Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский государственный университет им. А.Н. Косыгина» (Технологии. Дизайн. Искусство)

Институт Мехатроники и информационных технологий

Кафедра Информационных технологий

КУРСОВАЯ РАБОТА

БАКАЛАВРА

по дисциплине: «Лингвистическое обеспечение САПР»

на тему: «Разработка компилятора»

Направление подготовки: 09.03.01 Инфоpматика и вычислительная техника

Профиль: Системы автоматизированного проектирования

Выполнила Шишерина М.А.,

студентка группы МВС-117 3 курса очной формы обучения

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись фамилия, имя, отчество*

\_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020г.

*оценка дата сдачи*

Преподаватель \_\_\_\_\_\_\_\_\_\_ \_ /ст.пр. А.М. Козлов/\_\_\_\_\_

*подпись ученая степень, ученое звание, фамилия, инициалы*

Москва 2020

ОГЛАВЛЕНИЕ

[Введение 4](#_Toc43650150)

[1. Теория построения компиляторов 5](#_Toc43650151)

[1.1 Основные понятия и определения 5](#_Toc43650152)

[1.2 Этапы процесса компиляции 5](#_Toc43650153)

[1.3 Формальные грамматики 7](#_Toc43650154)

[1.3.1 Форма Бэкуса-Наура (БНФ) 9](#_Toc43650155)

[1.4 Грамматический разбор 10](#_Toc43650156)

[1.4.1 Левосторонний восходящий грамматический разбор 11](#_Toc43650157)

[1.4.2 Левосторонний нисходящий грамматический разбор 11](#_Toc43650158)

[1.5 Регулярные грамматики 12](#_Toc43650159)

[1.5.1 Конечные автоматы 12](#_Toc43650160)

[1.5.2 Лексические анализаторы 12](#_Toc43650161)

[1.6 Контекстно-свободные грамматики 13](#_Toc43650162)

[1.6.1 Автомат с магазинной памятью 13](#_Toc43650163)

[1.6.2 Синтаксические анализаторы LL(k)-грамматик 13](#_Toc43650164)

[2. Язык программирования RU 16](#_Toc43650165)

[2.1 Блок записи программы 16](#_Toc43650166)

[2.2 Типы данных 16](#_Toc43650167)

[2.2.1 Длинное целое положительное число 16](#_Toc43650168)

[2.2.2 Квадратная матрица 16](#_Toc43650169)

[2.3 Операторы ввода/вывода 17](#_Toc43650170)

[2.3.1 Оператор ввода 17](#_Toc43650171)

[2.3.1 Оператор вывода 17](#_Toc43650172)

[2.4 Цикл с предусловием 17](#_Toc43650173)

[2.5 Условный оператор 18](#_Toc43650174)

[2.6 Сортировка матрицы 18](#_Toc43650175)

[2.7 Комментарии 18](#_Toc43650176)

[3. Алгоритмизация 19](#_Toc43650177)

[3.1 Структуры данных 19](#_Toc43650178)

[3.2 Реализация трансляции 21](#_Toc43650179)

[3.3 Полезные функции 22](#_Toc43650180)

[3.3.1 Функция set\_id () 22](#_Toc43650181)

[3.3.2 Функция get\_id () 23](#_Toc43650182)

[3.3.3 Функции компиляции выражений 23](#_Toc43650183)

[3.3.4 Компиляция оператора присваивания 24](#_Toc43650184)

[3.3.5 Компиляция условного оператора 25](#_Toc43650185)

[3.3.6 Компиляция оператора цикла 25](#_Toc43650186)

[3.4 Список ошибок лексического анализатора 26](#_Toc43650187)

[3.5 Примеры программ 26](#_Toc43650188)

[4. Руководство пользователя 28](#_Toc43650189)

[4.1 Загрузка программ 28](#_Toc43650190)

[4.2 Сохранение программы 29](#_Toc43650191)

[4.3 Кнопка «Транслировать» 29](#_Toc43650192)

[4.4 Кнопка «Запустить» 30](#_Toc43650193)

[4.5 Поле ошибок 31](#_Toc43650194)

[4.6 Кнопка «Очистить ошибку» 32](#_Toc43650195)

[4.7 Кнопка «Закрыть» 32](#_Toc43650196)

# ВВЕДЕНИЕ

Компилятор – программа, переводящая текст программы, написанной на одном языке программирования, на другой, как более низкого, так и более высокого уровня.

Задачей данной курсовой работы является разработка компилятора с графическим интерфейсом, реализующего следующие возможности:

Язык относится к языкам высокого уровня и должен содержать набор операторов структурного программирования. Все операторы и ключевые слова имеют русские названия.

Базовые типы данных включают в себя:

* целый тип – длинное целое положительное число;
* матрица – квадратная матрица целых чисел.

Для целого типа данных выполнить операции сложения, вычитания, сравнения (больше, меньше, равно, не равно).

В языке должны быть реализованы следующие операторы:

* оператор ввода значения переменной и матрицы с клавиатуры,
* оператор вывода значения переменной и матрицы на экран,
* оператор цикла c предусловием,
* условный оператор,
* оператор сортировки строк матрицы.

Языки программирования: C, С++

Среда разработки: Borland C++ Builder

# Теория построения компиляторов

## 1.1 Основные понятия и определения

Транслятор – программа, которая переводит программу, написанную на одном языке, в эквивалентную ей программу, написанную на другом языке.

Компилятор – транслятор с языка высокого уровня на машинный язык или язык ассемблера.

