Лекция 6.

Общая структура языков программирования

Язык программирования — формальная знаковая система для записи компьютерных программ

```c
#include <locale.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    setlocale(LC_CTYPE, "Russian_Russia.1251");
    int N, k, n, m;
    printf("Введите N: ");
    scanf_s("%d", &N);
    for (k = 5; k < N; k++)
    {
        for (n = 3; n < N; n++)
            for (m = 4; m < N; m++)
                if (n * n + m * m == k * k && n < m)
                    printf("%d\t%d\t%d\n", n, m, k);
    }

    _getch();
    return 0;
}
```

Бронштейн **Любитель**

Сочи, 1950 г.

1.	e2—e4	e7—e5
2.	d2—d4	e5 : d4
3.	Фd1 : d4	Кb8—c6
4.	Фd4—a4	Кg8—f6
5.	Кb1—c3	d7—d5
6.	Сc1—g5	d5 : e4
7.	Кc3 : e4	Фd8—e7?
8.	0—0—0!	Фe7 : e4??
9.	Лd1—d8+!!	Крe8 : d8
10.	Фa4 : e4	Черные сдались.

1) $A = \frac{\sqrt{k}-k^2+2{,}3m}{m^3-5{,}7\sqrt{|k-m|}} + \frac{e^{-3m}}{(k^4-12{,}1)^2}$

2) $C = \frac{|k-7m^2|+e^{k-a}}{mk} - \frac{\sin a^2}{|a-4{,}2k|}$

3) $B = \sqrt{\frac{(a-k)^2}{(k-m)^2} + \frac{(m-a)^2}{(c-a)^2}}$

4) $K = \frac{1}{6}\pi h^3 + \frac{1}{2}\pi(r_1^2 + r_2^2)h$

5) $H = \left(\frac{\sqrt[4]{ax^3}-\sqrt[4]{a^3x}}{\sqrt{a}-\sqrt{x}} + \frac{1+\sqrt{ax}}{\sqrt[4]{ax}}\right)^{-2} \sqrt{1 + 2\sqrt{\frac{a}{x}} + \frac{a}{x}}$

Уравнения реакций	Типы	Условия
$H_2SO_4+Zn=ZnSO_4+H_2\uparrow$	Замещение	Без нагревания
$H_2SO_4+CuO=CuSO_4+H_2O$	Обмен	Нагревание
$2HCl+CuO=CuCl_2+H_2O$	»	Без нагревания
$H_2SO_4+2KOH=K_2SO_4+2H_2O$	»	» »
$2HCl+Cu(OH)_2=CuCl_2+2H_2O$	»	» »
$H_2SO_4+Ba(NO_3)_2=BaSO_4\downarrow+2HNO_3$	»	» »
$2HCl+CaCO_3=CaCl_2+H_2CO_3$ ($H_2CO_3 \rightarrow H_2O + CO_2\uparrow$)	»	» »

Сравнение ЯП с естественными яыками

```c
#include <locale.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    setlocale(LC_CTYPE, "Russian_Russia.1251");
    int N, k, n, m;
    printf("Введите N: ");
    scanf_s("%d", &N);
    for (k = 5; k < N; k++)
    {
        for (n = 3; n < N; n++)
            for (m = 4; m < N; m++)
                if (n * n + m * m == k * k && n < m)
                    printf("%d\t%d\t%d\n", n, m, k);
    }

    _getch();
    return 0;
}
```

КОМПАНИЯ ЭТНИЧЕСКОГО МЕНЬШИНСТВА/ALLISON: 30 90-224, 3/12/91

FADE IN (изображение из затемнения):

Звучит бодрая оптимистичная музыка. В этот же момент черный экран становится ЯРКИМ И РАЗНОЦВЕТНЫМ. После того, как установлен такой яркий пестрый фон, "МРАМОРНАЯ" ГРАФИЧЕСКАЯ РАМКА ВПЛЫВАЕТ В ЭКРАН. Внутри рамки - ЗАСТЫВШЕЕ ИЗОБРАЖЕНИЕ выбранного GTL ПОСТАВЩИКА женского/этнического предприятия, в ее рабочей обстановке.
Когда рамка достигает центра экрана, изображение поставщика ПРИХОДИТ В ДВИЖЕНИЕ. Она смотрит на интервьюера, остающегося за кадром. Она говорит интервьюеру (в свободной форме, не по сценарию) примерно такие слова:

ПОСТАВЩИК #1
Я работала с GTL несколько раз. Они искренни и честны в своей поддержке женских и этнических предприятий, и они действительно стремятся помочь..

Рамка с изображением этой женщины ПЕРЕВОРАЧИВАЕТСЯ НА ЭКРАНЕ, На ее обратной стороне - ВТОРОЙ ПОСТАВЩИК. Он говорит:

ПОСТАВЩИК #2
Конечно, здесь нужно работать. Нужно знать, чем ты занимаешься. Но GTL всегда готово дать тебе возможность доказать это.

