Министерство науки и Высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Курский государственный университет»

Факультет физики, математики, информатики

Кафедра программного обеспечения и администрирования информационных систем

КурсовАЯ РАБОТА

по дисциплине

Теория формальных языков и трансляций

на тему: РАЗРАБОТКА КОМПИЛЯТОРА МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Вариант 16

Обучающегося 3 курса

очной формы обучения

направления подготовки

02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Проектирование информационных систем и баз данных

Козявина Максима Сергеевича

Руководитель:

доцент кафедры АГиТОМ

Селиванова Ирина Васильевна

Допустить к защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_г.

Курск, 2023

Содержание

[Введение 3](#_Toc145498638)

[1 Задание 5](#_Toc145498639)

[2 Лексический анализ 7](#_Toc145498640)

[3 Синтаксический анализ 8](#_Toc145498641)

[4 Тестирование проекта 9](#_Toc145498642)

[Заключение 10](#_Toc145498643)

[Список использованных источников 11](#_Toc145498644)

[Приложение А 12](#_Toc145498645)

[Текст программы 12](#_Toc145498646)

[Приложение Б 13](#_Toc145498647)

[Внешний вид графического материала 13](#_Toc145498648)

# Введение

В наши дни, несмотря на огромное количество разработанных языков программирования и соответствующих им компиляторов, процесс создания новых приложений в этой области не замедляется. Это обусловлено как прогрессом в технологии производства компьютерных систем, так и увеличивающимися требованиями к решению сложных задач. Существуют различные причины для разработки нового компилятора, такие как функциональные ограничения, отсутствие локализации или низкая эффективность существующих решений. Именно поэтому основы теории языков и формальных грамматик, а также практические методы разработки компиляторов играют особую роль в инженерном образовании в области информатики и вычислительной техники.

Создание собственного компилятора способствует не только пониманию основ языков программирования, но и дает возможность разработать специфичный язык программирования. Это поможет лучше понять основы информатики и алгоритмизации, а также применить формальную логику, теорию языков программирования и оптимизацию кода. Разработка компилятора также позволит изучить особенности работы с низкоуровневым кодом, углубить знания по работе с памятью и регистрами процессора.

Цель данной курсовой работы включает: получение навыков разработки программы компилятора на основе полученных теоретических знаний в области формальных языков и трансляций, позволяющего получить ассемблерный код и имеющего понятный пользовательский интерфейс, позволяющий визуализировать результаты работы всех этапов компиляции.

В рамках этой цели можно выделить следующие задачи:

1. изучение особенностей разработки грамматик модельных языков программирования;
2. разработка алгоритмов и методов лексического, синтаксического и семантического анализов их программной реализации;
3. визуализация этапов выполнения лексического, синтаксического и семантического анализов при практическом применении предложенного языка программирования;
4. создание алгоритма для трансляции кода модельного языка на язык низкого уровня.

Объектом исследования выступают языки программирования.

Предметом исследования в курсовой работе выступают алгоритмы и методы разработки компиляторов языков программирования и их трансляции.

Работа выполнена на основе следующих структурных особенностей, указанных в варианте №16 задания к курсовой работе:

Выполняется задание согласно варианту 16:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№**  **Варианта** | Задания | | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 16 | 3 | 2 | 4 | 1 | 2 | 1 | 4 | 1 | 3 | 1 | 3 | 2 |

1. Структура программы

<программа> ::= {/ (<описание> | <оператор>) ; /}

2. Синтаксис команд описания данных

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

3. Синтаксис идентификаторов

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

4. Описание типов

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

5. Синтаксис составного оператора

<составной> ::= «{» <оператор> { ; <оператор> } «}»

6. Оператор присваивания

<присваивание> ::= <идентификатор> ass <выражение>

7. Оператор условного перехода

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

8. Синтаксис оператора

<фиксированного\_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

9. Синтаксис оператора

<условного\_цикла> ::= do while <выражение> «{» <оператор> “}”

10. Синтаксис оператора

<ввода> ::= read (<идентификатор> {, <идентификатор> })

11. Синтаксис оператора

<вывода>::= output (<выражение> { пробел <выражение> })

12. Признак начала комментария Признак конца комментария

/\* \*/

Аннотация: в данной пояснительной записке содержится информация о разработке программы компилятора модельного языка программирования. Приложение включает в себя множество функций, таких как: лексический, синтаксический, семантический анализ. Результатом работы приложения является поиск ошибок в написанном коде и перевод правильного кода в низкоуровневый язык.

Ключевые слова: компилятор, язык программирования, лексический анализ, синтаксический анализ, семантический анализ.

Abstract: this explanatory note contains information about the development of a model programming language compiler program. The application includes many functions, such as: lexical, syntactic, semantic analysis. The result of the application is to find errors in the written code and translate the correct code into a low-level language.