Ассемблер – транслятор с языка Ассемблера на машинный язык.

Интерпретатор – программа, которая принимает исходную программу и выполняет ее, не создавая программы на другом языке.

Макропроцессор (препроцессор – для компиляторов) – программа, которая принимает исходную программу, как текст и выполняет в нем замены определенных символов на подстроки. Макропроцессор обрабатывает программу до трансляции.

Любой язык обязательно подчиняется определенным правилам:

Синтаксис – это совокупность правил, определяющих допустимые конструкции языка, т. е. его форму.

Семантика – это совокупность правил, определяющих логическое соответствие между элементами и значением синтаксически корректных предложений, т. е. содержание языка.

## 1.2 Этапы процесса компиляции

Процесс компиляции предполагает распознавание конструкций исходного языка (анализ) и сопоставление каждой правильной конструкции семантически эквивалентной конструкций другого языка (синтез).

Он включает несколько этапов:

1. лексический анализ:

Процесс преобразования исходного текста в строку однородных символов. Каждый символ результирующей строки (токен) соответствует слову языка (лексеме) и характеризуется набором атрибутов, таких как тип, адрес и т. п., поэтому строку токенов часто представляют таблицей, строка которой соответствует одному токену.

Лексема обозначает простое понятие языка.

Существует 2 типа лексем:

а) лексемы, соответствующие символам алфавита языка, такие как «Служебные слова» и «Служебные символы»;

б) лексемы, соответствующие базовым понятиям языка, такие как «Идентификатор» и «Литерал».

1. синтаксический анализ:

Процесс распознавания конструкций языка в строке токенов. Главным результатом является информация об ошибках в выражениях, операторах и описаниях программы.

1. семантический анализ:

Процесс распознавания/проверки смысла конструкции. По результатам распознавания строится последовательность, приближенная к последовательности операторов будущей программы и выполняются предусмотренные проверки правильности программы.

1. распределение памяти:

Процесс назначения адресов для именованных констант и переменных программы.

1. генерация и оптимизация объектного кода:

Процесс формирования программы на выходном языке, которая семантически эквивалентна исходной программе. На этом этапе также обычно выполняется оптимизация генерируемого кода.

Лексический и синтаксический анализ предполагают выполнение грамматического разбора. При их построении используют специальный математический аппарат – формальные грамматики.

## 1.3 Формальные грамматики

Формальный язык — это математическая модель реального языка.

Чтобы лучше разобраться в том, как именно изучаются формальные языки, необходимо сначала понять, в чем заключаются особенности математических методов изучения. Согласно Колмогорову, математический метод, к чему бы он ни применялся, всегда следует двум основным принципам:

Обобщение (абстрагирование). Объекты изучения в математике — это специальные сущности, которые существуют только в математике и предназначены для изучения математиками.

Математические объекты образуются путем обобщения реальных объектов. Изучая какой-нибудь объект, математик замечает только некоторые его свойства, а от остальных отвлекается. Так, абстрактный математический объект «число» может в реальности обозначать количество гусей в пруду или количество молекул в капле воды; главное, чтобы о гусях и о молекулах воды можно было говорить, как о совокупностях. Из такой «идеализации» реальных объектов следует одно важное свойство: математика часто оперирует бесконечными совокупностями, тогда как в реальности таких совокупностей не существует.

Строгость рассуждений. В науке принято для выяснения истинности того или иного рассуждения сверять их результаты с тем, что существует в действительности, т.е. проводить эксперименты. В математике этот критерий проверки рассуждения на истинность не работает. Поэтому выводы не проверяются экспериментальным путем, но принято доказывать их справедливость строгими, подчиняющимися определенным правилам, рассуждениями. Эти рассуждения называются доказательствами и доказательства служат единственным способом обоснования верности того или иного утверждения.

Таким образом, чтобы изучать языки с помощью математических методов, необходимо сначала выделить из языка его свойства, которые представляются важными для изучения, а затем эти свойства строго определить. Полученная таким образом абстракция будет называться формальным языком — математической моделью реального языка.

В теории формальных языков представляется важным изучить законы расположения слов рядом друг с другом, т.е. синтаксические свойства текстов. Поэтому формальный язык задается как множество последовательностей, составленных из элементов конечного алфавита.

Алфавит – непустое конечное множество символов, используемых для записи предложений языка.

Строка – любая последовательность символов алфавита, расположенных один за другим.

Строки в теории формальных языков обозначаются строчными греческими буквами: α, β, γ и т.д.

Строка, содержащая 0 символов, называется пустой и обозначается символами e, ε или λ.

Множество всех составленных из символов алфавита A строк, в которое входит пустая строка, обозначают A\*.

Множество всех составленных из символов алфавита A строк, в которое не входит пустая строка, обозначают A+. Откуда: A\* = A+ ∪ {e}.

Формальным языком L в алфавите A называют произвольное подмножество множества A\*. Таким образом, язык определяется как множество допустимых предложений. Задать язык L в алфавите A значит либо перечислить все включаемые в него предложения, либо указать правила их образования. Перечисление бесконечно большого количества предложений невозможно. Определение правил образования предложений осуществляют с использованием абстракций формальных грамматик.

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка:

G = (VT, VN, P, S)

VT – алфавит языка или множество терминальных (незаменяемых) символов;

VN – множество нетерминальных (заменяемых) символов – вспомогательный алфавит, символы которого обозначают допустимые понятия языка, VT ∩ VN = ∅;

V = VT ∪ VN – словарь грамматики;

P – множество порождающих правил – каждое правило состоит из пары строк (α, β), где α ∈ V+ – левая часть правила, β ∈ V\* – правая часть правила: α → β, где строка α должна содержать хотя бы один нетерминал;

S ∈ VN – начальный символ – аксиома грамматики.