Рамка ПЕРЕВОРАЧИВАЕТСЯ СНОВА. На ее обратной стороне - образ начальника ОТДЕЛА КАДРОВ МАЙКА АРБЕЛА. Он говорит:

АРБЕЛ
Мы не раздаем возможности людям, только потому, что они женщины или принадлежат к этническим меньшинствам. Мы даем возможности людям, потому что они хорошо делают свою работу.

Рамка ПЕРЕВОРАЧИВАЕТСЯ СНОВА. Вместе с переворотом она превращается как бы в СЛИТОК ЗОЛОТА, НА КОТОРОМ ЧЕКАНКОЙ НАПИСАНО: "GTL - партнерства с женскими и этническими компаниями"

А затем слиток диагонально перекрывают написанные яркими курсивными буквами слова:
"Делая бизнес с лучшими!"

МУЗЫКА ДОСТИГАЕТ ПИКА; ФИНАЛЬНЫЙ ГРОМКИЙ АККОРД, мы...

FADE OUT (затемнение):

Сравнение ЯП с естественными яыками

Язык программирования — формальная знаковая система, предназначенная для записи (по определенным правилам) компьютерных программ.

Сходство с естественными языками:

- Текст состоит из слов, составленных из символов. Слова объединяются в "предложения".
- **Синтаксис (грамматика)**, определяющий структуру текста, отличается от **семантики**, определяющей смысл.
- Есть слова с предопределенным смыслом (if, do, while, ...). Можно определять собственные слова.
- Не зная алфавита и правил построения предложений, то есть грамматики, нельзя изучить языки

Синтаксис, грамматика и семантика

Глокая куздра штеко будланула бокра и курдячит бокрёнка

Синтаксис, грамматика и семантика

Глокая куздра штеко будланула бокра и курдячит бокрёнка

1. Тигрица быстро победила буйвола и грызет буйволенка.
2. Слон весело разбил бочку и катает бочонок.

Синтаксис, грамматика и семантика

Глокая куздра штеко будланула бокра и курдячит бокрёнка

1. Тигрица быстро победила буйвола и грызет буйволенка.
2. Слон весело разбил бочку и катает бочонок.

Синтаксис, грамматика и семантика

Глокая куздра штеко будланула бокра и курдячит бокрёнка

1. Тигрица быстро победила буйвола и грызет буйволенка.
2. Слон весело разбил бочку и катает бочонок

Грамматика определяет правила (законы) построения корректных предложений из произвольных слов.

Похоже на математические законы в формулах.

Z=X+Y

1. Z=10, X=2, Y=8
2. Z=10, X=2, Y=120

Сравнение ЯП с естественными яыками

- ЯП не позволяют выражать чувства или мысли. Они определяют объекты, представимые в компьютере, и задачи, выполнимые над этими объектами.
- ЯП выигрывают в точности. Синтаксис и семантика ЯП должны быть определены совершенно строго.

ЯП влияет, независимо от нашего желания, на наш способ мышления.

Естественные языки используются в комментариях и служат основой для построения идентификаторов (имен переменных, подпрограмм, методов, …).

Языки программирования: основные определения

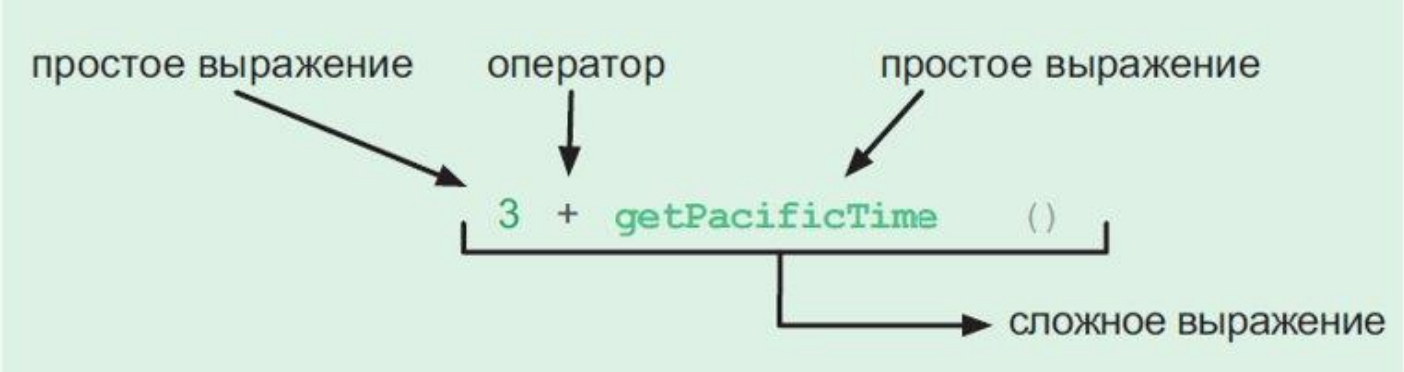
Все языки программирования управляют информацией.

- **Язык** – множество допустимых строк.
- **Значение** – представляет информацию.
- **Выражение** – создает значение. Можно написать **литерал** или вызвать **функцию**.

