

# Языки программирования высокого уровня

**Пример:** сложить содержимое двух разных ячеек и результат поместить в третью.

*(вспоминаем формат машинной команды (лекция 5))*

Код Операции	О	п	е	р	а	н	д
--------------	---	---	---	---	---	---	---

(пусть адреса ячеек равны 51, F2 и 93)

***Машинный код:***

```
0001 0001 0101 0001
0001 0010 1111 0010
0101 0011 0001 0010
0011 0011 1001 0011
```

***Мнемокод:***

```
LD    R1, 51
LD    R2, F2
ADD   R3, R1, R2
ST    R3, 93
```

***Язык программирования в/у:*** `c=a+b`

Пример иллюстрирует использование языков программирования трех поколений:

- машинные коды – первое поколение,
- языки ассемблера – второе поколение,
- языки высокого уровня – третье поколение .

Языки программирования третьего поколения используют ***машинно-независимые примитивы высокого уровня.***

Эти примитивы по смыслу эквивалентны основным алгоритмическим конструкциям (см. *Лекция 1*), имеют ту же ***семантику*** и называются ***операторами***.

Программа, преобразующая программу высокого уровня в машинный код, называется ***транслятором*** (в данном случае ***компилятором***).

Например, примитиву «присвоение»  $\langle \text{имя} \rangle = \langle \text{выражение} \rangle$  в частном случае  **$a=7$**  компилятор ставит в соответствие следующую цепочку машинных команд (виртуальная машина из лекции 5):

**0010 0000 0000 0111**  
**0011 0000 1111 0010**

$\left( \begin{array}{ll} \text{LD} & \text{R0, [7]} \\ \text{ST} & \text{R0, F2} \end{array} \right)$

- $\left( \begin{array}{l} 1. \text{ Загрузка в нулевой регистр битовой комбинации 0111.} \\ 2. \text{ Сохранение содержимого нулевого регистра} \\ \text{в ячейке памяти с адресом F2.} \end{array} \right)$

