

## 06. СИМВОЛЫ, СТРОКИ, ГРАММАТИКИ

*курс лекций по информатике и программированию  
для студентов первого курса ИТИС КФУ (java-поток)  
2023/2024*

**М.М. Абрамский**

кандидат технических наук, доцент кафедры программной инженерии

**ЧАСТНЫЙ СЛУЧАЙ МАССИВА – МАССИВ СИМВОЛОВ  
НО СНАЧАЛА РАЗБЕРЕМСЯ С СИМВОЛАМИ**

# Символ – хранение и отображение

Символ – это *номер* и *визуализация*

- Номер – хранится в памяти **68**
- Визуализация – то, что мы видим **D**

# На всякий случай

- Символы – **D**, **D**, **ᐃ**, **ᐃ** – это один и тот же символ
  - Одинаковые номер и визуализация, шрифт – это не ключевое свойство символа
- Символы **ᐃ** и **ᐃ** – разные символы
  - разные номера и разные визуализации
- Символы **А** (английская) и **А** (кириллица) – разные символы
  - разные номера, хотя и одна визуализация

# Кодировка

**примитивно:**

Таблица соответствия символов  
и номеров

**продвинуто:**

- Таблица соответствия  
символов и номеров
- Способ двоичного кодирования  
(представления) этих  
номеров

32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	\$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

# Какие кодировки мы слышали

- UTF-8, UTF-16
- win1251 (cp1251)
- cp866

В чем же отличие?

# Сначала – общее

- ASCII = American Standard Code for Information Interchange (1963)
- ОБЩАЯ ЧАСТЬ ВСЕХ КОДИРОВОК (сколько битов?)

**ASCII Code Chart**

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

строка – 1я цифра, столбец – 2я цифра (в 16й системе счисления)

# Что будет, если добавить еще 1 бит к 7ми?

Получатся 1-байтовые кодировки (1 символ – 1 байт)

*Например:*

- win1251 – кириллица
- win1250 – западно-европейская кодировка



# win1251

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8																
9																
A	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	-	Ў	Ц	
B	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
C	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
D	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F	№	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	џ	џ	џ



# win1252

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8	€	ƒ	„	”	•	—	—	ˆ	%	Š	ˆ	œ				
9	‘	’	“	”	•	—	—	ˆ	™	š	ˆ	œ				
A	ı	ç	£	¥	¥	§	©	®	™	š	ˆ	œ				
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

# «Проблемы» с кодировкой

Под одними номерами в разных кодировках скрываются  
разные символы

Михаил  
Михайлович,  
здравствуйте!



Ìèõàèë  
Ìèõàéëîâè÷,  
çăďââñòâóéòå!

# Проблемы с кодировками

- Обычные люди не разбираются в тонкостях win1251 и т.п.
- Как идентифицировать кодировку по файлу?
- Не все зарубежные разработчики (англоязычные) разрабатывали решения с поддержкой кодировок