Для описания синтаксиса языков с бесконечным количеством различных предложений используют рекурсию. При этом если нетерминал в порождающем правиле расположен справа, то рекурсию называют правосторонней, если слева, то – левосторонней, а, если между двумя подстроками, то – вложенной.

### 1.3.1 Форма Бэкуса-Наура (БНФ)

Форма Бэкуса-Наура (сокр. БНФ, Бэкуса — Наура форма) – формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. БНФ используется для описания контекстно-свободных формальных грамматик. Существует расширенная форма Бэкуса — Наура, отличающаяся лишь более ёмкими конструкциями.

БНФ-конструкция определяет конечное число символов (нетерминалов). Кроме того, она определяет правила замены символа на какую-то последовательность букв (терминалов) и символов. Процесс получения цепочки букв можно определить поэтапно: изначально имеется один символ (символы обычно заключаются в угловые скобки, а их название не несёт никакой информации). Затем этот символ заменяется на некоторую последовательность букв и символов, согласно одному из правил. Затем процесс повторяется (на каждом шаге один из символов заменяется на последовательность, согласно правилу). В конце концов, получается цепочка, состоящая из букв и не содержащая символов. Это означает, что полученная цепочка может быть выведена из начального символа.

БНФ-конструкция состоит из нескольких предложений вида

**<определяемый символ> ::= <посл.1> | <посл.2> | . . . | <посл.n>**

описывающих правила. Такое правило означает, что символ <определяемый символ> может заменяться на одну из последовательностей <посл.i>. Знак определения обычно выглядит как ::= или →, но возможны и другие варианты.

Некоторые специальные символы, как например <пусто>, означают какую-то последовательность (в данном случае — пустую).

Пример БНФ-конструкции, описывающей правильные скобочные последовательности:

<правпосл>::=<пусто> | (<правпосл>) | <правпосл><правпосл>

Это простая конструкция, состоящая всего из одного правила, утверждающего, что символ <правпосл> может замениться либо на пустое место, либо на этот же символ <правпосл>, заключённый в скобки, либо на два символа <правпосл> идущих подряд.

## 1.4 Грамматический разбор

Распознавание принадлежности строки конкретному языку осуществляется его выводом из аксиомы. Вывод представляет собой последовательность подстановок, при выполнении которых левая часть правила заменяется правой. Исходная строка, строки, получаемые в процессе вывода и аксиома, называются сентенциальными формами. Сентенциальная форма, содержащая только терминальные символы, называется допустимой и представляет собой предложение языка.

Грамматический разбор – процедура построения синтаксического дерева для конкретного предложения языка. Построение такого дерева позволяет однозначно доказать, что анализируемая строка языка является допустимой, т.е. принадлежит конкретному языку. Грамматический разбор может осуществляться в разной последовательности, причем дерево можно строить как «сверху» – от аксиомы, так и «снизу» – от предложения. Соответственно существуют нисходящий и восходящий методы разбора. При этом предложения языка можно рассматривать слева направо и наоборот. Соответственно различают левосторонний и правосторонний разбор. Левосторонний разбор используют чаще, т. к. мы читаем слева направо.

### 1.4.1 Левосторонний восходящий грамматический разбор

Метод разбора «слава-направо» применим, если грамматика не содержит правил с правосторонней рекурсией. При осуществлении грамматического разбора сентенциальные формы просматриваются слева направо и последовательно «свертываются» посредством замены подстроки, совпадающей с правой частью правила («основы») на левую часть того же правила. Правила подстановки должны проверяться, начиная с самого сложного, иначе сложные правила никогда не будут применены, и разбор не удастся. В общем случае при разборе возможны возвраты, поскольку может быть выбрано неподходящее правило подстановки.

### 1.4.2 Левосторонний нисходящий грамматический разбор

Метод разбора «сверху-вниз» применим, если грамматика не содержит правил с левосторонней рекурсией. Метод предполагает последовательное выдвижение гипотез, начиная с аксиомы, их доказательство посредством подбора правил, которым удовлетворяет разбираемый фрагмент, и выдвижения новых гипотез относительно не разобранной части предложения. Правила подстановки также должны проверяться, начиная с самого сложного, иначе цель разбора не будет достигнута. При наличии правил с левосторонней рекурсией процедура разбора становится бесконечной. В общем случае при разборе возможны возвраты, поскольку может быть выбрано неподходящее правило подстановки. Недостаток метода, как и в предыдущем случае, заключается в том, что в программе разбора следует хранить всю информацию о каждом его шаге. Выбор метода разбора для грамматики определяется видом правил продукции языка.

## 1.5 Регулярные грамматики

### 1.5.1 Конечные автоматы

Распознаватели регулярных грамматик строятся на конечных автоматах.

Конечный автомат – это математическая модель, свойства и поведение которой полностью определяются пятеркой:

M = (Q, Σ, δ, q0, F)

Q – конечное множество состояний;

Σ – конечное множество входных символов;

δ(qi, сk) – функция переходов (qi – текущее состояние, сk – очередной символ);

q0 – начальное состояние;

F = {qj} – подмножество допускающих состояний.

Конечные автоматы – одна из базовых моделей, используемых при проектировании программного и аппаратного обеспечения средств вычислительной техники.