  3 getPacificTime()

Языки программирования: основные определения

- **Оператор** – минимальная единица программного кода, объединяет простые выражения в сложные. Операторы могут быть одиночными и блочными.



- **Инструкция** – использует значение, чтобы дать команду компьютеру.

```
write("Привет!");
```

Языки программирования: основные определения

- **Синтаксис** программы = структура и форма записи ее текста. Синтаксис задается правилами, позволяющими определить, принадлежит ли выражение/инструкция языку или нет.

```
write("Привет!");
echo "Привет!";
```

Языки программирования: основные определения

- **Синтаксис** программы = структура и форма записи ее текста. Синтаксис задается правилами, позволяющими определить, принадлежит ли выражение/инструкция языку или нет.

```
write("Привет!");
echo "Привет!";
```

- Синтаксическая единица – **лексема** (слово).

Лексемы – последовательности символов языка, которые имеют смысл только как единое целое.

Выражение “2+3” – это не лексема, константа 23 - лексема, идентификатор `TForm` – лексема.

Языки программирования: основные определения

Семантика языка – описание смысла языковых выражений.

Семантику составляют правила, определяющие, к выполнению каких действий приведет то или иное выражение.

Разница между синтаксисом и семантикой

Оператор присваивания в Pascal

x:=y+z;

Синтаксически это правильное выражение?

Разница между синтаксисом и семантикой

Оператор присваивания в Pascal

x:=y+z;

Синтаксически это правильное выражение?

Может ли оно быть семантически неверным?

Разница между синтаксисом и семантикой

Оператор присваивания в Pascal

x:=y+z;

Синтаксически это правильное выражение?
Может ли оно быть семантически неверным?

- Существуют формальные способы описания синтаксиса, позволяющие выделить отдельные синтаксические конструкции. Поэтому можно делать вывод о синтаксической корректности отдельной части выражения языка.
- Формальных правил описания семантики нет.

Лексика, синтаксис и семантика

Для описания ЯП нужно определить:

- **алфавит** – набор символов, используемый при записи программ;
- **синтаксис** – правила, задающие внешний вид программы и определяющие допустимые тексты в языке;
- **семантику** – правила, определяющие действия, которые выполнит компьютер под управлением программы.

Также нужно представить **прагматику** - пояснение, для чего нужна та или иная языковая конструкция.

Спецификация языка программирования

Спецификация ЯП - это предмет документации, который определяет ЯП, чтобы пользователи и разработчики могли согласовывать, что означают программы на данном языке.

https://262.ecma-international.org/13.0/#sec-ecmascript-language-expressions

https://learn.microsoft.com/en-us/powershell/scripting/lang-spec/chapter-02?view=powershell-7.2

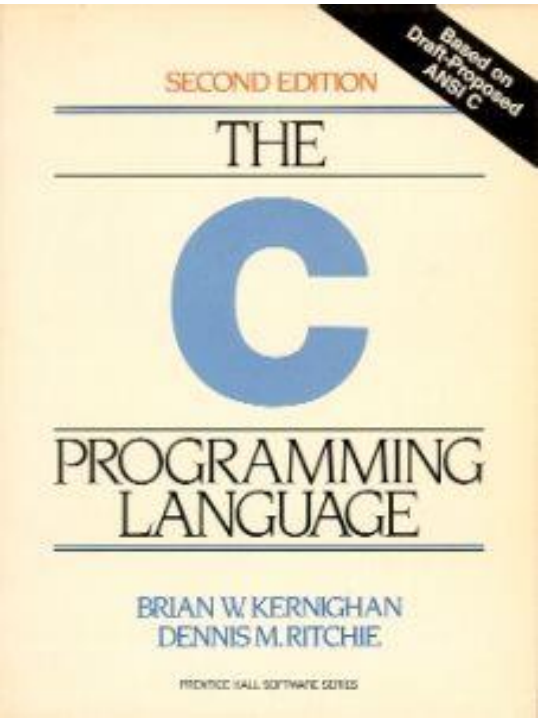
https://docs.python.org/3/contents.html

Язык PHP получил спецификацию в 2014 году, после 20 лет использования.

Спецификация языка программирования

- Синтаксис обычно описывается формально.
- Семантические определения могут быть написаны на естественном языке.

Спецификация и реализация языка программирования.



Анализ текста программы

Текст:

- Предложения (грамматика - синтаксис)
  - Слова (грамматика - лексика)
    - Буквы (алфавит)

- **Лексический анализ** – разделение выражения на отдельные лексемы. Также анализатор может удалять из выражения комментарии, лишние разделители и т.д.
- **Синтаксический анализ** – проверка правильности выражений, составленных из лексем.

