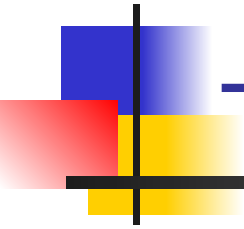


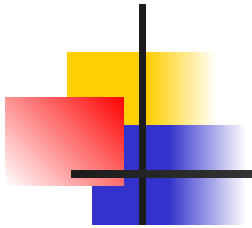
Теория автоматов и формальных языков

Карпов Юрий Глебович
профессор,

karpov@dcn.infos.ru

Структура курса

- Конечные автоматы-распознаватели – 5 л
  - Лекция 1. Формальные языки. Примеры языков. Грамматики. КА
  - Лекция 2. Теория конечных автоматов-распознавателей
  - Лекция 3. Трансляция автоматных языков
  - Лекция 4. Регулярные множества и регулярные выражения
  - Лекция 5. Язык MiLan и стековая машина
- Порождающие грамматики Хомского – 3 л
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции 2 л



Язык программирования MiLan (Mini Language)

Язык программирования MiLan

```
begin
  m:=read;
  n:=read;
  while m!=n do m:=1;
  if m>n
     then m:=m-read
     else n:=n-m;
  write(m)
end
```

- Язык Милан – простой паскалеподобный язык программирования:
  - один тип данных – целые;
  - в программе нет описаний переменных;
  - программа начинается ключевым словом begin, заканчивается словом end;
  - управление процессом вычислений: Условный оператор и Цикл:
    - if <УСЛОВИЕ> then <ОПЕРАТОРЫ> else <ОПЕРАТОРЫ>;
    - if <УСЛОВИЕ> then <ОПЕРАТОРЫ>;
    - while <УСЛОВИЕ> do <ОПЕРАТОРЫ>;
  - условия в операторах if и while – два арифметических выражения, связанные знаками отношения >, <, =, !=, >=, <=
  - оператор ввода read вводит очередное целое число из входного потока; оператор x:=read присваивает введенное число переменной с именем x;
  - оператор вывода write(<ВЫРАЖЕНИЕ>) выводит на печать значение, полученное как результат вычисления арифметического выражения.

Вопрос: почему begin, end, while, ... НЕ подчеркнуты???

Примеры программ на языке MiLan

P0::

```
begin
  m:=read;
  n:=read;
  while m!=n do
    if m>n then
       m:=m-read
    else n:=n-m;
  write(m)
end
```

P1::

```
begin
 x:=read;
 ;
 if     x>y2-3 then
   ;


 write(x +2 *y)
end
```

P2::

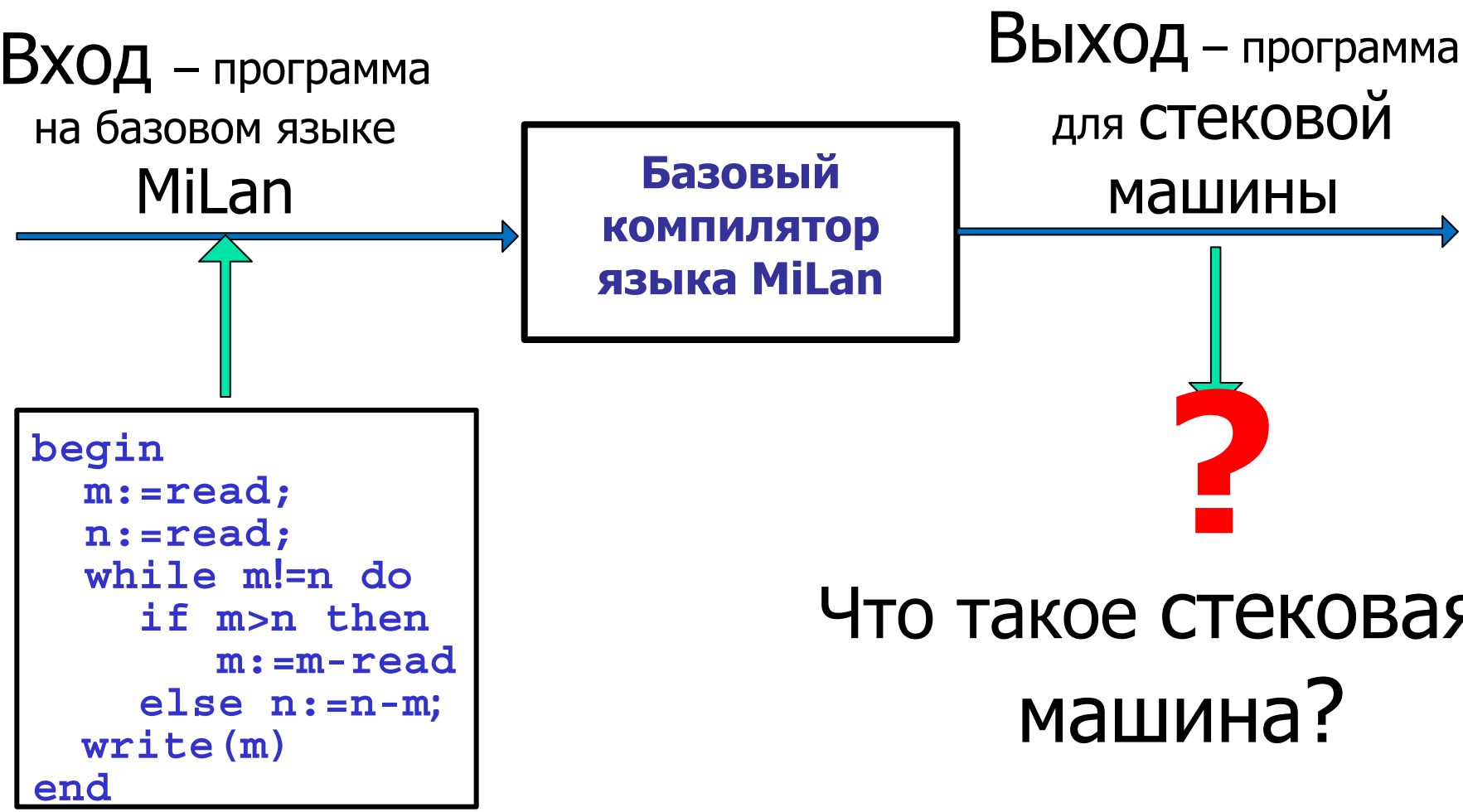
```
begin
 m:=34;
 n:=read *(  m+2);
 while m!=3*n do

 n:=m+       rea  d;
 wri te(m - 2*  n)
end
```

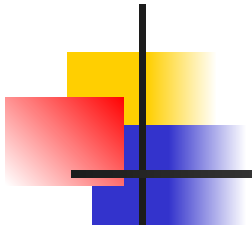
- программа на Милане не имеет блоков;
- в тело операторов Условный и Цикл могут быть включены любое число операторов языка;
- в языке НЕТ подпрограмм, методов, функций и процедур;
- начальные значения переменных устанавливаются равными нулю.

Как понимать программы P1, P2?

Транслятор с языка МИЛАН – на какой язык?



Что такое стековая машина?

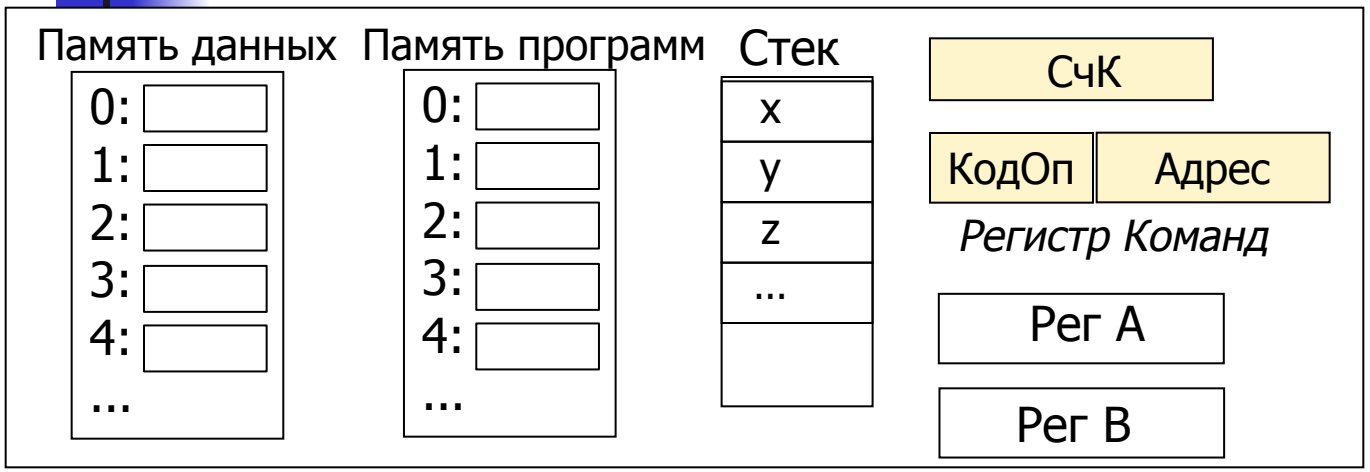


Виртуальная стековая машина

Стековая машина: архитектура и набор команд

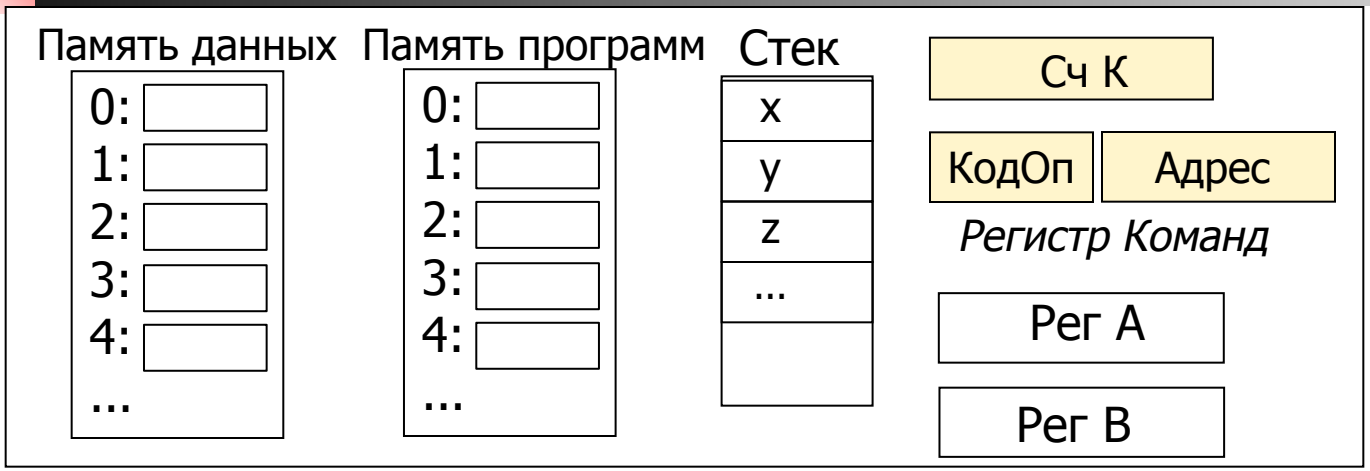
- Стековая машина – Виртуальный однопроцессорный компьютер с простой архитектурой. Содержит:
  - память данных (ПД) – линейная память, каждый регистр Памяти данных может хранить одно целое число;
  - память программ (ПП) - линейная память, каждый регистр Памяти программ может хранить одну команду (код операции и адрес);
  - стек – линейная память с доступом только к верхушке стека;
  - ‘счетчик команд’ (СчК); в нем хранится адрес выполняемой команды;
  - регистр Команд - в нем хранится выполняемая команда (ОДНОАДРЕСНАЯ);
  - регистры A, B – в них помещаются данные при выполнении команд.

Стековая машина: как выполняется программа?



- **Цикл выполнения команды** – всегда один и тот же:
  - адрес очередной выполняемой команды находится в Счетчике Команд; перед выполнением программы он устанавливается в 0;
  - (**1-й ШАГ**) из Памяти Программ по адресу, находящемуся в СчК, выбирается код (КодОп и Адрес) и помещается в *Регистр Команд*;
  - (**2-й ШАГ**) содержимое Счетчика Команд увеличивается на 1 (*готовимся к выполнению следующей команды*);
  - (**3-й ШАГ**) поле КодОп Регистра Команд дешифрируется и соответствующая операция выполняется (*например,* `Jump` `Addr` *реализуется так: Addr помещается в СчК*).

Стековая машина: архитектура и набор команд



Пересылки

LOAD a
STORE a

Арифметика:

ADD
SUB
MULT
DIV

Сравнение:

COMPARE k

LOAD a	загрузка из адреса ПД a в верхушку стека
STORE a	выталкивание значения из верхушки стека и запоминание его в ПД по адресу a
ADD, SUB, ...	сложение, вычитание, ... (операции со стеком)
COMPARE k (CMP k)	сравнение двух верхних значений, x,y в стеке: y q_k x. Рез-т сравнения (0 или 1) => в стек
JUMP m JUMP_NO m JUMP_YES m	Безусловный переход управления по адресу m. Переход по адресу m, если в верхушке стека 0. Переход по адресу m, если в верхушке стека 1 (не 0).

