|  |  |
| --- | --- |
|  | **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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***«Разработка компилятора языка miniJava»***

Студент **ИУ7-21М** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Коноваленко В.Д.**

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Ступников А.А.**

**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)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. В. Рудаков

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине Конструирование компиляторов

Студент группы ИУ7-21М

Коноваленко Владимир Денисович

(Фамилия, имя, отчество)

Тема курсового проекта Разработка компилятора языка miniJava

Направленность КП (учебный, исследовательский, практический, производственный, др.)

учебный

Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

***Задание*** Описать грамматику языка miniJava. Разработать компилятор языка miniJava на языке C#, использующий библиотеку ANTLR для синтаксического анализа потока входных данных и построения AST-дерева. В качестве целевой платформы использовать .NET для генерации промежуточного представления на основе составленного синтаксического дерева.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 20-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение и список литературы.

Дата выдачи задания «\_\_\_» 2024 г.

**Руководитель курсового проекта**  А.А.Ступников

(Подпись, дата) (И.О.Фамилия)

**Студент**  В.Д.Коноваленко

(Подпись, дата) (И.О.Фамилия)

**ОГЛАВЛЕНИЕ**

1. Аналитическая часть
   1. Принцип работы компиляторов
   2. Методы реализации лексического и синтаксического анализа
   3. Платформа .NET
2. Конструкторская часть
   1. IDEF0
   2. Язык miniJava
   3. Лексический и синтаксический анализаторы
   4. Семантический анализ
3. Технологическая часть
   1. Выбор языка программирования и средств разработки
   2. Генерация анализаторов
   3. Реализация семантического анализатора
   4. Примеры работы программы

**ВВЕДЕНИЕ**

MiniJava [1] является подмножеством языка программирования Java [2]. Каждая программа miniJava является программой с семантикой языка программирования Java.

Целью курсового проекта является разработка компилятора для языка программирования miniJava под платформу .NET.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области.
2. Повести анализ существующих методов реализации лексического и синтаксического анализаторов.
3. Разработать компилятор для языка программирования miniJava.
4. Подготовить набор тестовых данных для тестирования и демонстрации разработанного приложения.

# 1. Аналитическая часть

## 1.1 Принцип работы компиляторов

Компилятор – это программа, которая считывает текст программы на одном языке, называемом исходным, и транслирует или переводит его в эквивалентный текст на другом языке – целевом (рисунок 1) [3].

Компилятор должен сообщать об ошибках в исходной программе, обнаруженных в ходе трансляции.

|  |
| --- |
|  |
| Рисунок 1. Компилятор |

Отображение компилятором исходной программы в семантически эквивалентную ей целевую программу разделается на две части: анализ и синтез [8, 9].

Анализ разбивает исходную программу на составные части и накладывает на них грамматическую структуру. Далее эта структура используется для создания промежуточного представления исходной программы. В случае, если программа составлена некорректно, анализ должен выдать соответствующие информационные сообщения пользователю. Помимо этого, анализ собирает информацию о программе и сохраняет ее в таблице символов, которая передается вместе с промежуточным представлением синтезу [3]. Синтез строит целевую программу на основе полученного с предыдущего шага представления и таблицы символов. Как правило, анализ называют начальной стадией (front-end), а синтез – заключительной (back-end).

Фазы компиляции, представлены на рисунке 2 [3].

|  |
| --- |
|  |
| Рисунок 2. Фазы компиляции |

Далее более подробно рассмотрим, что происходит на каждой из этих фаз.

**Лексический анализ.** Является первой фазой компиляции. Также данная фаза называется сканированием. Лексический анализатор читает поток символов, которые составляют исходную программу, и группирует их в значащие последовательности, называемые лексемами. Для каждой построенной лексемы анализатор строит выходной токен вида:(имя\_токена, значение\_атрибута). [3]

Первый компонент токена представляет собой абстрактный символ, который используется во время синтаксического анализа, а второй компонент указывает на запись в таблице символов, которая соответствует данному токену.

**Синтаксический анализ.** Вторая фаза компиляции – синтаксический анализ или разбор. Анализатор использует первые компоненты токенов, полученных при лексическом анализе, для построения промежуточного древовидного представления, которое описывает грамматическую структуру потока токенов. Обычно для этой цели используется синтаксическое дерево, в котором каждый внутренний узел представляет операцию, а дочерние узлы – аргументы этой операции [8, 9].

**Семантический анализ.** Семантический анализатор использует синтаксическое дерево и информацию из таблицы символов. Данный этап собирает информацию о типах и сохраняет ее в синтаксическом дереве или в таблице символов для последующего использования в процессе генерации промежуточного кода [3].

Важной частью данной фазы является проверка типов. В процессе этой проверке компилятор проверяет, имеет ли каждый оператор операнды соответствующего типа [4].

**Генерация промежуточного кода.** В процессе трансляции исходной программы в целевой код компилятор может создавать одно или несколько промежуточных представлений различного вида. Синтаксические деревья являются видом промежуточного представления, они обычно используются в процессе синтаксического и семантического анализа [3].

После синтаксического и семантического анализа исходной программы многие компиляторы генерируют явное низкоуровневое или машинное промежуточное представление исходной программы, которое можно рассматривать как программу абстрактной вычислительной машины. Данное промежуточное представление должно легко генерироваться и транслироваться в целевой машинный язык.

**Оптимизация кода.** Данная фаза предназначения для улучшения промежуточного кода с целью получения более качественного целевого кода. Обычно под «более качественным» подразумевается более производительный код, но иногда встречаются и другие критерии [3].

**Генерация кода.** Генератор кода получает на вход промежуточное представление исходной программы и отображает его в целевой язык. Если целевой язык представляет собой машинный код, то для каждой переменной, которая используется в программе, должны быть выбраны соответствующие регистры или ячейки памяти. Далее промежуточные команды транслируются в последовательности машинных команд, выполняющих те же действия.

## 1.2 Методы реализации лексического и синтаксического анализаторов

Реализовывать лексический и синтаксический анализаторы можно вручную, с использованием стандартных алгоритмов анализа. Однако, гораздо легче и быстрее использовать готовые инструменты для их генерации. За счёт их использования, у разработчика нет необходимости реализовывать анализаторы самостоятельно. На вход такие инструменты принимают файл, описывающий грамматику языка, затем на основе этой грамматики они создают лексический или синтаксический анализатор на одном из языков программирования.

Далее будут рассмотрены наиболее популярные инструменты для генерации лексического и синтаксического анализаторов.

### 1.2.1 Генераторы лексического анализатора

**Lex** — программа для генерации лексических анализаторов, обычно используемая совместно с генератором синтаксических анализаторов yacc, является стандартным генератором лексических анализаторов в операционных системах Unix, а также включён в стандарт POSIX. Lex читает входной поток, описывающий лексический анализатор, и даёт на выходе исходный код на языке программирования Cи [5].

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Compiler project. [Электронный ресурс]. URL: <https://www.cs.unc.edu/~plaisted/comp455/Mini-Java%20Grammar.pdf> (дата обращения: 03.06.2024).
2. Java (programming language). [Электронный ресурс]. URL: <https://en.wikipedia.org/wiki/Java_(programming_language)> (дата обращения: 03.06.2024).
3. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.
4. АХО А., УЛЬМАН Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.1.: Синтаксичечкий анализ. - М.: Мир, 1978.
5. <https://ru.wikipedia.org/wiki/Lex>