Способы описания правил грамматики ЯП

Инструменты для описания конструкций ЯП:

- **Металингвистические формулы, текстовые формы Бэкуса-Наура (БНФ).** Предложена Джоном Бэкусом и модифицирована Питером Науром (для описания Алгола).
- **Визуальные синтаксические диаграммы.**

Джон Бэкус (*John Backus*, 3 декабря 1924 года — 17 марта 2007 года) — американский учёный в области информатики. Он был руководителем команды, разработавшей первый высокоуровневый язык программирования ФОРТРАН, изобретателем формы Бэкуса — Наура, одной из самых универсальных нотаций, используемых для определения синтаксиса формальных языков

Петер Наур (*Peter Naur*; 25 октября 1928, Фредериксберг — 3 января 2016, Херлев) — датский учёный в области информатики, один из пионеров компьютерной науки. Более всего известен как один из разработчиков первого языка структурного программирования Алгол 60 и, совместно с Бэкусом, как изобретатель формы Бэкуса — Наура.

Конструкции, используемые в формулах БНФ

- **Терминальные символы** — это отдельные символы или их последовательности, имеющие конкретные известные значения и являющиеся неразрывным целым, не сводимым к другим символам.
- **Нетерминальные символы (метапеременные)** – элементы языка, не имеющие заранее известного значения (формулы, команды и т.п.), которые раскрываются правилам грамматики.
- **::=** "есть по определению"
- **|** "или"
- **{a}** повтор a 0 или более раз
- **[a]** a входит 0 или 1 раз

Формулы БНФ

<идентификатор> ::= выражение

- идентификатор – имя метапеременной
- выражение – комбинация терминальных символов и метапеременных.

Примеры БНФ

<Separator> ::='.' (разделитель для дробной части числа)

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

Примеры БНФ

<Separator> ::='.' (разделитель для дробной части числа)

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

<Unsigned> ::= <Digit>|<Digit><Unsigned> (целое без знака)

<Unsigned> ::= <Digit>{<Digit>} (целое без знака)

Примеры БНФ

<Assignment> ::=<Var> ':=' <Expression> (синтаксис оператора присваивания в Pascal)

<if> ::= 'if' <Condition> 'then' <Operator> ['else' <Operator>]

<for> ::= 'for' <Var> ':=' <Expression> ('to' | 'downto') <Expression> 'do' <Operator>

Примеры БНФ

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

<Идентификатор> ::= <буква> | <идентификатор><буква> |
<идентификатор><цифра>

Основы БНФ

Задача. Описать с помощью БНФ синтаксис вещественного числа.

Основы БНФ

Задача. Описать с помощью БНФ синтаксис вещественного числа.

- Перед числом может стоять знак — плюс или минус.
- Затем идет одна или несколько цифр.
- Потом может следовать точка, после которой будет еще одна или несколько цифр. Затем может быть указан показатель степени "E" (большое или малое), после которого может стоять знак плюс или минус, а затем должна быть одна или несколько цифр".

Основы БНФ

Задача. Описать с помощью БНФ синтаксис вещественного числа.

- Перед числом может стоять знак — плюс или минус.
- Затем идет одна или несколько цифр.
- Потом может следовать точка, после которой будет еще одна или несколько цифр. Затем может быть указан показатель степени "E" (большое или малое), после которого может стоять знак плюс или минус, а затем должна быть одна или несколько цифр.

<Number> ::= [<Sign>] <Digit> {<Digit>}[<Separator> <Digit> {<Digit>}][<Exponent> [<Sign>] <Digit> {<Digit>}]

Основы БНФ

Задача. Описать с помощью БНФ синтаксис вещественного числа.

- Перед числом может стоять знак — плюс или минус.
- Затем идет одна или несколько цифр.
- Потом может следовать точка, после которой будет еще одна или несколько цифр. Затем может быть указан показатель степени "E" (большое или малое), после которого может стоять знак плюс или минус, а затем должна быть одна или несколько цифр.

<Number> ::= [<Sign>] <Digit> {<Digit>}[<Separator> <Digit> {<Digit>}][<Exponent> [<Sign>] <Digit> {<Digit>}]

<Digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<Sign> ::= '+' | '-'
<Separator> ::= '.'
<Exponent> ::= 'E' | 'e'

Основы БНФ

<Number> ::= [<Sign>] <Digit> {<Digit>}[<Separator> <Digit> {<Digit>}][<Exponent> [<Sign>] <Digit> {<Digit>}]

