НИУ ВШЭ – Пермь
Факультет бизнес-информатики
Кафедра информационных технологий в бизнесе

# Формальные языки и грамматики

Материалы курса «Теоретические основы информатики»

Лекция 6

*Лядова Л.Н.*

Пермь 2012

# Естественные и искусственные языки

***Естественный язык***: Язык, правила которого основываются на современном словоупотреблении без точного их описания.

***Искусственный язык***: Язык, правила которого четко устанавливаются до его использования.

***Язык программирования***: Искусственный язык для представления программ.

Традиционно программы для ЭВМ записываются в виде *строк символов некоторого алфавита*. Современные системы программирования и языки допускают использование *визуальных элементов* (окон, икон и др.) для построения программ, в частности, интерфейса с пользователем. Такие языки обычно называют ***языками визуального программирования***.

# Понятие языка программирования

***Язык программирования*** (ЯП) – формальная знаковая система, предназначенная для записи компьютерных программ.

Для каждого языка определяются:

1. ***Функция языка программирования***.
2. Задача языка программирования.
3. Исполнение языка программирования.

Язык программирования предназначен для *написания компьютерных программ*, которые представляют собой инструкции (алгоритмы) по выполнению вычислительного процесса и организации управления отдельными устройствами при решении задач с помощью компьютера.

# Понятие языка программирования

***Язык программирования*** (ЯП) – формальная знаковая система, предназначенная для записи компьютерных программ.

Для каждого языка определяются:

1. Функция языка программирования.
2. ***Задача языка программирования***.
3. Исполнение языка программирования.

Язык программирования отличается от естественных языков тем, что *предназначен для передачи команд и данных от человека к компьютеру*, в то время как естественные языки используются для общения людей между собой. Таким образом, ЯП – это способ передачи команд, чёткой инструкции к действию, а не средство обмена информацией, как естественные языки.

# Понятие языка программирования

***Язык программирования*** (ЯП) – формальная знаковая система, предназначенная для записи компьютерных программ.

Для каждого языка определяются:

1. Функция языка программирования.
2. Задача языка программирования.
3. ***Исполнение языка программирования***.

Язык программирования *предполагает использование специальных конструкции для определения и манипулирования структурами данных и управления процессом вычислений*.

# Спецификация и стандартизация языка программирования

Язык программирования может быть представлен в виде *набора спецификаций*, определяющих его.

Для многих широко распространённых ЯП созданы международные стандарты.

Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка.

На основе существующих языков появляются новые языки, с новыми свойствами и возможностями.

# Спецификация и стандартизация языка программирования

| Основные характеристики | Основные типы данных | Основная область применения | Происхождение | Разработчик | Стандарт | Язык |
|---|---|---|---|---|---|---|
| Параллельная многозадачность, модульность | Создает собственные типы данных | Обработка в реальном времени | Pascal, PL/1, Algol 68 | Минобороны США, 1970 г. | ISO 8652 | Ada |
| Интерпретирующего типа, не компилятор, расширенная графика, возможность генерации звука | Целое (байт, слово, длинное целое); слово с плавающей точкой, двойной точности | Учебное программирование | | Кемени и Куртц, 1960 г. | ISO 6373 | BASIC |
| Обеспечивает программирование на уровне машинного кода, ориентирован на системное программирование, позволяет использовать типы данных и записей, задаваемых программистом | Целое, с плавающей точкой (одинарной и двойной точности, символьные, структуры, указатели | UNIX, язык, альтернативный ассемблеру структурное программирование | B и BCPL | BELL Lab., 1970 г. | ISO 9899 | C |
| Использует структуры данных типа "запись" | Десятичные данные | Обработка коммерческой информации | | Минобороны США, ряд поставщиков систем, 1960г. | ISO 1989 | COBOL 85 |
| Позволяет программисту описывать вычисления в виде формул | Целое, с плавающей точкой удвоенной точности, булевские, символьные переменные | Научные и инженерные расчеты | | IBM, 1950 г. | ISO 1359 | FORTRAN 77 |
| Новые типы данных, определяемые программистом | Действительные, целые, булевские, символьные переменные | Язык общего применения | | Н. Вирт | ISO 7185 | Pascal |
| Высокий уровень машинной независимости, эффективное внедрение на машинах всех классов | Множественные типы данных, в т. ч. числа с плавающей и фиксированной точкой, двоичные и десятичные, символьные и битовые строки | Использовался для сложных программ на мейнфреймах | ALGOL 60 – блочная структура FORTRUN – арифметика COBOL – структура данных | IBM, 1960 г. | ISO 6160 | PL/1 |

# Описание языка программирования

*Описание языка программирования* складывается из четырех компонентов:

1) описание лексики,
2) описание синтаксиса,
3) описание семантики,
4) описание прагматики.