Решение - **унификация**

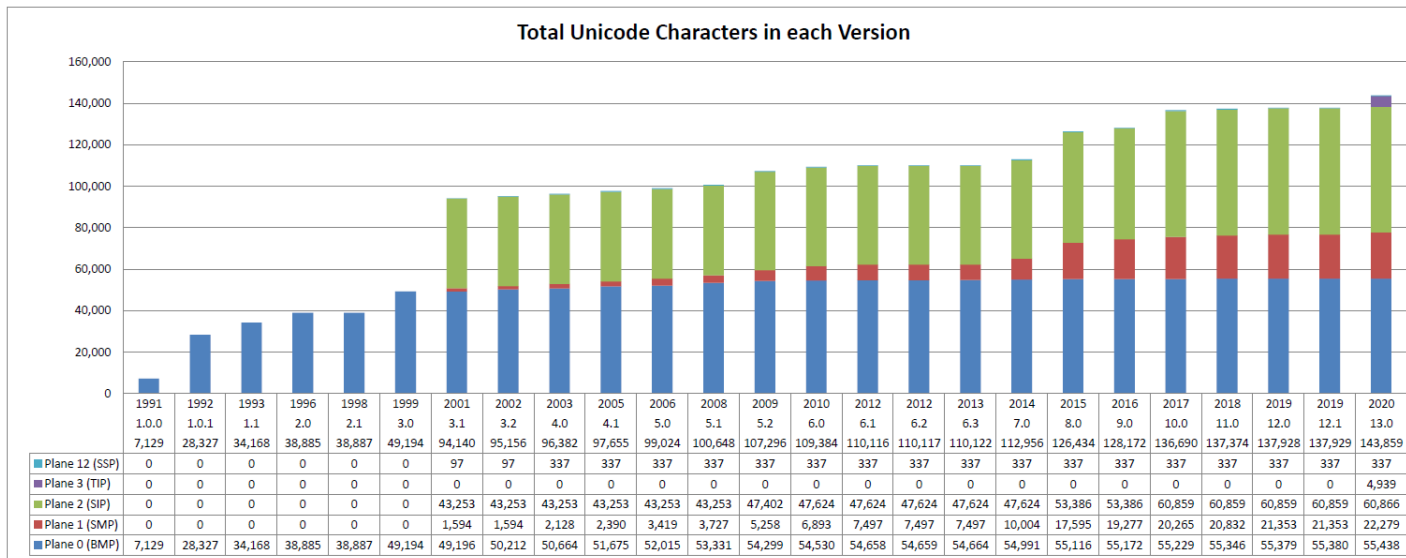
# Unicode

- Самый распространенный формат кодирования СИМВОЛОВ.
- Unicode – не кодировка (в «*непримитивном*» смысле)
  - Определяет только номер символа, не определяет способ его кодирования
- Unicode = UCS (Universal Coded Character Set) + семейства кодировок UTF.

# Кодировки UTF

- Способ хранения (представления) Unicode символов
- Общая их часть – UCS
- UTF-8 – от 1 до 4 байтов
- UTF-16 – 2 байта

# Сколько всего символов?



Тут унифицировали  
японский, китайский и  
корейский

Тут добавили эмодзи

# 143 859 символов

Сколько нужно памяти, чтобы хватило на 143 859?


$$\log_2 143859 \approx 17,1343 \leq 18 \text{ битов} \leq 3 \text{ байта}$$

**Как экономить память?**

# Пример одного из решений - «Суррогатные пары»

Два символа визуально объединяются в 1  
Классический пример – цвет кожи эмодзи



 + символ окрашивания





# ЭКОНОМИЯ

1 606 эмодзи людей + 6 видов цветов = **1 612** СИМВОЛОВ  
(храним в *Unicode*)

но комбинаций их пар:  $1\,606 * 6 = \mathbf{9\,636}$

(столько бы мы хранили уникальных эмодзи, если  
просто каждому дали бы номер)

# Перейдем к программированию

- Символьный тип – **char**

```
char c = 'a';
```

- Все символы имеют свой код

```
int code = (int) c;
```

# Операции с символами

`'a' < 'c' // символы можно сравнивать`

`'a' <= c && c <= 'z' // и двойным неравенством тоже`

**! Ни в какой задаче на символы вам не нужно знать коды конкретных символов !**

Т.е. если при написании кода у вас возникает вопрос «а какой код у символа с?», скорее всего вы делаете «нехорошо»

– Кстати, а что именно нехорошо?

# Unicode в Java

Unicode - стандартная кодировка Java программы.

К символу можно обратиться по его коду (16-ный):

```
char c = '\u0053'
```

Как бы мы не записали программу, символы, компилятор переводит их все в Unicode.

