. **Таблицы символов**

Таблица символов, как правило, применяется для хранения информации о различных конструкциях исходного языка. Содержимое таблицы просматривается каждый раз, когда в исходном тексте программы встречается некоторое имя. Каждый вновь объявленный в коде исходного языка именованный символ заносится в таблицу. Структура данных заполняется во время фазы анализа, и ее содержимое может быть использовано в дальнейшем при генерации целевого кода.

Механизм таблицы символов должен обеспечивать эффективный поиск и добавление в таблицу символов.

Таблица символов может служить следующим целям в зависимости от используемого языка:

* Хранить имена всех сущностей в структурированной форме в одном месте;
* Чтобы проверить, была ли объявлена ​​переменная;
* Чтобы реализовать проверку типов, путем проверки присваиваний и выражений в исходном коде семантически правильно;
* Чтобы определить область имени (разрешение области).
  + 1. **Линейные списки**
    2. **Хеш-таблицы**
    3. **Выбор реализации таблицы символов**

1. Конструкторский раздел

БЛАБЛАБЛА

1. Технологический раздел

БЛАБЛАБЛА

Заключение

Охуенный компилятор вышел

Список использованных источников

АХА УЛЬМАН