### 1.5.2 Лексические анализаторы

При построении лексических анализаторов конечные автоматы используют для распознавания лексем второго типа (базовых понятий языка). Эти понятия можно описать в рамках самых простых, регулярных грамматик и предложить для них эффективную технику разбора. Отделение сканера от синтаксического анализатора позволяет также сократить время распознавания лексем, так как при этом появляется возможность использования оптимизированных ассемблерных команд, специально предназначенных для сканирования строк. Алфавит автомата лексического анализатора – все множество однобайтовых (ANSI) или двухбайтовых (Unicode) символов. При записи правил обычно используются обобщающие нетерминалы вида «Буквы», «Цифры». В процессе распознавания может формироваться описываемый объект, например, литерал или идентификатор.

## 1.6 Контекстно-свободные грамматики

### 1.6.1 Автомат с магазинной памятью

Распознавание КС-грамматик выполняется автоматом с магазинной памятью. Автомат с магазинной памятью определяется семеркой:

PM = (Q, Σ, Г, δ, q0, z0, F)

Q – конечное множество состояний автомата;

Σ – конечный входной алфавит;

Г – конечное множество магазинных символов;

δ (q, ck, zj) – функция переходов;

q0 ∈ Q – начальное состояние автомата;

z0 ∈ Г – символ, находящийся в магазине в начальный момент;

F ⊆ Q – множество заключительных (допускающих) состояний.

### 1.6.2 Синтаксические анализаторы LL(k)-грамматик

Метод рекурсивного спуска LL(k)-грамматики – это класс КС-грамматик, гарантирующих существование детерминированных нисходящих распознавателей (L – левосторонний просмотр исходной строки, L – левосторонний разбор, k – количество символов, просматриваемых для определения очередного правила). В грамматиках этого класса для однозначного выбора правил должны:

1) отсутствовать правила с левосторонней рекурсией;

2) различаться k первых символов разных правил.

Реализация нисходящего разбора заключается в следующем:

В исходный момент времени в стек записывают аксиому. Затем выбирают правило подстановки аксиомы, сравнивая k символов исходной строки с тем же количеством символов правых частей правил. Правую часть правила подставляют в стек вместо левой части. Одинаковые первые символы стека и исходной строки удаляют. Затем вновь выбирают правило подстановки уже для нетерминала, который оказался в начале стека и т. д. Количество просматриваемых символов определяется конкретными правилами грамматики.

#### 1.6.2.1 Метод рекурсивного спуска

Метод рекурсивного спуска основывается на синтаксических диаграммах языка. Согласно этому методу для каждого нетерминала разрабатывают рекурсивную процедуру. Основная программа вызывает процедуру аксиомы, которая вызывает процедуры нетерминалов, упомянутые в правой части аксиомы и т. д. В эти же процедуры встраивают семантическую обработку распознанных конструкций.

1.7 Синтаксические анализаторы LR(k)-грамматик

LR(k)-грамматики – это класс КС-грамматик, гарантирующих существование детерминированных восходящих распознавателей (L – левосторонний просмотр, R – правосторонний разбор, k – количество символов, просматриваемых для однозначного определения следующего правила). В грамматиках этого класса отсутствуют правила с правосторонней рекурсией, и обеспечивается однозначное выделение основы. К этому классу, например, относятся грамматики с операторным предшествованием, простым предшествованием и расширенным предшествованием, обеспечивающие еще более простые алгоритмы распознавания. При восходящем разборе стек используют для накопления основы. Автомат при этом выполняет две основные операции: свертку и перенос. Свертка выполняется, когда в стеке накоплена вся основа, и заключается в ее замене на левую часть соответствующего правила. Перенос выполняется в процессе накопления основы и заключается в сохранении в стеке очередного распознаваемого символа сентенциальной формы. Основная проблема метода заключается в нахождении способа выделения очередной основы. Проще всего основу выделить для грамматик, получивших название «грамматики предшествования». Рассмотрим эти грамматики. Если два символа α, β ∈V расположены рядом в сентенциальной форме, то между ними возможны следующие отношения, названные отношениями предшествования: - α принадлежит основе, а β – нет, т. е. α – конец основы: α ⋅> β; - β принадлежит основе, а α – нет, т. е. β – начало основы: α α <⋅ β ;- α и β принадлежит одной основе, т. е. α =⋅ β;- α и β не могут находиться рядом в сентенциальной форме (ошибка).

Грамматикой с предшествованием называется грамматика, в которой существует однозначное отношение предшествования между соседними символами. Это отношение позволяет просто определить очередную основу, т. е. момент выполнения каждой свертки.

Различают:

1) грамматики с простым предшествованием, для которых α, β ∈V;

2) грамматики с операторным предшествованием, для которых α, β ∈VТ; т. е. отношение предшествования определено для терминальных символов и не зависит от нетерминальных символов, расположенных между ними;

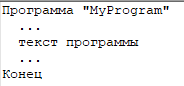
3) грамматики со слабым предшествованием, для которых отношение предшествования не однозначно – оно требует выполнения специальных проверок.

# Язык программирования RU

## 2.1 Блок записи программы

Оболочкой для программы является блок, начинающийся оператором **Программа "Название программы"** и оканчивающийся оператором **Конец**.

Внутри этого блока непосредственно располагается сам код программы.



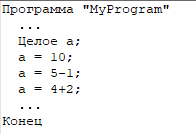
## 2.2 Типы данных

### 2.2.1 Длинное целое положительное число

Объявить целое число можно оператором **Целое** после которого следует название переменной целого типа.