# Поэтому - не только English

Юникод разрешает вот такие идентификаторы:

```
public class ЭтоЧтоКласс {  
  
    public static void main(String[] args) {  
  
        final int МОЯ_КОНСТАНТА = 23;  
  
    }  
  
}
```

```

3  * 11-401
4  * 045
5  */
6
7  import java.util.Scanner;
8
9  public class Task045 {
10     public static void main(String[] args) {
11         Scanner 入力 = new Scanner(System.in);
12         String[] サッカーチーム = 入力.nextLine().split(" ");
13         int[] 成果 = new int[サッカーチーム.length];
14         int サッカーの試合 = Integer.parseInt(入力.nextLine());
15         for (int カウント = 0; カウント < サッカーの試合; カウント++) {
16             String[] 文字列 = 入力.nextLine().split(" ");
17             int 最初, 第2;
18             for (最初 = 0; !サッカーチーム[最初].equals(文字列[0]) && (最初 < サッカーチーム.length); 最初++);
19             for (第2 = 0; !サッカーチーム[第2].equals(文字列[1]) && (第2 < サッカーチーム.length); 第2++);
20             String[] アカウント = 文字列[2].split(":");
21             int 違い = Integer.parseInt(アカウント[0]) - Integer.parseInt(アカウント[1]);
22             成果[最初] += 違い;
23             成果[第2] -= 違い;
24         }
25         for (int カウント = 0; カウント < サッカーチーム.length; カウント++) {
26             System.out.println(サッカーチーム[カウント] + "\t" + 成果[カウント]);
27         }
28     }
29 }

```

# Другой пример пары СИМВОЛОВ

Привет! **CR LF**



Windows  
Linux

\r\n  
\n

# Escape Characters

- Если в символе `\`, значит у него есть особый смысл:
  - `\n` – перенос строки
  - `\t` – табуляция
  - `\b` – отмена предыдущего символа
- Также `\` применяется, чтобы вывести символы, которые тяжело вывести обычным способом:
  - `\\`
  - `\"`
    - Т.к. `“”` неправильно понимается компилятором.
  - `\'`



# Массив символов

- `char []` – в С это и называлось строкой.
- Java: Строка – отдельный тип (ссылочный), у которого должно быть много полезных функций (методов)
  - Есть целый класс задач, который решается на тестовых данных (текст – массив символов).
- *Еще раз: по смыслу строка – массив символов, но с точки зрения реализации – нет.*

# Класс String

- Неизменяемая строка
  - **immutable**, нельзя `s[0] = 'a'`
  - есть изменяемые: `StringBuffer`, `StringBuilder`
- Доступ к символу: метод `charAt(i)`
- Длина: *метод* `str.length()`
  - не путать с массивом!
- Соединение строк: `+`
  - `String hi = "Hello" + ", " + "ITIS"`

# Объявление

- Как ссылочный тип по хорошему строку нужно было бы создавать вот так:
  - `String str = new String("Hello!");`
- Но только для строки введено сокращение:
  - `String str = "Hello!";`

*Но не все так просто.*

# Что происходит тут:

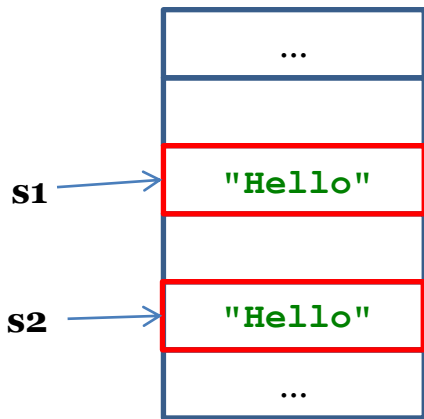
```
String s1 = new String("Hello");
```

- Объявляется ссылочная переменная (ссылка) s1 типа String
- new String(...) – создается объект класса String на основании содержимого строковой константы “Hello”
- Объект присваивается ссылке (теперь она на него указывает).

# Правда об операции == на ссылочных типах данных!

Операция == проверяет равенство ссылок (в одно и то же ли мы место ссылаемся или нет)

содержимое по ссылке на равенство не проверяется!

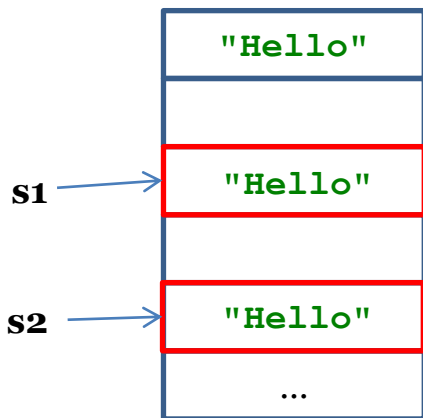


```
String s1 = new String("Hello");
String s2 = new String("Hello");
```

Чему равно `s1 == s2`?