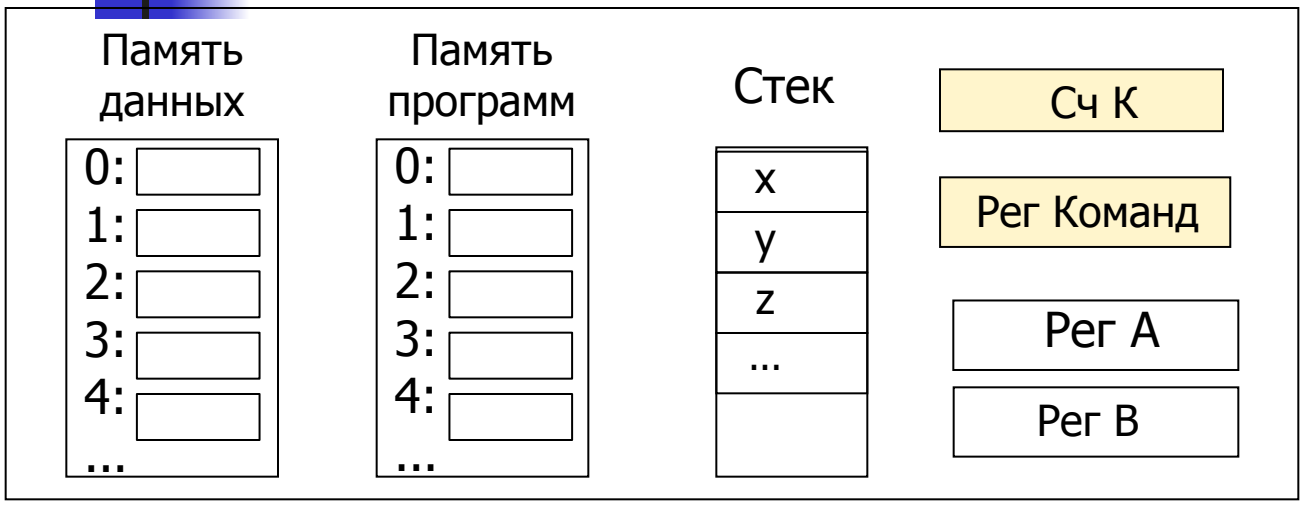
k=0? ‘=‘
k=1? ‘!=’
k=2? ‘<‘
k=3? ‘>’
k=4? ‘≤’
k=5? ‘≥’

Стековая машина: выполнение простых команд

a – b + d

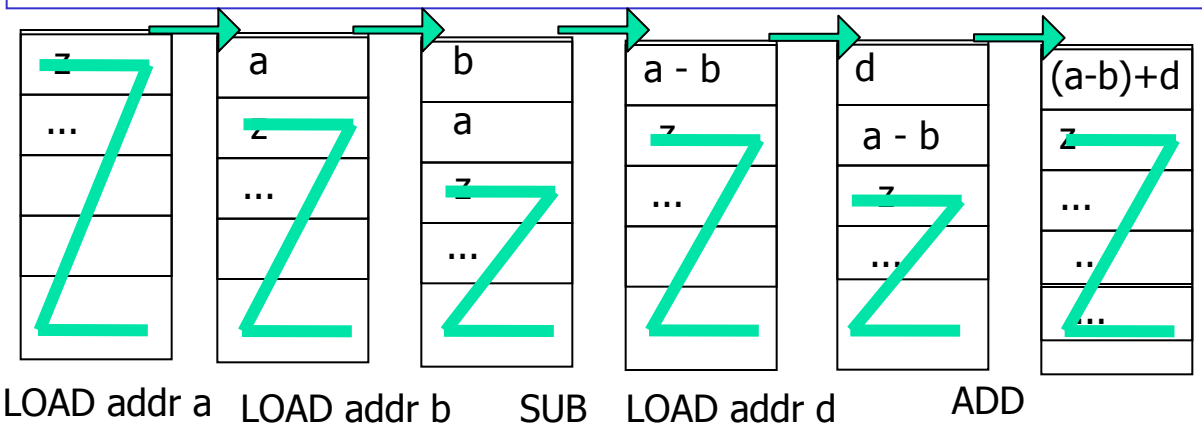
Результат компиляции Ар. Выр:

последовательность команд, помещающих в верхушку стека результат вычисления арифметического выражения




```
LOAD    addr a
LOAD    addr b
SUB
LOAD     addr d
ADD
```

a – b + d – выполнение программы стековой машины:



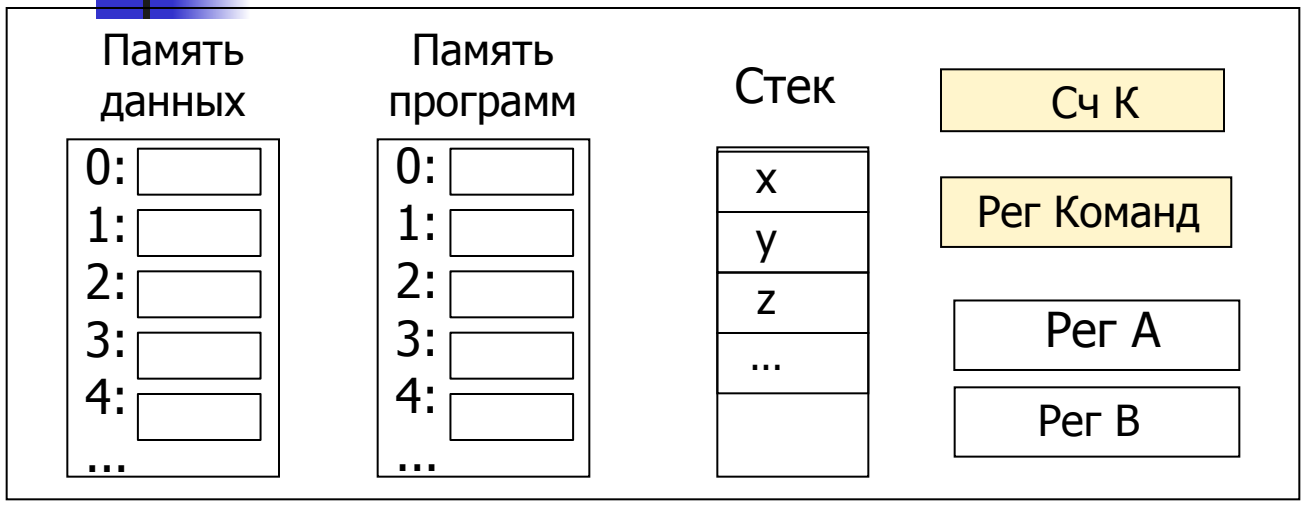
Первый операнд операции находится в стеке ниже, чем второй

Стековая машина: архитектура и набор команд

a := b - d * e

Результат компиляции оператора:

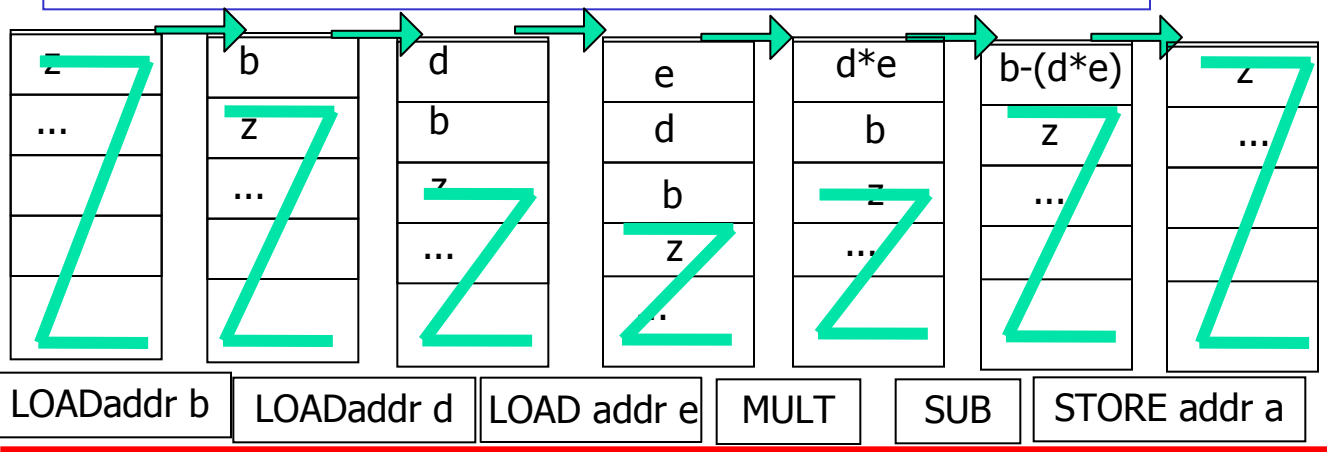
```
LOAD    addr b
LOAD    addr d
LOAD    addr e
MULT
SUB
STORE   addr a
```



последовательность команд, реализующих вычисление арифм. выражения и присваивание результата

a:=b-d*e

выполнение программы стековой машины:

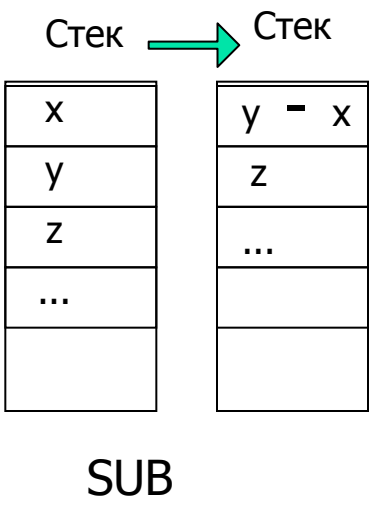


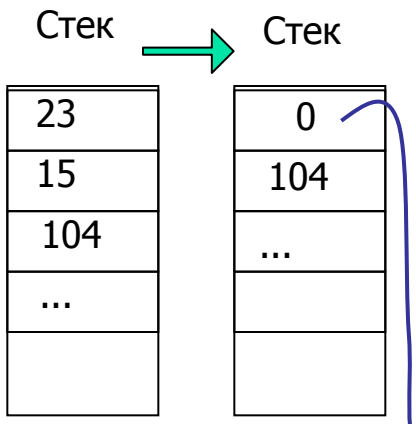
Внимание! Здесь учтены приоритеты операций

После выполнения присваивания стек остается в том же состоянии, в котором он был до начала оператора

Пример выполнения стековых команд

a – b > c





q_k = k=0? '='
k=1? '≠'
k=2? '<'
k=3? '>'
k=4? '≤'
k=5? '≥'

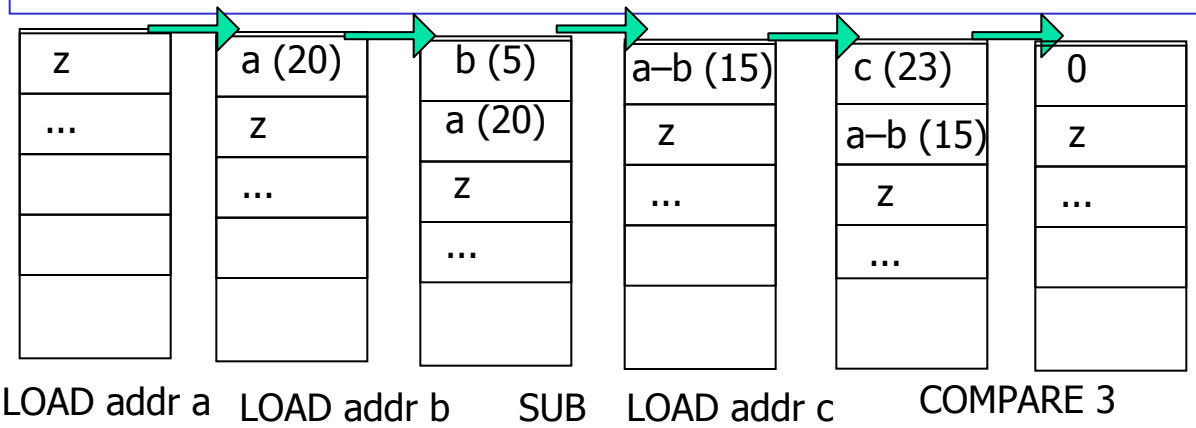
q_3 = '>'
15 > 23 (?)
нет

Результат компиляции:

```
LOAD    addr a
LOAD    addr b
SUB
LOAD    addr c
COMPARE 3
```