# Описание языка программирования

*Описание языка программирования* складывается из четырех компонентов:

1) ***описание лексики***,
2) описание синтаксиса,
3) описание семантики,
4) описание прагматики.

*Описание лексики* в языке программирования – это задание набора символов, которые можно использовать при написании программ, задание ключевых (зарезервированных) слов, зарезервированных сочетаний символов (например ‘<=’).

С точки зрения формального языка описание лексики – это просто *задание алфавита A*, т.е. множества его неделимых (терминальных) символов.

# Описание языка программирования

*Описание языка программирования* складывается из четырех компонентов:

1) описание лексики,
2) ***описание синтаксиса***,
3) описание семантики,
4) описание прагматики.

*Описание синтаксиса* – это задание правил построения различных *конструкций* языка (выражений, операторов цикла и т.д.).

Синтаксис описывается с использованием заданного алфавита, лексики языка.

Для формальных языков описание синтаксиса является основой описания семантики языка.

# Описание языка программирования

*Описание языка программирования* складывается из четырех компонентов:

1) описание лексики,
2) описание синтаксиса,
3) ***описание семантики***,
4) описание прагматики.

*Описание семантики* – придание смысла различным конструкциям ЯП.

Одним из способов описания семантики является задание абстрактного *Исполнителя* - идеализированной вычислительной машины с простейшими командами, смысл и результат действия которых очевиден. Для каждой конструкции ЯП составляется программа Исполнителя. Считается, что эта программа «объясняет» смысл конструкции.

# Описание языка программирования

*Описание языка программирования* складывается из четырех компонентов:

1) описание лексики,
2) описание синтаксиса,
3) описание семантики,
4) ***описание прагматики***.

*Описание прагматики* (применения) языка отвечает на вопрос: «Как писать программы на данном языке программирования?».

Описание прагматики невозможно формализовать – это «передача опыта», описание рекомендаций, как использовать конструкции языка для решения тех или иных задач для получения максимально эффективного решения.

# Описание лексики языка программирования

*Лексема*, в обычном понимании – это *словарная единица*.

С точки зрения компилятора – это «символ» языка.

Из лексем складываются конструкции языка, описываемые при определении синтаксиса языка программирования.

Так как лексемы могут представлять собой цепочки (сочетания) нескольких символов, которые вводятся с клавиатуры, то лексика может рассматриваться как «часть» синтаксиса: для описания «составных» символов задаются правила (например, задаются правила записи идентификаторов, констант различных типов).

Таким образом, происходит «расслоение» синтаксиса: сначала задаются правила записи простейших конструкций, которые рассматриваются как лексемы, а затем – более сложных конструкций (выражений, операторов языка и т.п.).

# Лексика и синтаксис языка программирования

На практике описание лексики языка отделено от описания синтаксиса (и, соответственно, лексический анализ программ обычно отделен от синтаксического).

Формально можно определить лексемы нескольких видов:

- знаки операций (‘+’, ‘-’ и др.),
- константы (знаковые и беззнаковые числа),
- идентификаторы и пр.

Другим принципиальным ограничением лексики является отсутствие при описании лексики вложенности фрагментов, составляющих лексему (точнее, возможно отслеживать вложенность на ограниченное количество уровней).

# Классификация языков программирования по уровням

Языки программирования разделяются на две основные категории:

- *Язык низкого уровня* (*low-level language*) – язык программирования, предназначенный для определенного типа процессоров и отражающий внутреннюю организацию, архитектуру компьютера. Каждая команда такого языка соответствует одной команде машинного языка – внутреннего языка процессора. Это «*машинно-ориентированный* язык».
- *Язык высокого уровня* (*high-level language*) – язык программирования, средства которого обеспечивают описание задачи в наглядном, легко воспринимаемом виде, *удобном для программиста*. Основная черта высокоуровневых языков – это *абстракция*. ЯВУ не только облегчают решение сложных программных задач, но и упрощают *портирование* ПО (обеспечивают *переносимость* программ – возможность переноса на другую платформу)

# Компиляция и интерпретация

ЯВУ не зависит от внутренних машинных языков целевых процессоров, поэтому программы, написанные на языках высокого уровня, *требуют перевода в машинные коды* с помощью специальных программ – *трансляторов* (*компиляторов* или *интерпретаторов*), т.е. языки программирования могут быть реализованы как компилируемые и интерпретируемые.

Программа на *компилируемом языке* при помощи компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код – двоичные коды инструкций конкретного процессора), который далее преобразуется (компонуется) в исполнимый модуль.