# Более того

Строковые константы (“Hello” в нашем примере) создаются в памяти как отдельный объект:



```
String s1 = new String("Hello");
String s2 = new String("Hello");
System.out.println(s1 == "Hello");
System.out.println(s1 == s2);
```

что увидим?

# Правильная проверка

```
s1.equals(s2), s1.equals("Hello")
```

Верно не только для строк:

пусть arr1, arr2 – массивы. Проверьте разницу между:

» arr1 == arr2

» Arrays.equals(arr1, arr2)?

# Ho!

```
String s1 = "Hello";  
String s2 = "Hello";  
System.out.println(s1 == "Hello");  
System.out.println(s1 == s2);
```



# Ho!

```
String s1 = "Hello";  
String s2 = "Hello";  
System.out.println(s1 == "Hello");  
System.out.println(s1 == s2);
```

true  
true

`String s1 = "Hello";` // Создается строковый объект “Hello” и он присваивается ссылке `s1`. Строковые константы создаются 1 раз – следующее их упоминание – уже созданный объект. Поэтому тот же объект “Hello” присваивается и `s2`. Поэтому все 3 ссылки указывают на один и тот же объект.

# Ho! [2]

```
String s1 = "Hello";  
String s2 = "Hell" + "o";  
System.out.println(s1 == s2);  
System.out.println(s2 == "Hello");
```

# Ho! [2]

```
String s1 = "Hello";  
String s2 = "Hell" + "o";  
System.out.println(s1 == s2);  
System.out.println(s2 == "Hello");
```

true

true

Hell + o создает должен порождать новый объект – строку Hello, но она уже есть, поэтому в s2 присваивается существующая Hello

# System.out.println (?)

```
int x = 2, y = 5;  
char a = 'a', b = 'b';  
System.out.println(x + y);  
System.out.println(x + y + "");  
System.out.println("" + x + y);  
System.out.println(x - y + "");  
System.out.println(a + b);  
System.out.println(a + b + "");  
System.out.println("" + a + b);  
System.out.println(a + y);  
System.out.println("" + a + y);
```

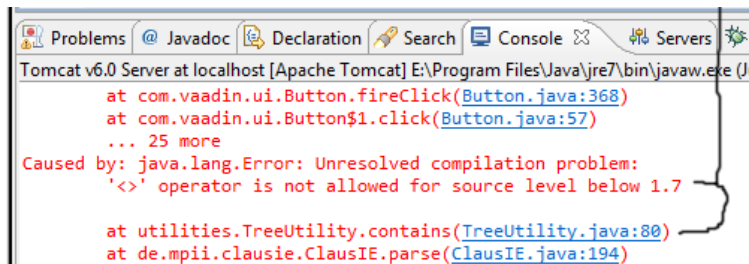
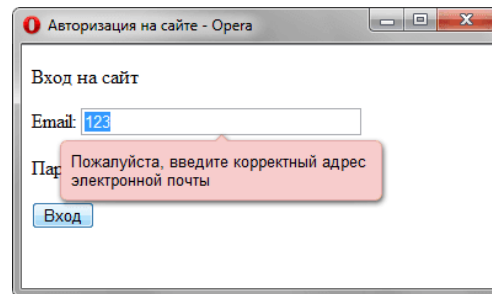
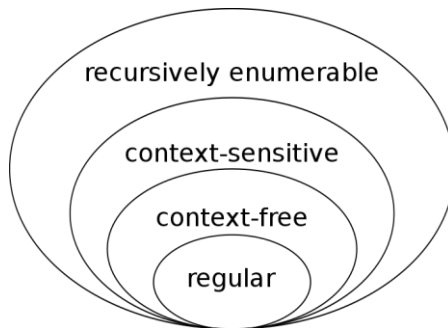
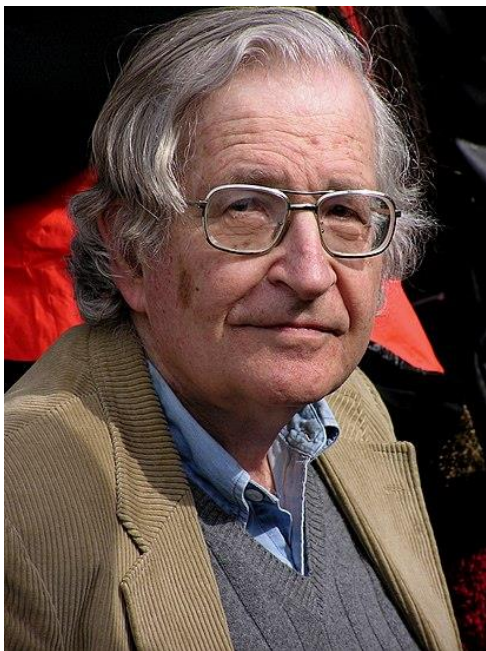
# Строковые методы