Результат: 0 или 1 помещается в верхушку стека

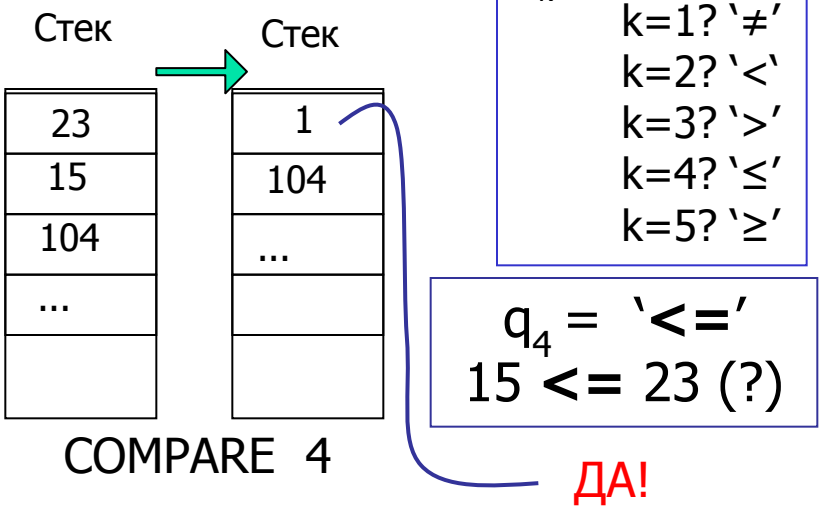
a - b > c выполнение программы стековой машины:



Значение 0 или 1 в верхушке стека может быть использовано следующей командой условного перехода

Пример выполнения стековых команд

a - b ≤ c + d * f



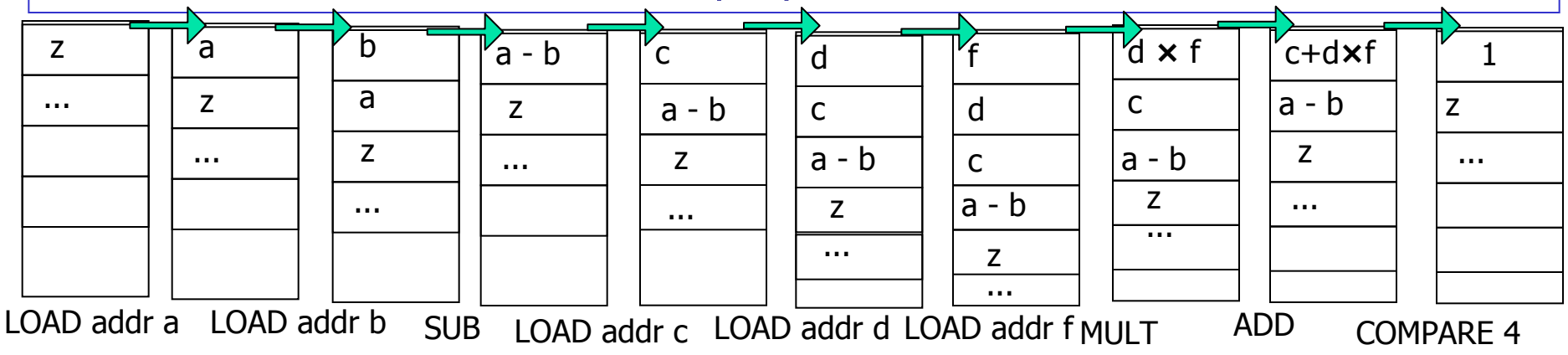
q_k = k=0? '='
k=1? '≠'
k=2? '<'
k=3? '>'
k=4? '≤'
k=5? '≥'

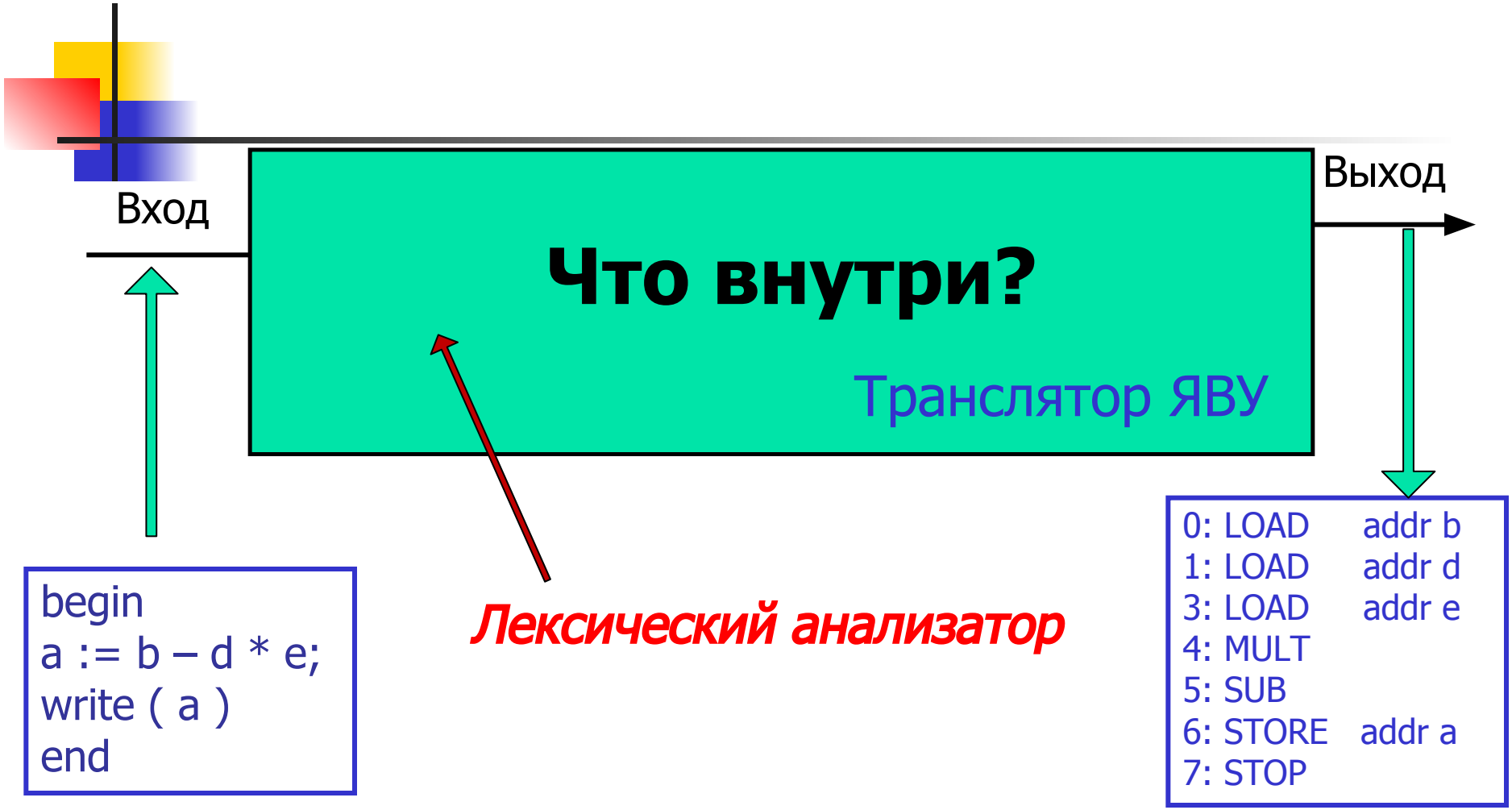
q_4 = '<='
15 <= 23 (?)

Результат компиляции:

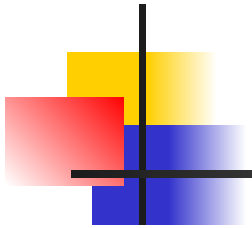
```
LOAD    addr a
LOAD    addr b
SUB
LOAD    addr c
LOAD    addr d
LOAD    addr f
MULT
ADD
COMPARE    4
```

a - b ≤ c + d * f – выполнение программы стековой машины:



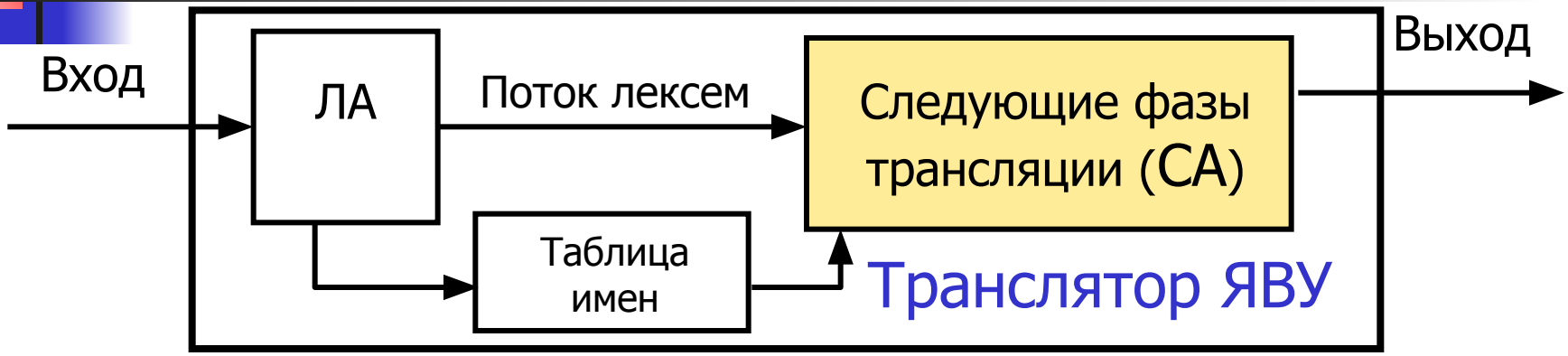


- Транслятор (КОМПИЛЯТОР) языка Милан переводит программу на этом языке в программу на языке стековой машины (промежуточный язык).
- Программа на языке стековой машины обычно затем интерпретируется. Такой интерпретатор пишется элементарно.



Лексический анализ языков программирования

Лексический анализатор (ЛА) – первый проход транслятора



Вход ЛА – цепочка кодов символов клавиатуры:

Лексема: | тип | номер |

begin m13 := read ; alpha237 := read ; while 243 < > alpha237 do if m13 > a2 then
... **символы ASCII**

Выход ЛА – цепочка лексем (во внутреннем представлении):

begin i_0 **ass read sc** i_1 **ass read sc while** c_0 q_1 i_1 **do if** i_0 q_2 i_1 **then** - **лексемы**

Таблица имен:

N	имя	доп. инф
0	m13	...
1	alpha237	...
2	a2	

i_0 i_1 i_2

Таблица констант:

N	симв	знач
0	243	...
1		
2		

c_0 c_1

Лексический анализатор языка Милан. ЗАЧЕМ?

- Язык программирования МИЛАН не является автоматными.
- Некоторые фрагменты языка описываются автоматами: имена, константы...

```
begin
/*************   ПРОГРАММА вычисления НОД на Милане  ****************/
    m13 := read;      /*  переменная m13 читается из входного потока */
    alpha237:=read;   /*  переменная alpha237 означает ширину */
    while m13 <>  alpha237 do
          if m13 >= alpha237  then  m13 : = m13 - alpha237
          else
     write(alpha237) ; alpha237:= alpha237 – m13;
end
```

Чем неудобна программа на входном языке МИЛАН?

1. Бессмысленны отдельные символы: например, **e** в словах b**e**gin, r**e**ad.
2. Каждое служебное слово представляет единый ‘символ’.
3. Все имена с точки зрения синтаксиса – одно и то же, но имеют разную семантику (также неразличимы и все константы).
4. Очень трудно формально описать включение произвольного числа пробелов и комментариев в любое место программы.
5. Целые группы входных символов представляют одну лексему: ‘:=‘ , ‘>=‘ ... .

Программа набирается на клавиатуре в коде ASCII

	.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	TAB	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

q_k

k=0? '='
k=1? '!='
k=2? '<'
k=3? '>'
k=4? '≤'
k=5? '≥'

- Если на клавиатуре наберем:

```
    if  m < 1 then max
:=     10      ;
```

i_k – k-е имя

c_k – k-я константа

q_k – k-е отношение

После лексического анализа – цепочка лексем:

if	i_0	q_2	c_0	then	i_1	:=	c_1	;

Назначение лексического анализа: ЛЕКСЕМЫ

- В реальных трансляторах ЯП первой фазой является так называемый лексический анализ входной программы - предварительная обработка входного текста с выделением в нем структурно значимых единиц – лексем.

Лексемы – минимальные единицы языка, которые имеют смысл.

- В естественном языке лексемами являются не буквы, а слова (словоформы), в языке программирования – не отдельные символы, а имена, служебные слова, константы, знак оператора присваивания из двух символов ':=' и т.д.
- На входе транслятора исходная программа

```
    for j  : = 1 to max do
  x2[j] : = 10;
```

представлена в виде неструктурированного потока байтов в коде ASCII:

66 6F 72 20 6A 20 3A 3D 31 64 6F 20 6D 61 78 20 64 6F 0D 0A 09 20
20 20 78 32 5B 6A 5D 20 3A 3D 20 31 30 3B

- Символы не имеют смысла, смысл имеют ЛЕКСЕМЫ – группы символов.

for | j | := | 1 | to | max | do | x2 | [| j |] | := | 10 | ;

- В этом фрагменте 14 лексем: служебные слова: for, to, do; 4 лексемы 3-х имен: j, max, x2; два вхождения лексемы присваивания := и т.д.

