Министерство образования Российской Федерации

Тверской государственный технический университет

Кафедра Программного обеспечения

Курсовая работа на тему:

Разработка транслятора из языка программирования в команды Assembly

Выполнил: Бакеев А.В.

Проверил: Калабин А.Л.

Тверь, 2024

Оглавление

[Введение 3](#_Toc184248180)

[Постановка задачи 4](#_Toc184248181)

[Теоретическая часть 5](#_Toc184248182)

[Входные и выходные данные 6](#_Toc184248183)

[Разработка 7](#_Toc184248184)

[Тестирование 10](#_Toc184248185)

[Выводы 12](#_Toc184248186)

[Список литературы 13](#_Toc184248187)

# Введение

В современном мире программирования трансляторы играют ключевую роль в разработке программного обеспечения, обеспечивая перевод кода, написанного на одном языке, в код на другом языке или в машинный код, исполняемый компьютером. Написание транслятора — это сложная, но увлекательная задача, требующая глубоких знаний как теории компиляции, так и практических навыков программирования. В данной курсовой работе будет рассматриваться процесс создания транслятора на языке программирования C#, который сочетает в себе мощные инструменты и удобные средства разработки.

C# — это высокоуровневый язык программирования, который используется для создания различных приложений, включая веб-приложения, мобильные приложения и игры. Его гибкость, объектно-ориентированный подход и хорошая поддержка разработчиков делают C# идеальным выбором для реализации транслятора. В данной работе будет подробно описан процесс проектирования и реализации транслятора, включая анализ исходного языка, разработку синтаксического анализатора и генерацию целевого кода.

Целью курсовой работы является не только написание работающего транслятора, но и более глубокое понимание принципов работы языков программирования и компиляторов. Кроме того, работа позволит на практике применить теоретические знания, полученные в ходе изучения курсов по программированию и компиляции. В процессе реализации транслятора будут рассмотрены актуальные проблемы и вызовы, с которыми сталкиваются разработчики, а также предложены возможные пути их решения.

# Постановка задачи

Спроектировать и разработать транслятор на языке программирования C# по указанному варианту.

Проанализировать теоретические подходы к разработке транслятора на языке программирования С#.

**Вариант 2**

<Программа> ::= <Объявление переменных> <Описание вычислений> <Оператор печати>

<Описание вычислений> ::= Begin <Список присваиваний> End

<Объявление переменных> ::= Int <Список переменных> ;

<Список переменных> ::= <Идент> | <Идент> , <Список переменных> <Список присваиваний>::= <Присваивание> |

<Присваивание> <Список присваиваний>

<Присваивание> ::= <Идент> := <Выражение> ;

<Выражение> ::= <Ун.оп.> <Подвыражение> | <Подвыражение> <Подвыражение> :: = ( <Выражение> ) | <Операнд> |

< Подвыражение > <Бин.оп.> <Подвыражение>

<Ун.оп.> ::= "-"

<Бин.оп.> ::= "-" | "+" | "\*" | "/"

<Операнд> ::= <Идент> | <Const>

<Идент> ::= <Буква> <Идент> | <Буква>

<Const> ::= <Цифра> <Const> | <Цифра>

<Оператор печати>::=Print <Идент>;

На одной строке может быть только объявление переменных или один оператор присваивания[2]

# Теоретическая часть

C# — это современный, объектно-ориентированный язык программирования, который я использовал при написании данной программы. Это язык разработанный Microsoft и впервые представленный в начале 2000-х годов как часть платформы .NET. Он характеризуется статической типизацией, что способствует раннему выявлению ошибок на этапе компиляции, и простым, понятным синтаксисом, удобным для изучения. C# поддерживает принципы объектно-ориентированного программирования и предлагает богатую экосистему библиотек и фреймворков, таких как ASP.NET.

Visual Studio — это интегрированная среда разработки (IDE), разработанная компанией Microsoft, предназначенная для создания программного обеспечения на различных языках программирования, включая C#, VB.NET, C++, Python и другие. Она была использована как основной инструмент написания на программы.

Ассемблер — это низкоуровневый язык программирования, который напрямую соответствует машинному коду определённой архитектуры процессора. Он предоставляет программистам возможность писать инструкции с использованием понятных человеку символов вместо двоичных кодов, что облегчает работу с аппаратным обеспечением и оптимизацию производительности. В нашем приложении в него будет преобразовываться язык описанный во варианте.

Транслятор — это программа, которая преобразует код, написанный на одном языке программирования (обычно высокоуровневом), в код на другом языке, чаще всего в машинный код, исполняемый процессором. В данном случаи, будет преобразовывать в ассембли код.