- !
- javadoc String

# Двигаемся дальше

- Часто у нас есть необходимость проверять именно строки на некую «корректность», на соответствие строки определенному шаблону:
  - «это не email, введите корректный email»
  - «правильный ли телефон?»
  - «правильно ли написано предложения на английском?»
  - «правильно ли написана программа?»
- Есть целый ряд подходов, как такие задачи решать, и они зависят от того, насколько сложно устроен *язык корректных строк для данной задачи*

# Ноам Хомский (р. 1928)



# Формальные языки

$$L = \langle A, X \rangle$$

- $A$  – алфавит
  - $w = a_1 a_2 \dots a_n$  – слово в алфавите,  $a_i \in A$
- $X$  – характеристическая функция
  - $X(w) = 1$ , если слово  $w$  принадлежит языку
  - $X(w) = 0$ , если слово  $w$  не принадлежит языку



# Заметка про «слово»

- Слово - набор символов алфавита.
- Если пробел разрешен (а в естественных языках он разрешен), то он – тоже символов алфавита. Это означает, что **«мама мыла раму»** - не 3 слова, а одно.
  - Да, предложение – это тоже слово
  - И абзац – тоже слово
  - И текст – слово

# Пример #1

Язык всех двоичных наборов

- $A = \{0, 1\}$
- $X(w) = 1 \quad \forall w, \text{ в } w \text{ все символы из } A$

# Пример #2

## Язык всех натуральных чисел

- $A = \{0, 1, 2, \dots 9\}$
- $X(w) = 1$  если
  - $w$  длины не менее 1,
  - первый символ слова с 1 ... 9,
  - следующие символы слова — 0 ... 9

# Распознавание языков (Language Recognition)

Решение задачи о принадлежности произвольного слова  $w$  языку  $L$ .

Фактически, решение задачи вычисления функции  $X$ .

*Как минимум, все задачи проверок текстовой информации на соответствие неким правилам сводятся к распознаванию языков (и не только они).*

# Как задавать $X$ для языка?

Определить функцию  $X$  можно формулой, но только для простых языков. А как для более сложных?

- Естественные языки?
- Языки программирования?

**ДАЛЕЕ ДЛЯ ПРИМЕРА ВОЗЬМЕМ «ЯЗЫК  
ВСЕХ ПРАВИЛЬНЫХ ПРОГРАММ НА JAVA»**

# Грамматика языка

$\langle T, N, S, P \rangle$

- **T – терминальные символы**
  - Алфавит языка
- **N – нетерминальные символы**
  - Общие понятия языка
    - » Для языка программирования: «оператор», «заголовок метода», «список параметров», «имя переменной», «строковая константа»
- **S – стартовый нетерминальный символ**
  - Понятие «правильного слова» в языке
    - » Для языка программирования – понятие «правильной программы»
- **P – правила вывода / продукции**
  - По каким правилам мы можем «получать» правильные слова в языке

# Пример #1

Грамматика для языка всех двоичных наборов:

- $T = A = \{0, 1\}$
- $N = \{S\}$
- $P$ :
  - $S \rightarrow 0$
  - $S \rightarrow 1$
  - $S \rightarrow S0$
  - $S \rightarrow S1$



# Пишут иногда так

$$\begin{array}{l} S \rightarrow 0 \\ S \rightarrow 1 \\ S \rightarrow S0 \\ S \rightarrow S1 \end{array} \quad \Rightarrow \quad S \rightarrow 0 \mid 1 \mid S0 \mid S1$$

# Пример вывода

$$1. S \rightarrow 0$$

$$2. S \rightarrow 1$$

$$3. S \rightarrow S0$$

$$4. S \rightarrow S1$$

**Вывод слова** – применение правил вывода к  $S$ , чтобы получить конкретное слово, допустимое языком

Вывод конкретной строки 10101 из  $S$ :

$$S - 4 -> S1 - 3 -> S01 - 4 -> S101 - 3 -> S0101 - 2 -> 10101$$

# Что такое слово из языка?

*(с точки зрения грамматик)*

Слово принадлежит языку, если для него можно построить его вывод из  $S$  с помощью правил вывода  $P$ .