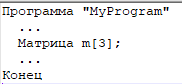
Присвоить значение переменной можно с помощью знака **=**.

Данный тип данных поддерживает следующие операции, проводимые над ним: +, -, >, <, >=, <=, =, !=



### 2.2.2 Квадратная матрица

Представляет собой двумерный массив с одинаковым количеством столбцов и строк. Задается оператором **Матрица**.



## 2.3 Операторы ввода/вывода

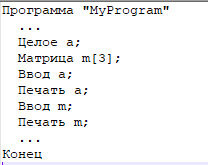
Применимы для всех типов данных.

### 2.3.1 Оператор ввода

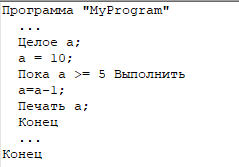
Объявить ввод переменной можно оператором **Ввод**.

### 2.3.1 Оператор вывода

Объявить вывод переменной можно оператором **Печать**.



## 2.4 Цикл с предусловием



Форма записи:

**Пока** <условие> **Выполнить**

...

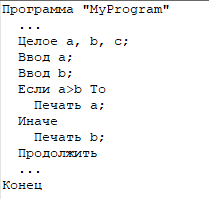
Тело цикла

...

**Конец**

Вложенность циклов поддерживается

## 2.5 Условный оператор



Форма записи:

**Если** <условие> **То**

...

Тело условного оператора

...

**Иначе**

...

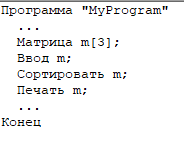
Тело условного оператора

...

**Продолжить**

Возможно применение операторов &&, ||

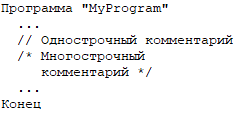
## 2.6 Сортировка матрицы



Форма записи:

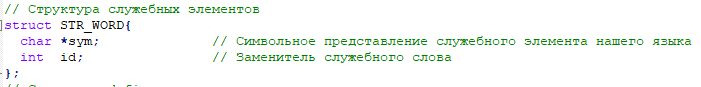
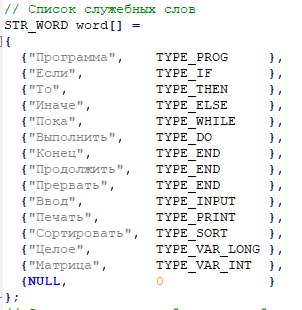
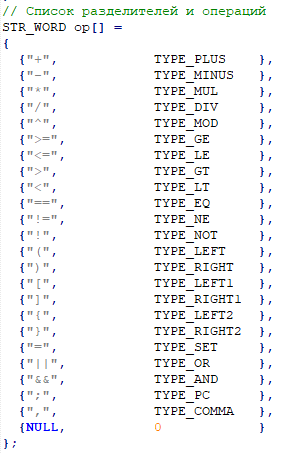
Сортировка матрицы вызывается оператором **Сортировать**

## 2.7 Комментарии



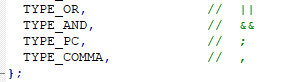
# Алгоритмизация

## 3.1 Структуры данных

Служебные слова и разделители хранятся в массивах данных, типом данных которых является структура

С помощью enum задаются лексемы и ошибки





Идентификаторы храним в таблице идентификаторов



## 3.2 Реализация трансляции

Объявления:

Целое x; 🡪 unsigned long x;

Матрица m[3]; 🡪 int m[3][3];

Ввод:

Ввод x; 🡪 printf ("Vvedite 'a'"); scanf ("%d", &a);

Ввод m; 🡪 for(int i=0; i<3;i++){

for(int j=0; j<3;j++){

printf("m[%d][%d]=", i, j);

scanf("%d", &m[i][j]);

}

}

Вывод:

Печать x; 🡪 printf("x = "); printf ("%d", x);printf("\n");

Печать m; 🡪 for(int i=0; i<3;i++){

for(int j=0; j<3;j++){

printf("m[%d][%d]=", i, j);

printf("%d",m[i][j]);

printf("\n");

}

}

Сортировка матрицы:

Сортировать m; 🡪 printf ("Sort 'm'\n");

for(int k=0; k<3\*3;++k){

for(int i=0; i<3;++i){

for(int j=0; j<3;++j){

if(j!=3-1){

if(m[i][j+1]<m[i][j]){

int t = m[i][j+1];

m[i][j+1] = m[i][j];

m[i][j] = t;

}

}else{

if((m[i+1][0]<m[i][j])&&(i != 3-1)){

int t = m[i+1][0];

m[i+1][0] = m[i][j];

m[i][j] = t;

}

}

}

}}

Цикл с предусловием:

Пока a >= 5 Выполнить while ( a >= 5 ) {

a=a-1; 🡪 a=a-1;

Конец }

Условный оператор:

Если a >= 5 То if ( a >= 5 ) {

a=a-1; 🡪 a=a-1;

Иначе }else{

a=a+1; a=a+1;

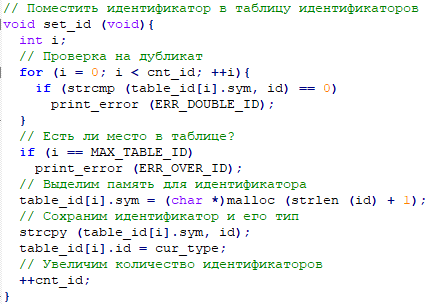
Продолжить }

Операторы +, -, =, !=, &&, ||, <, >, >=, <=, //, /\* транслируются в таком же виде

