|  |  |
| --- | --- |
|  | **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 6](#_Toc168507164)

[1.1 Принцип работы компиляторов 6](#_Toc168507165)

[1.2 Методы реализации лексического и синтаксического анализаторов 9](#_Toc168507166)

[1.2.1 Генераторы лексических анализаторов 9](#_Toc168507167)

[1.2.2 Генераторы синтаксических анализаторов 10](#_Toc168507168)

[1.3 Платформа .NET 11](#_Toc168507169)

[2. Конструкторская часть 12](#_Toc168507170)

[2.1 IDEF0 12](#_Toc168507171)

[2.2 Язык MiniJava 14](#_Toc168507172)

[2.3 Лексический и синтаксический анализаторы 14](#_Toc168507173)

[2.4 Семантический анализ 15](#_Toc168507174)

[3. Технологическая часть 15](#_Toc168507175)

[3.1 Выбор средств разработки 15](#_Toc168507176)

[3.2 Генерация лексического и синтаксического анализаторов 16](#_Toc168507177)

[3.3 Реализация семантического анализатора и кодогенерации 17](#_Toc168507178)

[3.4 Примеры работы программы 18](#_Toc168507179)

[3.4.1 Сортировка пузырьком 18](#_Toc168507180)

[3.4.2 Инвертирование односвязного списка 21](#_Toc168507181)

[3.4.3 Наследование 26](#_Toc168507182)

[ЗАКЛЮЧЕНИЕ 29](#_Toc168507183)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 30](#_Toc168507184)

[ПРИЛОЖЕНИЕ 1 32](#_Toc168507185)

**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 [3, 4].

**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 лексических анализаторов

**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].

**Flex** (Fast Lexical Analyzer) — генератор лексических анализаторов. Заменяет Lex в системах на базе пакетов GNU и имеет аналогичную функциональность. При этом Flex не является частью проекта GNU. На входе программа получает текст в свободном формате и правила выделения лексем, а на выходе даёт код анализатора, в виде функции на языке Си [6].

**ANTLR** (от англ. ANother Tool for Language Recognition — «ещё одно средство распознавания языков») — генератор нисходящих анализаторов для формальных языков. ANTLR преобразует контекстно-свободную грамматику в виде РБНФ в программу на C++, Java, C#, JavaScript, Go, Swift, Python. Используется для разработки компиляторов, интерпретаторов и трансляторов [7]. Ключевыми преимуществами данного инструмента является поддержка множества языков программирования, а также то, что он сочетает в себе генератор как лексических, так и синтаксических анализаторов.

### 1.2.2 Генераторы синтаксических анализаторов

**GNU Bison** — программа, предназначенная для автоматического создания синтаксических анализаторов по данному описанию грамматики. Программа bison относится к свободному ПО, разработана в рамках проекта GNU и портирована под все традиционные операционные системы. Программа bison во многом совместима с подобной программой yacc. Обычно используется в комплексе с лексическим анализатором flex [8].

**yacc** — компьютерная программа, служащая стандартным генератором синтаксических анализаторов (парсеров) в Unix-системах. Название является акронимом «Yet Another Compiler» («ещё один компилятор компиляторов»). Yacc генерирует парсер на основе аналитической грамматики, описанной в нотации BNF (форма Бэкуса-Наура) или контекстно-свободной грамматики. На выходе yacc выдаётся код парсера на языке программирования Си.

**ANTLR** был описан ранее, и также предоставляет функционал генерации синтаксических анализаторов на различных языках программирования.

Таким образом, в связи с использованием языка C# для написания программы-компилятора в рамках данной курсовой работы, наиболее подходящим вариантом генератора лексического и синтаксического анализатора является ANTLR.

## 1.3 Платформа .NET

.NET — это бесплатная платформа разработчика с открытым кодом для создания множества приложений. Платформа .NET разработана для обеспечения производительности, безопасности и надежности. Она обеспечивает автоматическое управление памятью с помощью сборщика мусора (GC). Это типобезопасный и безопасный для памяти, из-за использования GC и строгих компиляторов языка. Он включает в себя большой набор библиотек, имеющих широкие функциональные возможности и оптимизированный для производительности в нескольких операционных системах и архитектурах микросхем.

Платформа .NET состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET. Основой платформы .NET является среда CLR. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом средой накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом [9].