Программу на *интерпретируемом языке* непосредственно выполняет без предварительного перевода программа интерпретатор.

# Понятие транслятора

*Транслятор* – это программа или техническое средство, выполняющее преобразование программы, представленной на одном из языков программирования, в программу на другом языке и, в определённом смысле, равносильную первой.

Язык, на котором представлена входная программа, называется *исходным языком*, а сама программа – исходным кодом.

Выходной язык называется *целевым языком* или *объектным кодом*.

Транслятор обычно выполняет также диагностику ошибок, выдаёт тексты программы на исходном и объектном языке, и т.д.

Транслятор, который преобразует программы в машинный язык, принимаемый и исполняемый непосредственно процессором, называется *компилятором*.

# Структура компилятора и этапы компиляции

# Этапы компиляции

*Модуль ввода-вывода* считывает исходный текст программы и преобразует его, исключая из него, в частности, «лишние» символы (удаляя «лишние» разделители (пробелы, табуляции и пр.), комментарии и т.п.), а также формирует листинг программы.

Трансляция программы предполагает выполнение *проверки соответствия программы заданным описаниям языка программирования*, проведения

- лексического,
- синтаксического и
- семантического анализа.

Если в программе не обнаружены ошибки, компилятор генерирует объектный код.

Этапы компиляции могут совмещаться.

# Этапы компиляции

Этапы компиляции

- ***Лексический анализ***.
- Синтаксический анализ.
- Семантический анализ.
- Генерация кода.

Результат *лексического анализа* – последовательность неделимых символов языка (лексем): идентификаторов, ключевых слов, констант, знаков операций и пр.).

Если в результате работы анализатора выявляются ошибки, сообщения об этом выводятся как результат работы лексического анализатора.

Кроме последовательности символов, в которую преобразуется программа, лексический анализатор строит еще и различные таблицы, которые используются на последующих этапах анализа (таблицы ключевых слов, идентификаторов и пр.).

# Этапы компиляции

Этапы компиляции

- Лексический анализ.
- ***Синтаксический анализ***.
- Семантический анализ.
- Генерация кода.

Фаза *синтаксического анализа* предполагает синтаксический разбор полученной последовательности символов.

При этом определяется, является ли входное предложение языка правильным в данном языке, соответствует ли синтаксическим правилам.

Реальным результатом является внутренне представление программы (например, синтаксическое дерево), которое отражается структуру анализируемой программы: из каких синтаксических единиц она состоит, и в какой взаимосвязи они находятся.

# Этапы компиляции

Этапы компиляции

- Лексический анализ.
- Синтаксический анализ.
- ***Семантический анализ***.
- Генерация кода.

*Семантический анализатор* проверяет соответствие построенной программы описаниям семантики языка, синтаксическим единицам придается смысл. Семантические процедуры осуществляют анализ программы во внутреннем представлении, построенном синтаксическим анализатором (выполняют обход синтаксического дерева, задают «смысл» этой структуры). Их вызов – продолжение процесса трансляции.

# Этапы компиляции

Этапы компиляции

- Лексический анализ.
- Синтаксический анализ.
- Семантический анализ.
- ***Генерация кода***.

Если в результате анализа не выявляются серьёзные ошибки, то последний этап работы компилятора – *генерация машинного кода* целевого процессора.

Для того чтобы было возможно построить процедуры трансляции необходимо четко формализовать описание языка.

# Определение формального языка

Введем конечное множество символов

$$A = \{a_1, a_2, ..., a_n\},$$

которое назовем ***алфавитом языка***.

Из символов алфавита можно составлять всевозможные *цепочки* различной длины, записывая символы друг за другом.

Минимальная цепочка имеет длину 0, не содержит ни одного символа, называется *пустой* цепочкой и обозначается $\lambda$ (греческая лямбда). Ограничения сверху на длину цепочек нет.

Любой символ алфавита может входить в цепочку произвольное число раз.

Бесконечное множество всех возможных цепочек обозначается $A^*$.

Любое подмножество $L \subset A^*$ называется ***языком***.

# Определение формального языка

Одни цепочки символов принадлежат языку *L* (их называют *правильными*), а другие цепочки не принадлежат.

Практика обычно требует от любого языка существования двух механизмов:

1) механизма создания (генерации) правильных цепочек;

2) механизма проверки для любой данной цепочки, является ли она правильной.

Первая задача решается в CASE-средствах, которые по различным (обычно визуальным) описаниям строят программный код на языке программирования.

Вторая задача – это задача, решаемая на этапе трансляции программ.