Как проверить, что слово принадлежит языку?

- **способ #1** – вывести его из  $S$
- **способ #2** – наоборот, из него получить  $S$  (применением правил вывода в обратную сторону)

# Вообще говоря

Правила вывода  $P$  могут быть сложными, например:

$$AaSBb \rightarrow AaAbAac$$

Что предложил Хомский (!):

*«Сложность языка/грамматики – это сложность правил вывода  $P$ .  
Задавая ограничения на  $P$ , можно построить иерархию языков»*

# Иерархия Хомского

- **Рекурсивно перечислимые языки**
  - Никаких правил на  $P$  не накладывается
- **Контекстно-зависимые (КЗ) языки**
  - Правила вывода имеют вид:  $abcs\mathbf{B}abCs \rightarrow abcs\mathbf{A}VcabCs$
- **Контекстно-свободные (КС) языки**
  - Правила вывода имеют вид:  $\mathbf{B} \rightarrow \mathbf{A}V\mathbf{c}$  (в левой части всех правил вывода строго один нетерминал)
- **Регулярные (Reg) языки**
  - Правила вывода должны иметь только такой вид:  $(A \rightarrow a, A \rightarrow Vc)$  или  $(A \rightarrow a, A \rightarrow cV)$  (т.е. если выводится терминал и нетерминал, то терминал либо во всех правилах справа, либо во всех правилах слева)

Глядя на иерархию Хомского, можно сделать вывод:

**Чем сильнее ограничение,  
тем проще язык**

# Контекстно-свободные грамматики

Языки программирования – это контекстно-свободная грамматика.

- Не КЗ, т.к., например, вместо одной команды можно вставить целый набор любых разрешенных команд – и программа все равно останется синтаксически правильной.
- Не Reg, т.к. скобки:
  - Нужны правила вывода вида  $A \rightarrow \{B\}$  или  $A \rightarrow (C)$ , а такие правила не разрешены для регулярных грамматик.

# Пример #1

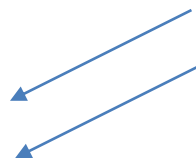
- Язык всех двоичных наборов – регулярный.

$S \rightarrow 0$

$S \rightarrow 1$

$S \rightarrow S0$

$S \rightarrow S1$

 *0 и 1 появляются справа*



# Пример #2

## Язык всех натуральных чисел

- $T = \{0, 1, 2, \dots 9\}$


- $N = \{S, A\}$

- $P:$

- $S \rightarrow 1A \mid 2A \mid 3A \mid \dots \mid 9A$

- $A \rightarrow 0A \mid 1A \mid 2A \mid 3A \mid \dots \mid 9A \mid 0 \mid 1 \mid 2 \mid \dots \mid 9$

Надо как-то запомнить, что первая цифра не должна быть 0, для этого используется нетерминал  $A$ .



**Тоже регулярная грамматика**

(цифры как терминалы везде упоминаются слева)

# Нетерминал 'A' в примере #2

$$S \rightarrow 1A \mid 2A \mid 3A \mid \dots \mid 9A$$
$$A \rightarrow 0A \mid 1A \mid 2A \mid 3A \mid \dots \mid 9A \mid 0 \mid 1 \mid 2 \mid \dots \mid 9$$

Смысл A - «набор любых цифр длиной  $\geq 1$ »

Следующий вопрос –

как запрограммировать функцию  $X$  для языка,  
учитывая его грамматику?

(т.е. проверку принадлежности слова  $w$  языку  $l$ )

# Подход #1

Явно строить вывод слова в грамматике (слова из  $S$  или  $S$  из слова).

Трудоемко, но используется.