Среда CLR управляет памятью, выполнением потоков, выполнением кода, проверкой безопасности кода, компиляцией и другими системными службами. Эти средства являются внутренними для управляемого кода, который выполняется в среде CLR.

Среда выполнения также обеспечивает надежность кода, реализуя инфраструктуру строгой типизации и проверки кода, которую называют системой общих типов (CTS). Система общих типов обеспечивает самоописание всего управляемого кода. Различные языковые компиляторы корпорации Microsoft и независимых изготовителей создают управляемый код, удовлетворяющий системе общих типов. Это означает, что управляемый код может принимать другие управляемые типы и экземпляры, при этом обеспечивая правильность типов и строгую типизацию.

Кроме того, управляемая среда выполнения исключает многие часто возникающие проблемы с программным обеспечением. Например, среда выполнения автоматически управляет размещением объектов и ссылками на объекты, освобождая их, когда они больше не используются. Автоматическое управление памятью исключает две наиболее часто возникающие ошибки приложений: утечки памяти и недействительные ссылки на память.

Среда выполнения разработана для повышения производительности. Хотя общеязыковая среда выполнения предоставляет многие стандартные службы времени выполнения, управляемый код никогда не интерпретируется. Средство компиляции по требованию (JIT) позволяет выполнять весь управляемый код на машинном языке компьютера, где он запускается. Между тем диспетчер памяти устраняет возможность фрагментации памяти и увеличивает объем адресуемой памяти для дополнительного повышения производительности [10].

# 2. Конструкторская часть

## 2.1 IDEF0

Высокоуровневая модель разрабатываемого компилятора в нотации IDEF0 представлена на рисунке 3.