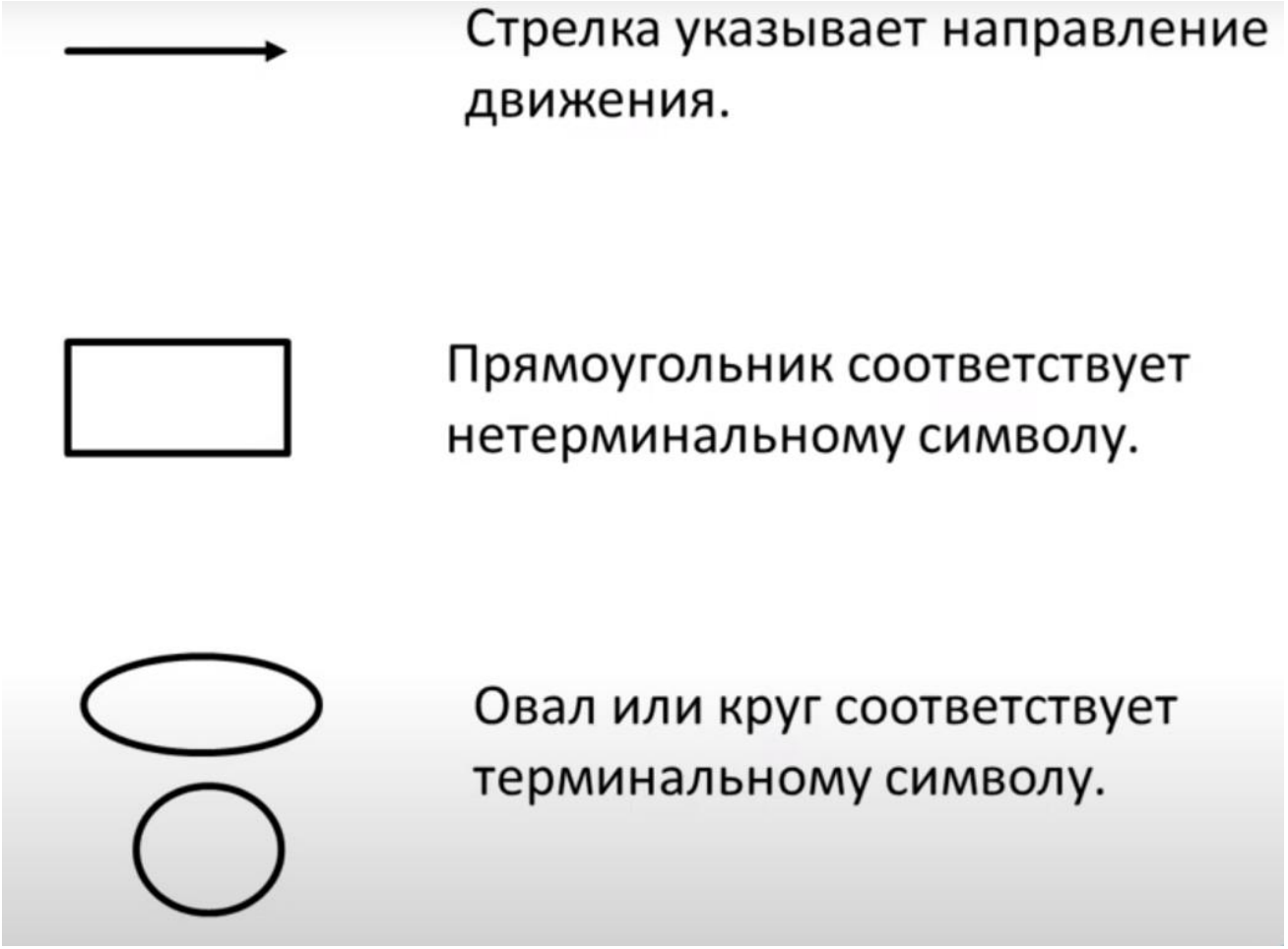
<Digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<Sign> ::= '+' | '-'
<Separator> ::= '.'
<Exponent> ::= 'E' | 'e'

Какие из записей являются вещественными числами?

- +3.14
- 3.14
- 3,14
- 4.
- -545
- 2e-5
- E3

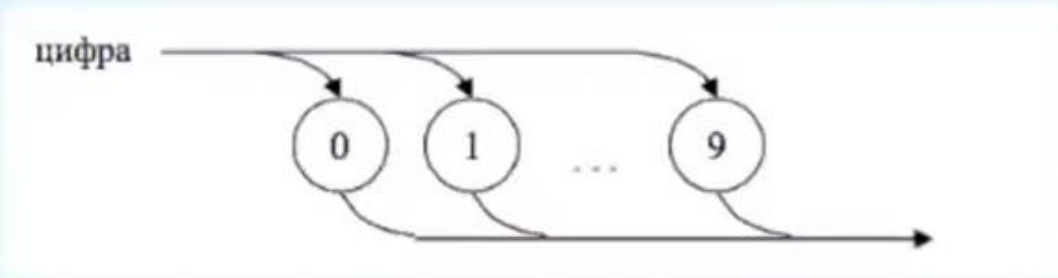
Синтаксические диаграммы

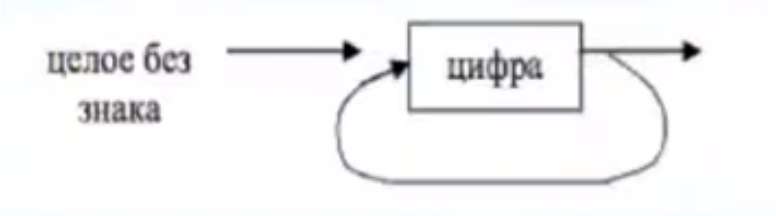
Графическое представление тех же формул и правил. Прямоугольники и овалы соединяются стрелками в нужной последовательности