# Входные и выходные данные

На входе программа получает текстовый файл и мы открываем его с приложения, обладающего GUI. Затем происходит считывание файла в специальное поле, где мы будем видеть исходный код, который можно редактировать. В этом же окошке можно будет запустить код на компиляцию, и, если все было верно – код на нашем языке программирования превратится в assembler код. Затем можно будет нажать еще одну кнопку, которая откроет программу DOSBox, где этот код запустится. В случаи, если что-то пойдет не так, внизу будет поле для вывода информации об ошибках. Ниже продемонстрированы скриншоты формы.

На рисунке 1 общий вид. Левая верхняя часть – входные данные. Правая верхняя часть – ассембли код. Нижняя часть – поле вывода информации об ошибках. Верхний menu-bar с кнопками Файл и Компиляция.

На рисунке 2 вид открывающегося под-окна при нажатии на кнопку «Файл».

На рисунке 3 вид открывающегося под-окна при нажатии на кнопку «Компиляция»

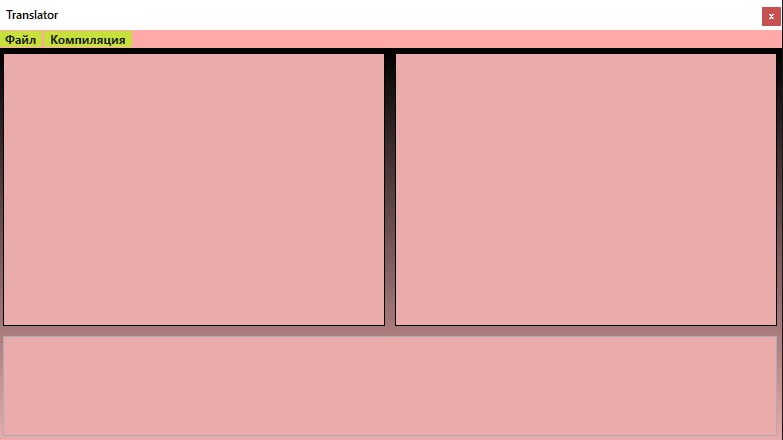


Рисунок 1. Общая форма

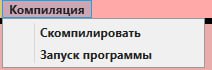
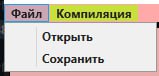


Рисунок 2. Меню работы с файлом Рисунок 3. Меню работы с компилятором и DOSBox

# Разработка

Разработка программной части проекта включает в себя несколько классов проекта – (Генератор кода, Лексемы, Лексический анализатор, Таблицу имен, Ридер, Синтаксический анализатор); Форму транслятора (GUI); Тестовая часть – проверяет корректность выбранных параметров при запуске формы (по сути – список тестов привязанных к форме, проверяющих корректность данных).

Проанализировав подходы к разработке транслятора, описанные в предоставленной методичке с лабораторными работами, я пришел к выводу, что в структуре приложения должен использоваться язык программирования C#, библиотека классов (которая будет описана ниже), класс формы и класс тестирования (привязанный) прямо к форме. При разработке и проектировании транслятора были создали следующие классы:

1. **CodeGenerator**.cs – класс отвечающий за генерацию ассембли кода. Здесь добавляются инструкции, метки, объявляются сегменты данных и прочие необходимые вещи.[3]
2. **Identifier**.cs – хранит информация о типах данных, какие допустимы в нашем языке и как они должны называться. Так же здесь перечисляются категории модификаторов – const, var, type.[3]
3. **Keyword**.cs – структура, отвечающая за доступ к словам и лексемам.[3]
4. **Lexems**.cs – лексемы, а точнее множество (enum), содержащее в себе все разрешенные в языке лексемы: Ключевые слова, арифметические операторы, операторы сравнения, специальные символы, служебные лексемы. А так же более сложные структуры – отсутствие лексемы (None), идентификатор (Name), число (Number), тип данных (Type – включает в себя все описанные в **Identifier**.cs типы данных).[3]
5. **LexicalAnalyzer**.cs - лексический анализатор, проверяет ключевые слова и сопоставляет их с нужными лексемами, парсит идентификаторы, числа, строки и символы. В соответствии с полученной информацией подбирает нужные лексемы.[3]
6. **NameTable.**cs – класс занимается хранением и управлением таблицей идентификаторов – добавляет новые в таблицу, ищет нужные среди уже имеющихся, получет все содержащиеся, по необходимости.[3]
7. **Reader**.cs – класс который отвечает за первичную обработку входного файла и считывания данных из него. Проверяет есть ли такой файл вообще и является ли он корректным.[3]
8. **SyntaxAnalyzer**.cs. – работает с синтаксической частью. По сути разбирает всевозможные входные операторы: присваивания, вывода на печать, арифметических операций, разбирает операции сложения, вычитания, деления, умножения, циклов, ветвлений, объявления переменных и т.д. Для примера приведен код разбора операторов if-then-else[3]:

/// <summary>

/// Разбирает оператор if и соответствующие ветки then и else.