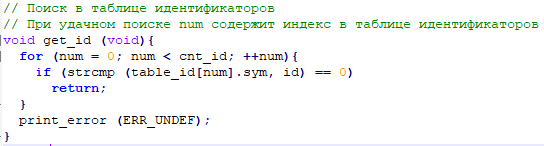
## 3.3 Полезные функции

### 3.3.1 Функция set\_id ()

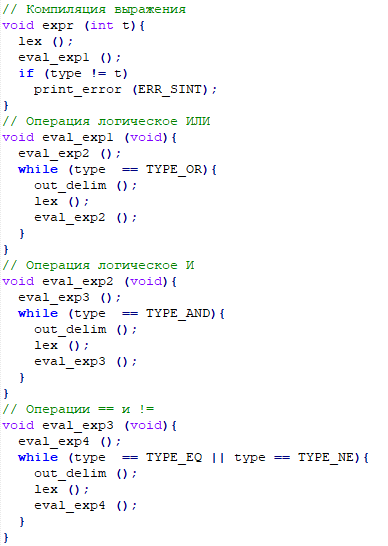
Помещает идентификатор в таблицу идентификаторов



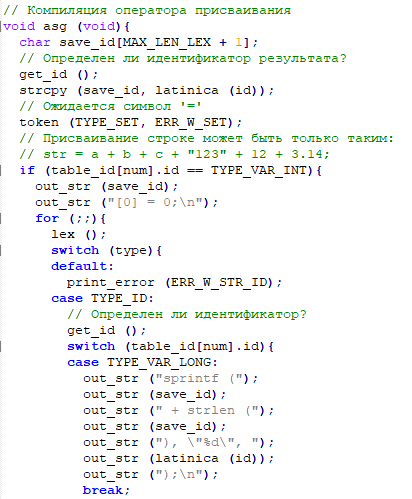
### 3.3.2 Функция get\_id ()

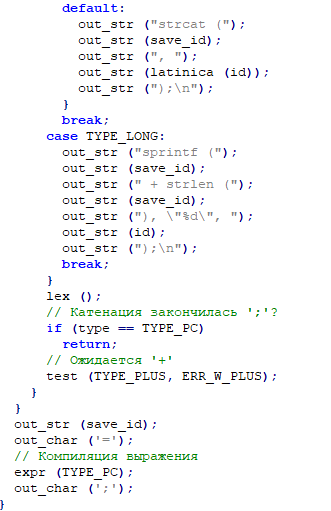
Ищет идентификатор в таблице идентификаторов.

### 3.3.3 Функции компиляции выражений

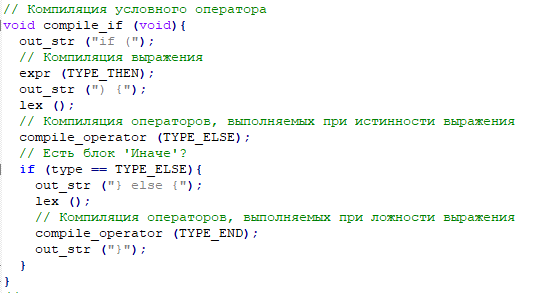


### 3.3.4 Компиляция оператора присваивания

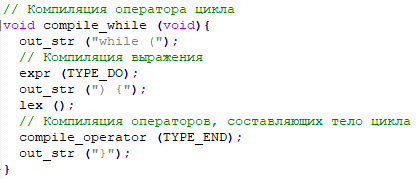




### 3.3.5 Компиляция условного оператора

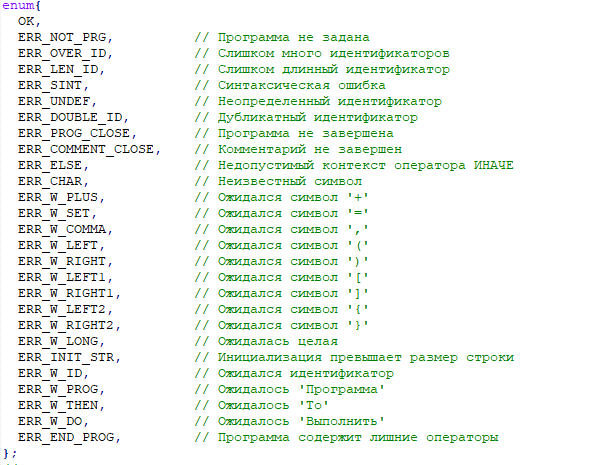


### 3.3.6 Компиляция оператора цикла



И еще много-много других. (см. код программы)

## 3.4 Список ошибок лексического анализатора



## 3.5 Примеры программ

Программа "Numbers"

Целое a, b;

b = 2+3;

Печать b;

b = 9-5;

Печать b;

Ввод a;

Печать a;

Конец

Программа "Matrix"

Матрица m[3];

Ввод m;

Сортировать m;

Печать m;

Конец

Программа "While\_Prog"

Целое a;

a = 10;

Пока a >= 5 Выполнить

a=a-1;

Печать a;

Конец

Конец

Программа "If\_Prog"

Целое a, b, c;

Ввод a;

Ввод b;

Если a>b То

Печать a;