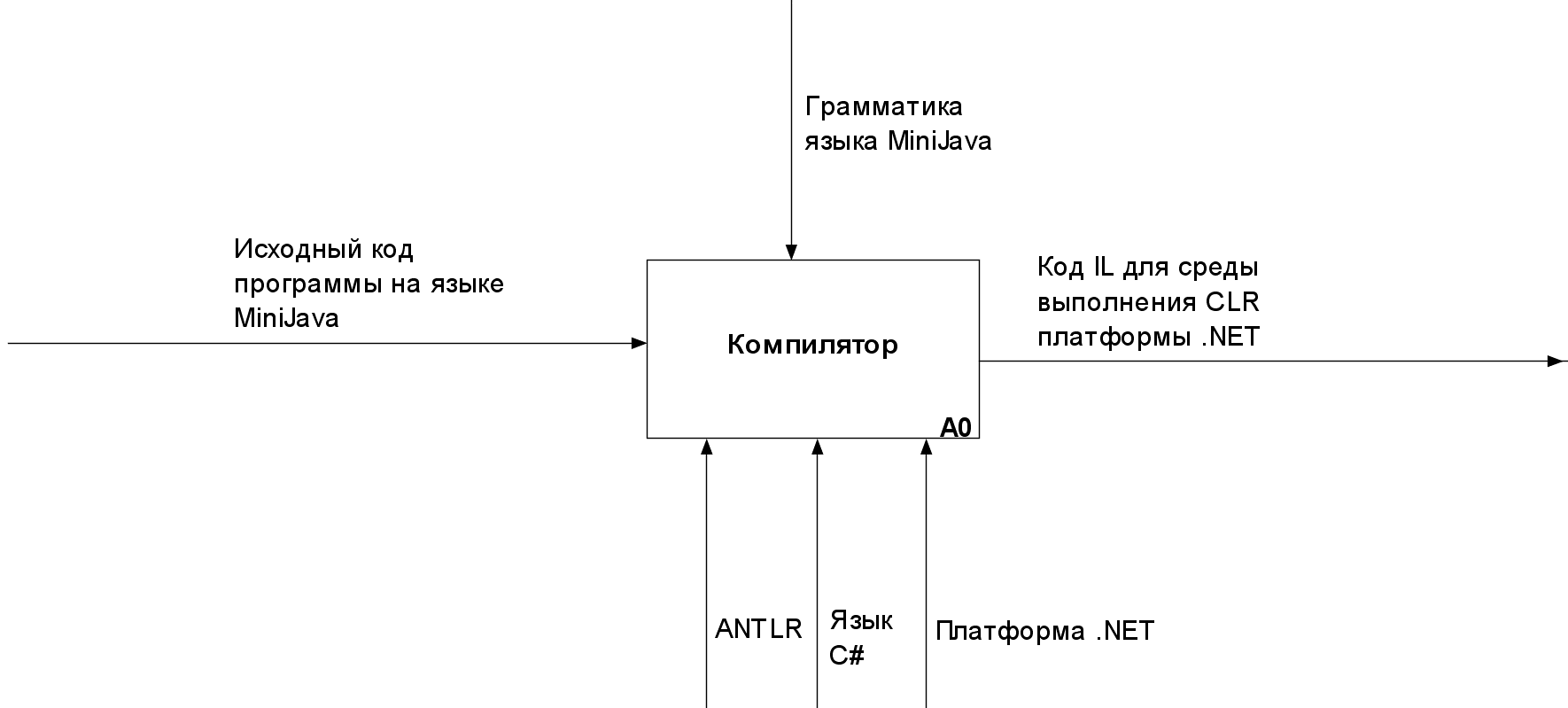


Рисунок 3. Высокоуровневая модель разрабатываемого компилятора в нотации IDEF0.

Разрабатываемый компилятор можно разделить на 3 компонента, обеспечивающие преобразование исходного кода в код IL для среды выполнения CLR. Детализированная модель разрабатываемого компилятора в нотации IDEF0 представлена на рисунке 4.