Key words: compiler, programming language, lexical, syntactic, semantic analysis.

# **1 Разработка грамматики модельного языка программирования**

**1.1 Форма Бэкуса-Наура**

<программа> ::= {/ (<описание> | <оператор>) ; /}

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

<составной> ::= «{» <оператор> { ; <оператор> } «}»

<присваивание> ::= <идентификатор> ass <выражение>

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

<фиксированного\_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

<условного\_цикла> ::= do while <выражение> «{» <оператор> «}»

<ввода> ::= read (<идентификатор> {, <идентификатор> })

<вывода>::= output (<выражение> { пробел <выражение> })

<выражение> ::= <число> | <идентификатор> | not (<идентификатор>, <выражение>, <булево значение>) | - (<идентификатор>, <выражение>, <число>) | (<идентификатор>, <число>) <знак> (<идентификатор>, <число>)

<число> ::= <непустая последовательность цифр> | <непустая последовательность цифр>.<непустая последовательность цифр>

<булево значение> ::= true | false

<последовательность цифр> ::= {<цифра>}

<непустая последовательность цифр> ::= {/<цифра>/}

<знак> ::= + | - | \* | / | > | < | <= | >= | = | and | or

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<начало\_комментария>::=</\*>

<конец\_комментария>::=<\*/>

**1.2 Формальная грамматика**

Грамматика представляет собой набор {T, N, P, J}, где T – множество терминальных символов.

T = { ;, +, -, \*, /, >, >=, <, <=, =, !=, , «,», (, ), {, }, if, else, elseif, for, to, do, while, ass, dim, read, output, and, or, not, ID, TYPE, BOOL, NUM }

N – множество нетерминальных символов.

N = { PROG, CMD, IDS, I, EXPR, ASSIGN, FORCOND, COND1, COND, CALC, ANY }

Программа

PROG : CMD

Команды  
CMD : EXPR ; CMD | EXPR ; | ASSIGN ; CMD | ASSIGN ;

Список идентификаторов  
IDS : IDS , ID | I | I , ID  
I : ID

Выражения  
EXPR : dim IDS TYPE | read ( IDS ) | output ( IDS ) | if ( COND ) { CMD } | if ( COND ) { CMD } else { CMD } | if ( COND ) { CMD } COND1 | do while ( COND ) { CMD } | for ( FORCOND ) { CMD }

Присвоение  
ASSIGN : ID ass CALC | ID ass ANY | ID ass I | ID ass COND

Условие фиксированного цикла  
FORCOND : ASSIGN to ANY | ASSIGN to I

Условия конструкций elseif, else  
COND1 : elseif ( COND ) { CMD } else { CMD } | elseif ( COND ) { CMD } COND1 | elseif ( COND ) { CMD }

Условие

COND : ANY = ANY | ANY != ANY | ANY < ANY | ANY <= ANY | ANY > ANY | ANY >= ANY | ANY and ANY | ANY or ANY | not ANY | not ID | ANY = ID | ANY != ID | ANY < ID | ANY <= ID | ANY > ID | ANY >= ID | ANY and ID | ANY or ID | ID = ANY | ID != ANY | ID < ANY | ID <= ANY | ID > ANY | ID >= ANY | ID and ANY | ID or ANY | ID = ID | ID != ID | ID < ID | ID <= ID | ID > ID | ID >= ID | ID and ID | ID or ID

Вычисление значения  
CALC : ANY + ANY | ANY - ANY | ANY \* ANY | ANY / ANY | ANY + ID | ANY - ID | ANY \* ID | ANY / ID | ID + ANY | ID - ANY | ID \* ANY | ID / ANY | ID + ID | ID - ID | ID \* ID | ID / ID

Константа числового или булевого типа  
ANY : NUM | BOOL

# 2 Лексический анализ

**2.1 Алгоритмы**

Алгоритмы лексического анализа могут основываться на разделении на лексемы с помощью символов разделителей, чтением исходного кода посимвольно или с использованием регулярных выражений. Комментарии могут быть удалены либо же проигнорированы непосредственно при лексическом анализе.

В данном алгоритме используется посимвольное чтение исходного текста и последующая проверка символа или их комбинации на наличие соответствующей лексемы языка. При несоответствии выдаётся ошибка. Во время лексического анализа комментарии игнорируются.

**2.2 Ошибки вывода**

При лексическом анализе может быть выявлено 2 типа ошибок:

Первый тип – неверный идентификатор. Ошибка выдаётся если встречено слово, не являющееся ключевым словом, описанным в словаре WORDS класса Lexer и не соответствующее правилам наименования идентификаторов языка. В этом случае будет выведена ошибка Lexer error: Unknown identifier "[имя идентификатора]" in line [номер строки].