# Определение формального языка: понятие грамматики

***Грамматика*** представляет набор из четырех элементов:

$$G = \{A, N, P, S\},$$

где $A$ – алфавит или множество *терминальных* символов,

$N$ – множество *нетерминальных* символов,

$P$ – множество *правил* грамматики,

$S \in N$ – *начальный (целевой) символ* грамматики.

*Начальный* символ $S$ грамматики представляет собой основное (базовое) понятие, базовая конструкция языка.

*Правила* (в литературе также встречается термин *продукции*) $P$ грамматики задают отношения между различными нетерминальными и терминальными символами, описывают правильные с точки зрения языка конструкции.

# Определение формального языка: понятие метаязыка

Для описания правил необходимо использовать специальные средства – свой формальный язык, который называется метаязыком.

***Метаязык*** – это язык, средствами которого производится описание структурных (а в общем случае и дедуктивных, семантических) свойств какого-либо другого (обычно формализованного) языка, являющегося предметом изучения.

Метаязык имеет свой алфавит, причем символы, используемые для формулирования правил метаязыка, не принадлежат описываемому языку. Эти символы называют *метасимволами*.

Существует два основных подхода к описанию языков: описание с помощью диаграмм (с использованием графических нотаций) и с помощью металингвистических формул (в виде текста).

# Определение формального языка с помощью металингвистических формул

Наиболее распространенным является описание языков программирования с помощью БНФ (*Форма Бэкуса-Наура*).

БНФ представляет собой разновидность *порождающей формальной грамматики*.

Для того чтобы облегчить чтение и понимание БНФ, а также возможности описания языков, используются расширения (или модификации), которые называются расширенными (модифицированными) БНФ, или РБНФ.

Далее рассматривается модификация БНФ, которая соответствует предложениям Н. Вирта и принята в качестве английского национального стандарта.

# Определение формального языка с помощью БНФ

Правила грамматики (продукции) формулируются в виде

$$\beta ::= \gamma,$$

где символ ::= (метасимвол) читается «*это есть*» или «*по определению есть*» (иногда используют другой символ – стрелку →), β – левая часть правила, которая содержит *определяемое понятие* (нетерминальный символ), а γ – правая часть, которая является *определением*.

В общем случае левая часть правила β является цепочкой символов расширенного алфавита $N \cup A$, т.е. $\beta \in (N \cup A)^*$. Здесь $A$ и $N$ – алфавит и множество нетерминальных символов определяемого языка.

Правая часть правила γ «устроена» более сложно. В ее состав, кроме цепочек из $(N \cup A)^*$, могут входить метасимволы.

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

Символ «*апостроф*» используется для выделения в записи правил символов терминального алфавита *A* описываемого языка.

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

*Угловые скобки* (символы ‘<’ и ‘>’) используются для выделения в тексте нетерминальных символов – нетерминальные символы заключаются в угловые скобки.

Например:

<присваивание> ::= <переменная> ':=' <выражение>

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

*Квадратные скобки* (символы ‘[’ и ‘]’) используются для указания на возможность опускать их содержимое в записи конструкции.

Например:

<ветвление> ::= '*if* ' <условие> '*then*' <оператор> ['*else*' <оператор>]

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

*Фигурные скобки* (символы ‘{’ и ‘}’) применяются для указания на возможность выписывания их содержимого нуль и более раз подряд в конструкции, заданной правилом.

Например:

<целое> ::= <цифра>{<цифра>}

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

Символ | (*вертикальная черта*, читается «или»), разделяющая цепочки символов в правой части правила (эти цепочки представляют альтернативы в описании конструкции).

Например:

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

<идентификатор> ::= <буква>{<буква>|<цифра>}

# Определение формального языка с помощью БНФ: метасимволы

***Метасимволы*** БНФ:

- ‘ (апостроф)
- **< >** (угловые скобки)
- **[ ]** (квадратные скобки)
- **{ }** (фигурные скобки)
- **( )** (круглые скобки)
- **|** (вертикальная черта)

В расширенной БНФ альтернативы могут быть «локализованы» – быть частью правила, если эта «часть» заключена в скобки.

*Круглые скобки* (символы ‘(’ и ‘)’) в расширенной нотации используются для группировки нескольких вариантов определения понятия (нескольких альтернатив).