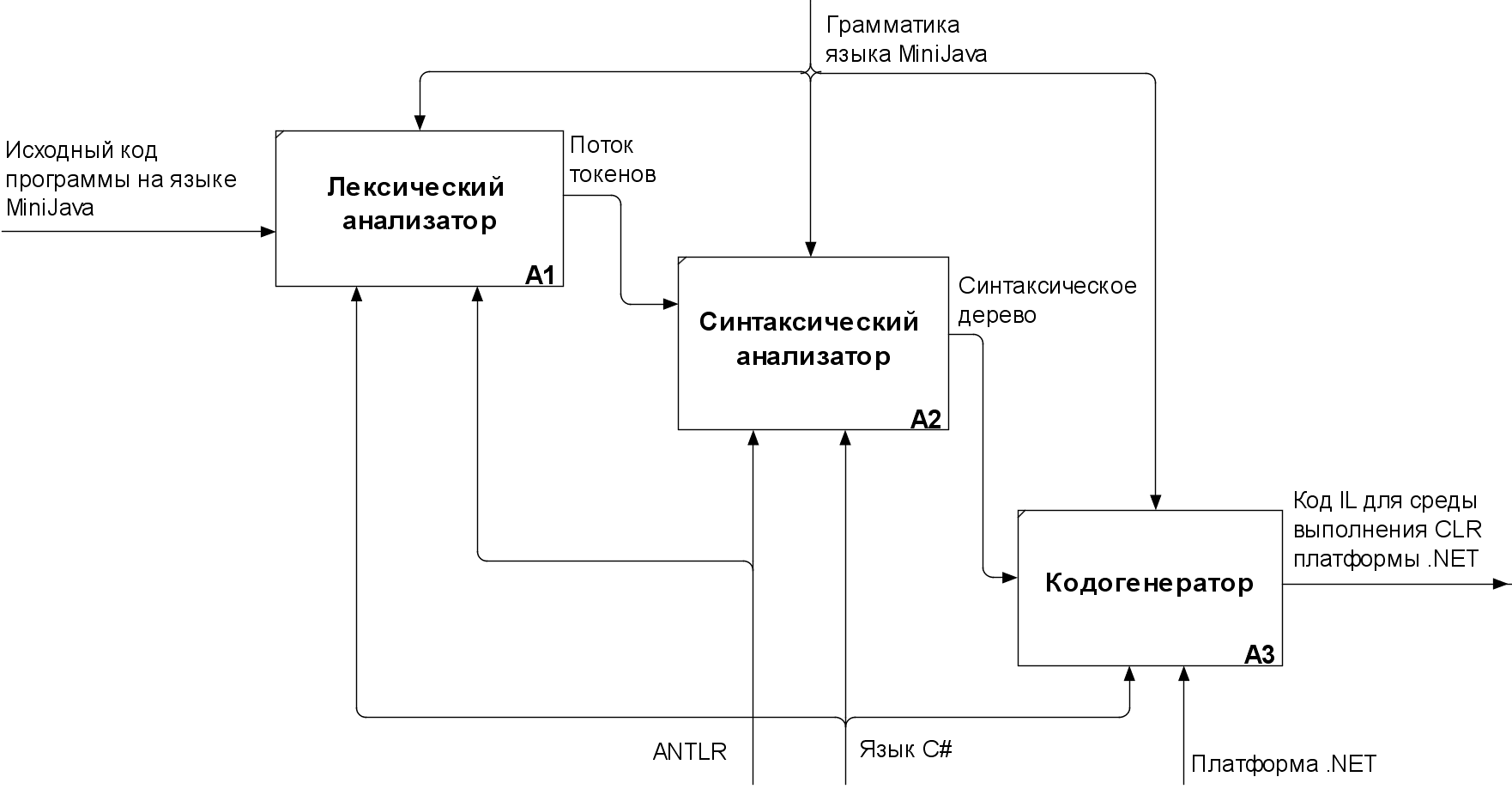


Рисунок 4. Детализированная модель разрабатываемого компилятора в нотации IDEF0.

## 2.2 Язык MiniJava

MiniJava является подмножеством языка программирования Java. Каждая программа MiniJava является программой с семантикой языка программирования Java.

В языке существуют следующие типы данных: integer, boolean и массив integer. Поддерживаются классы и механизм их наследования (однако, отсутствуют виртуальные методы). Также язык предоставляет функцию вывода в консоль, цикл while и условия if-else.

Грамматика MiniJava приведена в приложении А. Для удобства написания тестовых программ в грамматику были добавлены два правила:

1. Statement ::= Expression ";"
2. Expression ::= "System.randomInt(" Expression ")"

## 2.3 Лексический и синтаксический анализаторы

Код лексического и синтаксического анализатора генерируется при помощи программы ANTLR4. На вход она получает файл с описанием грамматики языка MiniJava, а в результате своей работы создаёт файлы, содержащие класс лексера и парсера на языке программирования C#. Также генерируется базовые классы для обхода синтаксического дерева, получаемого в результате работы парсера.

Для работы сгенерированных файлов их достаточно добавить в проект, после чего подать на вход лексеру исходный код программы, а полученный от него поток токенов передать на вход парсеру. В результате получится синтаксическое дерево, используемое для дальнейшего синтаксического анализа и кодогенерации.

## 2.4 Семантический анализ

ANTLR предоставляет два варианта базовых классов для обхода синтаксического дерева: Listener и Visitor.

Listener позволяет произвести обход дерева в глубину, без возможности переопределения способа обхода. Таким образом, обходится каждый узел дерева и вызываются обработчики при входе и выходе из этих узлов.

Visitor позволяет более гибко настраивать обход дерева – допускается переопределение порядка обхода узлов, либо не посещение какого-либо узла вообще. Данный подход более предпочтителен в рамках данной работы, так как не всегда для алгоритмов кодогенерации достаточного простого обхода дерева в глубину, без модификаций.

# 3. Технологическая часть

## 3.1 Выбор средств разработки

Для разработки компилятора был выбран язык программирования C#. C# является строго типизированным, объектно – ориентированным языком, призванным обеспечивать оптимальное сочетание удобства, простоты и производительности.

Основными преимуществами этого языка программирования являются:

1) Строгая и утиная типизация.

2) Наличие большого числа существующих библиотек и пакетов для платформы .NET, которые обеспечивают различные дополнительные возможности разработки.

3) Вариативность использования.

4) Лёгкость написания тестов.

5) Удобная собственная среда разработки.

6) Автоматический менеджмент памяти и сборка мусора.

7) Наличие встроенного обработчика событий.

8) Широкий функционал и возможности языка.

9) Отсутствие заголовочных файлов. Весь код распределен по пакетам и сборкам, что облегчает ситуацию при наличии перекрестных ссылок.