# Подход #2

Для каждого типа языков из иерархии Хомского есть своя модель вычислений, которая способна решать задачи распознавания языков.

Например, модель конечных автоматов для регулярных языков (но это уже совсем другая история).

# Регулярные языки очень распространены

- И что, все время строить для них автомат? Или вывод?
- Третий способ проверки принадлежности слова регулярному языку – регулярные выражения!

# Регулярные выражения

Строки, которые являются шаблонами других строк

Пример:

**[a-z]+** все строки из 1 и более строчных букв английского алфавита

**[1-9][0-9]\*** все строки, идентичные натуральным числам

# REGEX CHEAT SHEET

LOVES DATA

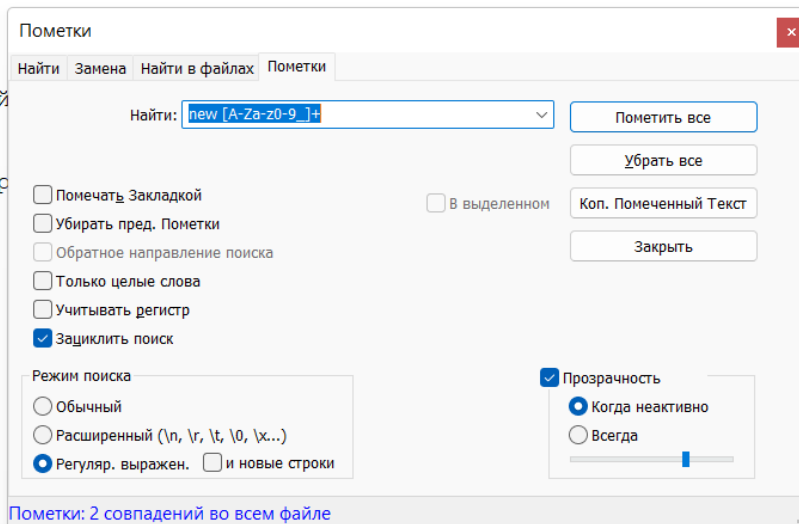
REGEX SYNTAX	MEANING	EXAMPLE	MATCHES	DOES NOT MATCH
.	Any single character	go.gle	google, goggle	gogle
[abc]	Any of these character	analy[zs]e	analyse, analyze	analyxe
[a-z]	Any character in this range	demo[2-4]	demo2, demo3	demo1, demo5
[^abc]	None of these characters	analy[^zs]e	analyxe	analyse, analyze
[^a-z]	Not a character in this range	demo[^2-4]	demo1, demo5	demo2, demo3
	Or	demo example	demo, demos, example	test
^	Starts with	^demo	demos, demonstration	my demo
\$	Ends with	demo\$	my demo	demonstration
?	Zero or one times (greedy)	demos?123	demo123, demos123	demoA123
??	Zero or one times (lazy)			
*	Zero or more times (greedy)	goo*gle	gogle, goooogle	goggle
*?	Zero or more times (lazy)			
+	One or more times (greedy)	goo+gle	google, goooogle	gogle, goggle
+?	One or more times (lazy)			
{n}	n times exactly	w{3}	www	w, ww
{n,m}	from n to m times	a{4, 7}	aaaa, aaaaa, aaaaaa, aaaaaaa	aaaaaaaa, aaa, a
{n,}	at least n times	go{2,}gle	google, gooogle, goooogle	ggle, gogle



# Использование регулярок

```
import java.io.*;

public class Main {
    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            System.err.println("Не указано имя файла.");
            return;
        }
        String filename = args[1];
        // Открытый файл будет автоматически закрыт по ошибке
        try (BufferedReader reader = new BufferedReader(new FileReader(filename)) {
            String line;
            for (int n = 1; (line = reader.readLine()) != null; ++n) {
                System.out.println(n + ": " + line);
            }
        } catch (FileNotFoundException e) {
            System.err.println("Указанный файл не найден");
        }
        // finally {
        //     reader.close(); // автоматическое закрытие
        // }
    }
}
```



# Регулярные выражения в Java

```
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
// шаблон натурального числа  
  
Pattern p = Pattern.compile("[1-9][0-9]*");  
Matcher m = p.matcher("123");  
System.out.println(m.matches());
```