Второй тип – непредвиденный символ. Ошибка выдаётся если встречен символ, не являющийся одним из символов-лексем языка, описанных в словаре SYMBOLS класса Lexer. В этом случае будет выведена ошибка Lexer error: Unexpected symbol "[символ]" in line [номер строки].

**2.3 Хэш-таблица**

Для разбиения исходного кода на лексемы используются встроенные в язык Python хэш-таблицы, именуемые словарями. В множествах и словарях языка Python используется метод открытой адресации. Он заключается в том, что в ячейки таблицы помещаются не указатели на списки, а сами пары ключ-значение. Значение зашифровывается хэш-функцией и при возникновении коллизии пара ключ-значение записываются в следующую пустую ячейку после той, которая получилась в результате работы хэш-функции. При извлечении данных из хэш-таблицы данные дополнительно сверяются с ключом, который так же хранится в хэш-таблице.

Хеш-функцией выступает метод \_\_hash\_\_ определённый в каждом хешируемом объекте языка Python. Эта функция может быть описана для любого типа данных, а для стандартных описана по стандарту. Для простых типов данных, например, int – результатом может быть само число, а для сложных может находиться комбинация хешей для составных частей и генератора случайных чисел, который гарантирует одинаковые значения для одного типа данных в рамках одного запуска программы.

**2.4 Реализация лексического анализа**

Лексический анализ реализует модуль lexer и, в частности, класс Lexer, содержащий хэш-таблицы символов и ключевых слов – SYMBOLS и WORDS. Входными данными для класса является текстовый поток input\_stream из файла текста программы. Метод getc() получает следующий символ из потока input\_stream. Метод set\_error() выставляет сообщение об ошибке. Метод next\_token() считывает следующую лексему посимвольно, определяет тип лексемы, игнорирует комментарии и вызывает метод set\_error() при возникновении ошибки. В результате в поле symbol выводится тип лексемы и в поле value значение для целочисленных, дробных и булевых литералов или имя идентификатора.

**2.5 Тестирование**

# 3 Синтаксический анализ

**3.1 Алгоритмы**

Существует несколько алгоритмов синтаксического анализа:

Алгоритм рекурсивного спуска – алгоритм нисходящего синтаксического анализа, реализуемый путём взаимного вызова процедур парсинга, где каждая процедура соответствует одному из правил контекстно-свободной грамматики или БНФ. Применения правил последовательно, слева-направо поглощают токены, полученные от лексического анализатора. Это один из самых простых алгоритмов парсинга, который является также малоэффективным т.к. может обрабатывать большое количество текста недостаточно быстро и может обрабатывать не все грамматики.

Алгоритм сдвиг-свёртки используется в данном алгоритме. Он используется для грамматики операторного предшествования. Для моделирования его работы необходима входная цепочка символов и стек символов, в котором автомат может обращаться не только к самому верхнему символу, но и к некоторой цепочке символов на вершине стека. Также необходимо построить матрицу операторного предшествования.

Этот алгоритм для заданной КС-грамматики можно описать следующим образом:

1. Поместить в верхушку стека символ «начало строки», считывающую головку МП-автомата поместить в начало входной цепочки. В конец входной цепочки надо дописать символ «конец строки».

2. В стеке ищется самый верхний терминальный символ sj при этом сам символ sj остается в стеке. Из входной цепочки берется текущий символ ai (справа от считывающей головки МП-автомата).

3. Если символ sj – это символ начала строки, а символ ai – символ конца строки, то алгоритм завершен, входная цепочка символов разобрана.

4. В матрице предшествования ищется клетка на пересечении строки, помеченной символом sj, и столбца, помеченного символом ai.

5. Если клетка, пустая, то значит, входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке.

6. Если клетка, содержит символ “=.” или “<.” то необходимо выполнить перенос. При выполнении переноса текущий входной символ ai помещается на верхушку стека, считывающая головка сдвигается на одну позицию вправо. После этого надо вернуться к шагу 2.

7. Если клетка, содержит символ “.>”, то необходимо произвести свертку. Для выполнения свертки из стека выбираются все терминальные символы, связанные отношением “=.”, начиная от вершины стека, а также все нетерминальные символы, лежащие в стеке рядом с ними. Эти символы вынимаются из стека и собираются в цепочку.

8. Во всем множестве правил грамматики ищется правило, у которого правая часть совпадает с цепочкой. Если правило найдено, то в стек помещается нетерминальный символ из левой части правила, иначе, если правило не найдено, это значит, что входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке. После выполнения свертки необходимо вернуться к шагу 2.

В данном алгоритме матрица операторного предшествования строится автоматически из правил грамматики перед выполнением алгоритма сдвиг-свёртки.