Иначе

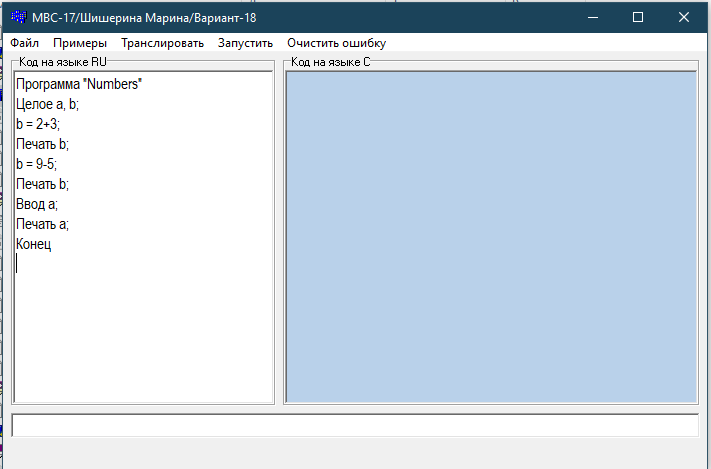
Печать b;

Продолжить

Конец

# Руководство пользователя

При запуске программы открывается ее окно



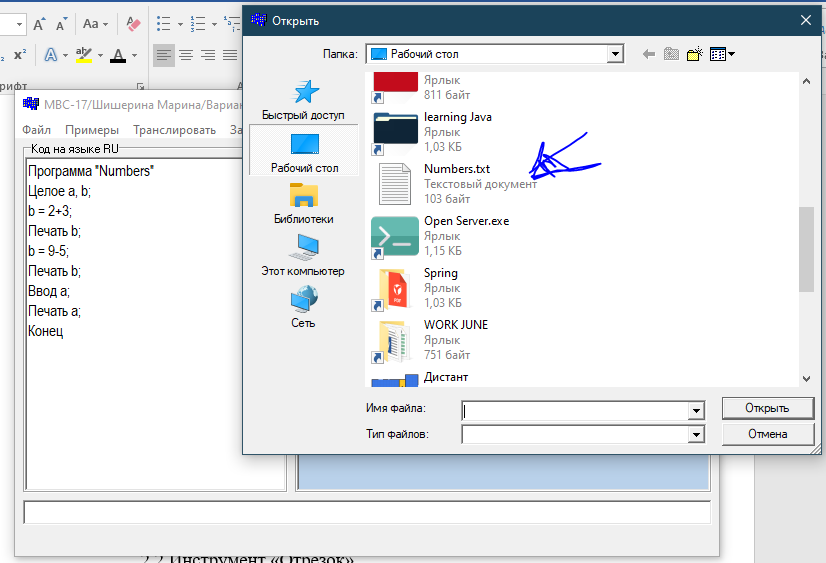
Поле вывода ошибок

Окно для вывода транслированной программы

Окно ввода программы на языке RU

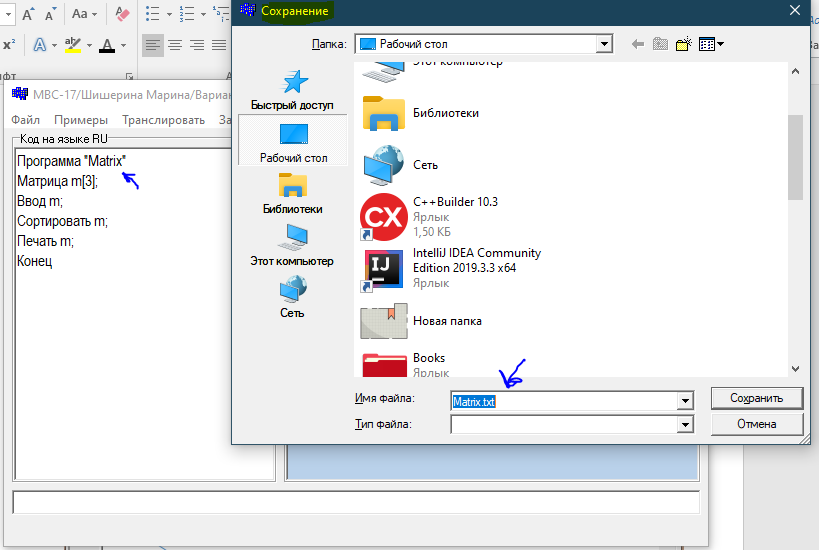
меню

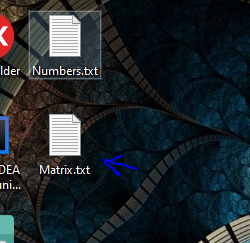
## 4.1 Загрузка программ

Программу можно загрузить из Примеры -> Программа или из файла: Файл -> Загрузить программу