/// </summary>

private void ParseIfStatement()

{

CheckLexem(Lexems.If);

// Генерация меток

string lowerLabel = CodeGenerator.GenerateLabel();

currentLabel = lowerLabel;

string exitLabel = CodeGenerator.GenerateLabel();

CheckLexem(Lexems.LeftPar);

ParseExpression();

CheckLexem(Lexems.RightPar);

CheckLexem(Lexems.Then);

ParseStatementSequence();

CodeGenerator.AddInstruction($"jmp {exitLabel}");

CodeGenerator.AddLabel(lowerLabel);

while (\_lexicalAnalyzer.CurrentLexem == Lexems.ElseIf)

{

\_lexicalAnalyzer.ParseNextLexem();

lowerLabel = CodeGenerator.GenerateLabel();

currentLabel = lowerLabel;

CheckLexem(Lexems.LeftPar);

ParseExpression();

CheckLexem(Lexems.RightPar);

CheckLexem(Lexems.Then);

ParseStatementSequence();

CodeGenerator.AddInstruction($"jmp {exitLabel}");

CodeGenerator.AddLabel(lowerLabel);

}

if (\_lexicalAnalyzer.CurrentLexem == Lexems.Else)

{

\_lexicalAnalyzer.ParseNextLexem();

ParseStatementSequence();

}

CheckLexem(Lexems.EndIf);

CodeGenerator.AddLabel(exitLabel);

}

# 

# Тестирование

**Кейс 1. Цикл while**

Входные данные:

Int a,b;

Bool x;

Begin

a = 1;

b = 4;

while (b>a)

a = a + 2;

b = b + 1;

Endwhile;

Print b;

if (x) then

Print a;

Endif

End

Выходные данные:



**Кейс 2. Sum 2 переменных**

Входные данные:

Int a,b,c;

Begin

a = 3;

b = 4;

c = a+b;

Print c;

End

Выходные данные:



**Кейс 3. Bool переменные**

Входные данные:

Int a,b;

Bool c;

Begin

c = True;

If (c) then

a = 20;

b = 130-a;

Endif;

print b;

End

Выходные данные:



**Кейс 4. Все вместе, операторы \* / + -, цикл, вывод, bool переменная**

Входные данные:

Int a,b;

Bool x;

Begin

a = 1;

b = 4;

while (b>a)

a = a \* 2;

b = b - 1;

Endwhile;

a = a/2;

b = b + 1;

Print b;

if (x) then

Print a;

Endif

End

Выходные данные:



# Выводы

В ходе выполнения курсовой работы по созданию транслятора на языке программирования C# было достигнуто множество поставленных целей и задач, а именно:

* На основании проведенного анализа методички и выполнения лабораторных работ был сформирован «скелет» транслятора, способный обрабатывать базовые операции и несколько типов данных.
* По сформированному скелету спроектировано и написано приложение по варианту, которое и является данной курсовой работой. Итоговое приложение корректно выполняет свою работу и проходит тестирование на предложенных преподавателем тест-кейсах.

Работа над проектом позволила глубже понять принципы функционирования языков программирования, а также выявить сложности и особенности, связанные с процессом компиляции. Мы также познакомились с актуальными инструментами, которые облегчают создание трансляторов, и оценили их эффективность в контексте решаемых задач.[1]

В процессе разработки транслятора были выявлены ряд проблем, связанных с обработкой различных конструкций языка и оптимизацией генерируемого кода. Это подтверждает, что создание эффективного компилятора — это не только техническая задача, но и область, требующая креативного подхода и серьезного анализа.

Таким образом - курсовая работа стала важным этапом в нашем обучении и подготовила нас к будущей профессиональной деятельности в области разработки программного обеспечения. Мы получили ценный опыт, который, без сомнения, будет полезен для дальнейшего освоения более сложных аспектов программирования и работы с языками низкого уровня.

# Список литературы

1. Ткаченко, В. Н. Трансляторы [Электронный ресурс] / В. Н. Ткаченко. — Режим доступа: <https://book.kbsu.ru/theory/chapter6/1_6_10.html> (дата обращения: 03.12.2024).
2. Рубцов, В. А. Compiler translation to assembly [Электронный ресурс]. // Stack Overflow. – 2010. – Режим доступа: <https://stackoverflow.com/questions/2612715/compiler-translation-to-assembly> (дата обращения: 04.12.2024).
3. Лабораторная работа по теории автоматов и формальным языкам [Электронный ресурс] / [б. а.]. — [место издания не указано], 2023. — Режим доступа: <https://cloud.mail.ru/public/KHBN/yd8aNW686/Lab_TLPMT_23_1.doc> (дата обращения: 03.12.2024).