## Типы трансляторов:

*Компилятор*

*Машинный код,  
автокод  
(мнемокод)*

Исходный код

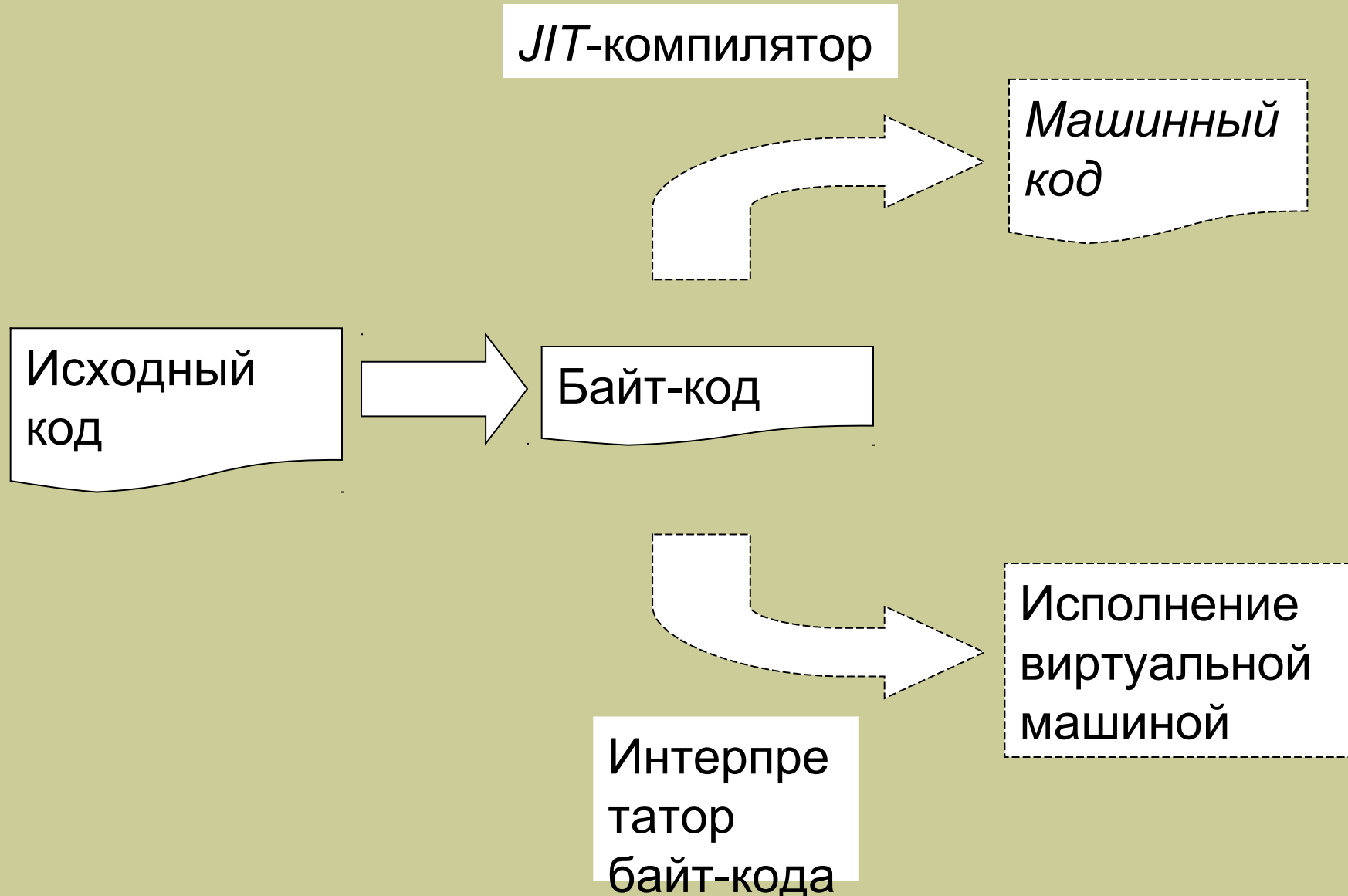
Объектный код

*Внутреннее  
представление*

*Исполнение*

*Интерпретатор*

Трансляторы, независимые от платформы и архитектуры.  
*Java, Perl, .Net*-языки и др.:



**Процесс компиляции (основные этапы):**

***Препроцессор:*** подстановка заголовочных файлов и макросов.

***Лексический анализ:*** преобразование последовательностей символов в лексемы программы (выявление «орфографических» ошибок).

***Синтаксический анализ:*** структурирование последовательности лексем во внутреннее представление программы в виде дерева синтаксического разбора.

***Генерация кода:*** преобразование внутреннего представления в объектный (целевой) код.

***Ассемблирование:*** преобразование автокода в машинный код.

***Компоновка:*** сборка всех объектных модулей и модулей, содержащих системные вызовы данной ОС.

***Исполнение:*** выполнение инструкций объектного языка.

После подстановки заголовочных файлов и макросов и после удаления комментариев текст программы высокого уровня представляет собой строку допустимых символов.

Это могут быть символы латинского алфавита, цифры, знаки арифметических операций и т.д.

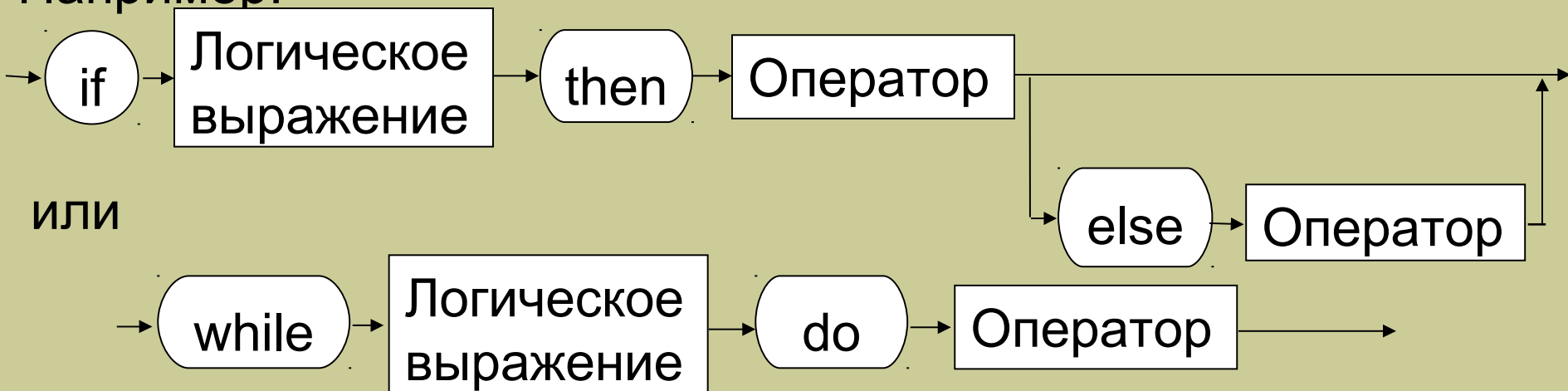
Последовательность символов, объединённых в единые с точки зрения синтаксиса сущности называются *лексемами*.

Это могут быть числа (например 328 или 3.141592, так называемые **числовые литералы**), ключевые слова (например, *if* или *while*), символы, заключённые в одинарные кавычки (например 'Q' или '9') – **символьные литералы**, имена переменных или знаки арифметических операций и т.п.