## 4.2 Сохранение программы

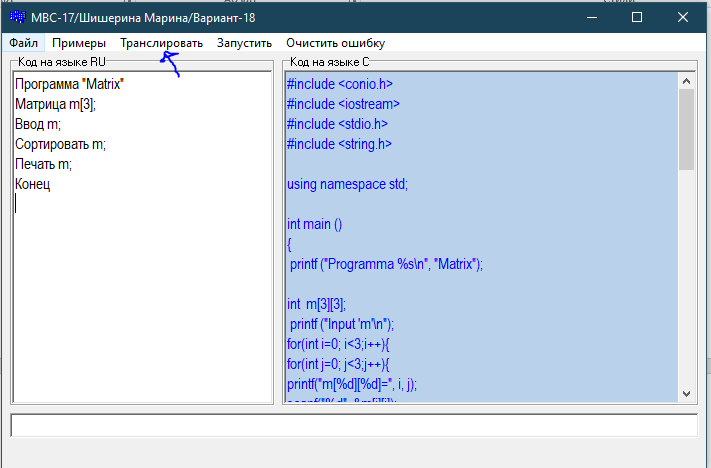
Файл -> Сохранить программу





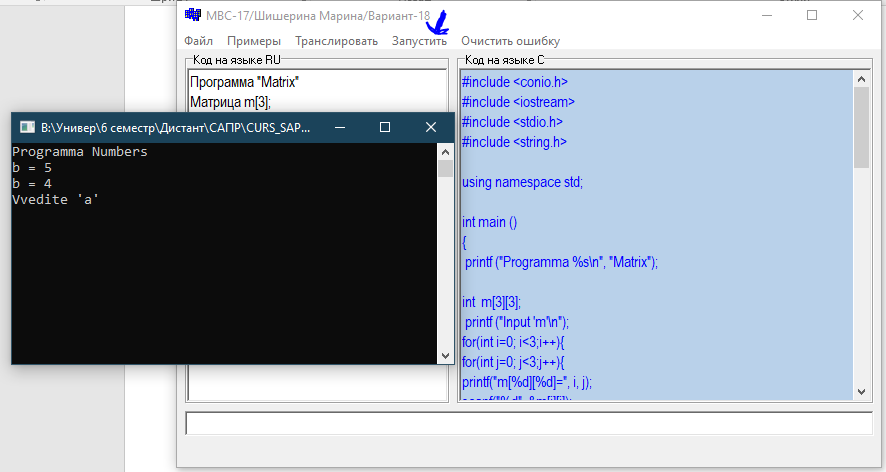
## 4.3 Кнопка «Транслировать»

Переводит код, написанный на языке RU в код языка С

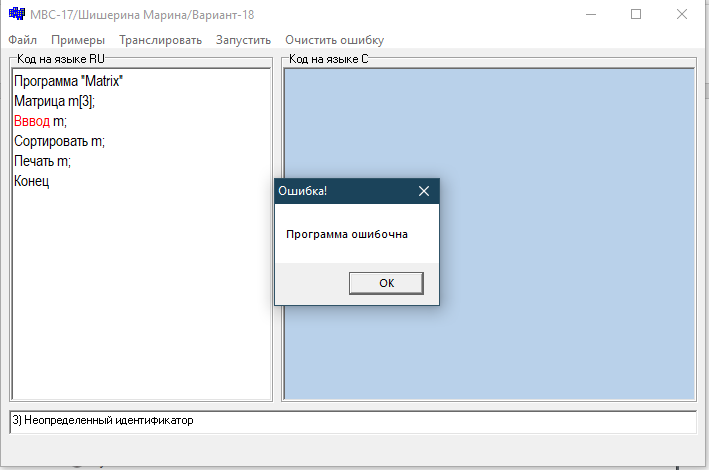


## 4.4 Кнопка «Запустить»

Транслирует код с языка RU на С и запускает консольную программу

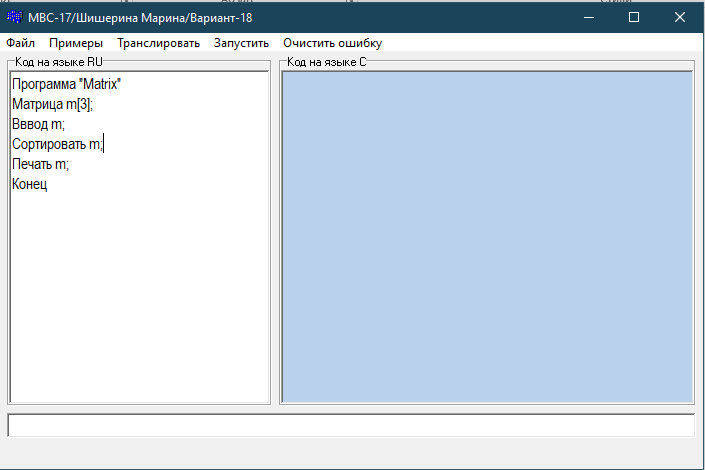


## 4.5 Поле ошибок

Ели программа некорректна, компилятор не позволит ее запустить и выведет текст ошибки в нижнее поле. Часть программы, написанная некорректно будет подсвечена красным.

## 4.6 Кнопка «Очистить ошибку»

Дает программе понять, что пользователь понял где ошибка. Убирает надпись из окна ошибок и снимает выделение синтаксиса.



## 4.7 Кнопка «Закрыть»

Файл -> Закрыть закрывает программу.

# СПИСОК ЛИТЕРАТУРЫ

1. Архангельский А.Я. C++Builder 6. Справочное пособие. Книга 1. Язык C++. – М.: БиномПресс, 2002 г. – 544 с.
2. Архангельский А.Я. C++Builder 6. Справочное пособие. Книга 2. Классы и компоненты. – М.: Бином-Пресс, 2002 г. – 528 с.
3. Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М. : ООО «И.Д. Вильямс», 2008 – 1184 с.
4. Хантер Р. Проектирование и конструирование компиляторов. Пер. с англ.; Предисл. В. М. Савинкова. — М.: Финансы и статистика, 1984 – 232 с.
5. Т. Пратт, М. Зелковиц - Языки программирования. Разработка и реализация. – Питер, 2002. – 690 с.
6. Страуструп Бьярне. Программирование: Принципы и практика использования С++ : Пер. с англ. – М.: ООО «И. Д. Вильямс», 2011. – 1248с.
7. Брайан Керниган, Деннис Ритчи - Язык программирования C. – Москва: Вильямс, 2015. — 304 с.