## 3.2 Генерация лексического и синтаксического анализаторов

В результате работы программы ANTLR4 на основе грамматики генерируются файлы:

* MiniJava.tokens и MiniJavaLexer.tokens. Данные файлы содержат список имен токенов и их числовых присвоений, сгенерированных ANTLR4.
* MiniJava.interp и MiniJavaLexer.interp. Данные файлы содержащее данные, которые позволяют запускать встроенный интерпретатор вместо сгенерированного парсера. В основном это используется для IDE, которые позволяют отлаживать. Файл содержит точно ту же информацию, что и сгенерированный файл синтаксического анализатора/лексера.
* MiniJavaLexer.cs – файл с реализаций лексического анализатора.
* MiniJavaParser.cs – файл с реализацией синтаксического анализатора.
* MiniJavaVisitor.cs – файл с интерфейсом класса для обхода синтаксического дерева.
* MiniJavaBaseVisitor.cs – файл с реализацией базового класса для обхода синтаксического дерева.

## 3.3 Реализация семантического анализатора и кодогенерации

Для семантического анализа и кодогенерации было реализовано три класса, каждый из которых является наследником MiniJavaBaseVisitor:

1. ClassDeclarationsVisitor. Данный класс обходит узлы описания главного и других классов (MainClassDeclaration и ClassDeclaration) для составления списка существующих в исходной программе классов. На этом этапе создаются только определения классов, без описания функций и полей внутри них. Если же функции и поля описывать сразу, то можно столкнуться с ошибкой, что класс не определен, хотя он просто описан в программе позже.
2. ClassesMethodsDeclarationsVisitor. Данный класс обходит узлы описания сигнатур методов (MainMethodDeclaration и MethodDeclaration), для составления списка методов в каждом из классов. Важно отметить, что на данном этапе рассматриваются именно сигнатуры методов, а не их реализации.
3. CodeGenerationVisitor. Данный класс на основе списка классов, списка методов каждого класса и обхода всех узлов синтаксического дерева генерирует IL код для среды выполнения CLR. Для генерации кода используется библиотека RunSharp [12], являющаяся обёрткой над .NET Reflection.Emit – стандартной библиотеки платформы .NET, содержащей классы, которые позволяют компилятору или сторонним инструментам создавать метаданные и промежуточный язык (IL) [13].

## 3.4 Примеры работы программы

Для тестирования работы компилятора были написаны программы на языке MiniJava, реализующие некоторые структуры данных и алгоритмы работы с ними.

### 3.4.1 Сортировка пузырьком

На листинге 1 в приложении Б представлен исходный код программы на языке MiniJava, которая заполняет массив чисел случайными числами, после чего сортирует их методом пузырька. На листинге 2 в приложении Б представлен сгенерированный промежуточный код для данной программы.

### 3.4.2 Инвертирование односвязного списка

На листинге 3 в приложении Б представлен исходный код программы на языке MiniJava, который создаёт односвязный список, заполняет его значениями, а затем инвертирует. На листинге 4 в приложении Б представлен сгенерированный промежуточный код для данной программы.

### 3.4.3 Наследование

На листинге 5 в приложении Б представлен исходный код программы на языке MiniJava для демонстрации наследования классов. На листинге 6 в приложении Б представлен сгенерированный промежуточный код для данной программы.

# ЗАКЛЮЧЕНИЕ

Таким образом, в рамках данной курсовой работы был рассмотрен принцип работы компиляторов, а также инструменты для генерации лексических и синтаксических анализаторов.

Был разработан компилятор языка MiniJava на языке C#, использующий библиотеку ANTLR для синтаксического анализа потока входных данных и построения AST-дерева. Для генерации промежуточного представления на основе составленного синтаксического дерева в качестве целевой платформы использовалась .NET.

# 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. Википедия: Lex [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Lex> (дата обращения: 03.06.2024)
6. Википедия: Flex [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Flex_(%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BB%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2)> (дата обращения: 03.06.2024)
7. Википедия: ANTLR [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/ANTLR> (дата обращения: 03.06.2024)
8. Википедия: GNU Bison [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/GNU_Bison> (дата обращения: 03.06.2024)
9. Microsoft: Введение в .NET [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/core/introduction> (дата обращения: 03.06.2024)
10. Microsoft: Общие сведения о платформе .NET [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/framework/get-started/overview> (дата обращения: 03.06.2024)
11. MiniJava Language Reference Manual [Электронный ресурс]. URL: <http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html> (дата обращения 03.06.2024).
12. GitHub: AqlaSolutions/runsharp [Электронный ресурс]. URL: <https://github.com/AqlaSolutions/runsharp> (дата обращения: 03.06.2024)
13. Microsoft: System.Reflection.Emit Namespace [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/api/system.reflection.emit?view=net-8.0> (дата обращения: 03.06.2024).
14. Хабр: Компилятор языка программирования MiniJava [Электронный ресурс]. URL: <https://habr.com/ru/articles/191766/> (дата обращения: 03.06.2024)
15. Хабр: Разрабатываем компилятор для учебного языка Cool на языке C# под .NET (Часть 1) [Электронный ресурс]. URL: <https://habr.com/ru/articles/136528/> (дата обращения: 03.06.2024)
16. Хабр: Разрабатываем компилятор для учебного языка Cool на языке C# под .NET (Часть 2 + Бонусы) [Электронный ресурс]. URL: <https://habr.com/ru/articles/136714/> (дата обращения: 03.06.2024)