**3.2 Ошибки вывода**

При построении матрицы операторного предшествования может быть выведены ошибки:

1. при встрече неизвестного символа - unknown symbol "[символ]";
2. при обнаружении неверно построенного файла правил вывода - wrong file formatting;
3. при конфликте в правилах вывода с указанием ошибки;

При выполнении алгоритма сдвиг-свёртки:

1. при отсутствии правила в исходных правилах вывода - Unable to locate rule [правило];
2. при отсутствии связи в таблице операторного предшествования - unknown construction [символ 1] [символ 2] in [фрагмент кода].

**3.3 Реализация синтаксического анализа**

Для построения матрицы операторного предшествования в класс Matrix передаётся текстовый поток input\_stream из текстового файла, в котором описаны все терминальные символы, все имена правил и перечислены все правила вывода причём имя и само правила разделяется символом «:». Все терминальные символы должны быть разделены пробелом. Метод print\_matrix() позволяет вывести результат на экран а Метод generate() генерирует матрицу операторного предшествования. Строятся множества крайних левых и крайних правых символов для всех символов и только для терминальных символов. После по этим множествам составляется матрица операторного предшествования.

Эта матрица передаётся для выполнения алгоритма сдвиг-свёртки в класс Parser. Перед выполнением сдвиг-свёртки все нетерминалы в правилах вывода заменяются на один нетерминал «Е». При вызове метода check выполняется проверка лексем на корректность по матрице операторного предшествования. Метод get\_t возвращает крайний терминал стека, а при передаче аргумента get\_t(n) может вернуть n-ый символ с конца стека. Для вывода ошибок используется метод set\_error.

**3.4 Тестирование**

# 4 Семантический анализ

**4.1 Алгоритмы**

Для семантического анализа используется простой алгоритм поиска в лексемах определённых закономерностей и проверка правильности типов и определения переменных в этих закономерностях.

**4.2 Ошибки вывода**

При попытке обратится к переменной, которая не была определена до этого будет выведена ошибка Undeclared variable [имя переменной].

При попытке присвоить переменной литерал несоответствующего типа или значение переменной несоответствующего типа будет выведена ошибка Wrong type: [имя переменной] is [тип переменной] unable to assign [тип, который пытался присвоить пользователь]

**3.3 Реализация семантического анализа**

Метод check проверяет входную цепочку лексем на соответствие типов и ведёт учёт переменных, определённых в исходном коде. Считывание происходит поэлементно. Алгоритм ищет цепочку вида dim [имена переменных] [тип] для занесения переменной в список определённых, цепочки вида [переменная] ass [переменная], [переменная] ass [литерал] или [переменная] ass [переменная или литерал] [операция] [переменная или литерал] для контроля соответствия типов. Любые другие вхождения переменных будут сопоставляться со списком определённых в программе переменных. В классе определены поля-массивы операций RETURN\_BOOL и RETURN\_NUM, которые определяют какой тип возвращают соответствующие операции. Для вывода ошибок используется метод set\_error.

# 4 Тестирование проекта

Проверка функционирования приложения в нормальных условиях при обычных значениях входных данных представлена на рисунках 14 –17.

# Заключение

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме проекта: грамотное и рациональное распределение задач, документация по среде разработки MS Visual Studio 2019, документация по языку C++; был разработан интерфейс приложения, разработаны структуры данных, подобраны алгоритмы, удобные для работы с выбранной структурой данных. Так же были изучены библиотеки языка, работающие с потоками файлов, вводом и выводом информации.

Программа может быть использована пользователем, для записи, хранения и работы с текстовой информацией.

Усовершенствовать данное приложение можно с помощью улучшения интерфейса, добавления меню, панели инструментов, контекстного меню, справочной системы, всплывающих подсказок.

# Список источников

1. Страуструп, Бьярне. С83 Программирование: принципы и практика с использованием С++, 2-е изд. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 20 1 6. - 1 328 с. : ил. - Парал. тит. англ.
2. Справка по C++ – Pointer declaration [Сайт]. URL: https://en.cppreference.com/w/cpp/language/pointer (дата обращения 14.03.2023)
3. Курносов М.Г. К93 Введение в структуры и алгоритмы обработки данных. – Новосибирск: Автограф, 2015. – 179 с.
4. Селиванова, И. А. Построение и анализ алгоритмов обработки данных: учеб.-метод. пособие / И. А. Селиванова, В. А. Блинов. — Екатеринбург : Изд-во Урал. ун-та, 2015. — 108 с.
5. Вендров А.М. В29 Проектирование программного обеспечения экономических информационных систем: Учебник. - 2-е изд., перераб. и доп. — М.: Финансы и статистика, 2006. — 544 с: ил

# Приложение А

# Текст программы

# Приложение Б

# Внешний вид графического материала

На рисунке Б.1 представлена диаграмма классов.

На рисунке Б.2 представлена диаграмма компонентов.

Рисунок Б.1 – Диаграмма классов

Рисунок Б.2 – Диаграмма компонентов