Лексический анализ языков программирования

- Задача лексического анализа – представить исходную программу как последовательность лексем (лексических единиц).
- Лексемы и являются терминальными символами грамматики языка.

```
begin
   /******* ПРОГРАММА вычисления НОД на Милане *********/
    m13 := read;          //  переменная m13 из входного потока
    alpha237:=read;       /*    alpha237 означает ширину */
    while m13 <> alpha237 do
       if m13  > alpha237 then m13:= m13 + alpha237   ;
```

По цепочке (последовательности) байтов на входе лексический анализатор должен построить цепочку лексем:

$$\mathbf{begin}\ \ i_0\ \mathbf{ass\ read\ sc}\ i_1\ \mathbf{ass\ read\ sc\ while}\ i_0\ q_1\ i_1\ \mathbf{do\ \ if}\ i_0\ q_2\ i_1\ \mathbf{then}\ i_0\ \mathbf{ass}\ i_0\ \mathbf{addop}_0\ i_1\ \ldots\ \ \ldots$$

Каждая лексема имеет свой смысл, так же, как при трансляции естественного языка каждое СЛОВО имеет смысл (буквы смысла не имеют)

Лексемы языка MiLan

Лексема: | тип | номер |

Программа вычисления НОД двух чисел на языке MiLan:

```
begin
m:=read;
n:=    read;
while m≠n do
     if m>n then
m : = m-n
     else n:=n-m;
write     (m)
end
```

Каждый студент в качестве части курсовой работы строит лексический анализатор языка Милан.

Цепочка лексем программы

begin i_0 **ass read sc** i_1 **ass read sc while** i_0 q_1 i_1 **do if** i_0 q_2 i_1 **then** i_0 **ass** i_0 $addop_1$ i_1 **else** i_1 **ass** i_1 $addop_1$ i_0 **sc write lb** i_0 **rb end**

begin – служебное слово begin

read – служебное слово read

…

i_0, i_1, … Имена переменных с номерами 0, 1, …

ass – assign (присваивание :=)

sc – semicolon – точка с запятой (;)

q_0, q_1, q_2, … – отношения =, <>, >, <, …

$addop_0$, $addop_1$ – операция типа сложения (+ и -)

…

где можно вставлять пробелы?

где нужно вставлять пробелы?

основные вопросы ЛА

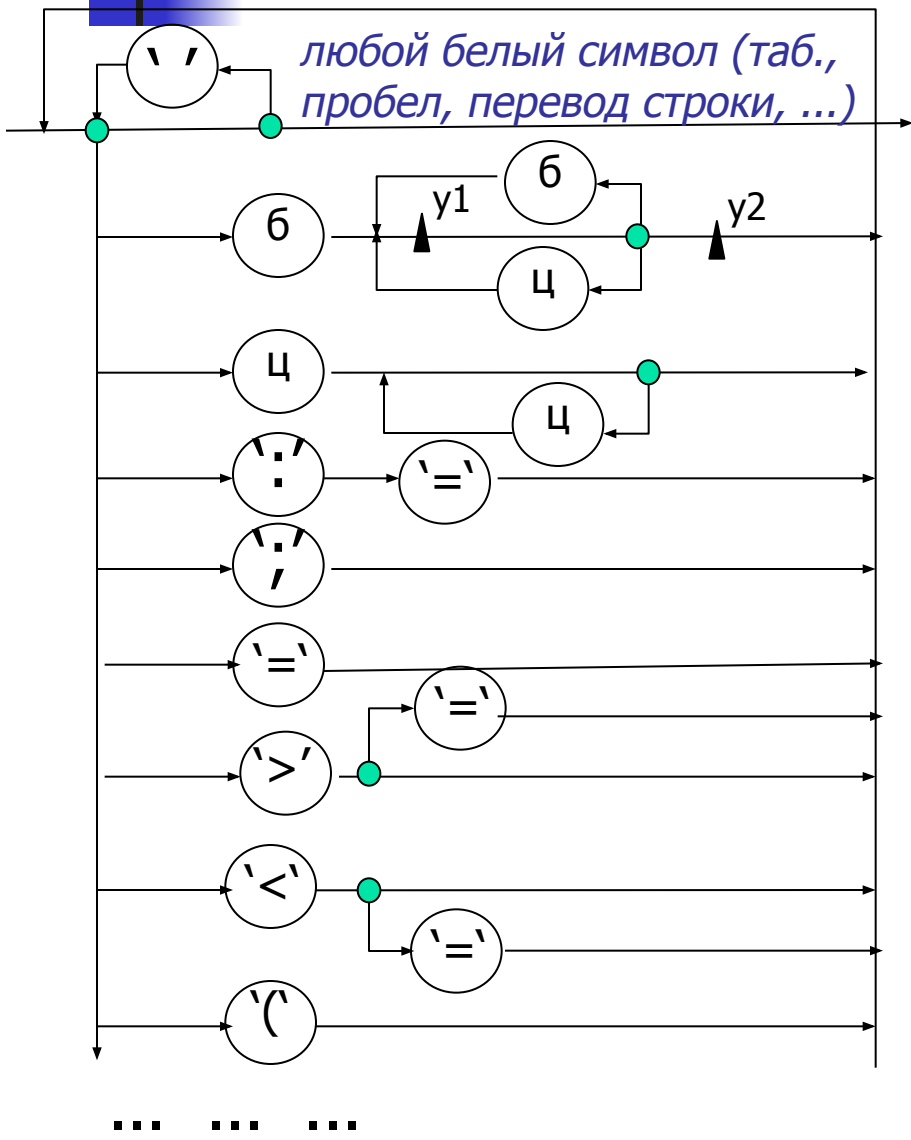
Кодировка лексем языка Milan

Лексема: | тип | номер k |

тип	Лексема	Представление	Номер в типе	Кодировка
0	begin	**begin**		<0, 0> тип 0
1	while	**while**		<1, 0> тип 1
…				
21	имена	i_k	k – номер в таблице имен	<21,3>- третье имя (тип=21)
22	конст	c_k	k – номер в таблице констант	<22,5> – пятая константа
23	:=	**ass** assign		<23,0>
24	;	**sc** (semicolon)		<24,0>
25	=,!=,>=, >, <=, <	q_k	‘=‘ k=0, ‘!=‘ k=1, ‘>=‘ k=2, ‘>’ k=3, …	<25,2> – тип25, отн. N2, ‘>=’
26	-, +	**addop**$_k$	‘-’ k=0, ‘+’ k=1	<26,1> – это ‘+’

Лексемы на языке построения транслятора – перечислимый тип

Лексический анализатор языка MiLan (транслятор автоматного языка лексем Милана)



Лексема: | тип | номер |

- y1: символ - в буфер
- y2: это служебное слово? Если нет, то это имя. Проверить/добавить в таблице имен. Выдать соотв. лексему.

буфер | a l p h a 2 3 7 |

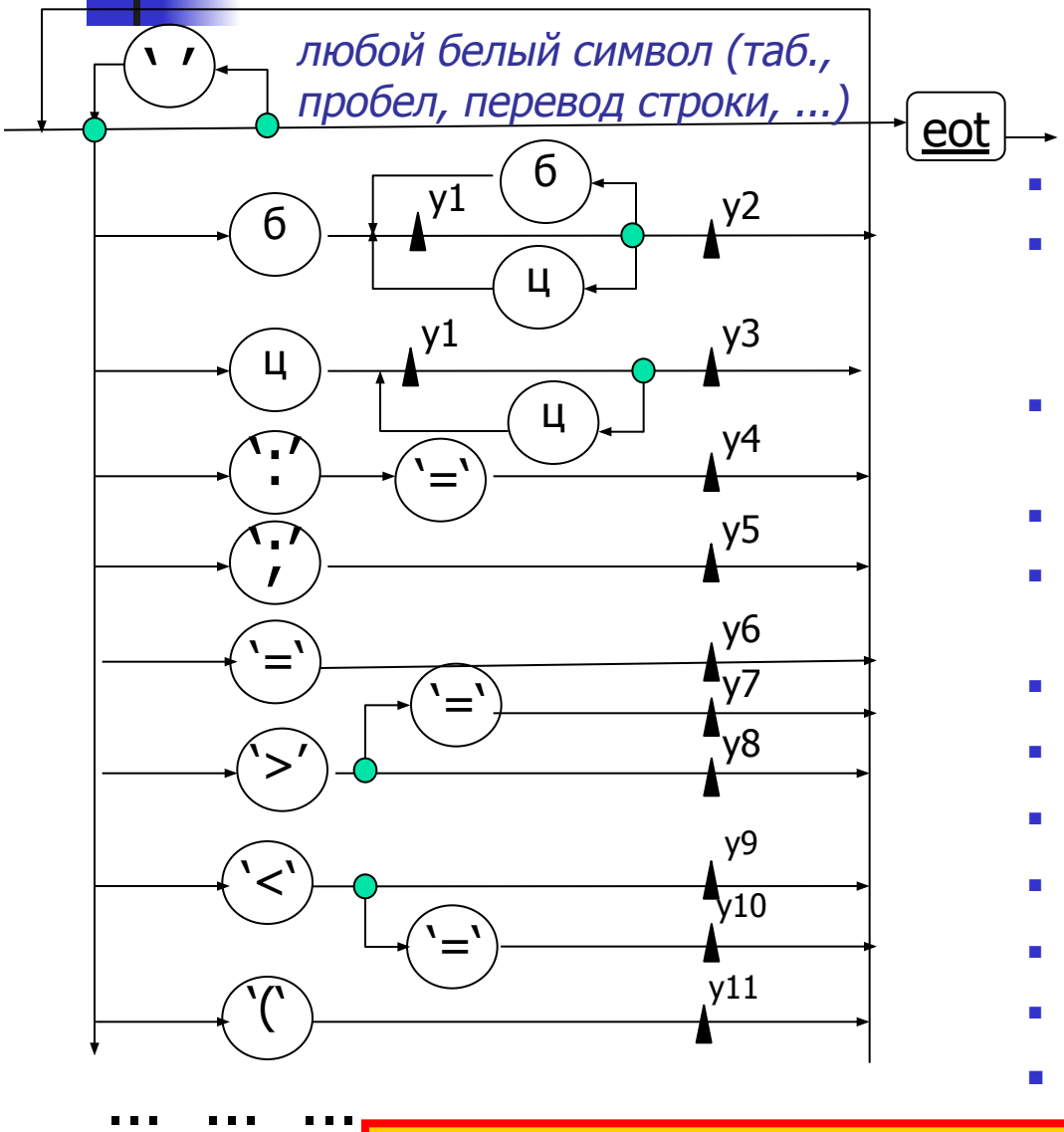
Таблица служебных слов:

N	служебное слово
0	b e g i n
1	w h i l e
2	... ...

Таблица имен:

N	имя	доп. инф.
0	m 1 3	
1	a l p h a 2 3 7	
2	... ...	

Лексический анализатор языка MiLan (транслятор автоматного языка лексем Милана)



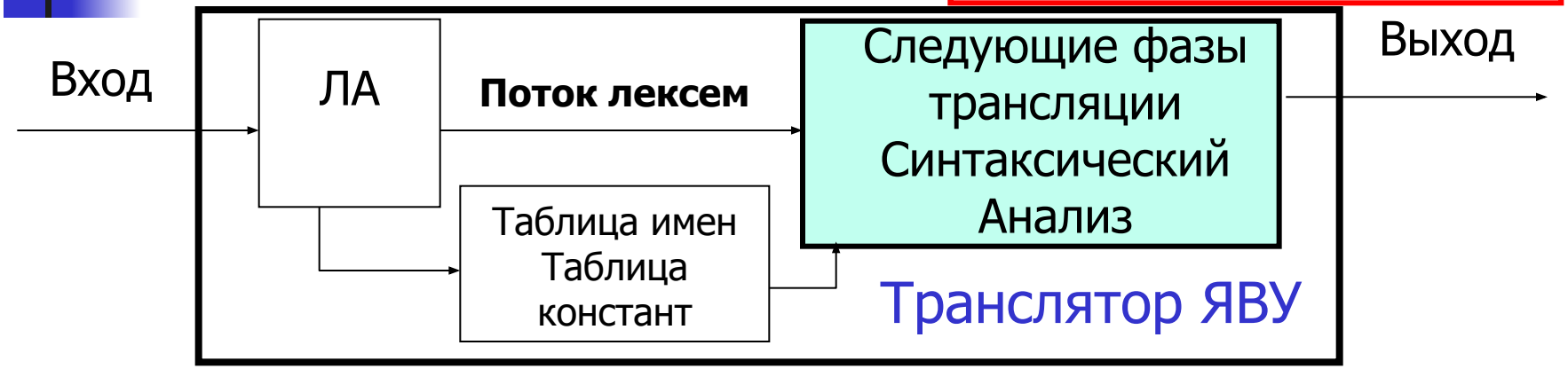
Лексема: | тип | номер |

- y1: символ - в буфер
- y2: это служебное слово? Если нет, то это имя. Проверить/добавить в таблице имен. Выдать соотв. лексему.
- y3: добавить в таблицу констант. Выдать лексему <22, k>
- y4: Выдать лексему <23,0> (**ass** ':=')
- y5: Выдать лексему <24,0> (**semicolon** – ';')
- y6: Выдать лексему <25,0> ('=')
- y7: Выдать лексему <25, 2> ('≥')
- y8: Выдать лексему <25,3> ('>')
- y9: Выдать лексему <25,5> ('<')
- y10: Выдать лексему <25,4> ('≤')
- y11: Выдать лексему <28,0> ('(')
- **... ... ...**

какие ошибки может найти этот ЛА?