Например:

<сравнение> ::=
<выражение> ('<' | ‘<=' | '=' | ‘<>' | ‘>=' | '>') <выражение>

# Определение формального языка с помощью БНФ: примеры

В различных источниках для выделения символов языка (терминальных и нетерминальных) используются различные способы. Например:

<условный оператор> ::=
***if*** <условие> ***then*** <оператор> |
***if*** <условие> ***then*** <оператор> ***else*** <оператор>

или

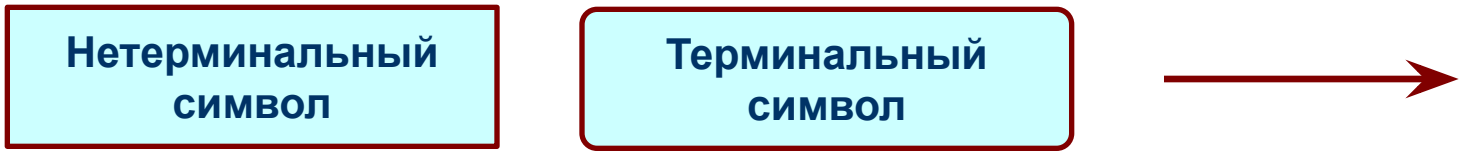
<условный оператор> ::=
*if* <условие> *then* <оператор>
[ *else* <оператор> ]

Здесь *if*, *then*, *else* ∈ *A* – терминальные символы, <условный оператор>, <условие>, <оператор> ∈ *N* – нетерминальные символы (понятия).

# Определение формального языка с помощью диаграмм Вирта

*Диаграммы Вирта* – визуальный язык описания синтаксиса языков программирования.

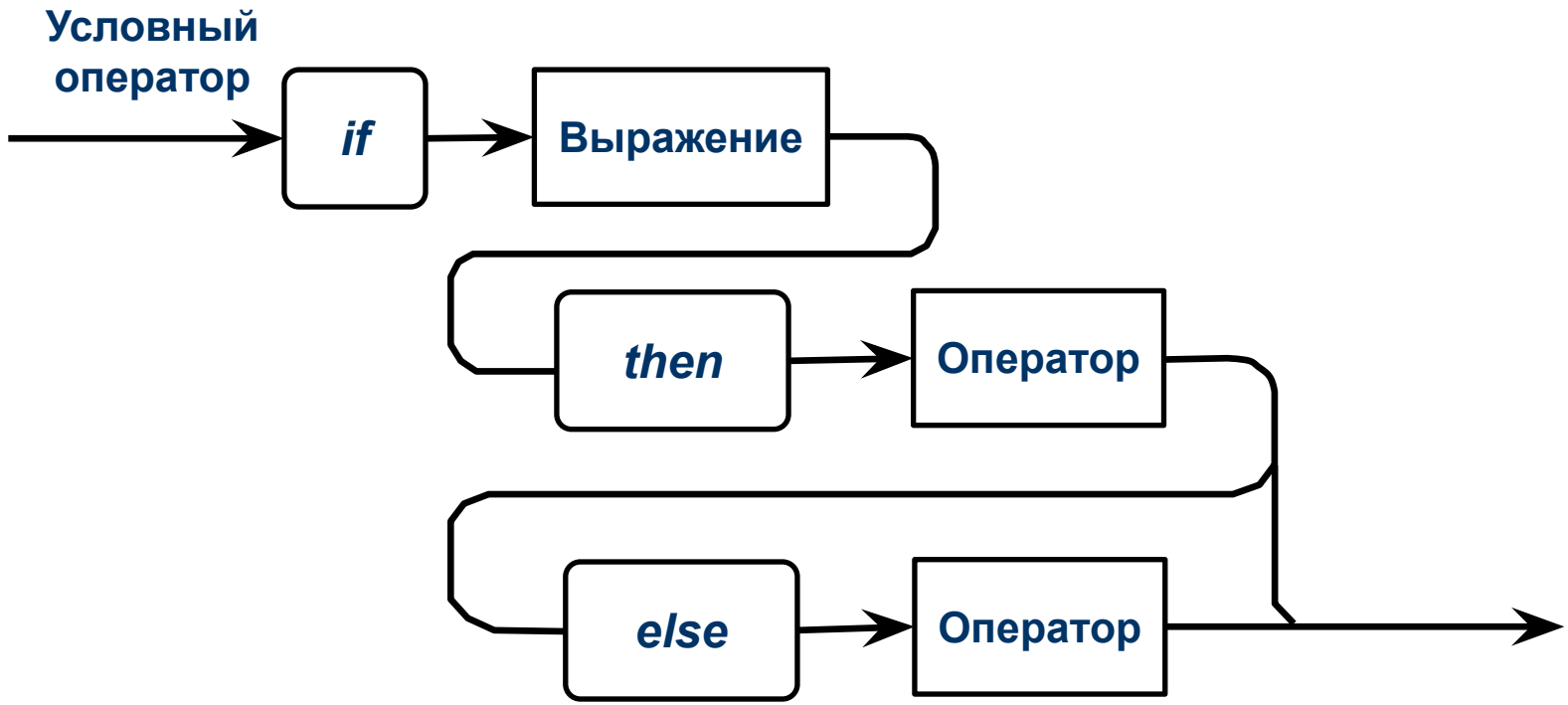
Основные элементы диаграмм, представляющие элементы языка:



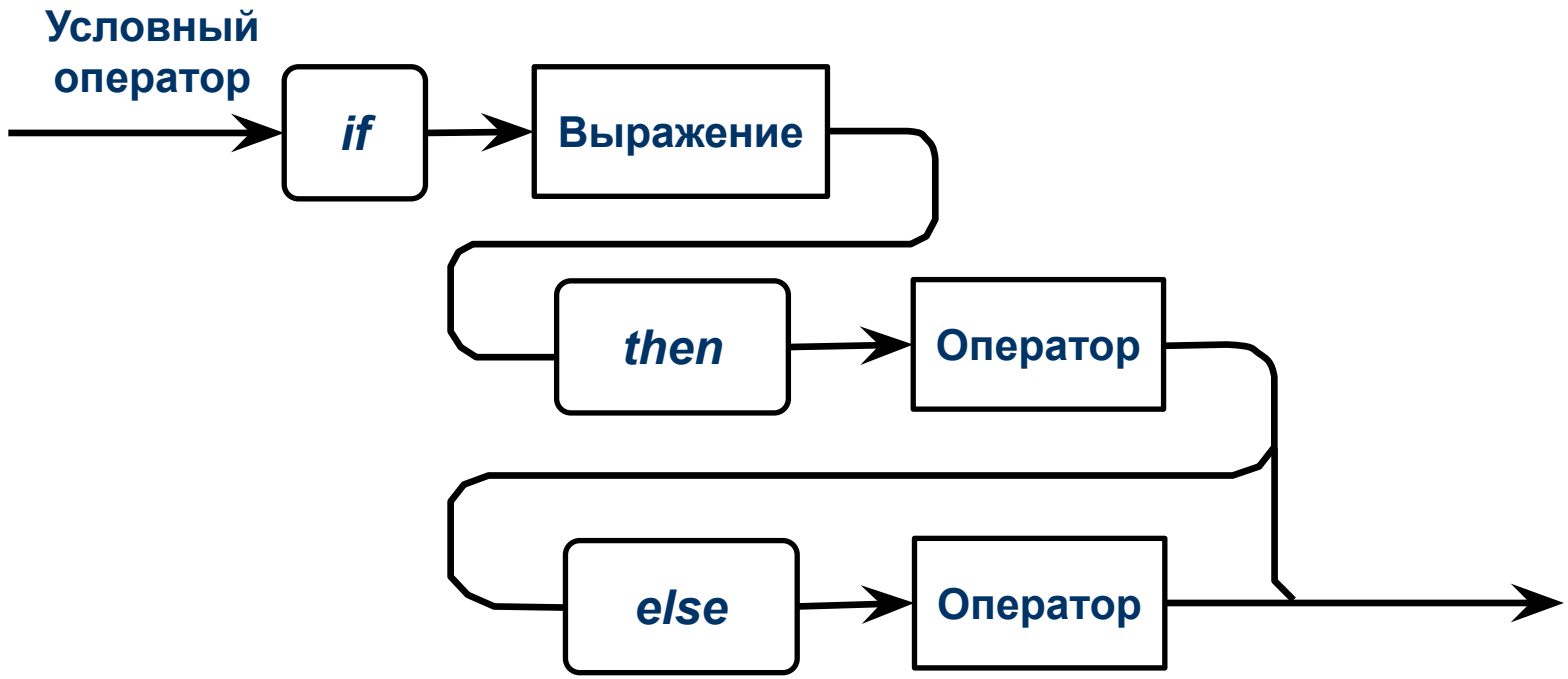

Стрелки показывают порядок следования терминальных или нетерминальных символов в конструкциях языка.

Возможные альтернативы показываются ветвлением в диаграмме.