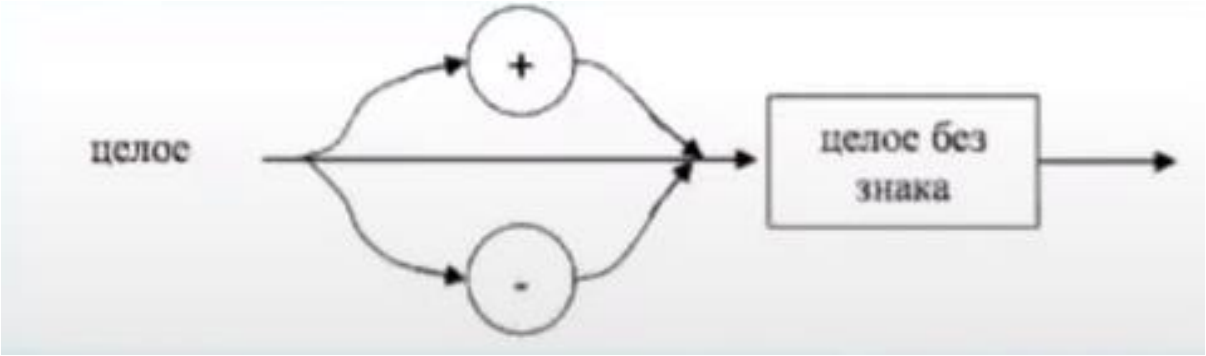


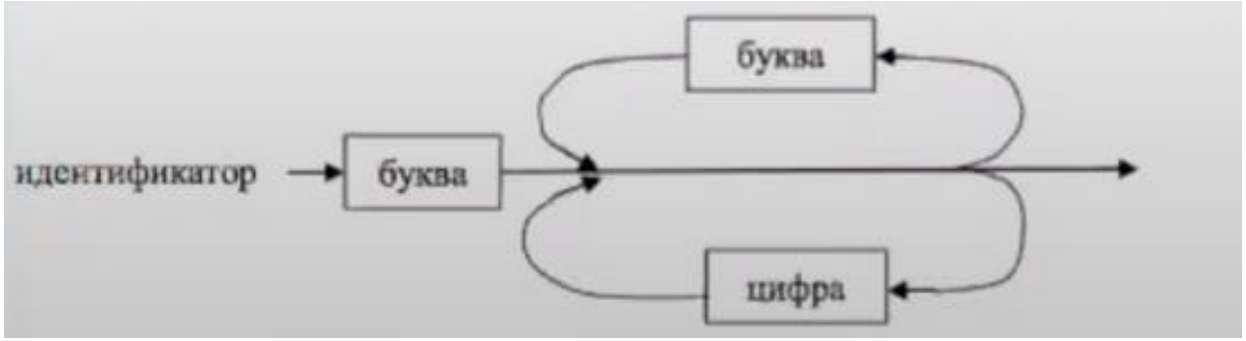
Синтаксические диаграммы

Текст удовлетворяет диаграмме, если, двигаясь по стрелкам от левого конца, мы найдем путь к правому концу.

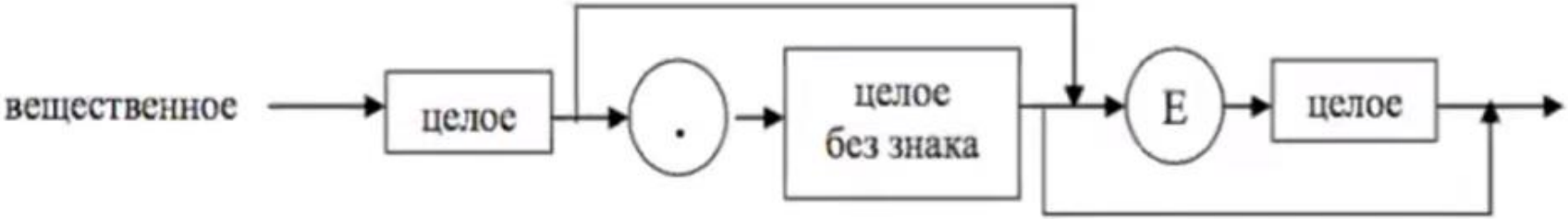








Синтаксические диаграммы



Какие из записей являются вещественными числами?

- +3.14
- 3.14
- 3,14
- 4.
- -545
- 2e-5
- E3

БНФ и анализ синтаксиса

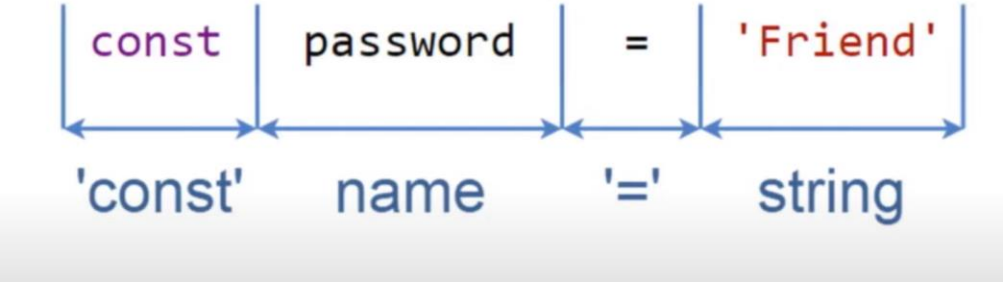
Имея описание выражений с помощью БНФ, можно писать

- **программы-лексеры** для разбора синтаксиса выражений;
- **программы-парсеры**, понимающие грамматику выбранного языка программирования.