# ПРИЛОЖЕНИЕ А

|  |  |  |
| --- | --- | --- |
| Program | ::= | [MainClass](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod2) ( [ClassDeclaration](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod3) )\* <EOF> |
| MainClass | ::= | "class" [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) "{" "public" "static" "void" "main" "(" "String" "[" "]" [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) ")" "{" [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) "}" "}" |
| ClassDeclaration | ::= | "class" [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) ( "extends" [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) )? "{" ( [VarDeclaration](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod6) )\* ( [MethodDeclaration](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod7) )\* "}" |
| VarDeclaration | ::= | [Type](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod8) [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) ";" |
| MethodDeclaration | ::= | "public" [Type](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod8) [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) "(" ( [Type](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod8) [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) ( "," [Type](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod8) [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) )\* )? ")" "{" ( [VarDeclaration](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod6) )\* ( [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) )\* "return" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ";" "}" |
| Type | ::= | "int" "[" "]" |
|  | | | "boolean" |
|  | | | "int" |
|  | | | [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) |
| Statement | ::= | "{" ( [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) )\* "}" |
|  | | | "if" "(" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ")" [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) "else" [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) |
|  | | | "while" "(" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ")" [Statement](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod5) |
|  | | | "System.out.println" "(" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ")" ";" |
|  | | | [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) "=" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ";" |
|  | | | [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) "[" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) "]" "=" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ";" |
| Expression | ::= | [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ( "&&" | "<" | "+" | "-" | "\*" ) [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) |
|  | | | [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) "[" [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) "]" |
|  | | | [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) "." "length" |
|  | | | [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) "." [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) "(" ( [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) ( "," [Expression](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod9) )\* )? ")" |
|  | | | <INTEGER\_LITERAL> |
|  | | | "true" |
|  | | | "false" |
|  | | | [Identifier](http://www.cs.tufts.edu/~sguyer/classes/comp181-2006/minijava.html#prod4) |
|  | | | "this" |
|  | | | "new" "int" "[" Expression "]" |
|  | | | "new" Identifier "(" ")" |
|  | | | "!" Expression |
|  | | | "(" Expression ")" |
| Identifier | ::= | <IDENTIFIER> |

# ПРИЛОЖЕНИЕ Б

class Program

{

    public static void main(String[] a)

    {

        new BubbleSort().Start(10);

    }

}

class BubbleSort

{

    public int Start(int size)

    {

        int[] array;

        array = this.InitArray(size);

        this.PrintArray(array);

        this.SortArray(array);

        System.out.println(false);

        this.PrintArray(array);

        return 0;

    }

    public int[] InitArray(int size)

    {

        int i;

        int[] array;

        array = new int[size];

        i = 0;

        while (i < size)

        {

            array[i] = System.randomInt(20);

            i = i + 1;

        }

        return array;

    }

    public int PrintArray(int[] array)

    {

        int i;

        i = 0;

        while(i < array.length)

        {

            System.out.println(array[i]);

            i = i + 1;

        }

        return 0;

    }

    public int SortArray(int[] array)

    {

        int i;

        int j;

        int tmp;

        i = 0;

        while (i < array.length - 1)

        {

            j = i + 1;

            while (j < array.length)

            {

                if (array[j] < array[i])

                {

                    tmp = array[j];

                    array[j] = array[i];

                    array[i] = tmp;

                }

                else {}

                j = j + 1;

            }

            i = i + 1;

        }

        return 0;

    }

}

Листинг 1. Исходный код программы для сортировки методом пузырька.

using System;

public class Program

{

  public static void Main(string[] a) => new BubbleSort().Start(10);

  public Program() => this.$$ctor();

  void $$ctor()