Лексический анализатор (ЛА) – первый проход транслятора ЯВУ

Лексема: | тип | номер |



Вход ЛА – цепочка кодов символов клавиатуры:

begin m13 := read ; alpha237 := read ; while m13 < > alpha237 do if m13 > 237 then … – ЛЮБАЯ цепочка из **символов ASCII.** В этой цепочке 120 символов ASCII.

Выход – цепочка кодировок лексем (во внутреннем представлении):

$\underline{\text{begin}}\ i_0\ \underline{\text{ass}}\ \underline{\text{read}}\ \underline{\text{sc}}\ i_1\ \underline{\text{ass}}\ \underline{\text{read}}\ \underline{\text{sc}}\ \underline{\text{while}}\ i_1\ q_1\ i_1\ \underline{\text{do}}\ \underline{\text{if}}\ i_0\ q_2\ i_1\ \underline{\text{then}}$ - 19 лексем

Таблица имен:

N	имя	доп. инф.
0	m 1 3	
1	a l p h a 2 3 7	
2		

Таблица констант:

N	константа	внутр.код
0	2 3 7	1001101 …
1		
2		

Упражнение. Лексический анализ программы языка MiLan

P0::

```
begin
  m:=read;
  n:=read;
  while m!=n do
    if m>n then
       m:=m-n
    else n:=n-m;
write(m)
end
```

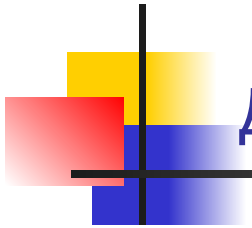
P1::

```
begin
m:=read;   n:=re
ad;
while m!=n do if
m>n then m:=m-n
else  n := n-
m;write ( m)end
```

P2::

```
begin
m:=read;
n:=    re ad;
while m!=   n do
 if m>n then m:=
m-n else n : = n
-m; write(     m
) end
```

- Даны три записи программы на Милане в виде цепочек символов клавиатуры с пробелами. Что будет выдавать ЛА для каждой из них?
- Какая расстановка дополнительных пробелов в программе P0 приведет к обнаружению в ней ошибок лексическим анализатором?
- Какая программа будет обработана ЛА, но в ней обнаружит ошибки синтаксический анализатор?
- Вставьте (уберите) максимальное число пробелов в P0 так, чтобы:
  а) не нарушить синтаксическую корректность P0;
  б) нарушить синтаксическую корректность P0.



Доп. курсовая

- Нужно полностью понимать, как выполняется лексический анализ, уметь по любой входной цепочке символов ASCII построить результирующую цепочку лексем языка Милан.

- **САМОСТОЯТЕЛЬНО:**
  построить ПОЛНЫЙ лексический анализатор языка Паскаль

Лексический анализатор (ЛА) – первый проход транслятора ЯВУ

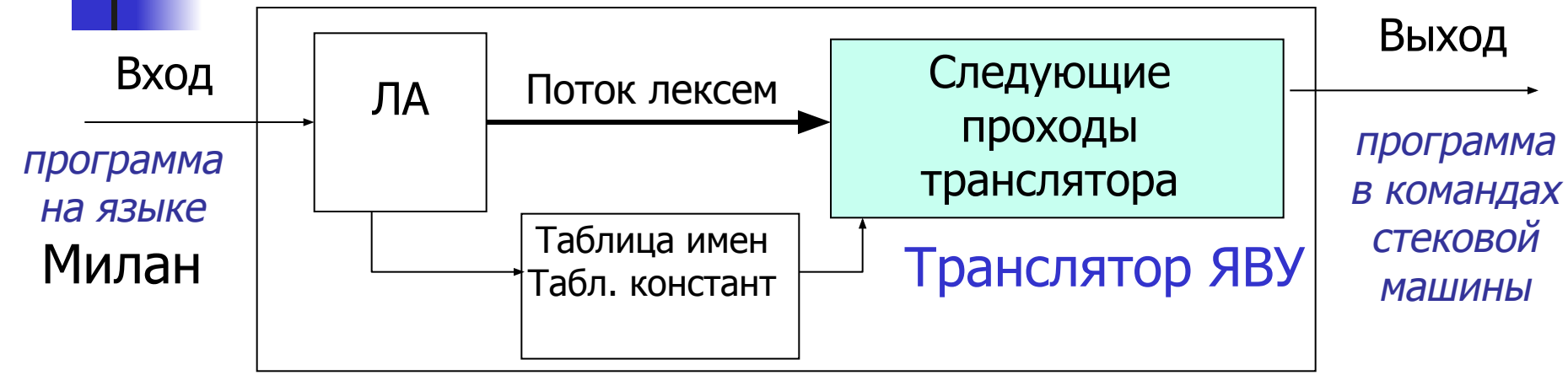


Таблица имен:

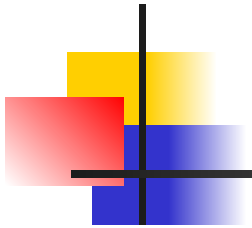
N	имя	
0	m 1 3	
1	a l p h a 2 3 7	
2		

Таблица констант:

N	симв	внутр. предст
0	3 2	… … …
1	1	… … …
2		

Вход следующей фазы трансляции – цепочка кодировок лексем.
Выход – программа СТЕКОВОЙ МАШИНЫ

под переменные отводим первые адреса Памяти Данных
под константы – следующие адреса ПД



- Пример трансляции СЛОЖНОЙ программы Милана в команды стековой машины

Пример результата лексического анализа программы языка Милан

исходная программа:

```
begin
 m:=read; n:=read;
 while m!=n+33 do
  if m>n then m:=m-2;
    write(n+142)
end
```

результат лексического анализа:

begin i_0 *ass read sc* i_1 *ass read sc*
while i_0 q_1 i_1 $addop_0$ c_0 *do if* i_0 q_3 i_1
then i_0 *ass* i_0 $addop_1$ c_1 *sc*
write (c_1 $addop_0$ *c2*) *end*

- *begin*, *read*, *while*, ... – лексемы служебных слов;
- i_k – лексема *имя* с номером k в Таблице Имен, в стековой машине ей отводим адрес k Памяти Данных (ПД);
- c_k – лексема константы с номером k в Таблице Конст, ей отводим адрес Ni +k ПД; (Ni – число имен в программе)
- q_k – лексема '*отношение*' с номером k;
- $addop_k$ – лексема операции '+' при k=0, '-' при k=1;
- $multop_k$ - лексема операции '*' при k=0, '/' при k=1;

q_k – лексема 'отношение'

k=0? '='
k=1? '!='
k=2? '<'
k=3? '>'
k=4? '≤'
k=5? '≥'

Лексический анализатор (ЛА) – первый проход транслятора ЯВУ

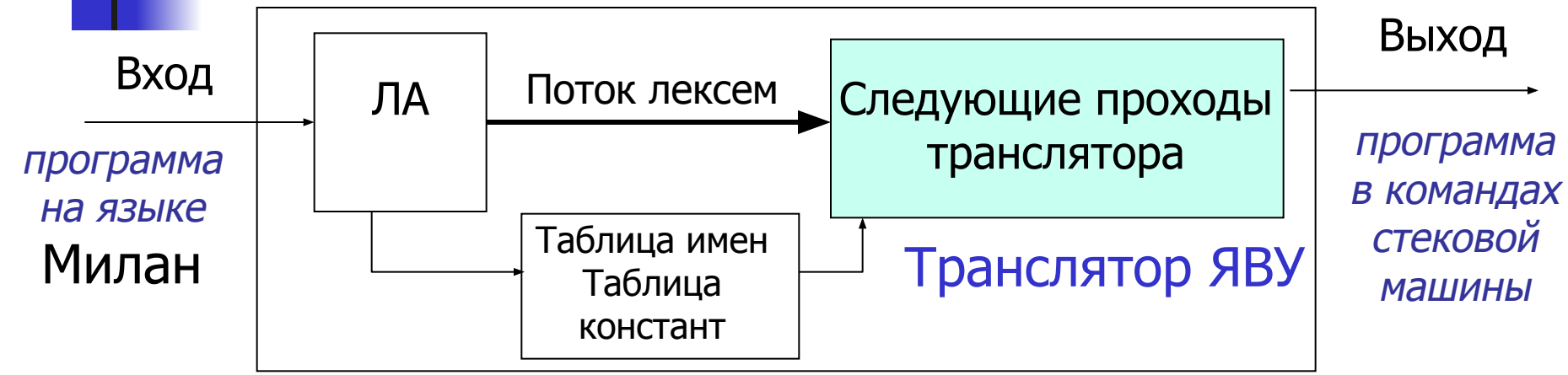


Таблица имен:

N	имя	
0	m 1 3	
1	a l p h a 2 3 7	
2		

Таблица констант:

N	симв	внутр. предст
0	32	… … …
1	1	… … …
2		

Вход следующей фазы трансляции – цепочка кодировок лексем.
Выход – программа СТЕКОВОЙ МАШИНЫ

Язык MiLan представлен, стековая машина описана, что должны получить на выходе транслятора - знаем.

Как строить следующие фазы трансляции?

Пример трансляции программы на Милане в программу стековой машины

исх. программа:

```
begin
 m:=read; n:=read;
 while m!=n+33 do
  if m>n then m:=m-2;
   write(n+142)
end
```

Таблица имен:

N	имя	
0	m	в адр 0 ПД
1	n	в адр 1 ПД

Таблица констант:

N	симв	внутр. предст.
0	3 3	... в адр 2 ПД
1	2	... в адр 3 ПД
2	142	... в адр 4 ПД

адреса переменных 0 и 1,
адреса констант – 2, 3 и 4

программа на языке стековой машины

```
0:  INPUT
1:  STORE   0
2:  INPUT
3:  STORE   1
4:  LOAD    0
5:  LOAD    1
6:  LOAD    2
7:  ADD
8:  COMPARE 1
9:  JUMP_NO 23
10: LOAD    0
11: LOAD    1
12: COMPARE 3
13: JUMP_NO 18
14: LOAD    0
15: LOAD    3
16: SUB
17: STORE   0
18: LOAD    1
19: LOAD    4
20: ADD
21: PRINT
22: JUMP    4
23: STOP
```

ввод m

ввод n

начало цикла

m != n + 33 ?

1 – сравнение по != (не равно)
если нет, то выход из цикла
начало оператора if-then

3 – сравнение m > n ?
если нет, то на write
часть then оператора if-then
m:=m-2

write (n + 142)

последний оператор цикла
возврат на начало цикла
выход из программы в ОС

Как задать язык Милан?

P::

```
begin
  m:=read;
  n:=read;
  while m!=          n
    do
     if m >n
       then
         m:=m-n
       else
         n:= n-m;
         write(m);
end
```

Как задать язык Милан?

Полностью язык Милан можно задать только ГРАММАТИКОЙ. Но язык Милан НЕ автоматный, конечным автоматом его задать нельзя.

Нужна другая грамматика

- Можно ли по приведенным программам полностью понять, что такое язык Милан?
- Нужно ли ставить ‘;’ после оператора write?

  Можно ли НЕ ставить?
- Как строить условия для цикла и условного оператора?

Понимаем ли мы точно, что такое язык Милан?

НЕТ!

```
begin
 m:=read; n:=read;
 while m!=n+33 do
  if m>n then m:=m-2;
       write(n+142*m)
end
```

Язык MiLan НЕ ЗАДАН ФОРМАЛЬНО!

Нужна другая грамматика, более мощная, чем автоматная

- НАПРИМЕР:
  В ЭТОЙ программе оператор while включает write ИЛИ нет?
- Оператор if-then включает write ИЛИ нет??
- Когда выполнять write – внутри цикла, внутри if??

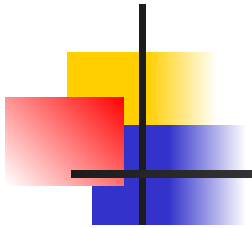
Как задать язык MiLan формально?

Какие другие виды грамматик существуют?
Как задать сложную семантику (например, для оператора цикла или для арифметических выражений)?

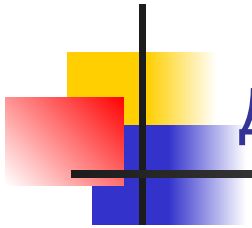
НАМ ПРЕДСТОИТ ДОЛГИЙ ИНТЕРЕСНЫЙ ПУТЬ!

Мы только приоткрыли дверь в эту страну формальных языков и грамматик.

Но мы уже многое умеем!