# Определение формального языка с помощью диаграмм Вирта: примеры

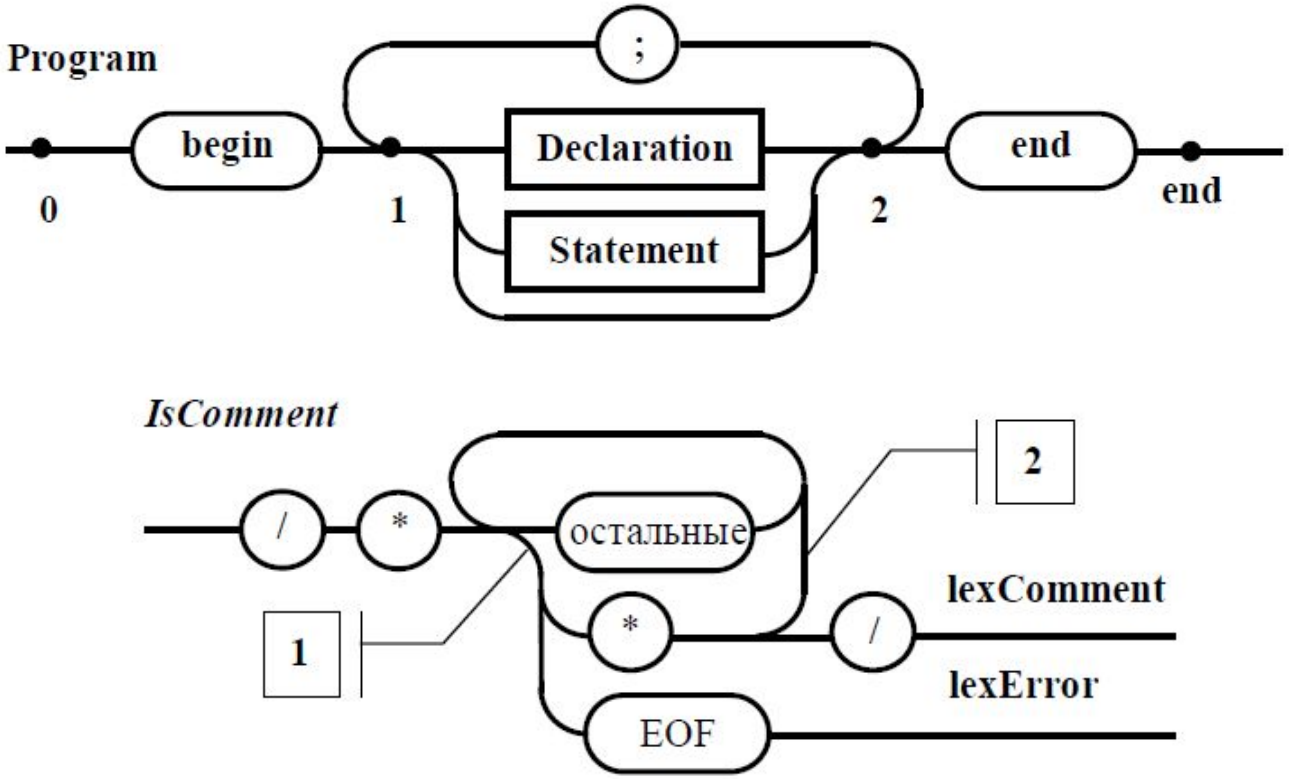


# Сравнение диаграммы Вирта и БНФ: пример




<Условный оператор> ::=
*if* <Выражение> *then* <Оператор>
[ *else* <Оператор> ]

# Определение формального языка с помощью диаграмм Вирта: примеры



# Определение формального языка: формальные грамматики

*Формальная грамматика является языком программирования синтаксиса* и, как любой язык программирования, она *не отвечает* на два вопроса:

- как написать программу под заданные требования, и
- какой содержательный смысл имеет написанная программа.

Какими свойствами обладают формальные языки, описываемые грамматиками?

# Формальные грамматики: классификация языков

Формальные языки принято классифицировать по виду правил

$$\beta ::= \gamma$$

описывающей их грамматики (*порождающей* грамматики), т.е. по виду цепочки **β** и цепочек, составляющих правую часть продукций **γ**.

Хомский ввел следующие классы грамматик:

0) без ограничений на вид правил;

1) цепочка **β** имеет вид $vBw$, где $B \in N$, $v, w \in A^*$ – произвольные цепочки, состоящие только из терминальных символов, возможно, пустые, а цепочка **γ** имеет вид $v\alpha w$, где α – непустая цепочка;

2) цепочка **β** имеет вид $B$, где $B \in N$ – нетерминальный символ;

3) цепочка **β** имеет вид $B$, где $B \in N$ – нетерминальный символ; правая часть **γ** состоит из цепочек, имеющих вид $a$, $Ca$ или

# Формальные грамматики: классификация языков

Класс 1 называется классом *контекстно-зависимых* языков, так как определение понятия *B* может зависеть от контекста – от окружающих его цепочек *v* и *w*.

Класс 2 называется классом *контекстно-свободных* языков, *КС-языков*, так как определение понятия *B* не зависит от контекста.

Класс 3 называется классом *регулярных* или *автоматных* языков. Первое название связано с тем, что все цепочки языка класса 3 могут быть заданы с помощью так называемых регулярных выражений (будут определены ниже). Второе название связано с задачей распознавания: принадлежность (или непринадлежность) цепочки терминальных символов языку класса 3 может быть проверена с помощью конечного автомата.