Анализ лексики (токенизация)

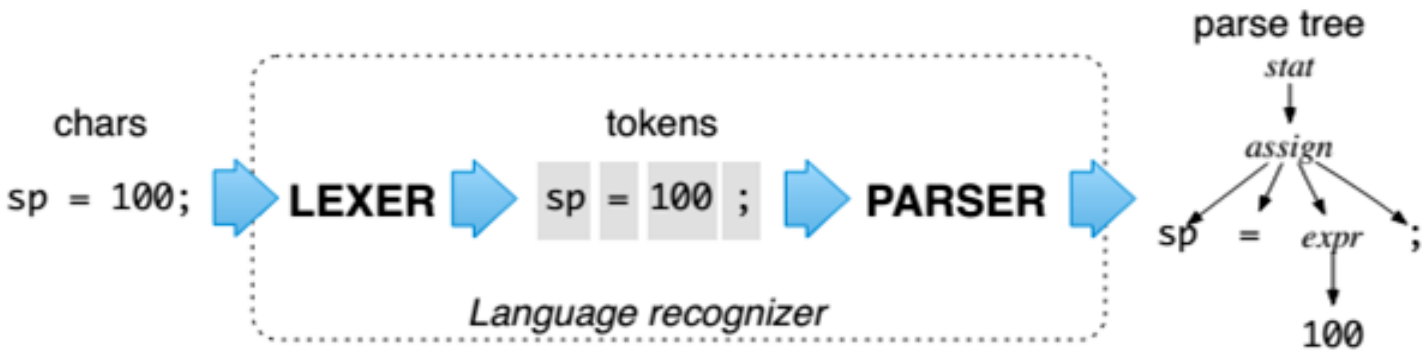
Лексер – программный модуль для разбора текста и представления его в виде массива токенов.

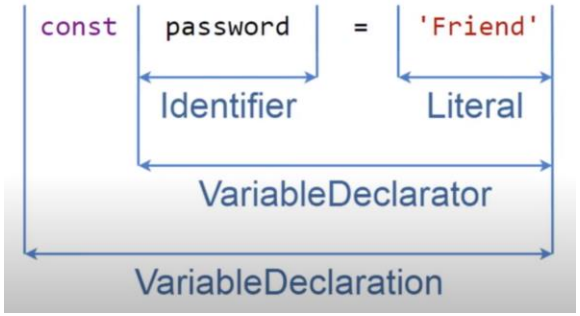
Токен – совокупность значения лексемы (значимого слова), ее типа и другой метаинформации.



Анализ синтаксиса (парсинг)

Парсер – группирует по поступающие из лексера токены в дерево разбора, отражающее иерархию элементов программы и связи между ними.