Дополнительные курсовые: построение трансляторов автоматных языков



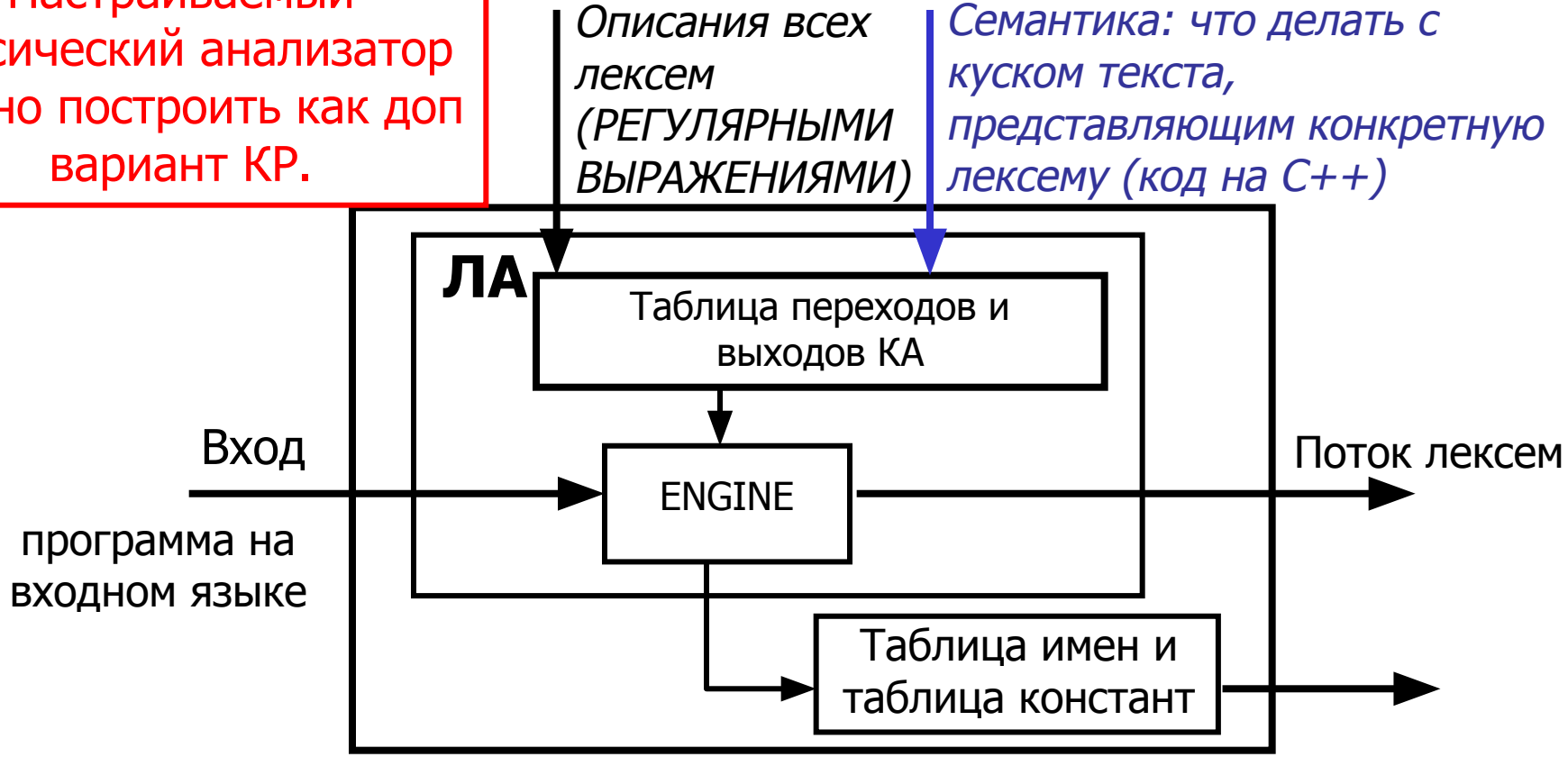
Доп. курсовая

- Построить ПОЛНЫЙ лексический анализатор языка Паскаль

Lex - настраиваемый лексический анализатор

Все лексемы нового языка можно описать (например, КА) и для каждой задать семантику (какую функцию вызывать, если распознана эта лексема – например, обратиться к таблице служебных слов и искать там, если нет, то ...).

Настраиваемый лексический анализатор можно построить как доп вариант КР.



Вход – цепочка символов; выход – цепочка лексем

Доп. курсовая

- Построить транслятор на язык стековой машины

Consider the following program, which computes the greatest common denominator (GCD) of two natural numbers *A* and *B*; when the program terminates, *A* = *GCD* of the two original input values:

L0: start
L1: read A;
L2: read B;
L3: if A = B then goto L10 else goto L4;
L4: if A < B then goto L5 else goto L8;
L5: C:= A;
L6: A:= B;
L7: B:= C;
L8: A:= A-B;
L9: goto L3;
L10: write A;
L11: end.

The program is written in a very simple programming language known as flowchart language, where a label is attached to each instruction. An ordering among labels is assumed, as indicated by their numbering. L0 is called the start label, and L11 is called a terminal label (in principle, there may be several 'termination points' in programs).

Пример транслятора простого языка присваиваний

- Пример программы:

```
begin
    x0 := input;
    x1 := -23;
    x2 := x1*56;
    print (x1, ++x0)
end
```

В выражении не больше одной операции

Варианты расширения языка:

- разные типы (int, real) и операции;
- логические переменные и операторы
- условное присваивание;
- логические переменные и операции;
- инкремент и декремент;
- комментарии;
- не только десятичные числа;
- …

Для самостоятельной проработки

1. Построить транслятор этого языка

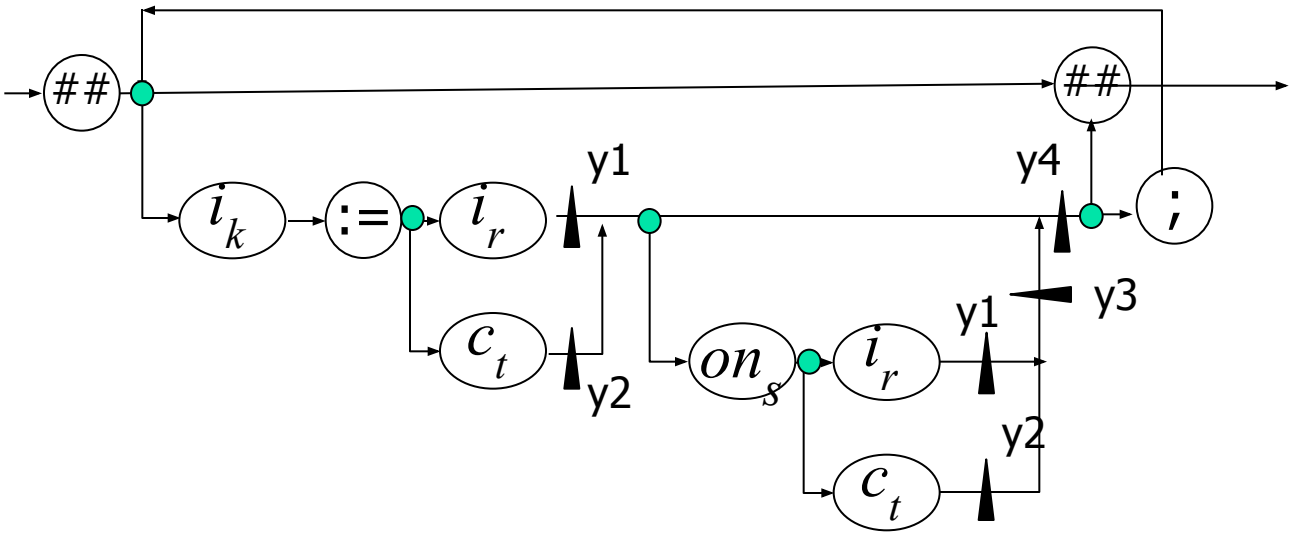
2. В качестве дополнительного варианта можно использовать бестиповые переменные: переменная считается объявленной при первом использовании, тип переменной определяется типом присвоенного выражения. Инициализированной переменной можно присваивать значения типов, отличающихся от первоначально присвоенного

Транслятор языка присваиваний арифметических выражений только с одним либо двумя операндами

Для вычисления
f:= a – b * 25 + d
должны написать программу:

```
##
   x := b * 25;
   y := a – x;
   z := y + d;
   f := z
##
```

Этот язык – автоматный!



Семантики:

y1: Gen: <C:LOAD r>; C++

y2: Gen: <C:LOAD t+Ni>; C++

y3: Gen: <C: ОП s >; C++

y4: Gen: <C: STORE k >; C++

C – счетчик сгенерированных команд стековой машины. Вначале он 0.

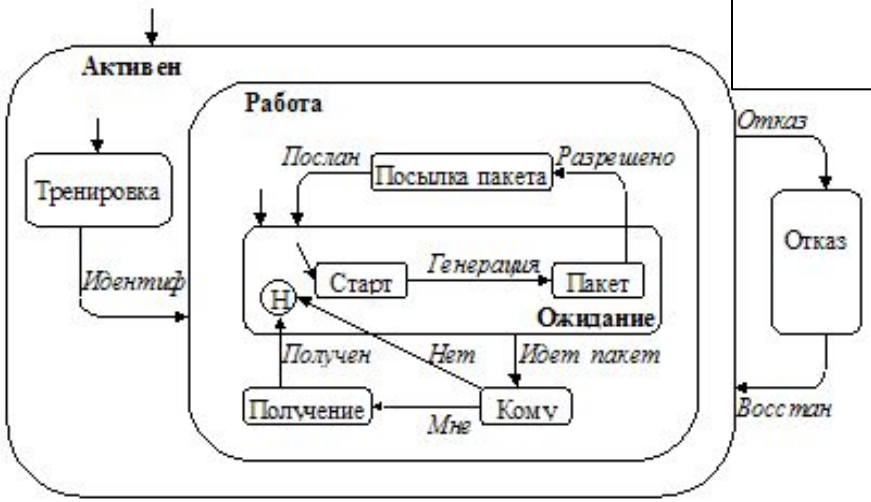
Один из вариантов доп. курсовой работы

- Построить транслятор с графического языка задания алгоритма обработки данных с помощью графа переходов конечного автомата (либо в виде стейтчартов - карт состояний) в программу на языке Java, либо C, либо Паскаль, ...

Вход: стейтчарт или граф переходов конечного автомата

Транслятор

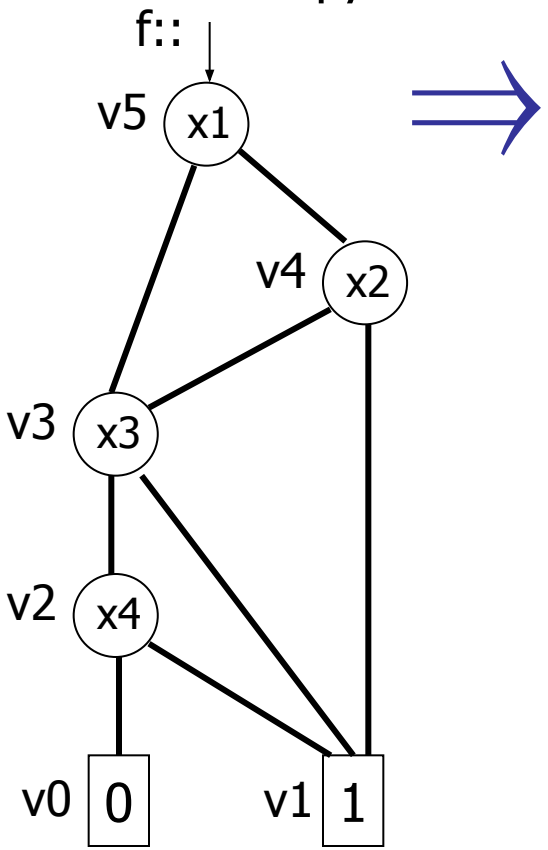
Выход: программа, имитирующая динамику работы автомата




```
begin
  S := STATE_0;    /* установка начального состояния*/
  while true do    /* бесконечный цикл */
      Next(x);     /* в x – очередной входной символ */
      if S= STATE_0 then
          if x=a then y3(); S:= STATE_2; break fi;
          if x=b then y0(); S:= STATE_1; break fi;
          Err0( )    /* ОШ: в сост s0 вход – не a и не b */
      fi;
      if S= STATE_1 then
          if x=a then y3(); S:= STATE_2; break fi;
          if x=b then y1(); S:= STATE_3; break fi;
          Err1( )
       fi;
...   ...
       if S= STATE_3  then
          if x=a then y3(); S:= STATE_1; break fi;
          if x=b then y2();                break fi;
          Err3( )
       fi;
  od;
end
```

Трансляция BDD в программу вычисления значений

- Построить транслятор из описания BDD в программу вычисления значений логической функции