# Формальные грамматики: классификация языков – пример 1

Алфавит терминальных символов $A$ = {0, 1, $a$, $b$}, множество нетерминальных символов $N$ = {$S$, $B$, $C$, $D$} и $S$ - начальный нетерминальный символ грамматики. Множества правил грамматик обозначим, соответственно, $P1$, $P2$, $P3$.

$P1$ :

1) $S ::= DBC$

2) $aBb ::= aCb$

3) $0B1 ::= 0bD1 \mid 0Ca1$

4) $C ::= b \mid 1 \mid aC$

5) $D ::= a \mid 0$

# Формальные грамматики: классификация языков – пример 1

Алфавит терминальных символов $A = \{0, 1, a, b\}$, множество нетерминальных символов $N = \{S, B, C, D\}$ и $S$ - начальный нетерминальный символ грамматики. Множества правил грамматик обозначим, соответственно, $P1$, $P2$, $P3$.

$P1$ :

1) $S ::= DBC$

2) $aBb ::= aCb$

3) $0B1 ::= 0bD1 \mid 0Ca1$

4) $C ::= b \mid 1 \mid aC$

5) $D ::= a \mid 0$

**Множество $P1$ задает контекстно-зависимую грамматику**

# Формальные грамматики: классификация языков – пример 1

Алфавит терминальных символов $A$ = {0, 1, $a$, $b$}, множество нетерминальных символов $N$ = {$S$, $B$, $C$, $D$} и $S$ - начальный нетерминальный символ грамматики. Множества правил грамматик обозначим, соответственно, $P1$, $P2$, $P3$.

$P2$ :

1) $S ::= BC \mid DS$

2) $B ::= aDC \mid 1$

3) $C ::= 01D$

4) $D ::= b \mid 1C$

**Множество *P2* задает контекстно-свободную грамматику**

# Формальные грамматики: классификация языков – пример 1

Алфавит терминальных символов $A = \{0, 1, a, b\}$, множество нетерминальных символов $N = \{S, B, C, D\}$ и $S$ - начальный нетерминальный символ грамматики. Множества правил грамматик обозначим, соответственно, $P1$, $P2$, $P3$.

$P3$ :

1) $S ::= aB \mid bC$

2) $B ::= 0 \mid 1 \mid aC$

3) $C ::= 1D$

4) $D ::= a \mid 0$

**Множество *P3* задает автоматную грамматику**

# Формальные грамматики: классификация языков – пример 2

Примеры – грамматики целых чисел.

Рассмотрим грамматики, задающие правила записи целых десятичных чисел – язык целых десятичных чисел.

Все грамматики имеют один и тот же алфавит:

$$A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

# Формальные грамматики: классификация языков – пример 2

*Вариант* 1.

Множество нетерминальных символов:

$$N = \{S, T, F\},$$

(здесь для сокращения записи введены следующие обозначения нетерминальных символов: *S* – целое число со знаком, *T* – целое число без знака, *F* – цифра).

Множество продукций:

$P = \{S::=T|+T|–T,$
$T::=F|FT,$
$F::=0|1|2|3|4|5|6|7|8|9\},$

Эта грамматика является контекстно-свободной (КС-грамматикой), но не автоматной, т.к. имеется правило $T::=F|FT$.

# Формальные грамматики: классификация языков – пример 2

*Вариант* 2.

Множество нетерминальных символов:

$$N = \{S,\ T\},$$

(здесь для сокращения записи введены следующие обозначения нетерминальных символов: $S$ – целое число со знаком, $T$ – целое число без знака).

Множество продукций:

$P = \{S ::= T \mid +T \mid -T,$
$T ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0T \mid 1T \mid 2T \mid 3T \mid 4T \mid 5T \mid 6T \mid 7T \mid 8T \mid 9T\},$

Эта грамматика является регулярной праволинейной.

# Формальные грамматики: классификация языков – пример 2

*Вариант* 3.

Рассмотрим пример эквивалентной леволинейной регулярной грамматики. Множество нетерминальных символов:

$$N = \{S, Sign\},$$

(здесь для сокращения записи введены следующие обозначения нетерминальных символов: *S* – целое число со знаком, *T* – целое число без знака).

Множество продукций:

*P* = {<*Sign*>::=+|–|λ,

*S*::= <*Sign*>0|<*Sign*>1|<*Sign*>2|<*Sign*>3|<*Sign*>4|
<*Sign*>5|<*Sign*>6|<*Sign*>7|<*Sign*>8|<*Sign*>9|
*S*0|*S*1|*S*2|*S*3|*S*4|*S*5|*S*6|*S*7|*S*8|*S*9|},

Эта грамматика является регулярной.