**На этапе лексического анализа выявляются недопустимые символы и выделяются лексемы языка.**

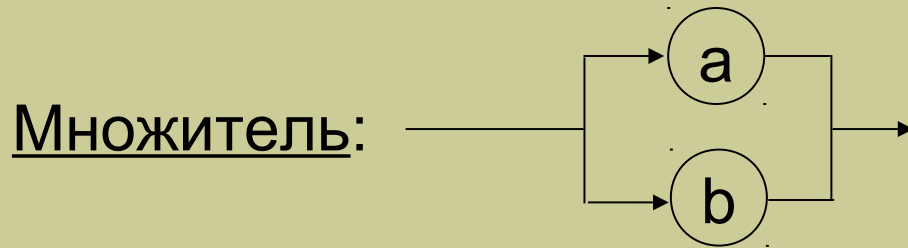
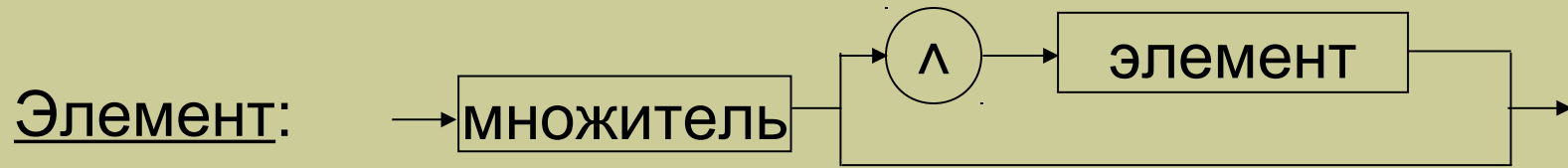
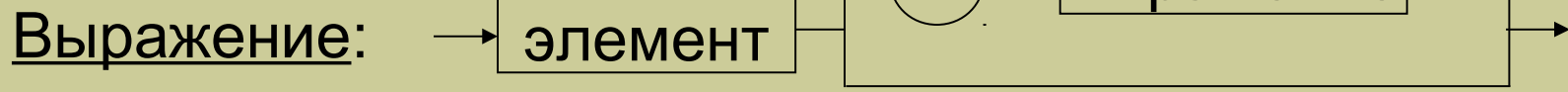
Лексемы должны располагаться в тексте в правильной последовательности, в соответствие с **синтаксисом** языка. Синтаксис языка программирования задается **грамматикой**. Правила грамматики могут представляться графически в виде **синтаксических диаграмм**.

Например:



В овальных контурах – лексемы, прямоугольники изображают другие синтаксические конструкции, требующие развертывания в синтаксическую диаграмму.





С помощью этой диаграммы можно генерировать и распознавать логические выражения содержащие переменные  $a$  и  $b$  и знаки логических операций дизъюнкции и конъюнкции, без скобок и знака отрицания.

*Упражнение 1:* достроить диаграмму для генерации выражений со скобками и знаком операции отрицания.

*Упражнение 2:* построить синтаксическую диаграмму арифметических выражений.

Набор лексем и грамматика, задающая синтаксическую структуру языка программирования, являются основой его описания (*спецификации*).

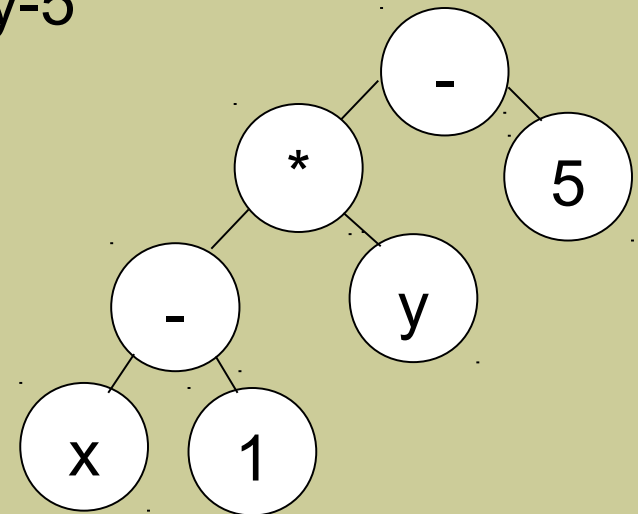
На стадии **синтаксического разбора** проверяется согласие последовательностей лексем с синтаксисом и правильные последовательности лексем структурируются во внутреннее представление программы в виде дерева синтаксического разбора.

линейное представление

арифметического выражения:  $(x-1)*y-5$

иерархическое представление

арифметического выражения:



# Классификация языков программирования

## Парадигмы программирования (подходы к программированию)

- I. Императивное программирование (*FORTRAN, Pascal, C* )
- II. Декларативное (*Prolog, SQL*)
- III. Объектно-ориентированное (*C++, Java, C#*)

## Стили программирования

- I. Неструктурное программирование (*FORTRAN, BASIC (goto)*)
- II. Структурное программирование (Э. Дейкстра, Н. Вирт) (*Pascal, C*)

## **Технологии программирования:**

- I. Многопоточное программирование
- II. Событийно-управляемое программирование  
(*функции обратного вызова; делегаты, события*)
- III. Многопроцессное программирование
  - Параллельное программирование
  - Распределенное программирование

## **Технологии разработки программ:**

Визуальное программирование (*MS Visual Studio, DELPHI, C/C++ Builder, Qt3/4 Designer* )