Эквивалентная программа

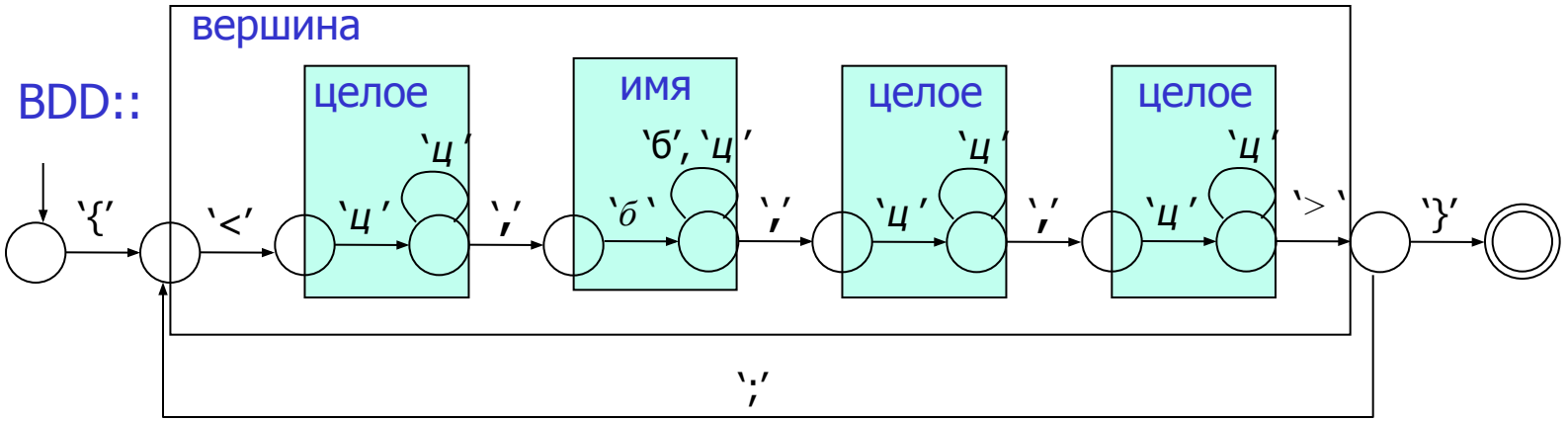
```
v5: if(x1 == 0) goto v3;
    else goto v4;
v4: if(x2 == 0) goto v1;
    else goto v4;
v3: if(x3 == 0) goto v2;
    else goto v1;
v2: if(x4 == 0) goto v0;
    else goto v1;
v1: return (1);
v0: return (0);
```

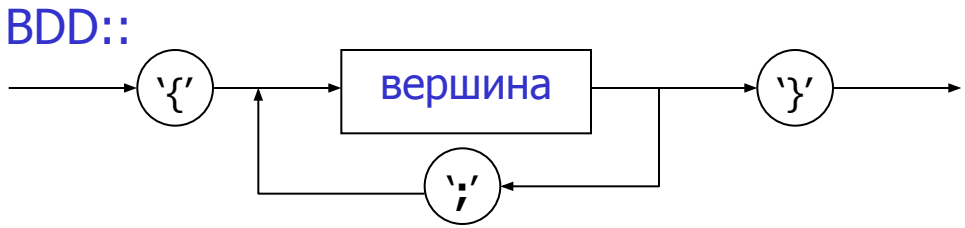
Использовать текстовое представление BDD:

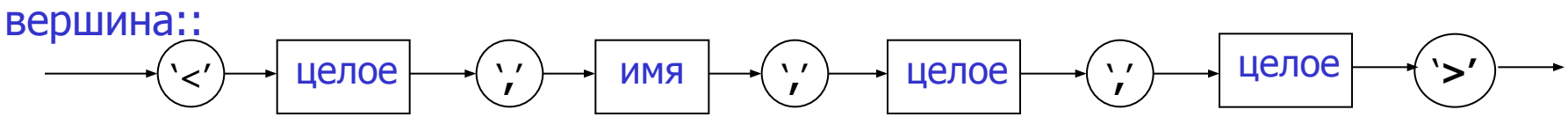
BDD:: f={<5, x1, 3, 4>; <4, x2, 3, 1>; <3, x3, 2, 1>; <2, x4, 0, 1>}

От распознающего автомата к синтаксическим диаграммам

BDD:: f={<5, x1, 3, 2>; <2, x2, 3, 1>; <3, x3, 4, 1>; <4, x4, 0, 1>}

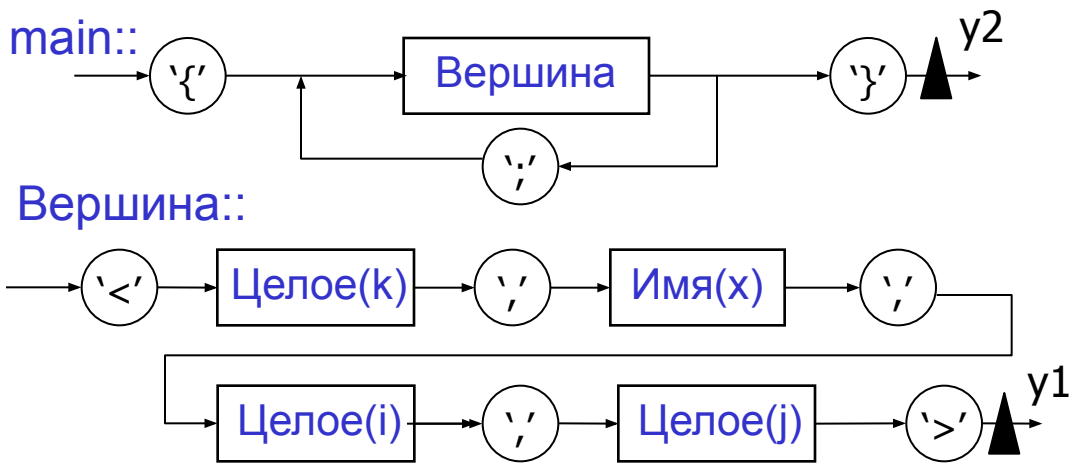






Трансляция BDD в программу вычисления значения булевых функций при заданных значениях аргументов

BDD:: f= {<5, x1, 3, 2>; <2, x2, 3, 1>; <3, x3, 4, 1>; <4, x4, 0, 1>}



Целое(k):: ...

Имя (a):: ...

```
v5: if(x1 == 0) goto v3;
    else goto v4;
v4: if(x2 == 0) goto v1;
    else goto v4;
v3: if(x3 == 0) goto v2;
    else goto v1;
v2: if(x4 == 0) goto v0;
    else goto v1;
v1: return (1);
v0: return (0);
```

Семантика – всего две семантических функции (кроме Целого и Имени) генерации текстов:

y1: Ген(' v_k: if(x == 0 goto v_i; else goto v_j; ')

y2: Ген(' v_1: return (1); '); Ген('v_0: return (0); ')

- По введенной записи логической формулы транслятор выдает программу на языке стековой машины, подсчитывающую значение формулы при заданных значениях переменных.
- Можно использовать пакет Buddy для построения BDD по исходной формуле.

Язык систем линейных алгебраических уравнений

- Пример программы:

Решить систему 3 уравнений:

-3.56 X[1] + 13.0 X[3] = 0.342;

0.236 X[2] + 223.8 X[3] = -233.3;

8.12 X[1] - 6.6 X[2] = 5.1

методом Гаусса.

Как задать все возможные программы?

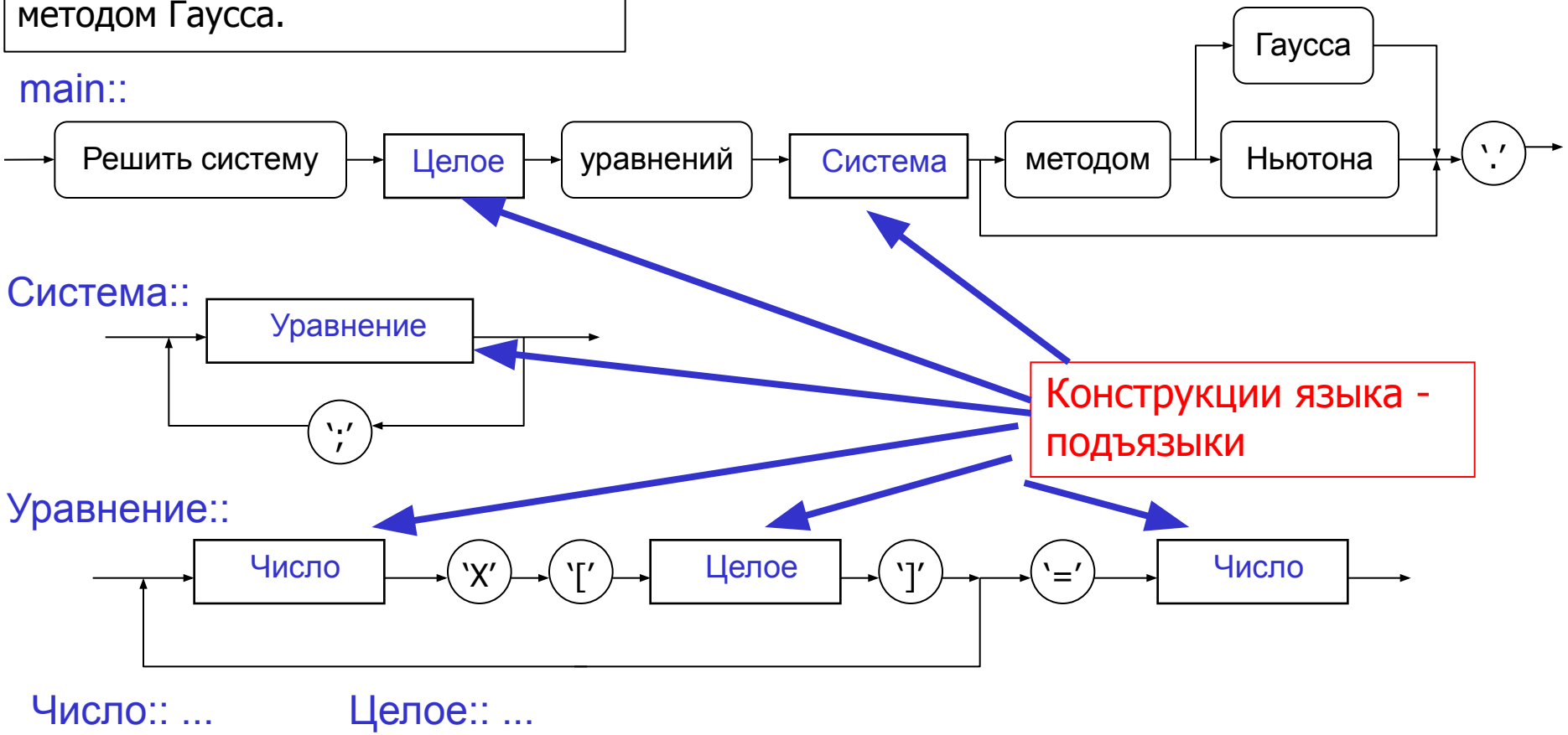
Как описать все возможные системы уравнений?

Грамматикой!

Грамматика языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:
-3.56 X[1] + 13.0 X[3] = 0.342;
0.236 X[2] + 223.8 X[3] = -233.3;
8.12 X[1] - 6.6 X[2] = 5.1
методом Гаусса.

как построить грамматику, задающую все возможные правильные тексты такого вида?



Распознаватель языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:

-3.56 X[1] + 13.0 X[3] = 0.342;

0.236 X[2] + 223.8 X[3] = -233.3;

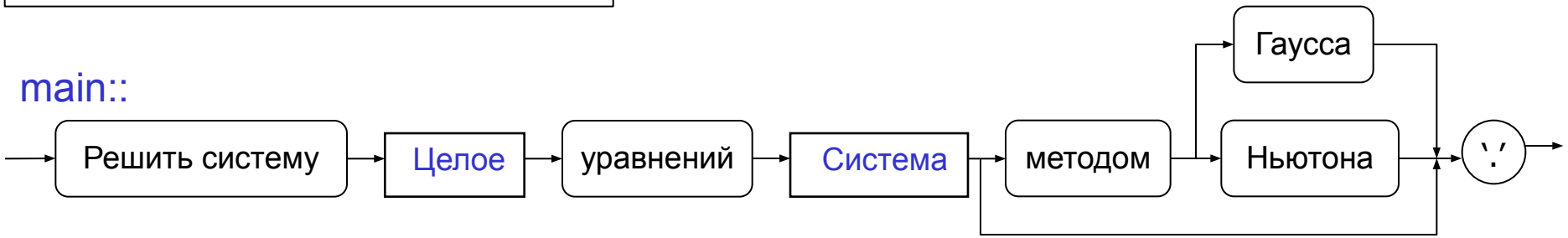
8.12 X[1] - 6.6 X[2] = 5.1

методом Гаусса.

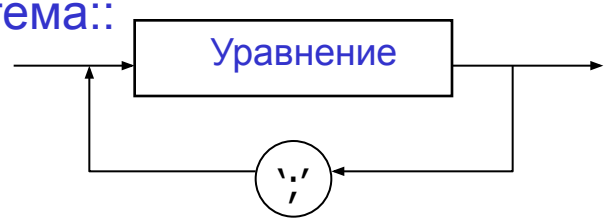
Распознаватель строится однозначно по синтаксическим диаграммам не всегда!

Только тогда, когда выбор в каждом разветвлении синтаксической диаграммы однозначен

main::



Система::



Уравнение::

Число:: ... Целое:: ...