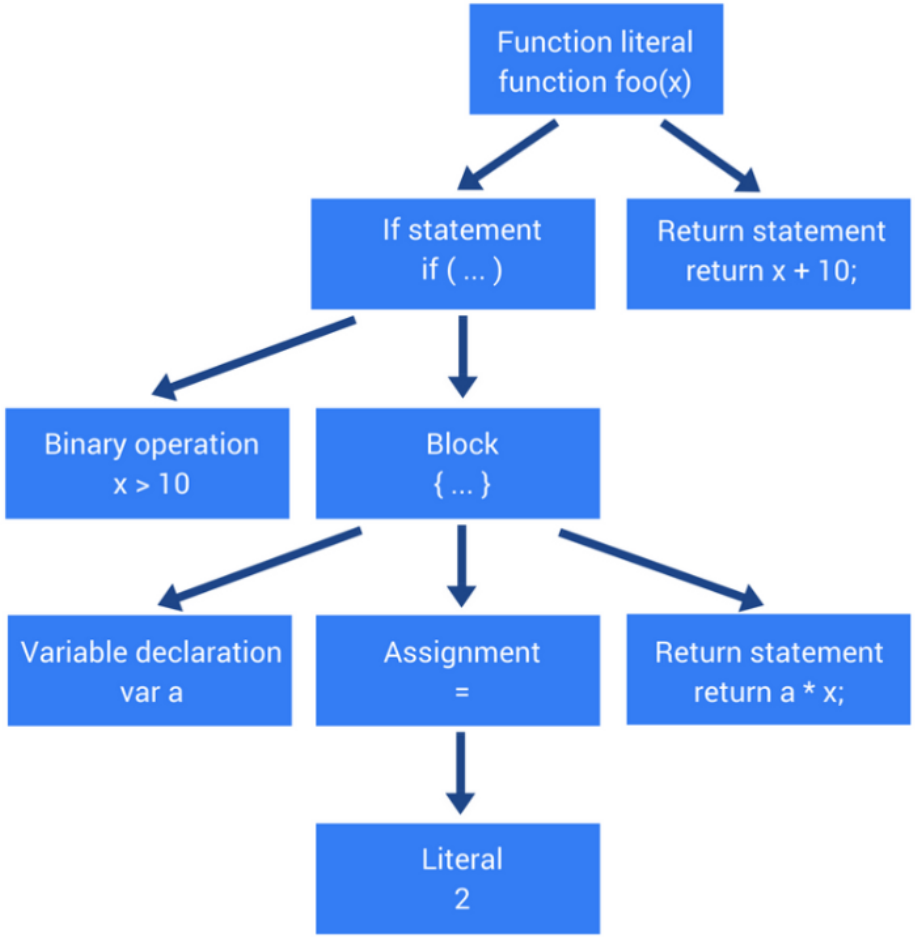


Анализ синтаксиса (парсинг)

Абстрактное синтаксическое дерево (AST) — дерево разбора, из которого удалены незначимые токены (скобки, запятые, …).

Это структурное представление исходного кода в виде дерева, где каждая вершина обозначает различные типы конструкций языка (выражение, переменную, оператор и т.п.)

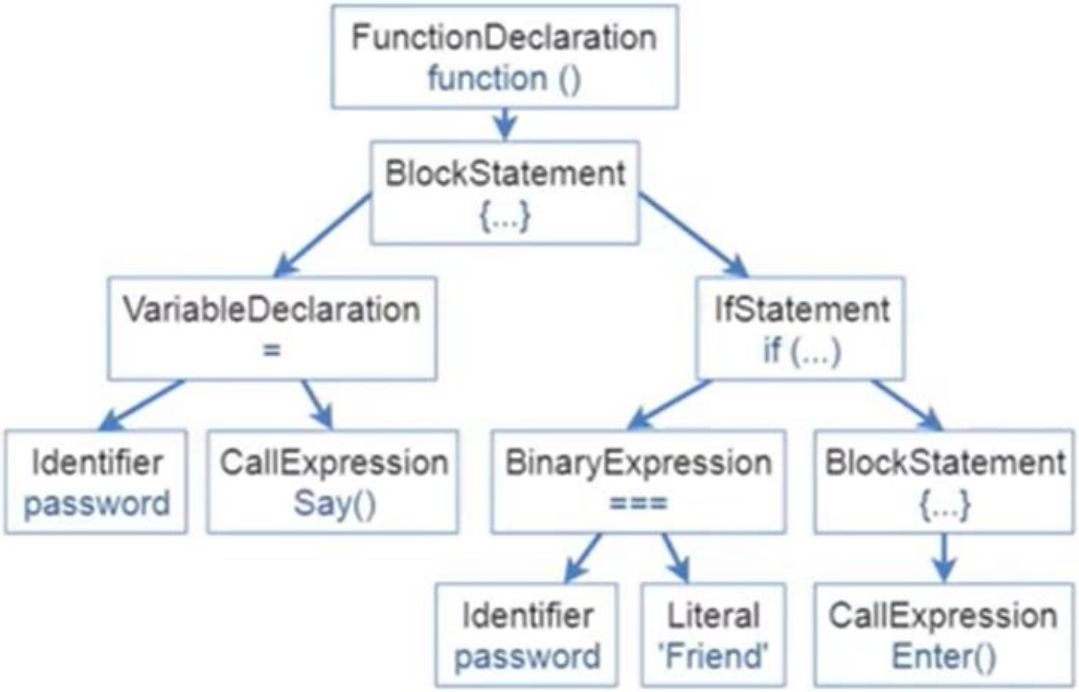
```
function foo(x) {
    if (x > 10) {
        var a = 2;
        return a * x;
    }

    return x + 10;
}
```



Abstract Syntax Tree

- Семантический анализ, генерация кода для интерпретатора или компилятора
- Статический анализ текста в IDE
- Линтинг
- Транспиляция

```
function DoorsCheck() {
  const password = Say();
  if (password === 'Friend') {
    Enter();
  }
}
```



AST для JavaScript

https://astexplorer.net/

```
function foo(x) {
    if (x > 10) {
        var a = 2;
        return a * x;
    }

    return x + 10;
}
```

AST Explorer | Snippet | JavaScript | acorn | Transform | default

```
1 function foo(x) {
2     if (x > 10) {
3         var a = 2;
4         return a * x;
5     }
6
7     return x + 10;
8 }
```

Tree | JSON

Autofocus | Hide methods | Hide empty keys | Hide location data | Hide type

```
- Program {
    type: "Program"
    start: 0
    end: 104
  - body: [
    - FunctionDeclaration {
        type: "FunctionDeclaration"
        start: 0
        end: 104
      + id: Identifier {type, start, end, name}
        expression: false
        generator: false
        async: false
      + params: [1 element]
      - body: BlockStatement {
          type: "BlockStatement"
          start: 16
          end: 104
        - body: [
          + IfStatement {type, start, end, test, consequent, ... +1}
          + ReturnStatement {type, start, end, argument}
          ]
        }
      }
    ]
    sourceType: "module"
  }
```