  {

  }

}

public class BubbleSort

{

  protected Random random;

  public BubbleSort() => this.$$ctor();

  public int Start(int size)

  {

    int[] array = this.InitArray(size);

    this.PrintArray(array);

    this.SortArray(array);

    Console.WriteLine(false);

    this.PrintArray(array);

    return 0;

  }

  public int[] InitArray(int size)

  {

    int[] numArray = new int[size];

    for (int index = 0; index < size; ++index)

      numArray[index] = this.random.Next(20);

    return numArray;

  }

  public int PrintArray(int[] array)

  {

    for (int index = 0; index < array.Length; ++index)

      Console.WriteLine(array[index]);

    return 0;

  }

  public int SortArray(int[] array)

  {

    for (int index1 = 0; index1 < array.Length - 1; ++index1)

    {

      for (int index2 = index1 + 1; index2 < array.Length; ++index2)

      {

        if (array[index2] < array[index1])

        {

          int num = array[index2];

          array[index2] = array[index1];

          array[index1] = num;

        }

      }

    }

    return 0;

  }

  void $$ctor() => this.random = new Random();

}

Листинг 2. Сгенерированный промежуточный код для программы сортировки метода пузырьком (в формате High-Level C#).

class Program

{

    public static void main(String[] args)

    {

        new ReverseLinkedList().Start(10);

    }

}

class LinkedListNode

{

    int \_data;

    LinkedListNode \_next;

    boolean \_hasNext;

    public int Init(int data)

    {

        \_data = data;

        \_hasNext = false;

        return 0;

    }

    public int GetData()

    {

        return \_data;

    }

    public LinkedListNode GetNext()

    {

        return \_next;

    }

    public boolean HasNext()

    {

        return \_hasNext;

    }

    public int SetNext(LinkedListNode node)

    {

        \_next = node;

        \_hasNext = true;

        return 0;

    }

    public int RemoveNext()

    {

        \_hasNext = false;

        return 0;

    }

}

class LinkedList

{

    LinkedListNode \_root;

    public int Init(int size)

    {

        int i;

        LinkedListNode current;

        LinkedListNode next;

        \_root = new LinkedListNode();

        \_root.Init(System.randomInt(20));

        i = 1;

        current = \_root;

        while (i < size)

        {

            next = new LinkedListNode();

            next.Init(System.randomInt(20));

            current.SetNext(next);

            current = next;

            i = i + 1;

        }

        return 0;

    }

    public int Print()

    {

        LinkedListNode current;

        current = \_root;

        while (current.HasNext())

        {

            System.out.println(current.GetData());

            current = current.GetNext();

        }

        System.out.println(current.GetData());

        return 0;

    }

    public int Reverse()

    {

        LinkedListNode current;

        LinkedListNode prev;

        boolean hasPrev;

        LinkedListNode next;

        current = \_root;

        prev = new LinkedListNode();

        hasPrev = false;

        while (current.HasNext())

        {

            next = current.GetNext();

            if (hasPrev)

                current.SetNext(prev);

            else

                current.RemoveNext();

            prev = current;

            hasPrev = true;

            current = next;

        }

        current.SetNext(prev);

        \_root = current;

        return 0;

    }

}

class ReverseLinkedList

{

    public int Start(int size)

    {

        LinkedList list;

        list = new LinkedList();

        list.Init(size);

        list.Print();

        list.Reverse();

        System.out.println(false);

        list.Print();

        return 0;

    }

}

Листинг 3. Исходный код программы для инвертирования односвязного списка.

using System;

public class Program

{

  public static void Main(string[] args) => new ReverseLinkedList().Start(10);

  public Program() => this.$$ctor();

  void $$ctor()

  {

  }

}

public class LinkedListNode

{

  protected int \_data;

  protected LinkedListNode \_next;

  protected bool \_hasNext;

  public int Init(int data)

  {

    this.\_data = data;

    this.\_hasNext = false;

    return 0;

  }

  public int GetData() => this.\_data;

  public LinkedListNode GetNext() => this.\_next;

  public bool HasNext() => this.\_hasNext;

  public int SetNext(LinkedListNode node)

  {

    this.\_next = node;

    this.\_hasNext = true;

    return 0;

  }

  public int RemoveNext()

  {

    this.\_hasNext = false;

    return 0;