Распознаватель языка систем линейных алгебраических уравнений

```
int Ук:=0;

void main( ) {
 if( s[ Yk ] <> 'Решить систему') then Ош(1) ;
       else Yk +=16; Целое( );
 if( s[ Yk ] <> 'уравнений:') then Ош(2) ;
        else Yk +=10; Система( ); ...
};

void Система( ) {
 m: Уравнение( );
 if( s[Ук] == ';') then  { Ук++;  goto m } else skip
};

void Уравнение( ) {
  m: Число( );
  if( s[Ук] == 'X') then Ук++;    else Ош3( ); );
  if( s[Ук] == '[') then Ук++;     else Ош4( ); ;
  void Целое( );
  if( s[Ук] == ']') then Ук++;     else Ош5( );
  if( s[Ук] <> '=') goto m;   else {Ук++; Число() }
};
```

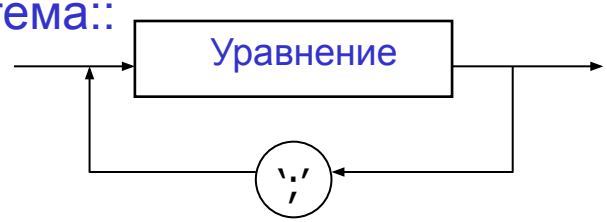
Решить систему 3 уравнений:

$$-3.56\,X[1] + 13.0\,X[3] = 0.342;$$
$$0.236\,X[2] + 223.8\,X[3] = -233.3;$$
$$8.12\,X[1] - 6.6\,X[2] = 5.1$$

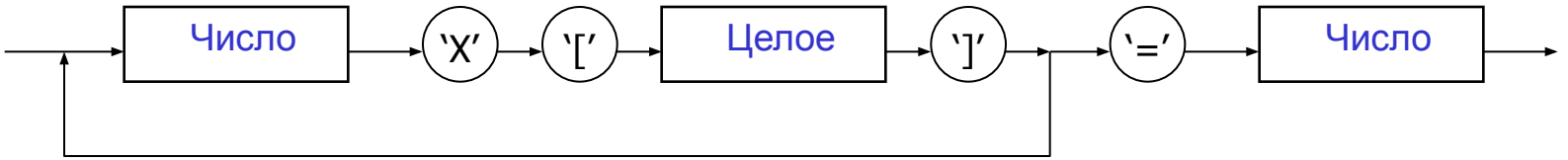
методом Гаусса.

Какие ошибки выявит распознаватель?

Система::



Уравнение::

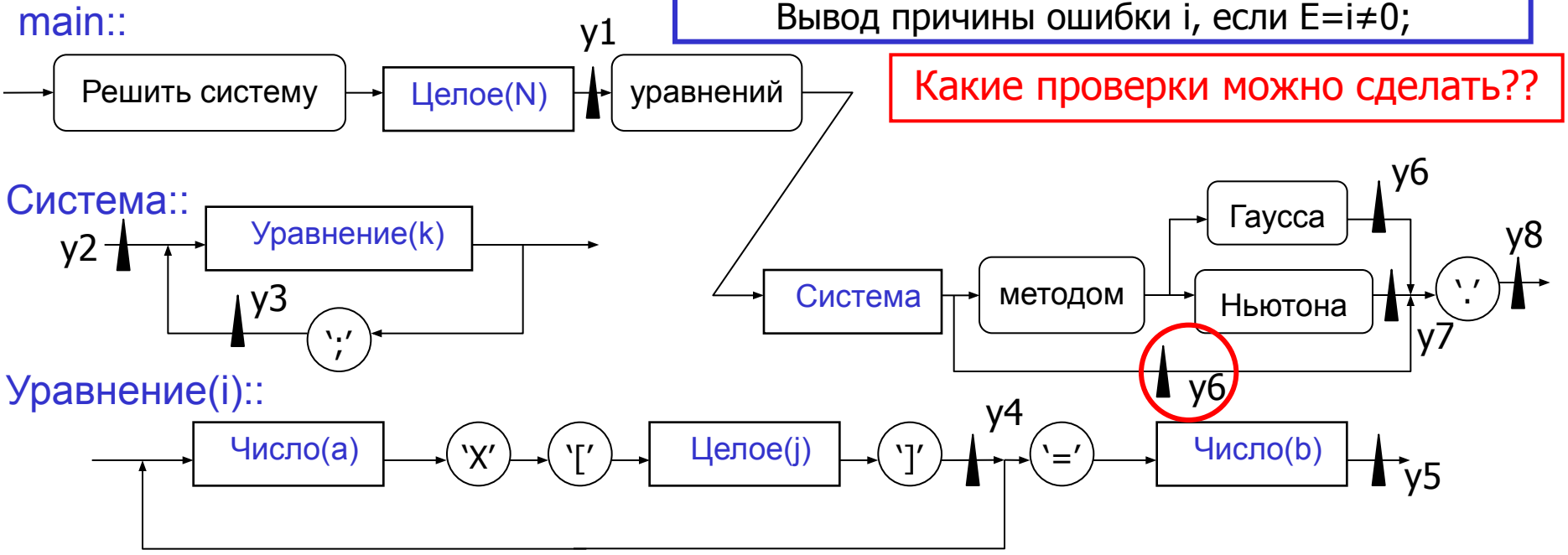


Семантика языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:
-3.56 X[1]+13.0 X[3]= 0.342;
0.236X[2]+223.8 X[3]= -23.3;
8.12 X[1] - 6.6 X[2] = 5.1
методом Гаусса.

y1: Объявить матрицы A[N,N] =0, B[N] и X[N];
y2: k:=1;
y3: k++;
y4: A[i,j] := a;
y5: B[i] := b;
y6: ГАУСС(A, B, N, X, E); (E – тип ошибки)
y7: НЬЮТОН(A, B, N, X, E);
y8: Вывод результата X, если E=0;
Вывод причины ошибки i, если E=i≠0;

Какие проверки можно сделать??

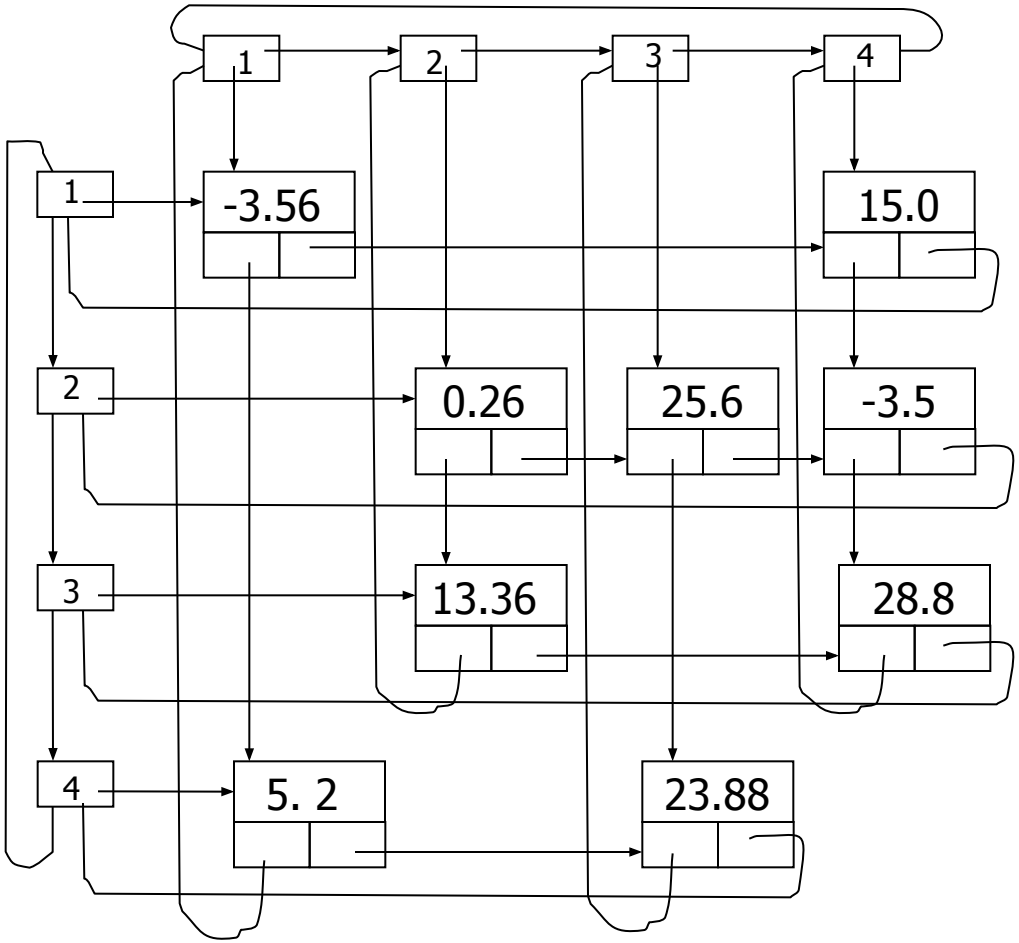


Пример транслятора: Построение структуры данных для разреженной матрицы коэффициентов

Вход – разреженная матрица:

```
-3.56 a[1] + 15.0 a[4];
0.26 a[2]+ 25.6 a[3]-3.5 a[4];
13.36 a[2] +28.8 a[4];
15.2 a[1]+23.88 a[3];
```

Выход – списочная структура

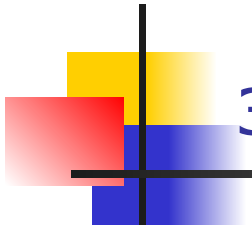


Для самостоятельной проработки

Пример. Интерпретатор калькулятора

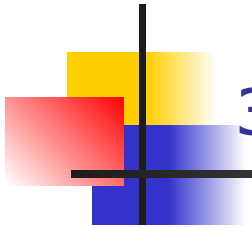
- 1. Реализовать калькулятор телефона Samsung.
  Примеры цепочек:
  3 + 14 – 5 + 6 = // *одноприоритетные операции*
  32 + 2 * 13 * 4 - 5÷2 = // *операции двух разных приоритетов*
  2*(3-1*4)+(5) = // *скобки (вложенные скобки не допускаются)*

- Реализовать калькулятор смартфона SAMSUNG GALAXY NOTE 5 (или какого-нибудь другого смартфона)



Заключение

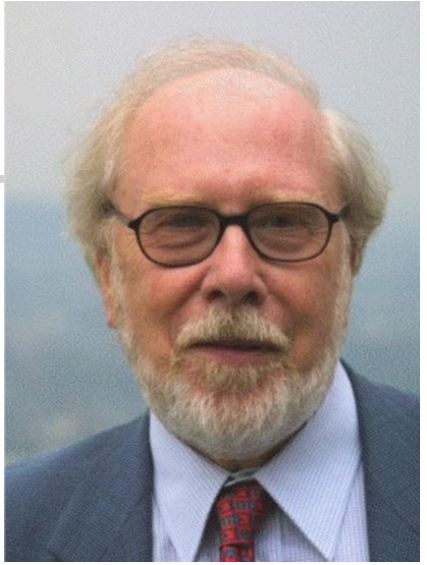
- Модель детерминированного КА используется не только для распознавания, но и для трансляции языков. Для этого на переходах распознающего КА выполняются семантические действия
- Схему распознавателей и трансляторов автоматных языков удобно представлять с помощью синтаксической диаграммы
- Синтаксические диаграммы для сложного автоматного языка удобно строить иерархически, выделяя конструкции языка, и строя синтаксическую диаграмму, задающую каждую конструкцию
- Для трансляции в любое место синтаксической диаграммы может быть помещено семантическое действие
- Для реализации транслятора для каждой конструкции языка по ее синтаксической диаграмме строится своя распознающая процедура. Такая процедура может быть построена по синтаксической диаграмме, которая соответствует детерминированному конечному автомату
- С помощью этого подхода можно строить трансляторы довольно сложных языков. Большинство скриптовых языков - автоматные



Заключение (2)

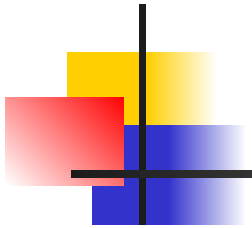
- Лексемы – это минимальные единицы языка, имеющие смысл. Лексемами являются имена, константы, символы, представляемые группами (например, ‘+=‘, ‘++’, ‘:=‘ и т.д.)
- Лексический анализатор выбрасывает комментарии, ‘белые’ символы, и т. п., строит лексемы из нескольких символов (например, служебные слова), представляя программу во внутреннем виде как последовательность лексем (терминальных символов грамматики)
- Лексический анализ выполняется с помощью алгоритма, реализующего функционирование конечного автомата, поскольку лексемы – это автоматные языки

"Мы стоим на плечах гигантов!" Персоналии: Никлаус Вирт



Никлаус Вирт (р.1934) – швейцарский ученый, внесший огромный вклад в теорию и технологию программирования

- Разработал структурное программирование (вместе с Э. Хоаром и Э. Дейкстрой)
- Разработал языки Паскаль, Модула, Оберон, Ада, …
- Разработал пи-код (**код витруальной стековой машины**)
- **Придумал синтаксические диаграммы и использовал их для формального описания синтаксиса языка Паскаль**
- Награды:
  - Премия Тьюринга (1984)
  - ACM Award for Outstanding Contributions to Computer Science Education
  - ACM Outstanding Research Award in Software Engineering (1999)
  - …



Спасибо за внимание