# Развлекайтесь

<https://regexcrossword.com/>

(EG BEEE)[WIO]*				
(WE GA AL)T*O+				
[HAS]*(SN PA)				
(.)T*E*\1				
[ABC]*(.)\1(ME UO)				
[QA].[WEST]*				
(HE RT TK)*.				
(RE QR)[QUART]*				
[EUW]*S[RITE]*				
(.)(.)\2\1[WE]				

[ON](PR AX TR)+				
(KT AL ET)+G				
(MN VO FI)[EU]{2,}				
(BG ON KK)+[RLF]+				
[RUH]*(OE EO)[RB]*				
[IT](O)*(BE AD)*\1				
[NORMAL]+T{2}				
.*(XA BE).*				
(EG UL){2}[ALF]*				
[REQ]*(G P)(.)+				

# БОНУС

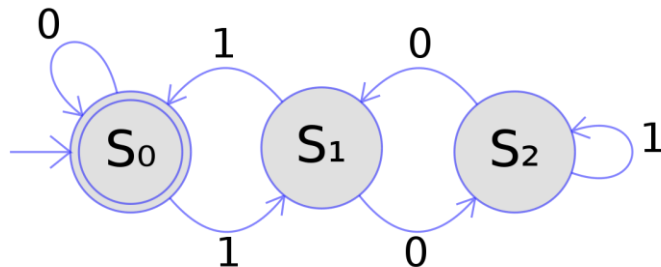
# КДА (конечные детерминированные автоматы)

Похож на Машину Тьюринга, но немного урезаны возможности

$$\text{КДА} = (A, S, \delta, s_0, F)$$

- $A$  – алфавит
- $S$  – множество состояний
- $\delta: A \times S \rightarrow S$  – функция переходов (в МТ это была таблица)
- $s_0$  – начальное состояние
- $F \subseteq S$  – множество финальных состояний

# Как работает КДА, пример



	S0	S1	S2
0	s0	s2	s1
1	s1	s0	s2

# Легко запрограммировать

	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>
0	S <sub>0</sub>	S <sub>2</sub>	S <sub>1</sub>
1	S <sub>1</sub>	S <sub>0</sub>	S <sub>2</sub>

```

int [] input = ...
int [][] f = {{0, 2, 1}, {1, 0, 2}};
int s = 0
for (int c : input) {
    s = f[c][s];
}
значение s - ответ
    
```

# Что есть состояние?

- Память о проведенной работе, закодированная в число.



# Теорема

## Теорема

Конечный автомат распознает только регулярные языки

Идея доказательства:

Правила вывода регулярной грамматики:

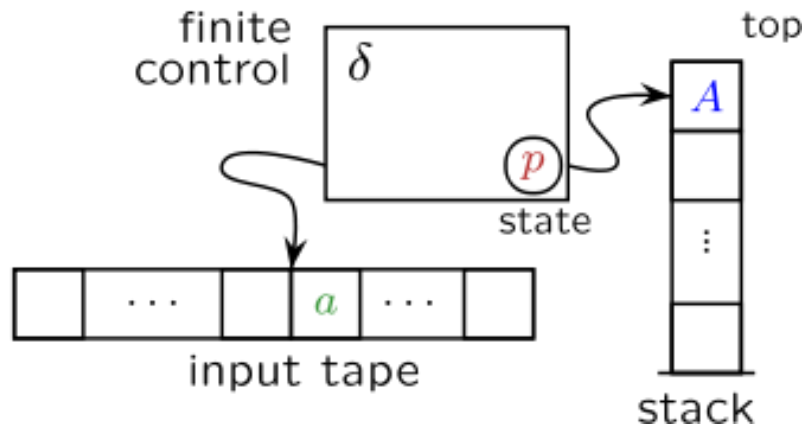
$A \rightarrow oB$

фактически являются частью диаграммы переходов автомата

$A \xrightarrow{o} B$  (из состояния  $A$  по символу  $o$  перейти в состояние  $B$ )

# КДА не умеет в КС

- Типичный пример: понять, что во входной строке одинаковое количество нулей и единиц
- Это может его модификация – магазинный автомат



**TO BE CONTINUED**