  }

  public LinkedListNode() => this.$$ctor();

  void $$ctor()

  {

  }

}

public class LinkedList

{

  protected LinkedListNode \_root;

  protected Random random;

  public int Init(int size)

  {

    this.\_root = new LinkedListNode();

    this.\_root.Init(this.random.Next(20));

    int num = 1;

    LinkedListNode linkedListNode = this.\_root;

    for (; num < size; ++num)

    {

      LinkedListNode node = new LinkedListNode();

      node.Init(this.random.Next(20));

      linkedListNode.SetNext(node);

      linkedListNode = node;

    }

    return 0;

  }

  public int Print()

  {

    LinkedListNode linkedListNode;

    for (linkedListNode = this.\_root; linkedListNode.HasNext(); linkedListNode = linkedListNode.GetNext())

      Console.WriteLine(linkedListNode.GetData());

    Console.WriteLine(linkedListNode.GetData());

    return 0;

  }

  public int Reverse()

  {

    LinkedListNode linkedListNode = this.\_root;

    LinkedListNode node = new LinkedListNode();

    bool flag = false;

    LinkedListNode next;

    for (; linkedListNode.HasNext(); linkedListNode = next)

    {

      next = linkedListNode.GetNext();

      if (flag)

        linkedListNode.SetNext(node);

      else

        linkedListNode.RemoveNext();

      node = linkedListNode;

      flag = true;

    }

    linkedListNode.SetNext(node);

    this.\_root = linkedListNode;

    return 0;

  }

  void $$ctor() => this.random = new Random();

  public LinkedList() => this.$$ctor();

}

public class ReverseLinkedList

{

  public ReverseLinkedList() => this.$$ctor();

  public int Start(int size)

  {

    LinkedList linkedList = new LinkedList();

    linkedList.Init(size);

    linkedList.Print();

    linkedList.Reverse();

    Console.WriteLine(false);

    linkedList.Print();

    return 0;

  }

  void $$ctor()

  {

  }

}

Листинг 4. Сгенерированный промежуточный код для программы инвертирования односвязного списка (в формате High-Level C#).

class Program

{

    public static void main(String[] args)

    {

        new Inheritance().Start();

    }

}

class A

{

    int value;

    public int SetValue(int number)

    {

        value = number;

        return 0;

    }

    public int DoNothing()

    {

        return 55;

    }

    public int DoSomething(int number)

    {

        return number \* number;

    }

}

class B extends A

{

    public int DoSomething(int number)

    {

        return number \* value;

    }

}

class Inheritance

{

    public int Start()

    {

        A a;

        B b;

        A ab;

        a = new A();

        b = new B();

        ab = b;

        b.SetValue(3);

        System.out.println(a.DoSomething(5));

        System.out.println(b.DoSomething(5));

        System.out.println(ab.DoSomething(5));

        System.out.println(a.DoNothing());

        System.out.println(b.DoNothing());

        System.out.println(ab.DoNothing());

        return 0;

    }

}

Листинг 5. Исходный код программы для демонстрации наследования классов.

using System;

public class Program

{

  public static void Main(string[] args) => new Inheritance().Start();

  public Program() => this.$$ctor();

  void $$ctor()

  {

  }

}

public class A

{

  protected int value;

  public int SetValue(int number)

  {

    this.value = number;

    return 0;

  }

  public int DoNothing() => 55;

  public int DoSomething(int number) => number \* number;

  public A() => this.$$ctor();

  void $$ctor()

  {

  }

}

public class B : A

{

  public new int DoSomething(int number) => number \* this.value;

  public B() => this.$$ctor();

  new void $$ctor()

  {

  }

}

public class Inheritance

{

  public Inheritance() => this.$$ctor();

  public int Start()

  {

    A a1 = new A();

    B b = new B();

    A a2 = (A) b;

    b.SetValue(3);

    Console.WriteLine(a1.DoSomething(5));

    Console.WriteLine(b.DoSomething(5));

    Console.WriteLine(a2.DoSomething(5));

    Console.WriteLine(a1.DoNothing());

    Console.WriteLine(b.DoNothing());

    Console.WriteLine(a2.DoNothing());

    return 0;

  }

  void $$ctor()

  {

  }

}

Листинг 6. Сгенерированный промежуточный код для программы демонстрации наследования (в формате High-Level C#).